# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

| Fecha | Duración | | Tema | Profesor |
|---|---|---|---|---|
| 25 Feb. | 17 a 21 h | | INTRODUCCION A LA COMPUTACION DIGITAL | M. en C. MARCIAL PORTILLA ROBERTSON |
| 26 Feb. | 9 a 13 h | | LENGUAJE | ING. HERIBERTO OLGUIN ROMO |
| 26 Feb. | 14 a 18 h | | FORTRAN | ING. RICARDO CIRIA MARCE |
| 4 Marzo | 17 a 21 h | | SOLUCION DE SISTEMAS DE ECUACION CON FORTRAN | ING. ARMANDO TORRES FENTANE |
| 5 Marzo | 9 a 13 h | * | APLICACION EN LA COMPUTADORA DIGITAL DEL LENGUAJE FORTRAN | ING. HERIBERTO OLGUIN ROMO |
| 5 Marzo | 14 a 17 h | * | APLICACIONES NO MATEMATICAS (BUSQUEDAS, ACOMODOS) | M. EN C. VERONICA CZITROM |
| 11 Marzo | 17 a 21 h | | APLICACIONES MATEMATICAS | ING. ARMANDO TORRES FENTANE |
| 12 Marzo | 9 a 14 h | * | PROBLEMAS DE INVENTARIOS Y APLICACIONES | M. EN C. MARCIAL PORTILLA ROBERTSON |
| 12 Marzo | 15 a 18 h | * | RUTA CRITICA | DR. VICTOR GEREZ GREISER |
| 18 Marzo | 17 a 21 h | | PROGRAMACION LINEAL EL SIMPLEX Y PROBLEMA DE TRANSPORTE | DR. VICTOR GEREZ GREISER |
| 19 Marzo | | * | APLICACIONES DE UN PAQUETE DE PROGRAMACION LINEAL (FAC. DE ING.) | M. EN C. VERONICA CZITROM |

* Estos temas se darán en el Centro de Cálculo de la Fac. de Ing., UNAM.

'edcs.

# DIRECTORIO DE PROFESORES DEL CURSO: INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA


ING. RICARDO CIRIA MARCE
DEPARTAMENTO DE PROCESAMIENTO
CENTRO DE CALCULO
FACULTAD DE INGENIERIA
UNAM
TEL.: 548.65.60 E.261

M. EN C. VERONICA CZITROM
PROFESOR DE MATEMATICAS
D E S F I
UNAM
TEL.: 548.09.50

DR. VICTOR GEREZ GREISER
PROFESOR TITULAR
INGENIERIA MECANICA Y ELECTRICA
FAC. DE ING.
UNAM
TEL.:50.52.15 E.3750

ING. HERIBERTO OLGUIN ROMO
JEFE DEL CENTRO DE CALCULO
FACULTAD DE INGENIERIA
UNAM
TEL.: 548.65.60 E.261

M. EN C. MARCIAL PORTILLA ROBERTSON
JEFE DE LA SECCION DE COMPUTACION
EDIFICIO DE INGENIERIA MECANICA Y ELECTRICA
FAC. DE ING., UNAM
TEL.:550.52.15 E.3750 y 3746

ING. ARMANDO TORRES FENTANES
PROFESOR
EDIFICIOS DE INGENIERIA MECANICA Y ELECTRICA
FACULTAD DE INGENIERIA
UNAM
TEL.: 550.52.15 E.3751


'edcs. 17, VIII.77.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

## INTRODUCCION A LA COMPUTACION DIGITAL

"Tomado de: Qué hacen las Computadoras y Cómo lo Hacen",
Por IBM, Argentina

Dr. Marcial Portilla Robertson

Febrero-Marzo 1977

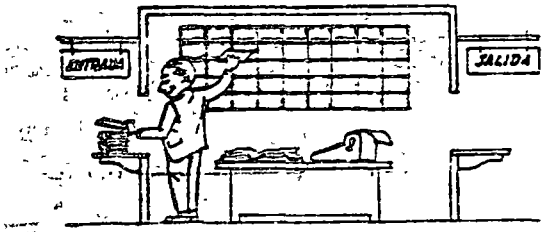
# CONCEPTO DE COMPUTADORA

## OBJETO :

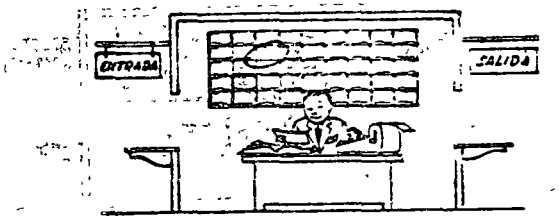
El objeto de esta breve reseña sobre las computadoras electrónicas y sus múltiples aplicaciones al servicio del hombre, es transmitir al lector una completa visión de conjunto, mediante un lenguaje sencillo que permita comprender conceptualmente los temas tratados, sin necesidad de conocimientos previos en la materia.

Esperamos que estas páginas, muy simples en apariencia pero con profundo contenido, permitan, a quienes las lean, ingresar al maravilloso mundo de las máquinas automáticas.

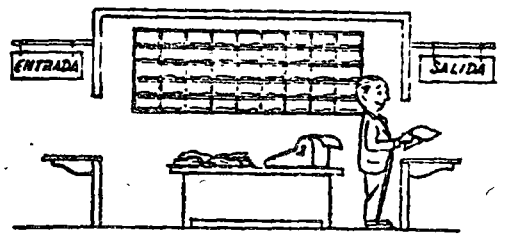

Este señor se llama Control. Trabaja en una pequeña habitación. Tiene a su disposición una máquina de calcular que suma, resta, multiplica y divide. Tiene también el señor Control un archivo parecido al casillero que existe en los trenes para clasificación postal.
Hay, además, en la habitación, dos ventanillas identificadas con sendos carteles: "Entrada" y "Salida".
El señor Control tiene un manual que le indica cómo debe desenvolverse con estos elementos, si alguien le pide que le haga un trabajo.



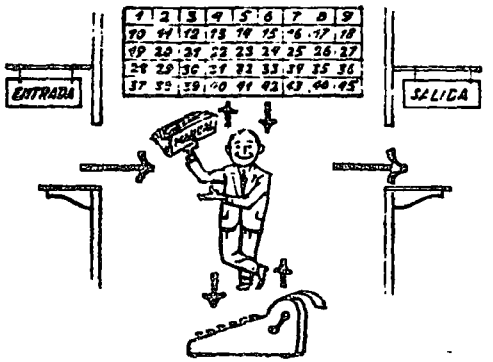

Una persona quiere saber el resultado de un complicado cálculo. Para ello, escribe ordenada, precisa y detalladamente, cada una de las operaciones que, en conjunto, integran ese cálculo, anota cada instrucción elemental en una hoja de papel y coloca todas las hojas en orden en la ventanilla "Entrada".
El señor Control, al ver las hojas, lee en su manual que debe tomar esas hojas con instrucciones, una por una, y colocarlas correlativamente en su archivo. Y así lo hace.




Una vez ubicadas todas las instrucciones en el archivo, el señor Control consulta nuevamente el manual. Allí se le indica que, a continuación, debe tomar la instrucción de la casilla 1 y ejecutarla luego, la de la casilla 2 y ejecutarla, y así sucesivamente hasta ejecutar la última instrucción. Algunas instrucciones indicarán que hay que sumar una cantidad a otra ( instrucciones aritméticas ); otras, que el señor Control debe ir a la ventanilla "Entrada" para buscar algún dato que intervenga en el cálculo ( instrucciones de "entrada/salida" ), dato que la persona que le formuló el problema habrá colocado ya en dicha ventanilla, en otra hoja de papel.
Finalmente, otras instrucciones indicarán que debe elegirse una de entre dos alternativas ( instrucciones lógicas ): por ejemplo, supongamos que una parte del cálculo - desde la instrucción que está en la casilla 5 del archivo hasta la que está en la casilla 9 debe ejecutarse 15 veces porque el cáculo así lo exige .
En tal caso, la instrucción que está en la casilla 10 indicará que, si los pasos 5 a 9 se han ejecutado menos de 15 veces, se debe volver al paso 5. Cuando se hayan realizado las 15 repeticiones y no antes, el señor Control seguirá con la instrucción de la casilla 11.




Después de ejecutar todas las instrucciones del archivo, haciendo con la maquina de calcular las operaciones en ellas indicadas, el señor Control entrega, a través de la ventanilla "Salida", los resultados obtenidos . . . y se sienta a esperar un nuevo trabajo




Obsérvese que la actuación del señor Control es puramente mecánica: sólo sigue las indicaciones de su [illegible] y cumple de acuerdo con ellas las instrucciones que recibe a través de la ventanilla "Entrada". Toma decisiones, pero solamente cuando se le señalan las alternativas que existen y con qué criterio debe elegir una de ellas.
El señor Control puede resolvernos cualquier problema, por complicado que éste sea. Pero para ello debemos indicarle paso a paso, en la forma más elemental y detallada, todo lo que debe hacer para resolverlo, sin olvidarnos absolutamente nada porque, en ese caso, el señor Control no sabría continuar por sí mismo.
Haga el [illegible] la prueba de formular un problema cualquiera de modo tal que una persona que no conozca nada acerca de ese problema pueda resolverlo sin necesidad de hacer consultas. Verá que es una experiencia interesantísima .

El esquema que acabamos de representar mediante el señor Control y sus elementos de trabajo, corresponde exactamente al esquema de funcionamiento de una computadora electrónica.

A continuación presentaremos una breve descripción de los elementos de la computadora que corresponden a los elementos de trabajo del señor Control.

Las unidades de Entrada ( representados por la ventanilla "Entrada" ) :

Son en la computadora, dispositivos capaces de leer información ( Instrucciones o Datos ) con el objeto de procesarla. Existen una gran variedad de elementos de entrada, entre los cuales tenemos :

Tarjetas de Cartulina y Cintas de Papel: Que son perforadas de manera que cada perforación representa un número, una letra ó un símbolo especial de acuerdo con un código predeterminado.

Cintas magnéticas: Conocidas como "memorias externas" tienen la ventaja de permitir almacenar la información en forma mas concentrada — ( a razón de 80 a 2400 caracteres por pulgada de longitud ) y de ser — mas veloces, ya que pueden enviar o recibir información a la unidad de control y velocidades que van de 10,000 a 680,000 caracteres por segundo. Pueden llegar a tener hasta 730 m. de longitud.

Disco Magnético : También conocidos como "Memoria externa" en general tienen un diámetro aproximado de 30 cms. y pueden grabar hasta 400,000 letras, numeros, Y caracteres especiales, formando palabras, cifras, ó registros completos se pueden grabar o leer a razón de 77,000 a 312,000 caracteres por segundo y su tiempo de acceso a un registro alcanza un promedio de 60 mili-segundos.

.....

Una diferencia importante entre las cintas y los discos es la siguiente:

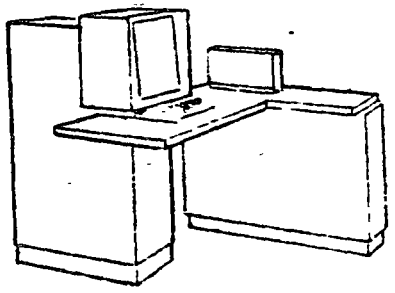
En las cintas los registros se graban o leen secuencialmente.

En los discos se tiene "Libre Acceso" a un registro cualquiera, en forma inmediata, pues cada registro se localiza por su posición física dentro del disco.

Lectora Optica de Caracteres Impresos : Puede leer un documento impreso por una máquina de escribir, o por una máquina de contabilidad o por la impresora de una computadora a una velocidad de 30,000 caracteres por minuto .

Unidad de Representación Visual : Esta unidad de entrada/salida sirve para hacer consultas a la computadora, por medio de un teclado de máquina de escribir, y obtener la respuesta reflejada en una pequeña pantalla de televisión.

La imagen está formada por hasta 12 renglones de hasta 80 caracteres ( letra, numeros, ó signos especiales ) cada uno.



Vemos aquí otra Unidad de Representación Visual, más evolucionada que la anterior, la comunicación hombre-máquina puede establecerse en ella por medio de gráficos, es decir que la entrada y la salida de datos se hacen por medio de imágenes.

. . . . . .

Cuenta esta unidad para ello con un dispositivo con forma de lápiz, que tiene en su punta una célula fotoeléctrica. Un delgado haz de luz parte en determinado momento de un punto de la pantalla y la recorre en forma de zig-zag. Si se apoya el "lápiz" en cualquier posición de la pantalla, su célula fotoeléctrica detectará en algún momento el haz de luz

Por el tiempo transcurrido desde que el haz de luz comenzó su "barrido" hasta que fue detectado, la computadora determina en qué punto de la pantalla se encuentra apoyado el "lápiz".

Como el barrido dura una fracción de segundo y se realizan muchos barridos por segundo, se puede "escribir" con el "lápiz" sobre la pantalla y el dibujo "ingresa" en la memoria de la computadora como una sucesión de puntos codificados.

La pantalla está imaginariamente dividida en 1.040.576 puntos, de manera que los trazos que se obtienen son prácticamente continuos.

Pueden dibujarse así curvas, estructuras, letras, números y cualquier tipo de gráfico, y esa información ingresa automáticamente a la computadora.

Por otra parte, los resultados obtenidos por la computadora son representados en la pantalla también como curva, letras, etc., bajo control del programa almacenado en la memoria.

Lectora Optica de Manuscritos : Salvo algunas pequeñas restricciones en cuanto al formato de los caracteres, esta unidad puede "leer" documentos escritos por cualquier persona y con cualquier ejemplo a una velocidad aproximada de 30,000 caracteres por minuto.

. . . . . .

El registrador/analizador Fotográfico es una Unidad de Entrada/Salida de datos que realiza las siguientes funciones.

1) Registra los resultados de la computadora sobre microfotografías, mediante un tubo de rayos catódicos, que inciden sobre una película fotográfica, y cuyo haz electrónico actúa gobernado por el Programa Almacenado. La película se revela automáticamente dentro de la unidad y 48 segundos después está lista para ser proyectada.
2) Proyecta sobre una pantalla translúcida las microfotografías registradas.
3) Analiza imágenes reproducidas en negativo sobre película transparente, las digitaliza y las transmite a la Unidad Central de Procesamiento.

La película utilizada tiene 30,5 milímetros de ancho y 120 metros de longitud. La Entrada o Salida de imágenes puede consistir en letras, números, símbolos, dibujos, gráficos, mapas, curvas, etc. En una microfotografía de 30,5 mm X 30,5 mm pueden registrarse hasta 30,600 letras y números, o hasta 16.777.216 puntos correspondientes a imágenes. La velocidad de Registración/Análisis es de 40.000 letras, números y símbolos por segundo, o su equivalente si se trata de imágenes.

Maquina de Escribir ( Teletipo ). Las unidades de almacenamiento o memorias ( Representadas por el archivo del señor Control , permiten registrar las instrucciones y los datos para resolver un problema; entre estas se tienen.

. . . . . .

Los Anillos Magnetizantes : Estos pueden magnetizarse en un sentido ó en otro "Recordando" así un 1 o un 0 respectivamente. Con 8 de éstos anillos se forma una posición de memoria, en la cual puede registrarse una letra, un dígito ó un caracter especial ,según las distintas combinaciones de anillos- " En 1 " y "En 0", de acuerdo a un código predeterminado .

Las Memorias de Flip - Flops

Las Cintas Magnéticas

Los Discos Magnéticos

El dispositivo aritmético ( representado por la máquina de cálcular ) realiza las cuatro operaciones aritméticas.

- Las unidades de salida ( representadas por la ventanilla "Salida" ), que pueden ser :

Impresoras

Maquinas de Escribir ( Teletipos )

Grabadoras de Cintas Magnéticas

Grabadoras de Discos Magnéticos

Unidad de Representación Visual

Registrador Analizador Fotográfico

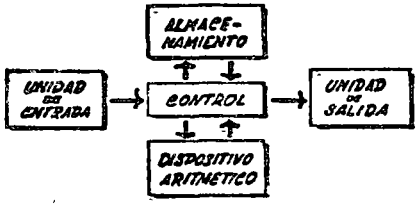
Unidad de Respuesta Oral con la cual la Computadora puede hablar en todo el sentido de la palabra.

Contiene una Cinta magnetofónica en la cual un locutor ha grabado un diccionario de una gran variedad de palabras, en cualquier idioma.

. . . . . .

Finalmente, un dispositivo electrónico de control( representado por el señor control ) ayudado de un programa especial o sistema operativo ( representado por el manual del señor Control ) , gobierna todas las operaciones de todas las unidades que componen la computadora.

Habiendo descrito las partes que componen la computadora podemos mostrar el siguiente esquema que lo representa :




O en forma más resumida :


Siendo :

UNIDAD CENTRAL { ALMACENAMIENTO, CONTROL, DISPOSITIVO ARITMETICO

. . . . . .



Hemos hablado hasta este momento de la computadora electónica desde el punto de vista conceptual. Durante las dos últimas décadas se han producido avances tecnológicos tan extraordinarios en materia de electrónica que la computadora ha sufrido enormes transformaciones. Veamos ahora cómo se ha ido modificando la idea original hasta llegar a las más modernas sistemas de procesamiento de datos.

Las primeras computadoras tenían circuitos con válvulas de vacio. Los tiempos de operación se median en ellas en milisegundos ( milésimas de segundo ) . Cuando aparecieron los transistores, el diseño de los circuitos se mejoró notablemente y la duración de las operaciones en las computadoras que utilizaban esta "Tecnología de Estado Sólido " se midio en microsegundos ( millonésimas de segundo ) .

El hecho de que las nuevas máquinas fueran miles de veces más rápidas que las anteriores trajo aparejada la creación de unidades de entrada, salida y memoria externa mucho más veloces.

La invención de un nuevo tipo de transistor ( "chip " ) provocó una verdadera revolución en los circuitos electrónicos y sus procesos de fabricación el nuevo elemento es tan pequeño que en un dedal de costura caben más de 50,000 chips. Puede observarse en la figura, marcado con un circulo, un circuito completo basado en esta nueva "Tecnología de Lógica Sólida". Debido a su tamaño, se los denomina circuitos microminiaturizados o microcircuitos. Los tiempos de operación se miden ahora en nanosegundos ( milmillonésimas de segundo ) . Ha nacido en esta forma la tercera generación de computadoras, y las altas velocidades alcanzadas posibilitaron un nuevo enfoque en el diseño de los sistemas de procesamiento de datos.

. . . . . .

Enunciaremos brevemente los adelantos que esta tercera generación ha introducido con respecto a la tecnología anterior :

. La computadora se autogobierna y trabaja sin detenerse, pasando de un trabajo a otro sin demora alguna.

. El Operador interviene sólo cuando algún problema excepcional ocurre. La comunicación entre hombre y máquina se realiza sólo sobre la base de "Informes por Excepción" .

. Si ocurre una falla en los circuitos o en la parte electromecánica la máquina realiza un autodiagnóstico e indica cuál es la anomalía.

. La velocidad de Entrada-Proceso-Salida se ha incrementado extraordinariamente.

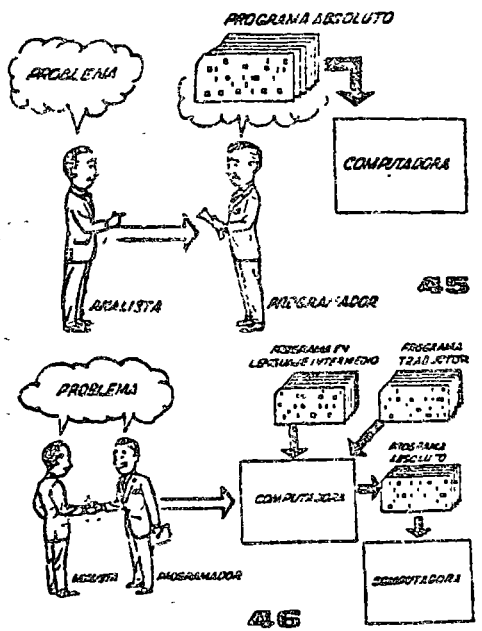
. Todas las operaciones del sistema se realizan en forma simultánea.

. Los lenguajes de programación han evolucionado de manera notable.

. El autocontrol y la autoverificación de operaciones han alcanzado niveles insospechados.

. Pueden realizarse, con máximo rendimiento, varios trabajos distintos simultáneamente.






Hasta ahora hemos visto muchas unidades que, en distintas combinaciones, configuran computadoras electrónicas para las más variadas aplicaciones. Ahora nos detendremos para analizar el manejo de dichos sistemas.

El Programa de Instrucciones almacenado en la Unidad Central de Procesamiento, consta de una secuencia de órdenes y comandos, expresados según una codificación especial denominada "Lenguaje Absoluto de Máquina". Las primeras computadoras se "programaban" en este complejo lenguaje. Había entonces una enorme diferencia entre nuestro idioma y aquél según el cuál debíamos comunicarnos con la máquina. Esto obligaba a un gran esfuerzo común entre el analista que conocía el problema, y el programador que conocía la computadora, pues ambos hablaban del mismo proceso pero en distintos lenguajes.

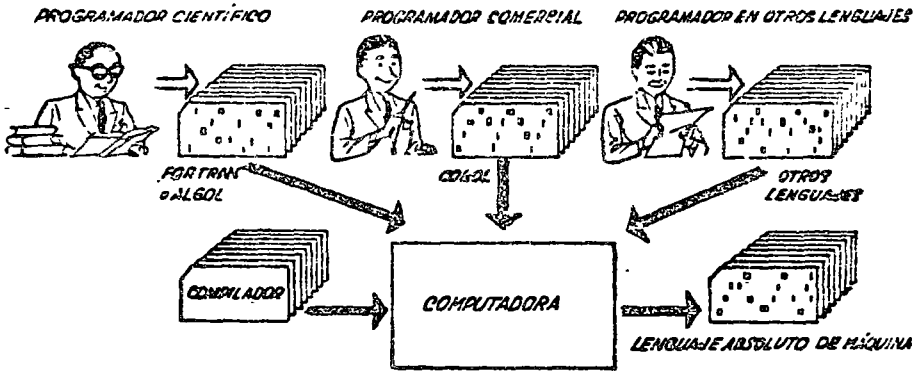
Se crearon, para solucionar el problema, lenguajes intermedios cada vez más parecidos a nuestro idioma. Es decir que cada nuevo lenguaje intermedio se acercaba más al problema y se alejaba más de la máquina. Para cada uno de estos lenguajes se creó un programa traductor llamado "Compaginador" o "Compilador", que tenía la misión de traducir el lenguaje intermedio al absoluto de máquina. Ahora, el analista y el programador "hablan un mismo idioma":
ambos conocen el problema y la solución.

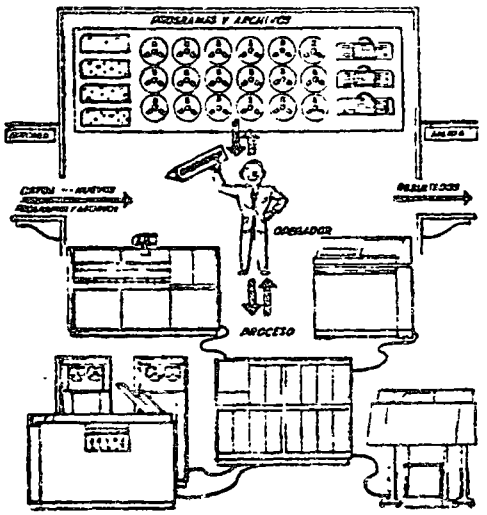
Pero la computadora seguía desarrollándose, y pronto los lenguajes intermedios fueron insuficientes para formular intrincados problemas científicos o comerciales. Nacieron, entonces, lenguajes especializados: dos de ellos, el FORTRAN y el ALGOL, permiten programar problemas científicos-técnicos utilizando una notación casi idéntica a la notación matemática común. El COBOL es un lenguaje comercial cuyas sentencias configuran oraciones y frases en forma tal que una persona que no sabe qué es una computadora, pueda leer un programa y entender perfectamente qué es lo que hará la máquina cuando lo tenga almacenado.

Cada uno de estos lenguajes tiene un programa Compilador para cada tipo distinto de computadora capaz de procesarlo. Esto significa que un programador que sabe FORTRAN, por ejemplo, puede programar una computadora aún sin conocerla. Es decir que estos tres lenguajes constituyen un "esperanto" de las máquinas.-

La tercera generación de computadoras permitió abordar complejos problemas que incluían, entre otros, aspectos comerciales y científicos. No había un lenguaje que abarcara todas las especialidades.
Entonces se reunieron todos los lenguajes conocidos en un superlenguaje llamado PL/1, cuyo compilador es tan poderoso que posibilita la sectorización de la programación en la forma que muestra el dibujo: varios programadores pueden programar distintas partes del proceso, incluso en diferentes lenguajes, y el programa compilador entregará como resultado las instrucciones del proceso completo, en Lenguaje Absoluto de Máquina.
Hemos llegado así a que la computadora nos "entienda", en lugar de que se limite a recibir órdenes en su idioma.



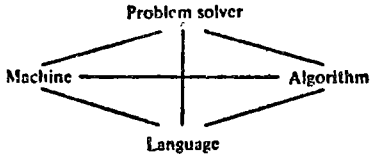


# Chapter 1 Algorithms and computers

Computers arouse curiosity in most of us. Articles in popular magazines and newspapers, current books, and TV shows heighten this curiosity, but such sources cannot be expected to present information in the carefully ordered sequences that is possible in a book like this. Whether you are drawn by curiosity alone, or economic necessity, or both, conscientious study of this book will help you to break through to a new level of understanding about computers, their uses, and their consequences.

Computer science deals with people who have problems to solve and with algorithms, the solutions to these problems. The solutions are expressed in special languages that represent stored data and communicate to machines the manipulations that are to be carried out on that data.




**Elements of computer problem solving.**

Each of these four elements (problem solver, algorithm, language, and machine) affects the others in interesting ways. For example, depending on its richness, a language can either limit or extend our ability to express complex plans of action effectively. And, depending on its capabilities (i.e., its architecture), a machine can execute some plans of action on certain data representations more effectively than on others. The loop of interaction closes when the problem solver changes the plan of action, the language, or the machine architecture to suit his purpose.

This book introduces all four components of this interaction. Every chapter takes you around this "four-cornered race track" and, with every circumnavigation, you gain a deeper and clearer understanding of the interplay among the four elements. You, of course, play the problem solver using a computer. To get the most out of this experience, laboratory practice is almost indispensable. But, even if you can't have actual computer experience, a careful reading of this book should illuminate the computer science scene far better and far beyond what you have previously perceived.

## 1·1 Algorithms and flowcharts

What is an algorithm? An *algorithm* is a list of instructions for carrying out some process step by step. A recipe in a cookbook is an excellent example of an algorithm. The preparation of a complicated dish is broken down into simple steps that every person experienced in cooking can understand. Another good example of an algorithm is the choreography for a classical ballet. An intricate dance is broken down into a succession of basic steps and positions of ballet. The number of these basic steps and positions is very small but, by putting them together in different ways, an endless variety of dances can be devised.

In the same way, algorithms executed by a computer can combine millions of elementary steps, such as additions and subtractions, into a complicated mathematical calculation. Also by means of algorithms, a computer can control a manufacturing process or coordinate the reservations of an airline as they are received from ticket offices all over the country. Algorithms for such large-scale processes are, of course, very complex, but they are built up from pieces, as in the example we will now consider.

If we *can* devise an algorithm for a process, we can usually do so in many different ways. Here is one algorithm for the everyday process of changing a flat tire.

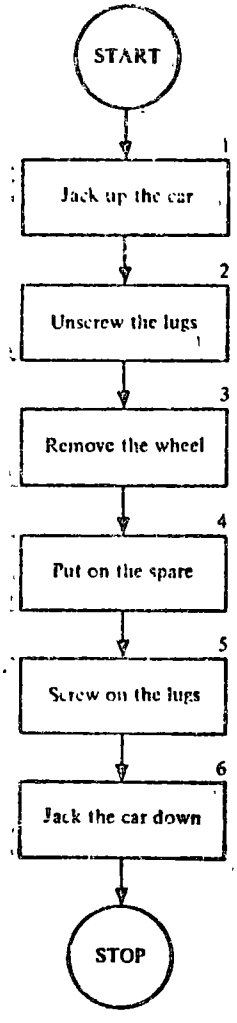
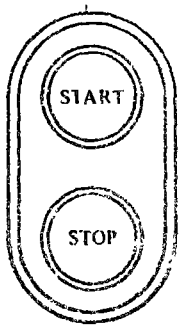


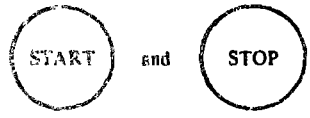

FIGURE 1·1
First flat-tire flowchart.

1. Jack up the car.
2. Unscrew the lugs.
3. Remove the wheel.
4. Put on the spare.
5. Screw on the lugs.
6. Jack the car down.

We could add many more details to this algorithm. We could include getting the materials out of the trunk, positioning the jack, removing the hubcaps, and loosening the lugs before jacking up the car, for example. For algorithms describing mechanical processes, it is generally best to decide how much detail to include. Still, the steps we have listed will be adequate to convey the idea of an algorithm. When we get to mathematical algorithms, we will have to be much more precise.

A *flowchart* is a diagram representing an algorithm. In Figure 1·1 we see a flowchart for the flat-tire algorithm.

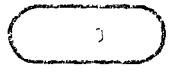



The  and  in the flowchart remind us of the buttons used to start and stop a piece of machinery. Each instruction in the flowchart is enclosed in a frame or "box." As we will soon see, the shape of the frame indicates the kind of instruction written inside. A rectangular frame indicates a command to take some action.

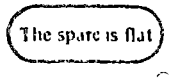



To carry out the task described by the flowchart, we begin at the start button and follow the arrows from box to box, executing the instructions as we come to them.

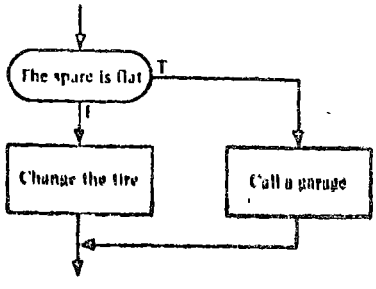
After drawing a flowchart, we always look to see whether we can improve it. For instance, in the flat-tire flowchart we neglected to check whether the spare was flat. If the spare *is* flat, we will not change the tire; we will call a garage instead. This calls for a decision between two courses of action. For this purpose we introduce a new shape of frame into our flowchart.



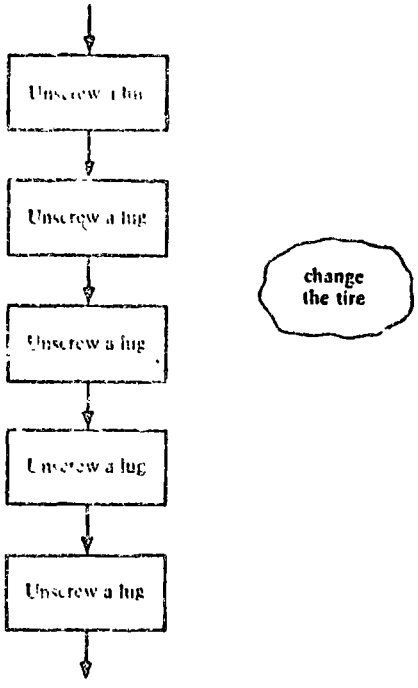
Inside this oval frame we will write an assertion instead of a command.

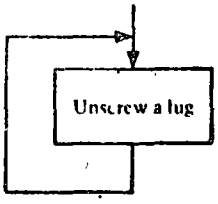



This is called a *decision box* and will have two exits, labeled T (for true) and F (for false). After checking the truth or falsity of the assertion, we choose the appropriate exit and proceed to the indicated activity. Incorporating the flowchart fragment on the left into Figure 1·1, we obtain the flowchart in Figure 1·2.



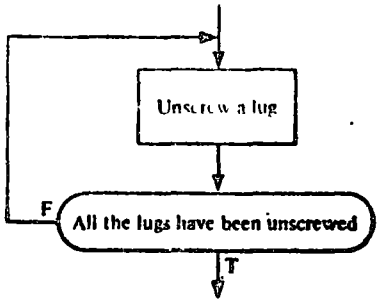

There is another instructive improvement possible. The instruction in box 2 of our flowchart actually stands for a number of repetitions of the same task. To show the additional detail we could replace box 2 by a step for each lug:




The awkwardness of this repeated instruction can be eliminated by introducing a *loop*.

As we leave the box, we find that the arrow leads us right back to repeat the task again. However, we are caught in an endless loop, since we have provided no way to get out and go on with the next task. To correct this situation, we require another decision box, as shown on the left.




Replacing box 2 of our flowchart with this mechanism and making a similar replacement for box 5, we get the final result shown in Figure 1·3.

Now that you have followed the development of the flat-tire flowchart, try to devise one of your own. In the algorithm of the following exercise, you will probably discover some decisions and loops. There are many different ways of flowcharting this algorithm, so many different-looking flowcharts will be created.

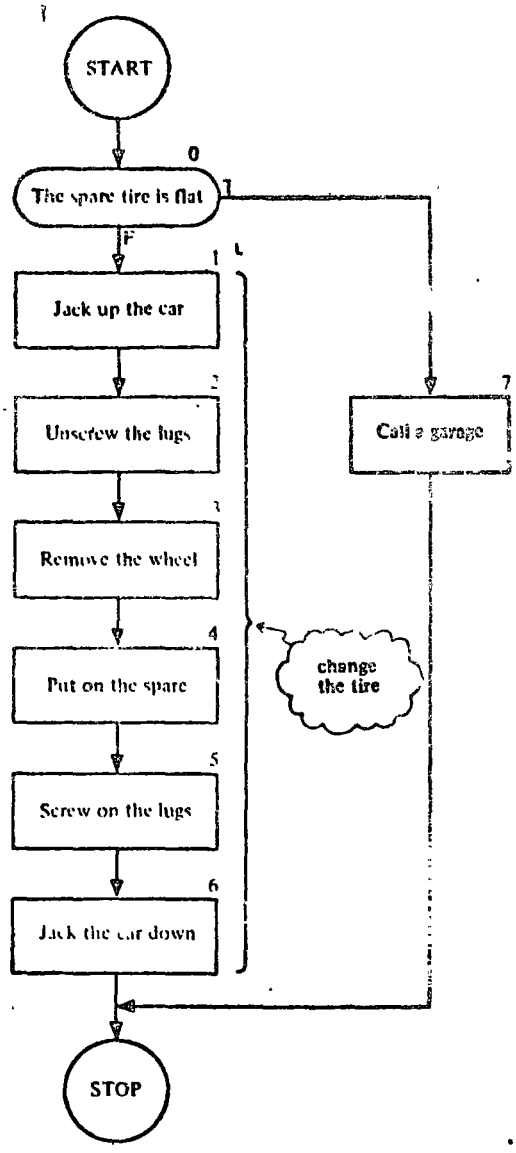



FIGURE 1·2
Second flat-tire flowchart.

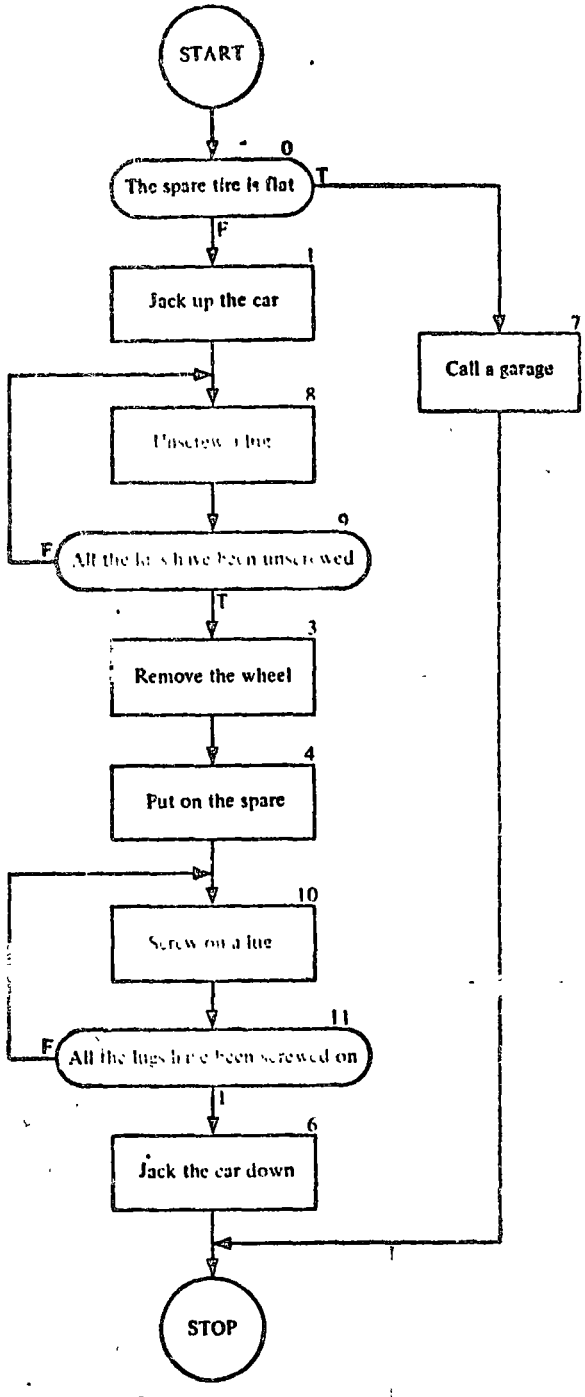



FIGURE 1·3
Final flat-tire flowchart.

EXERCISE 1·1

1. Prepare a flowchart representing the following recipe.

*Mrs. Good's Rocky Road*

Ingredients:

| | |
|---|---|
| 1 cup chopped walnuts | ½ cup evaporated milk |
| ¼ pound block of baker's chocolate | ½ cup corn syrup |
| ½ pound of marshmallows cut in halves | 1 teaspoon of vanilla |
| 3 cups sugar | ¼ pound of butter |
| | ½ teaspoon of salt |

Place milk, corn syrup, sugar, chocolate, and salt in a four-quart pan, and cook over a high flame, stirring constantly until the mixture boils. Reduce to medium flame and continue boiling and stirring until a drop of syrup forms a soft ball in a glass of cold water. Remove from the flame and allow to cool for 10 minutes. Beat in butter and vanilla until thoroughly blended. Stir in walnuts. Distribute marshmallow halves over the bottom of a 10-inch square, buttered baking pan. Pour syrup over the marshmallows. Allow to cool for 10 minutes. Cut in squares and serve.

## 1·2 A numerical algorithm

Now we are ready to examine an algorithm for a mathematical calculation. As a first example, we consider the problem of finding terms of the Fibonacci sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . .

In this sequence, or list of numbers, the first two terms given are 0 and 1. After that, the terms are constructed according to the rule that each number in the list is the sum of the two preceding ones. Check that this is the case. Thus, the next term after the last one listed above is

$$34 + 55 = 89$$

Clearly, we can keep on generating the terms of the sequence, one after another, for as long as we like. But, in order to write an algorithm for the process (so that a computer could execute it, for example), we have to be much more explicit in our instructions.

Before subjecting this process to closer scrutiny, let us review a little of the interesting history of this sequence. It was introduced in 1202 A.D. by the Italian mathematician, Fibonacci, to provide a model of population growth in rabbits.

His assumptions were: (1) it takes rabbits one month from birth to reach maturity; (2) one month after reaching maturity, and every month thereafter, each pair of mature rabbits will produce another *pair* of rabbits; and (3) rabbits never die.

One senses that this model is not completely realistic. But the essence of mathematical modeling is to start with a crude model that emphasizes the important aspects of the situation and suppresses less important information. A more refined model can be developed later, profiting from the experience with the crude model. Thus we might eventually improve the Fibonacci model by obtaining more accurate figures on the birth rate, taking mortality into account, considering the limitations of food supply, the effects of predators, disease, and overcrowding, and the like.

In spite of its frivolous origins, the Fibonacci sequence has many fascinating properties and plays a role in the solution of a number of seemingly unrelated mathematical problems. There is currently a published quarterly journal entirely devoted to the properties and applications of the Fibonacci sequence.

After this long digression, let's see how the rabbit-pair population model gives rise to the Fibonacci sequence. Fibonacci starts with one pair of newborn rabbits at the beginning of month *one*, and he then lets nature take its course. This is shown in Table 1·1, which we now explain.

TABLE 1·1
Rabbit Population

| Beginning of Month | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Infant rabbit pairs | 1 | 0 | 1 | 1 | 2 | 3 | 5 | 8 |
| Mature rabbit pairs | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 |
| Total rabbit pairs | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 |

Look at the arrows in the table. The number of pairs of infant rabbits in any month (after the first) is equal to the number of pairs of mature rabbits in the preceding month (condition 2 in the Fibonacci model). This explains the green arrows. In each month after the first, the number of pairs of mature rabbits will equal the total number of rabbit pairs in the preceding month (condition 1 in the Fibonacci model). This

explains the gray arrows. Following the arrows, we see that, from the third month onward, the total in any month is the sum of the totals in the two preceding months. Thus the rabbit population model generates the Fibonacci sequence except for the initial zero, which can be taken as the total number of rabbits in month zero.

Eliminating the reference to rabbits, we can tabulate the calculation of the terms of the Fibonacci sequence in Table 1·2.

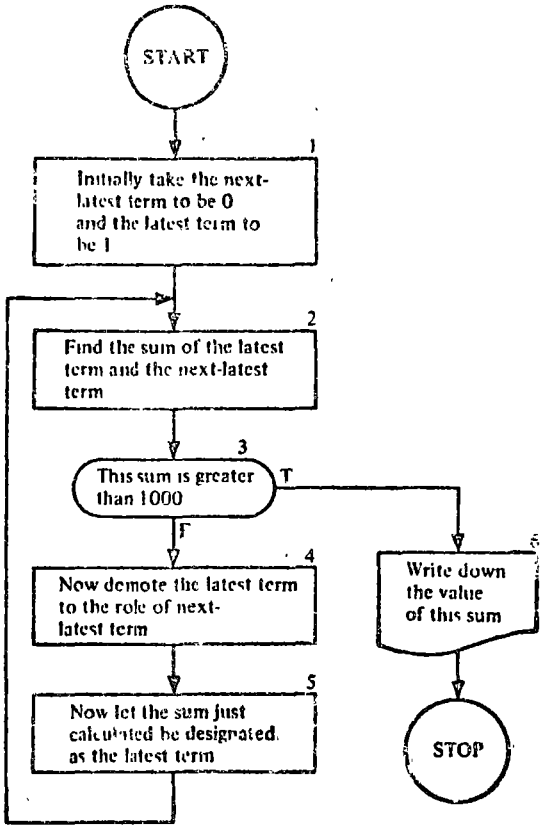



FIGURE 1·4
Flowchart for Fibonacci sequence.

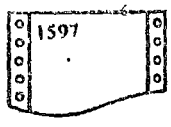
TABLE 1·2

| Next Latest Term | Latest Term | Sum |
|---|---|---|
| 0 | 1 | 0 + 1 = 1 |
| 1 | 1 | 1 + 1 = 2 |
| 1 | 2 | 1 + 2 = 3 |
| 2 | 3 | 2 + 3 = 5 |
| 3 | 5 | 3 + 5 = 8 |
| 5 | 8 | 5 + 8 = 13 |
| 8 | 13 | 8 + 13 = 21 |

We can see that in each step the latest term gets "demoted" to the role of next latest term and the sum becomes the new latest term.

Let's construct a flowchart for finding the first term to exceed 1000 in the Fibonacci sequence (Figure 1·4).

After going through the loop of flowchart boxes numbered 2 to 5 enough times (it happens to be 15 times), we eventually emerge from box 3 at the T exit and proceed to box 6. This box is seen to have a different shape because it calls for a different kind of activity—that of writing down our answer. The shape is chosen so as to suggest a page torn off a line printer, once the most common of computer output devices.




EXERCISES 1·2

1. (a) Suppose in the rabbit problem we had started in month one with one pair of infant rabbits and three pairs of mature rabbits. Make a table similar to Table 1·1 to show the state of the population over the first eight months.
   (b) How would you modify the flowchart of Figure 1·4 so as to generate the first term of this modified sequence greater than 1000?
2. Repeat Problem 1 with three pairs of infant rabbits and one pair of mature rabbits.
3. (a) For the Fibonacci sequence in Table 1·1, calculate from month two through month twelve the ratio, r, of the total number of

rabbits in the current month to that in the preceding month. Express each ratio as a decimal and carry out the calculation to the nearest thousandth.

(b) Express in your own words what seems to be happening to these ratios.

(c) Find the reciprocals of each of the ratios in Problem 3a.

(d) What relationship between the ratio r and its reciprocal 1/r seems to be becoming more and more true? Express this relationship as an equation.

(e) If this relationship held exactly, what would be the exact value of r? That is, solve the equation for r.

4. Repeat Problem 3 using:

(a) The table in Problem 1.

(b) The table in Problem 2.

## 1·3 SIMPLOS, a conceptual model of a computer

The algorithm of the preceding section can be expressed in much simpler notation that is, at the same time, more nearly acceptable by a computer as a set of instructions. To do this we must introduce a conceptual model of how a computer works. This conceptual model is so extraordinarily simple that we will call it the SIMPLOS computer. It is amazing, but true, that such a simple view of how a computer works is completely adequate for this entire course. We will present a more realistic picture of a computer in later sections of this chapter.

### Variables

In computing work, a *variable* is a letter or a string of letters used to stand for something. For now, this "something" that a variable stands for will always be a number. (As we progress through this book, we will take an ever broadening view of the sort of thing a variable can stand for.) In the formula

$$A = L \times W$$

the letters A, L, and W are variables. In the formula

$$DIST = RATE \times TIME$$

DIST, RATE, and TIME are variables.

At any particular time, a variable will stand for one particular number, called the *value* of the variable, which may change from time to time during a computing process. The value of a variable may change millions of times during the execution of a single algorithm.

In our conceptual model of a computer we associate with

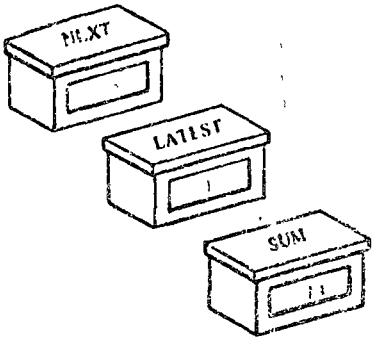



FIGURE 1·5
Storage.

each variable a *storage box*. On the top of each box there is a removable gummed sticker with the associated variable inscribed on it, and inside the box there is a strip of paper with the *present value* (or current value) of the variable written on it. The variable is a *name for the number* that currently appears inside.

Each box has a lid that may be removed when we wish to assign a new value to the variable. Each box has a window in the side so that we may read the value of a variable with no danger of altering its value. These boxes constitute the *storage* of our computer. In Figure 1·5 we see one stage in the execution of the Fibonacci sequence algorithm of the preceding section. Here NEXT stands for "next latest term" and LATEST stands for "latest term."

To summarize, the *data storage* of a computer is to be thought of as subdividable into a number of information containers or boxes. Each such storage box may be given a meaningful name (sticker), and each may be given (assigned) a value.

Some people view a computer as an electronic and mechanical system having a data storage similar to that just described, along with a number of other interconnected units or modules, each with a special set of functions that, when activated appropriately, carry out algorithms. Figure 1·6 is one way to depict the organization of such a computer system. If we were to pursue the explanation of this system according to the module view, it would be necessary to define the functions of each module and explain the significance of the arrowed lines into and out of each box. But it would also be

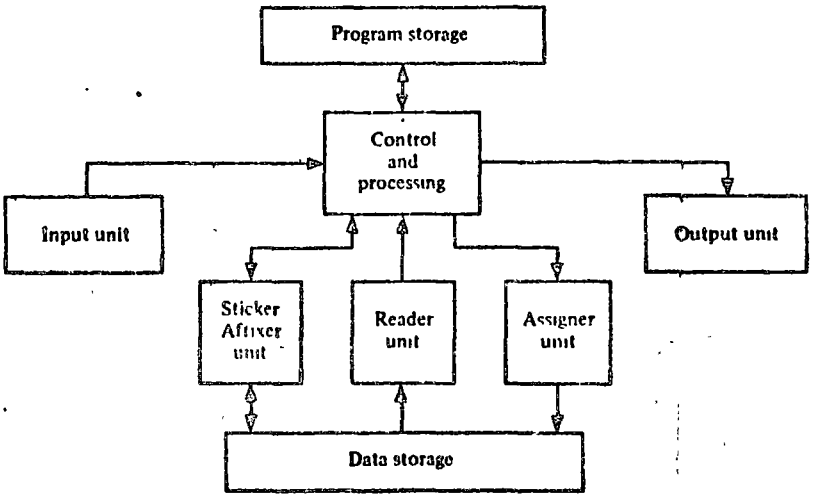



FIGURE 1·6
View of a SIMPLOS system as a set of interconnected modules.

necessary to bring the diagram to life by explaining the action sequences that occur in which each module serves the needs of the others so that the overall effect is to process information (i.e., to compute) in the desired fashion.

A second way to view a computer is to picture the active modules as robots working as a team. The actions of each robot always follow a fixed pattern, according to a set of relatively simple rules. We shall take this view in our conceptual model, SIMPLOS.

## The Model and How It Works

We visualize a computer as a number of storage boxes together with a staff of four robots—the *Master Computer* and three assistants, the *Assigner,* the *Reader,* and the *Sticker Affixer.* All these components are quartered in one room, isolated from those who will use the computer.

The Master Computer corresponds to the control and processing unit in Figure 1·6. He has a flowchart on his desk that sets forth the instructions according to which he delegates certain tasks to his assistants (Figure 1·7). "Note that the flowchart corresponds to the information kept in the *program storage* module of SIMPLOS."

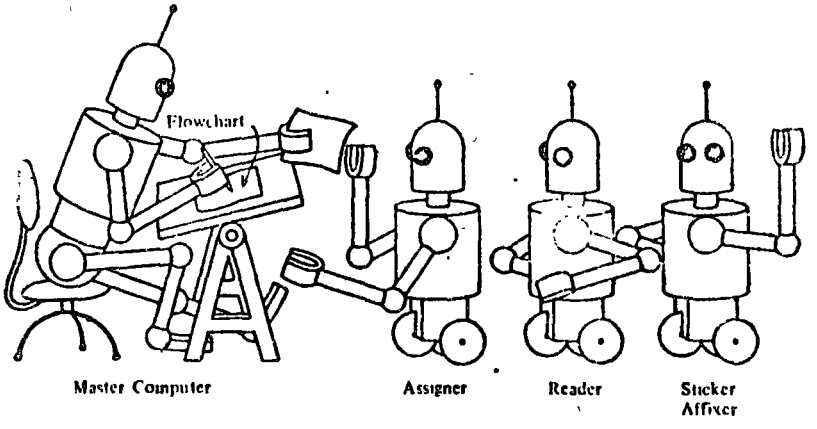
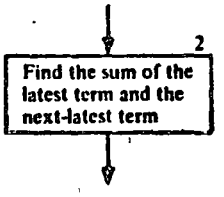
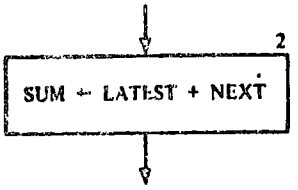



**FIGURE 1·7**
**The Master Computer and his staff.**

To see how this team operates, let us suppose the computer is in the midst of executing the Fibonacci sequence algorithm of Figure 1·4. One of the instructions in this algorithm was:

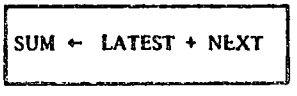


In a simplified flowchart notation, this instruction will take the form:




Inside this flowchart box we find an *assignment statement.* Reading this statement aloud, we would say, "Assign to SUM the value of LATEST plus NEXT," or more simply, "Assign LATEST + NEXT to SUM." The arrow pointing left is called the *assignment operator* and is to be thought of as an order or a command. Rectangular boxes in our flowchart language will always contain assignment steps and will therefore be called *assignment boxes.*

To see what takes place when the Master Computer comes to the above statement in the flowchart, let us assume that the variables LATEST and NEXT (but not SUM) have the values seen in Figure 1·5. The computation called for in the assignment statement is spelled out on the right-hand side of the arrow, so the Master Computer looks there first.




He realizes that he needs to know the values of the variables LATEST and NEXT, so he sends the Reader out to fetch copies of these values from storage.

The Reader then goes and finds the storage boxes labeled

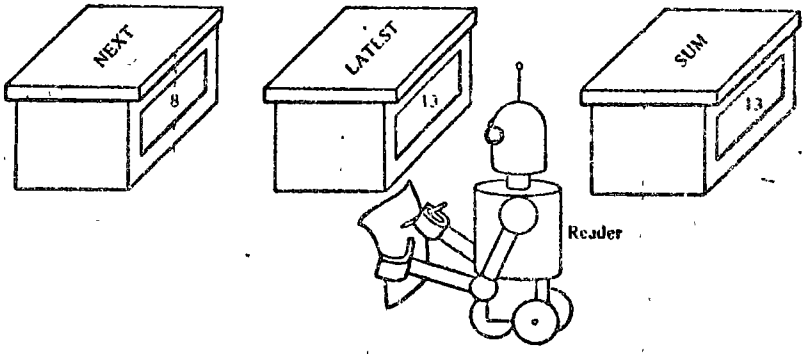



FIGURE 1·8
The Reader copying a value from storage.

LATEST and NEXT. He reads the values of these variables through the windows (Figure 1·8), jots down the values, and carries them back to the Master Computer (Figure 1·9).

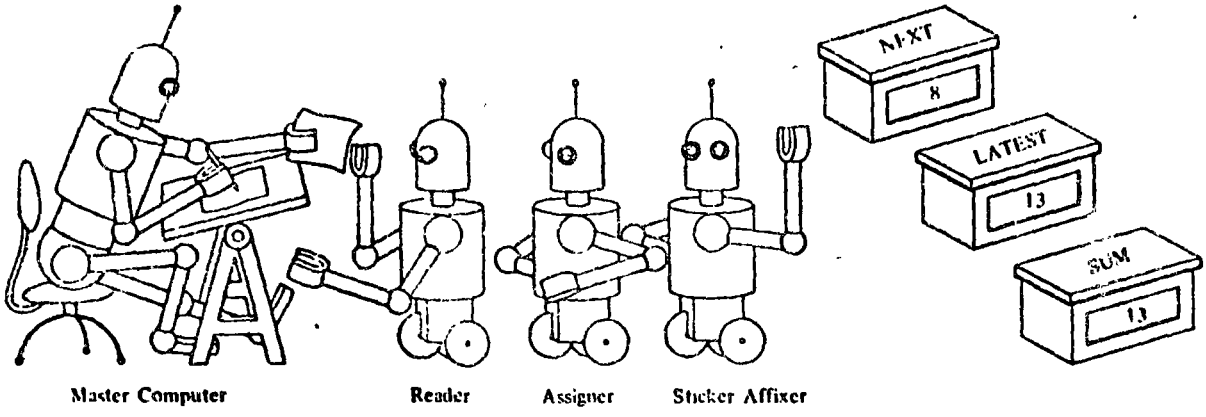



FIGURE 1·9
The Master Computer receives the copy.

The Master Computer computes the value of LATEST + NEXT using the values of these variables brought to him by the Reader:

$$8 + 13 = 21$$

What does he do with this value?

The Master Computer now looks to the left of the assignment arrow in his instruction.

```
SUM ← LATEST + NEXT
```

He sees that he must assign the computed value of LATEST + NEXT, namely, 21, to SUM so he writes "21" on a slip of paper, calls the Assigner, and instructs him to assign this value to the variable SUM.

The Assigner goes to storage, finds the box labeled SUM, and dumps out its contents (Figure 1·10). Then he places in the box the slip of paper containing the new value, closes the lid, and returns to the Master Computer for a new task.

In other words, assignment is the process of giving a value to a variable. We say that assignment is *destructive* because it displaces the former value of the variable. Reading is *nondestructive* because the process in no way alters the values of any of the variables in storage.

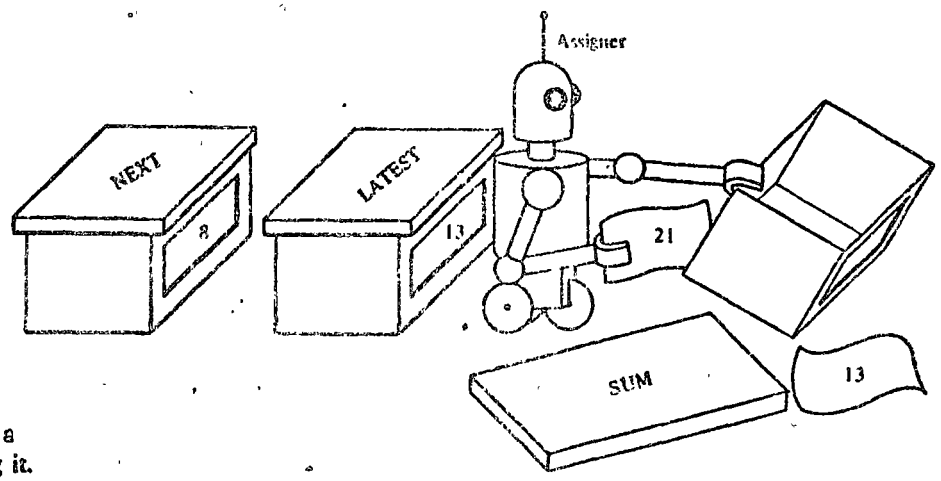



FIGURE 1·10
The Assigner emptying a storage box and refilling it.

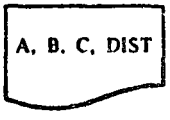
In Figure 1·11 we present the entire flowchart of Figure 1·4 in simplified flowchart language. The old and new flowcharts are placed side by side for easy comparison.

The translation requires very little explanation. It should be obvious that the statement in box 1 on the left is equivalent to the two statements in box 1 on the right. The new version of box 2 has been discussed in detail.

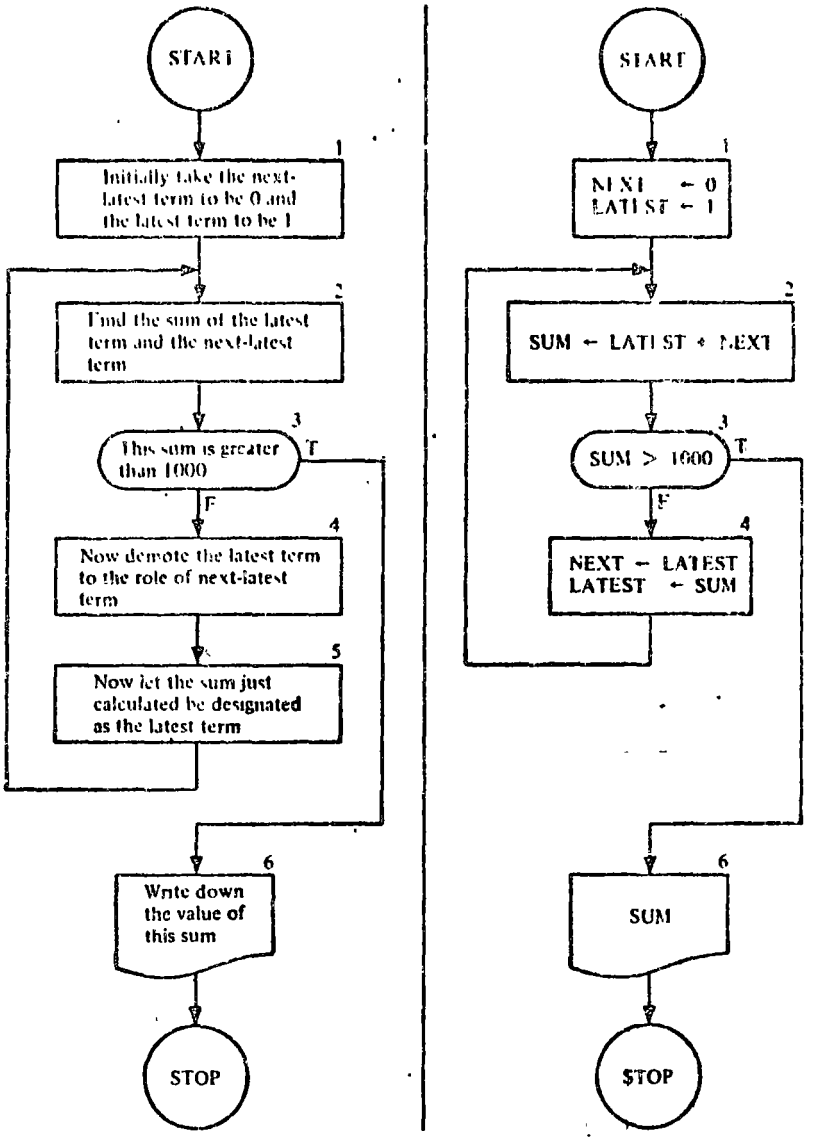
We see that the two statements in boxes 4 and 5 of the old flowchart are compressed into one box, box 4 of the new flowchart. This is permissible whenever we have a number of assignment statements with no other steps in between. However, it is very important to understand that these assignment statements must be executed in order from top to bottom, not in the opposite order and not simultaneously. The order in which things are done may be extremely important.

You can see that the statements in box 4 involve no computation but merely change the values in certain storage boxes. This sort of activity occurs frequently in flowcharts.

In box 6 of the flowchart we see only the word SUM. The shape of the box (called an *output box*) tells us that the value of the variable SUM is to be written down or displayed. If, in some other algorithm, we wished to write down the values of several variables, we would list these variables in an output box separated by commas, as illustrated on the left.




We will now describe the duties of the Sticker Affixer. We consider that the computation is begun by the transmittal of a flowchart to the Master Computer. The first thing the Master Computer does is to scan the flowchart, making a list




(*a*) Old. (*b*) New.

FIGURE 1·11
Translation of Fibonacci sequence flowchart into formal flowchart language.

of all the variables used. In the case of the Fibonacci sequence flowchart of Figure 1·11*b*, this list would have the form

NEXT
LATEST
SUM

The Master Computer hands this list to the Sticker Affixer, who now springs into action. He inscribes each of these variables on a sticker, goes to a bin of unlabeled storage boxes, and slaps one of these stickers on each of three boxes (Figure 1·12).

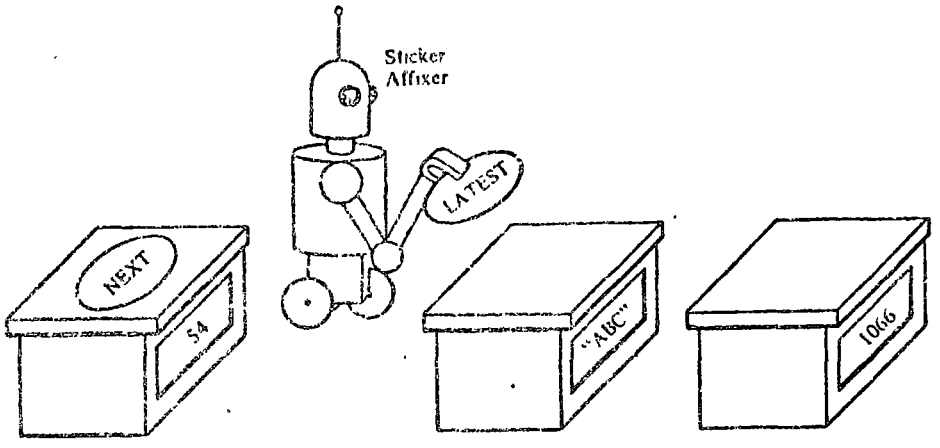



FIGURE 1·12
Sticker Affixer at work.

Now the instructions in the flowchart are executed until the STOP instruction is reached. At this juncture, the Master Computer directs the Affixer to unpeel all the labels and throw them into a recycle bin.

## Tracing the Flowchart

To understand better what our flowchart in Figure 1·11*b* does, let us trace through it, executing the steps as the Master Computer and his assistants do them (see Table 1·3).

In this trace, for ease of reading, the values of the variables are reproduced only when assignments are made to them. In between such steps, the values of the variables do not change and therefore have the most recently recorded values. For example, in step 33, where a test is performed, the values of the variables are

$$\text{NEXT} = 55, \qquad \text{LATEST} = 89, \qquad \text{SUM} = 144$$

In step 34 the values are

$$\text{NEXT} = 89, \qquad \text{LATEST} = 144, \qquad \text{SUM} = 144$$

You can see that in step 48 in the execution of our algorithm we finally leave box 3 by the *true* exit and pass on to box 6, where we output the answer, 1597, and stop.

The utter simplicity of our conceptual model avoids and removes certain pitfalls. There is an ever-present danger of thinking of assignment as equality or substitution. (We will say

TABLE 1·3
Tracing of the Flowchart of Figure 1·11*b*

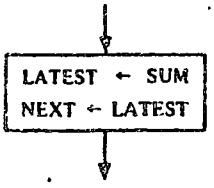
| Step Number | Flowchart Box Number | Values of Variables | | | Test | True or False |
|---|---|---|---|---|---|---|
| | | NEXT | LATEST | SUM | | |
| 1 | 1 | 0 | 1 | | | |
| 2 | 2 | | | 1 | | |
| 3 | 3 | | | | 1 > 1000 | F |
| 4 | 4 | 1 | 1 | | | |
| 5 | 2 | | | 2 | | |
| 6 | 3 | | | | 2 > 1000 | F |
| 7 | 4 | 1 | 2 | | | |
| 8 | 2 | | | 3 | | |
| 9 | 3 | | | | 3 > 1000 | F |
| 10 | 4 | 2 | 3 | | | |
| 11 | 2 | | | 5 | | |
| 12 | 3 | | | | 5 > 1000 | F |
| 13 | 4 | 3 | 5 | | | |
| 14 | 2 | | | 8 | | |
| 15 | 3 | | | | 8 > 1000 | F |
| 16 | 4 | 5 | 8 | | | |
| 17 | 2 | | | 13 | | |
| 18 | 3 | | | | 13 > 1000 | F |
| 19 | 4 | 8 | 13 | | | |
| 20 | 2 | | | 21 | | |
| 21 | 3 | | | | 21 > 1000 | F |
| 22 | 4 | 13 | 21 | | | |
| 23 | 2 | | | 34 | | |
| 24 | 3 | | | | 34 > 1000 | F |
| 25 | 4 | 21 | 34 | | | |
| 26 | 2 | | | 55 | | |
| 27 | 3 | | | | 55 > 1000 | F |
| 28 | 4 | 34 | 55 | | | |
| 29 | 2 | | | 89 | | |
| 30 | 3 | | | | 89 > 1000 | F |
| 31 | 4 | 55 | 89 | | | |
| 32 | 2 | | | 144 | | |
| 33 | 3 | | | | 144 > 1000 | F |
| 34 | 4 | 89 | 144 | | | |
| 35 | 2 | | | 233 | | |
| 36 | 3 | | | | 233 > 1000 | F |
| 37 | 4 | 144 | 233 | | | |
| 38 | 2 | | | 377 | | |
| 39 | 3 | | | | 377 > 1000 | F |
| 40 | 4 | 233 | 377 | | | |
| 41 | 2 | | | 610 | | |
| 42 | 3 | | | | 610 > 1000 | F |
| 43 | 4 | 377 | 610 | | | |
| 44 | 2 | | | 987 | | |
| 45 | 3 | | | | 987 > 1000 | F |
| 46 | 4 | 610 | 987 | | | |
| 47 | 2 | | | 1597 | | |
| 48 | 3 | | | | 1597 > 1000 | T |
| 49 | 6 | | | 1597 | | |

more about this later.) This and other potential sources of confusion, such as the effect of a certain sequence of flowchart statements, can be cleared up by thinking in terms of the SIMPLOS model, which will always give the right answers.

In fact, an excellent way to understand these ideas of reading and assigning values to variables is to make some storage boxes and, with some friends, work through several algorithms as described in this section.

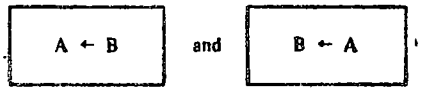
## EXERCISES 1·3

1. What is the effect of changing the order of the two assignment statements in box 4 of Figure 1·11*b* so as to appear as seen below?




Trace through the flowchart with this modification until you find the answer.

2. (a) To compare the effects of the assignment statements




find the missing numbers in the table below.

| Values Before Execution of Assignment | | Assignment To Be Executed | Values After Execution of Assignment | |
|---|---|---|---|---|
| A | B | | A | B |
| 7 | 13 | A ⟵ B | ? | ? |
| 7 | 13 | B ⟵ A | ? | ? |

(b) In which of the two cases is it true that A = B *after* assignment?
(c) Are the effects of the two assignment statements the same or different?

3. Modify the flowchart in Figure 1·11*b* so as to carry out the algorithm of Problem 1, Exercises 1·2.

4. Modify the flowchart of Figure 1·11*b* so as to output each term of the Fibonacci sequence starting with the third (i.e., omit the initial 0, and 1).

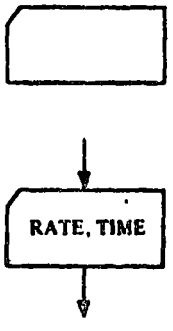
5. Revise the flowchart of Problem 4 to calculate the ratio, r, of LATEST to NEXT (as calculated in Problem 3a, Exercises 1·2) and output this ratio (as well as LATEST) at each step.

6. Revise the flowchart of Problem 5 to calculate at each step the reciprocal of r. Add this value to the output list.

## 1·4 Input/output

Imagine that you are a bookkeeper in a large factory. You have records of the hourly rate of pay and the number of hours worked for each employee, and you have to calculate the week's wages. Of course, this can be done by hand, but assume there are nearly 1000 workers in the plant, so that the job would be quite tedious. Naturally you prefer to have the computer execute this task for you, but you will have to devise a flowchart to convey the instructions to the computer.

How will the hourly wages and the hours worked come into our computation? Must each new value of RATE and TIME be represented by a separate assignment box? This is certainly a possibility, but it would require thousands of flowchart boxes—a most undesirable state of affairs. This unpleasant necessity can be eliminated by using the concept of *input*.




We now introduce a new shape of frame, the *input box*, into the flowchart language. The input box has this shape to suggest a "punch card" (a frequently used input medium, but not the only one). Inside the box will appear a single variable or a list of variables separated by commas.

What happens in our SIMPLOS model when the Master Computer encounters such an instruction? To answer this question, we must endow the SIMPLOS model with an additional feature not previously needed (Figure 1·13). SIMPLOS has a conveyer belt (called the *input belt*) that carries slips of paper from outside the room into the environment of the computing staff. On the outside end of the belt the "user" or "programmer" (who is not a member of the computer staff) places these slips of paper, with values written on them, on the conveyer belt in the order in which he wants them to be used.

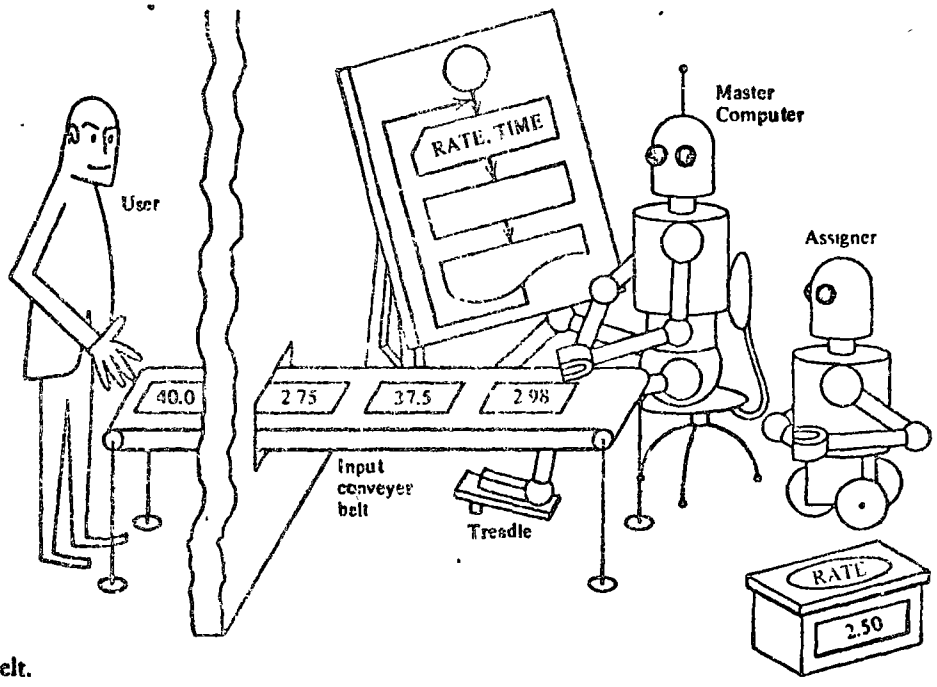
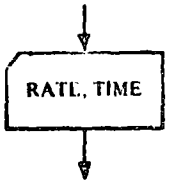



FIGURE 1·13
SIMPLOS with Input belt.




When the Master Computer comes to the input instruction he does the following.

1. Steps on a treadle running the conveyer belt until the next slip of paper comes within reach.

2. Remove his foot from the treadle, stopping the belt.

3. Takes a slip of paper from the belt and hands it to the Assigner with instructions to assign the value thereon to the variable, RATE.

When the Assigner returns from this task, the Master Computer repeats the above process, but this time tells the Assigner to assign the new value to TIME. When this is done, the Master Computer follows the arrow in his flowchart to the next instruction.

We see that an input box is a command to make assignments, but this command is essentially different from that in an assignment box. In an assignment box the values to be assigned are to be found in computer storage or are computed from values already stored, whereas with an input box the values to be assigned are obtained from outside the computer. No calculation is called for in an input box. Moreover, the

*values* to be input never appear in the flowchart itself. Only the *variables* to which these values are to be assigned appear in the input boxes of the flowchart.

In an actual computer (not our conceptual one) the distinction between the two kinds of assignment need not be so sharp. Assignments called for in an input box *usually* involve some mechanical motion such as transporting a punched card or other unit of recorded information past a reading station where the coded contents may be copied. But to gain speed the data often are transported into a special section of storage called an *input buffer*, well before the data are actually needed by the executing algorithm. In this case, when the input step is executed, what actually happens is that data values are simply copied at electronic speed from storage boxes of the input buffer to storage boxes of the variables that are specified in the input step of the algorithm.

Now let's see how to use the input box in our hourly rate and payroll problem. Should we input the data from all the cards before we start our calculations? If so, we would need a great many storage boxes in which to store all these data. Instead, we will calculate the wages after each data set is read. A description of our method is as follows.

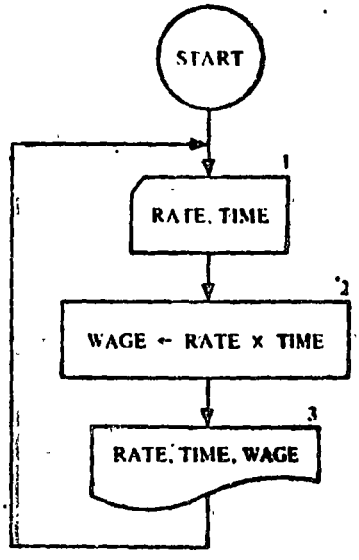



FIGURE 1·14
Payroll algorithm.

1. Input one value of RATE and one value of TIME by the process described above.
2. Multiply the RATE by the TIME to get the WAGE.
3. Output the values of RATE, TIME, and WAGE.
4. Return to step 1.

In the flowchart of Figure 1·14 each of the first three steps of the above list appears in a similarly numbered box. Step 4 is represented by the arrow returning from box 3 to box 1.

You may wonder why the flowchart does not have a stop button. SIMPLOS always terminates execution of an algorithm when an input step is being executed, and the input belt contains too few values to match the variables in the input box. Execution of the payroll algorithm will therefore always halt after the last *rate, time* pair of data values has been processed and control once again reaches box 1, where it is discovered that the input belt is empty.

It will also be useful to visualize output in a similar way.

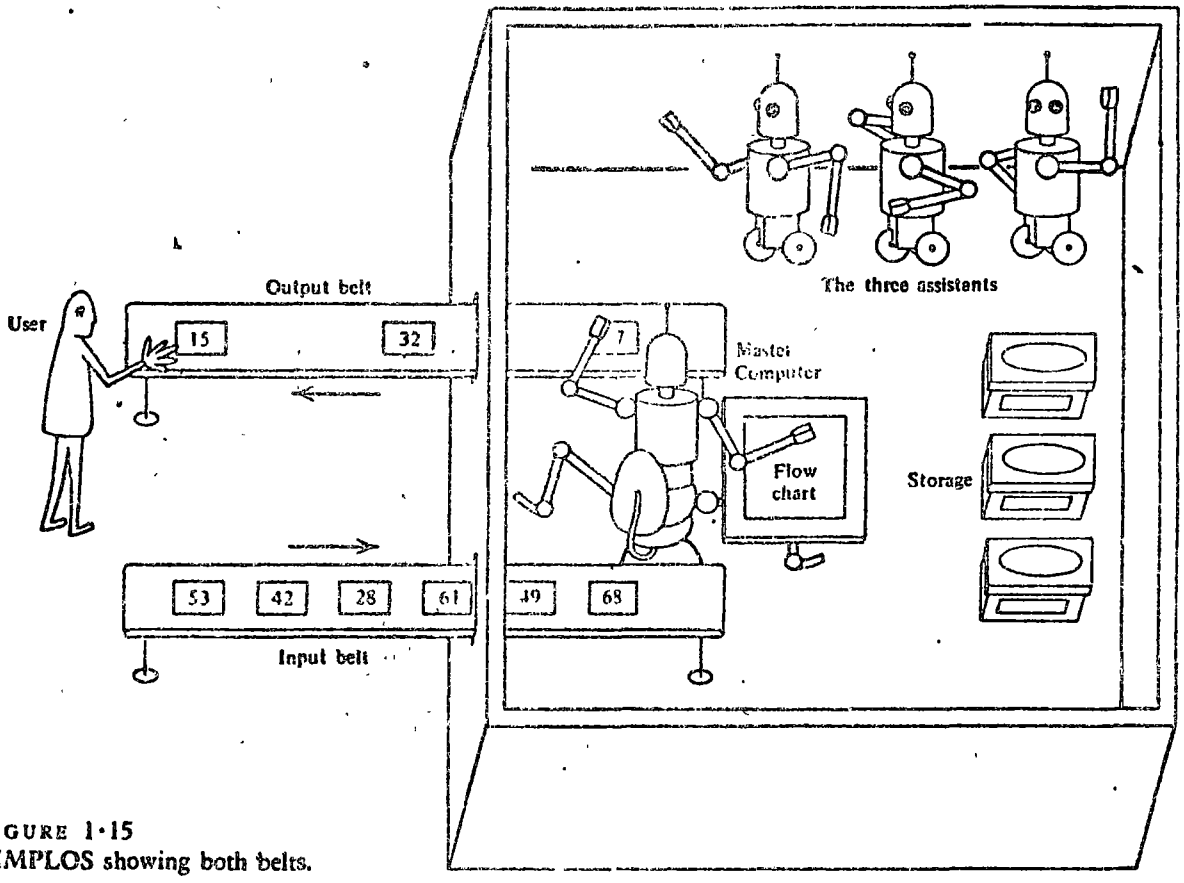


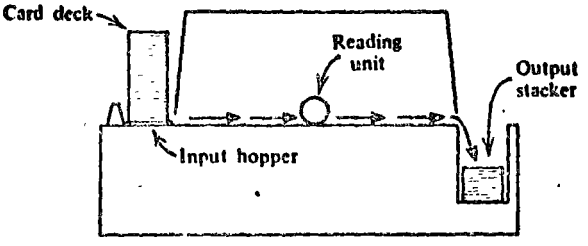

FIGURE 1·15
SIMPLOS showing both belts.

We endow SIMPLOS with a second conveyer belt, the *output belt*. This belt runs *out* instead of in and runs continuously—it needs no treadle. Each time the flowchart calls for output, the Master Computer writes the proper value on a slip of paper and drops it on the output belt, which carries the slip through the wall to the outside environment of the user. A view of this situation from the top is seen in Figure 1·15.

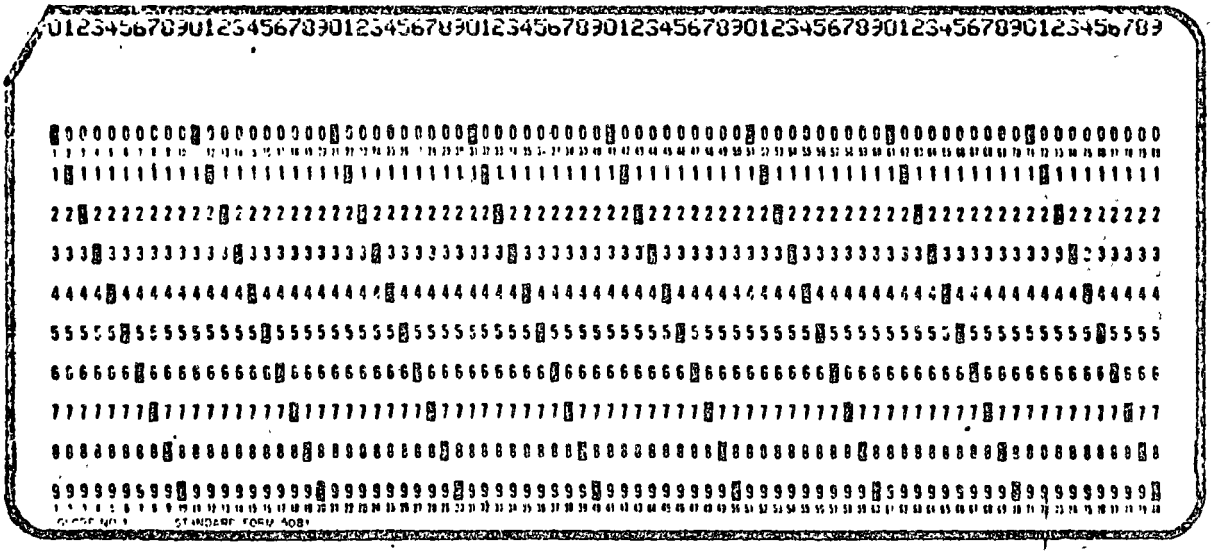
## Records versus Streams

Our conveyer belt model of input-output suggests that data values move as a stream into and out of a computer system. Although in actual computers this is not always strictly the case, the analogy nevertheless is quite close. To pursue this idea let us consider the punched card reader, one of the most common input devices on actual computers. First, a sequence of data values is punched on cards. The cards are then placed in proper order in the *input hopper* of the card reading device.

Usually the card deck is placed face down so that the bottom (first) card in the deck is the first one to be read. Each time more input data are required, another card is drawn from the bottom of the deck and its contents are read, either electromechanically or photo-optically. Once read, the card is dropped into an *output stacker* and thus discarded.




The information contained on a single punched card need not in principle be limited to one value. For example, depending on what a program is designed to expect, an 80-column card may contain up to eight 10-digit integers or up to twenty 4-digit integers.



When preparing data cards one is always faced with the decision of whether to utilize their capacity fully or to punch on each only the values required for the execution of one input step in the algorithm. The latter choice, although somewhat wasteful of card space, makes the data cards easier to check.

Our payroll problem provides us with a case in point. If,


for convenience in locating data preparation errors, we restrict the contents of each card to one *rate, time* pair, then each time box 1 is executed, one and only one data card will be drawn off the input deck, read, and discarded. We could be more wasteful and punch only one data value on each card. Then successive cards would contain first a *rate* value, then a *time* value, and so on. In this case, each execution of box 1 must cause *two* data cards to be drawn from the deck and read, and the analogy between the input conveyer belt and the card reading activity is very close indeed. That is, when the Master Computer hits the foot treadle to bring in one data value, the actual computer will signal the card reader to draw off one card and read it.

The analogy is less apparent if we allow the data cards to contain more than one *rate, time* pair and if we expect the pairs to be considered in turn during successive executions of box 1. In this case, box 1 can no longer mean "read a card" but, instead, "assign respectively to *rate* and *time* values from the next data pair in the deck. If the next pair cannot both come from the current data card, then draw off another card from the deck and read it. If, on the other hand, there is at least one more data pair yet to be processed from the most recently read card, then process that data pair." This interpretation assumes that an input buffer is filled (and refilled) with data from each newly read data card and that values are assigned to *rate* and *time* by simply copying information from this buffer into the respective program variables, always remembering for future use which items in the buffer have not yet been copied.

We see, therefore, that using an input buffer guarantees that each data pair in the sequence will be processed in turn, no matter how many pairs are punched on each data card. None will be missed or skipped over. It is in this sense that the stream analogy is preserved even though the sequence of data items is grouped into arbitrary-sized *card records.*

The SIMPLOS model is a primitive machine. It has only stream-oriented input and output. After values are placed on the conveyer belt to be output and are carried to the outside environment of the user, how are they displayed? We certainly are aware that in actual computer systems all values are printed or displayed on a screen in some sort of "format" with a

particular number of columns, but the only fact we are interested in with respect to SIMPLOS is that the values *are* output.


When it comes to interpreting output boxes of a flowchart, the situation is somewhat different. The output box on the left is considered to be a command to print the current values of the three variables, RATE, TIME, and WAGE *on one line.* (If the list won't fit on one print line, more lines are used.) Furthermore, if the same output box is executed again, the next set of three values will appear on a new line below the first set. If the three variables appeared in three individual boxes instead of in one single box, then each would be printed on a separate line. Thus each execution of an output box is considered to begin printing a new line.

**Importance of the Conceptual Model**

No doubt you have wondered why, at the very start of our study, so much attention has been given to a conceptual model of a computer and its details. Can any model, especially this one, which seems so simple and at the kindergarten level for some readers, be that important or that valuable to us? You may develop similar doubts about the value of flowcharts as you proceed further.

The model and the flowcharts we develop are *abstractions* of real machines and of real computer programs. Once we see the connection between an abstraction and the *concrete* or real thing, we can often gain more understanding of the real thing by studying and manipulating its abstract counterpart. So, high on our list of priorities should be an attempt to understand and appreciate the connections between the abstract and the concrete. For example, in the next sections of this chapter we examine how an actual computer is organized and how it works. Thereafter, it will be easier to see why the conceptual model, no matter how silly it may have first appeared, is a very useful, simplified view of a real computer. Likewise, just as soon as we try to write and test actual computer programs, we shall see that the flowchart gives us a simpler but more revealing way to think about computer programs for most purposes.

Experience has taught us that problem solving with computers is very effective if we can work first with a simplified model of a machine and a simple descriptive algorithmic lan-

guage in which to express our problem solutions. Then it is comparatively easy to *map* these solutions over to programs written in some convenient programming language such as BASIC, FORTRAN, ALGOL, or COBOL, so that the programs can be executed on some real, convenient computer.

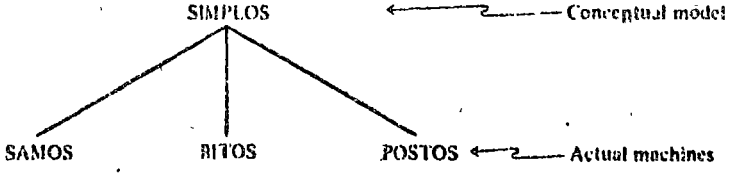
EXERCISES 1·4

1. Modify the flowchart of Figure 1·14 to provide for an overtime feature. All hours in excess of 40 are to be paid at time and a half. You will have to place a decision box somewhere in the flowchart to determine whether the worker actually put in any overtime. The formula by which his wages are computed will depend on the outcome of this test.

## 1·5 Actual computers

Now we are ready to examine how our conceptual model of a computer can be realized in an actual machine. For the first 25 years of modern computer history (1949 to 1974), nearly all actual machines were built following a more or less stereotyped pattern suggested by John Von Neumann (1903–1957). A prototype machine following this pattern is discussed in this and following sections. We will call it SAMOS. SAMOS is a very simple machine; that is, it is stripped down to the bare essentials. Some features of its operation are described in considerable detail, while others are glossed over. The programming of SAMOS is described briefly in Section 1·6 and in more detail in the Appendix, the purpose of which is to help the reader see a closer connection between language for expressing algorithms and machines that execute them.

It would be foolhardy to assume that SAMOS-like machines are the "be all and end all of computers," since the architecture of computers is still undergoing rapid change. For this reason, aspects of two other machines are discussed briefly in this book. One machine, called BITOS, appears later in this chapter; the other, called POSTOS, is considered in Chapter 8. Each of the three machines exhibits certain distinct characteristics for the implementation of our conceptual model, SIMPLOS.



In order to study this book it is useful, although not essential, to gain a good understanding of how an actual computer works. We suggest that you read once through the material of the next two sections without attempting to master it. As you work exercises that relate to SAMOS, or have occasion to study SAMOS in the Appendix, you will no doubt come back to the next two sections for a more careful study.

## 1·6 SAMOS

How are all those storage boxes of SIMPLOS realized in actual practice? The storage of actual computers is built of electronic components in a variety of ways and with a variety of materials. Here we describe one way that a SAMOS storage can be built.

### Core Storage

SAMOS storage, packaged in a rectangular box, is an arrangement of tiny magnetic doughnuts as small as 1/40 of an inch in diameter. These doughnuts are called *cores* (Figure 1·16). The cores are laid out in 61 horizontal layers or trays called *core planes*. On each of these layers, wires are strung evenly in two directions like the lines on a sheet of graph paper. There are 100 wires in each direction. At each point where two wires cross, the wires are threaded through a core, like the thread passing through the eye of a needle (Figure 1·17). (Still other wires are threaded diagonally through each core within each plane. Their function is not important to the discussion that follows, and they are therefore ignored.)

FIGURE 1·16
A magnetic core.

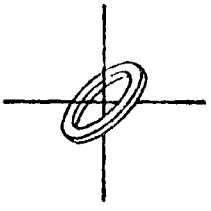


FIGURE 1·17
Where two wires cross.

Figure 1·18 is a picture of a core plane from an actual computer built in the mid-1960s. Since there are $100 \times 100$ crossings in each SAMOS core layer, we see that there are 10,000 cores in each core plane and hence $61 \times 10{,}000 = 610{,}000$ cores in the entire SAMOS *store* (storage).

These cores are capable of being magnetized in either the clockwise or the counterclockwise sense (Figure 1·19). Because of this a core can store information. We could think of clockwise magnetization as meaning "yes" and counterclockwise as meaning "no." We will instead think of clockwise as standing for "0" and counterclockwise for "1." In any event, the information contained in the magnetization of a core is the smallest

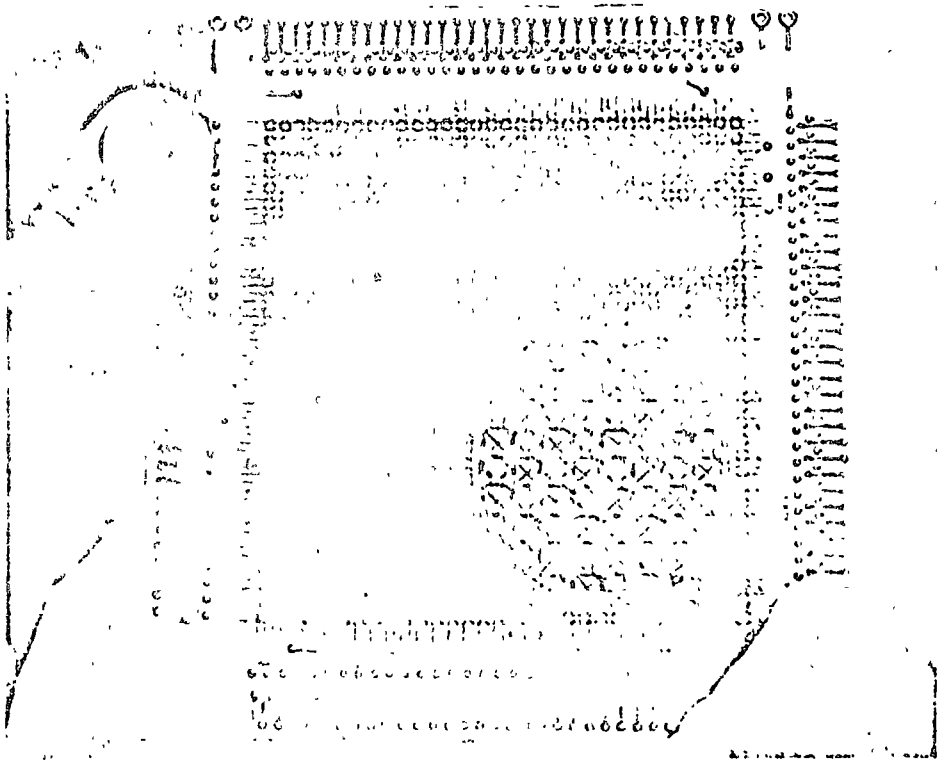


FIGURE 1·18
An actual core plane.
(Courtesy of IBM).

unit of information and is called a *bit* of information. We see that one core can store one binary digit, 0 or 1, but a collection of cores can store a very large number of bits. We will discuss this idea later, after a digression on how the cores get their magnetism.

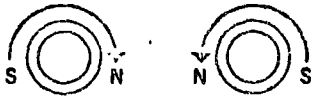



FIGURE 1·19
Magnetization of cores.

First, you must know that a pulse of electric current moving along a wire generates a magnetic field running around the wire, as shown in Figure 1·20. The strength of the magnetic field is strongest near the wire and dies away as we move further from the wire.

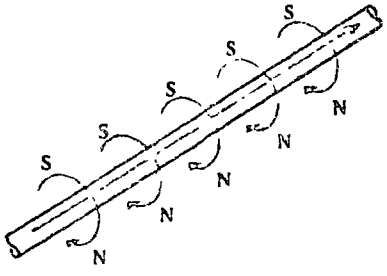



FIGURE 1·20
A magnetic field resulting from a pulse of electric current.

If the direction of the current is reversed, the direction of the magnetic field is also reversed (Figure 1·21).

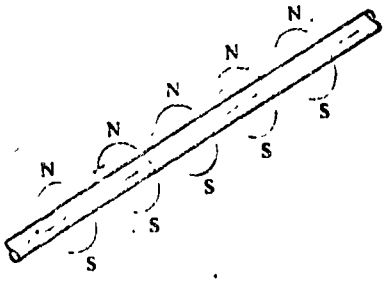



FIGURE 1·21
Reversing the direction of the magnetic field.

Thus, when a pulse of current passes through a core, the core will become magnetized in one direction or the other, depending on the direction of the current (Figure 1·22).

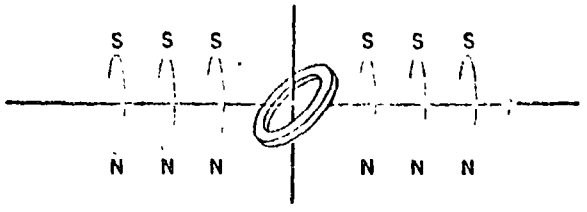



FIGURE 1·22
A core in a magnetic field.

But how can we manage to magnetize just one core instead of the whole string of cores (Figure 1·23) through which the pulse passes? The answer lies in the magnetic properties of the material from which the core is made. In this material,

FIGURE 1·23
A row of cores in a magnetic field.

if the pulse is too weak, the direction of magnetization of the core is only *temporarily* altered, and after the pulse of current has passed by, the core merely returns to its former magnetic condition, whatever that was.

On the other hand, if the current is strong enough, the core remains magnetized in the sense established by the direction of the current, regardless of the former magnetic condition of the core. The situation is analogous to trying to throw a ball from the ground to the flat roof of a building. If you have enough power in your throw, the ball will land on the roof; otherwise it will bounce against the wall and fall back to the ground.

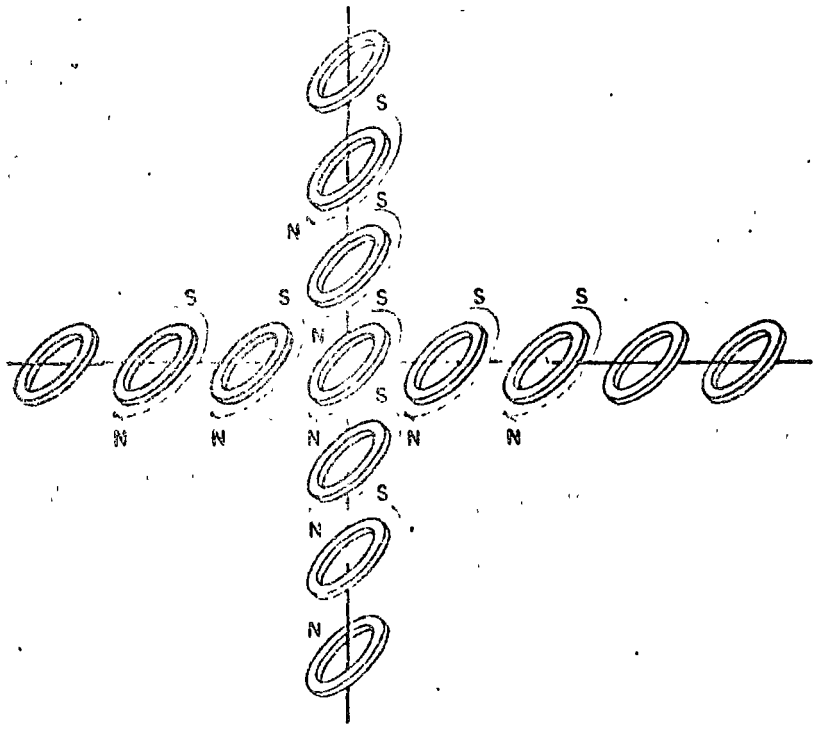



FIGURE 1·24
Doubling the magnetic field at the wire crossing.

The strength of the pulses is carefully regulated so that one pulse is not sufficient to permanently magnetize a core, but two pulses acting simultaneously will exceed the threshold strength and result in permanent magnetization. Thus, pulses passing along two of the wires (Figure 1·24) will permanently magnetize just the one core that is located where the wires cross.

## SIMPLOS and SAMOS Stores Compared

Let's leave the individual core planes and consider the entire store of the SAMOS computer, composed of the 61 core planes (Figure 1·25). Each vertical column of 61 cores constitutes a *computer word*. Thus, the storage of the computer is composed of 10,000 words. These words have addresses that are four-digit numbers from 0000 to 9999 and, like house numbers, the addresses identify the words. Each of the 10,000 dots suggested on the top of the box is the top of a vertical column of 61 cores (or a word). The method of assigning the addresses is indicated in the figure.

Each of these words corresponds to a storage box in our conceptual SIMPLOS model. For each variable in the flow-

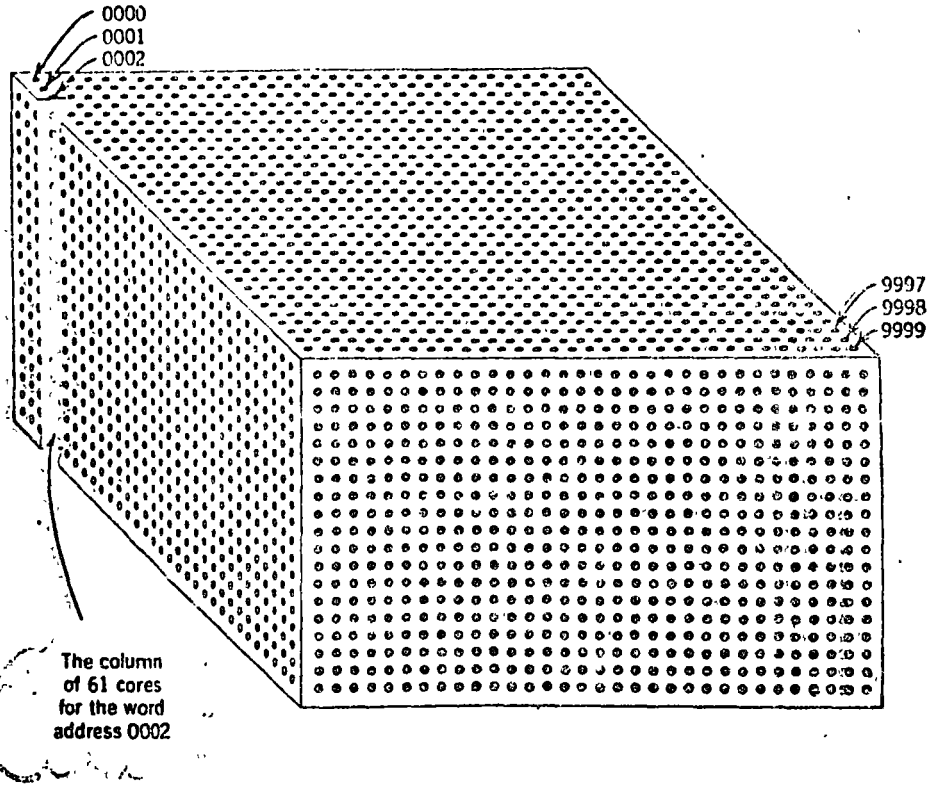



FIGURE 1·25
Cores of one computer word.

chart there is a SAMOS word with a definite address. The word contains a certain pattern of bits determined by the directions of magnetization of its cores and representing the value of that variable. "Assigning a value to a variable" is effected by putting a certain pattern of bits into a word (Figure 1·26).

When we say "the Master Computer tells the Assigner to assign the value 1597 to the variable SUM," what actually takes place is this. The variable SUM is represented inside the machine by means of its address; suppose it is 0103. Now all the $61 \times 2 = 122$ wires passing through cores in the word addressed 0103 are energized with pulses of current in the proper directions so as to achieve the pattern of bits representing the number 1597. In a modern computer this assignment process can be performed in a fraction of a microsecond; a microsecond is a millionth of a second.

## Characters

In the binary system of representation, a number such as 1597 is coded as a string of 1's and 0's, for example:

1 1 0 0 0 1 1 1 1 0 1

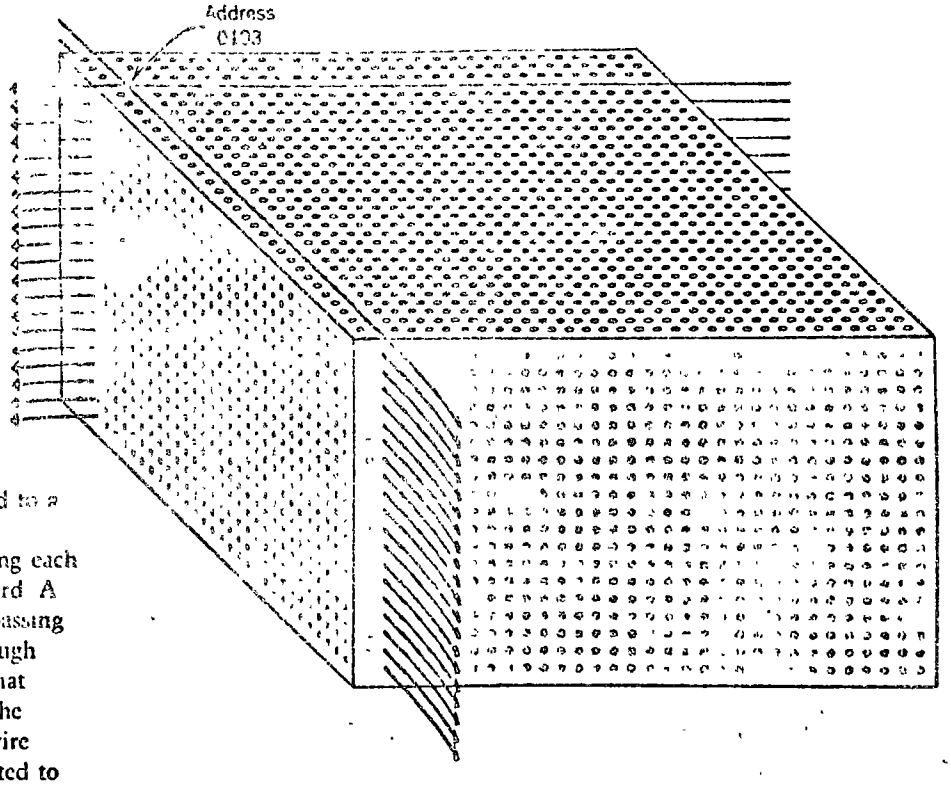
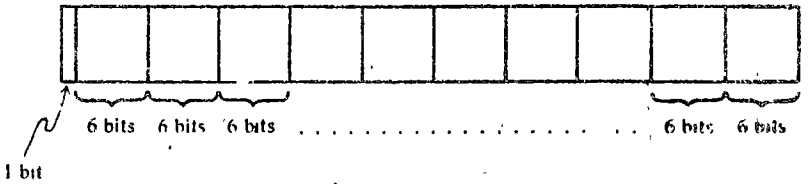



FIGURE 1·26
A bit pattern is assigned to a computer word by appropriately magnetizing each of the cores for that word. A core is magnetized by passing an electronic pulse through each of the two wires that intersect at that core. The diagram identifies the wire pairs that must be selected to assign a bit pattern for the word at address 0103.

For SAMOS any string would be preceded by a string of zeros to fill out all the bit positions of the word of storage. While numbers are coded in binary form in many computers, binary is certainly not the only choice. In a machine such as SAMOS, for instance, computation is carried out in the decimal system, which means that bit patterns in a word of storage must be coded to represent decimal digits instead of binary digits. Moreover, we want to store letters as well as decimal digits. For this reason, we subdivide our 61-bit SAMOS words into 11 character positions as shown below.




The first position (one bit only) is reserved for a code that designates the sign, + or −. Here 0 is sufficient to represent the + character and 1 signifies the − character. Each of the other positions consists of six bits and can be used to store

| Character | Code | Character | Code | Character | Code | Character | Code |
|---|---|---|---|---|---|---|---|
| 0 | 00 0000 | | | | | □ | 11 0000 |
| 1 | 00 0001 | A | 01 0001 | J | 10 0001 | | |
| 2 | 00 0010 | B | 01 0010 | K | 10 0010 | S | 11 0010 |
| 3 | 00 0011 | C | 01 0011 | L | 10 0011 | T | 11 0011 |
| 4 | 00 0100 | D | 01 0100 | M | 10 0100 | U | 11 0100 |
| 5 | 00 0101 | E | 01 0101 | N | 10 0101 | V | 11 0101 |
| 6 | 00 0110 | F | 01 0110 | O | 10 0110 | W | 11 0110 |
| 7 | 00 0111 | G | 01 0111 | P | 10 0111 | X | 11 0111 |
| 8 | 00 1000 | H | 01 1000 | Q | 10 1000 | Y | 11 1000 |
| 9 | 00 1001 | I | 01 1001 | R | 10 1001 | Z | 11 1001 |

FIGURE 1·27
Character code

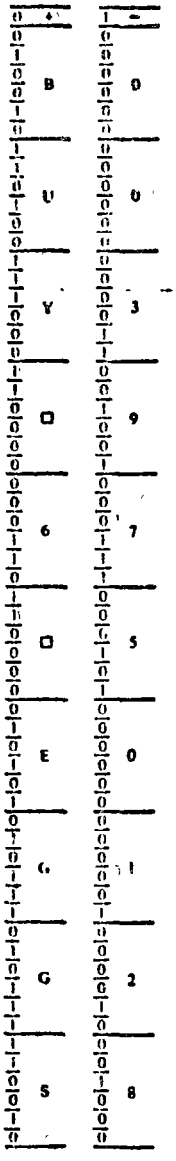



FIGURE 1·28
Detailed bit patterns for two computer words.

a digit or a letter, that is, *character,* according to the code shown in Figure 1·27.

For each group of six bits, $2^6$ or 64 distinct combinations of zeros and ones are possible. In Figure 1·27 we have used up only 37 of the 64 combinations possible with a six-bit code. This leaves 27 additional combinations for other special symbols such as +, ≥, and the like. One of the 37 combinations of special interest is the blank space, □, which is coded as

1 1 0 0 0 0

With this code you can see that the two 61-bit computer words displayed vertically in Figure 1·28 turn out to be

| + | B | U | Y | □ | 6 | □ | E | G | G | S |
|---|---|---|---|---|---|---|---|---|---|---|

and

| - | 0 | 0 | 3 | 9 | 7 | 5 | 0 | 1 | 2 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|

From now on we shall represent our SAMOS computer words as strings of 11 characters instead of strings of 61 bits. In a number of conventional computers of similar design eight instead of six bits are grouped to represent character codes, making it possible to distinguish among a considerably larger set of characters than is the case in SAMOS. This distinction, however, has absolutely no effect on the principles of character representation and manipulation that occur in ensuing chapters.

The construction of the main storage for any actual computer is of great interest mainly to computer engineers and designers. Storage components currently are built from various

types of physical devices and materials, including magnetic cores, magnetic thin films, and transistor flipflops. There is considerable variety in the circuitry used to organize and utilize such components and in the methods of packaging and miniaturizing them. Their physical characteristics, such as size, speed of access for storing and retrieving information, energy requirements to operate them, and cost of fabrication, vary also. Nevertheless, schemes similar to that used in the word-organized core storage of SAMOS have been used to assemble and incorporate all of these types of storage units into conventional computer systems. You might be surprised at how much understanding of this subject you can gain with a relatively small investment of study time. (See, for example, one of the references on this topic in the reading list at the end of this book.)

## SAMOS Processing Unit

Now that we have seen how the SAMOS storage is structured, we will consider how the storage is used in executing an algorithm.

Our computer has several other components besides the store. These are shown in Figure 1·29.

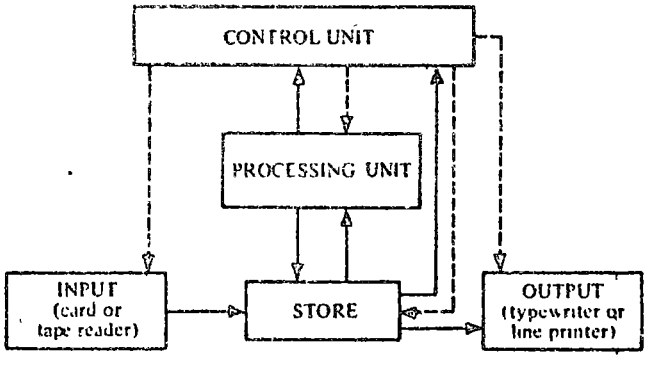
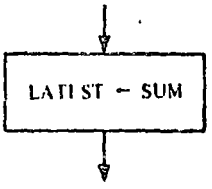



FIGURE 1·29
The principal components of SAMOS.

The solid lines indicate the directions in which values or instructions may be transferred. The dashed lines indicate the exercise of control. The control unit and the processing unit perform the duties of the "Master Computer" and his helpers.

An important part of the processing unit is the *accumulator*. This special computer word holds the result of each arithmetic operation.

Furthermore, a simple assignment such as

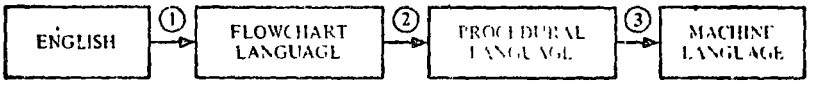



is carried out by first obtaining a copy of the value of SUM, placing it in the accumulator, and then copying the value in the accumulator into the computer word belonging to the variable LATEST. The value of SUM is unchanged in this process. Notice, however, that values to be input or output do not pass through the SAMOS accumulator but go directly into and out of storage.

When the control unit receives and interprets an *order,* some computer operation is activated. The orders are in the form of coded instructions stored in the computer; we will see about them presently.

## Machine Language

Getting an algorithm into a form a machine can execute involves several translations that we can represent as follows:




You have already had a little experience with the first translation step. The second translation step is the process of converting a flow chart into a procedural language such as FORTRAN, ALGOL, COBOL, or PL/I. You learn how to do this in your language manual. If approached properly, this translation step is quite mechanical and can be performed by a person (or by a machine) who has no idea what the algorithm is all about.

In many computers of advanced design, the third translation process can be omitted because the machine's language and the procedural language are effectively identical. When the third translation step is necessary, and it is for a computer such as SAMOS, the process is completely mechanical and is normally done by the computer itself. This process is called *compiling.*

It is not necessary right now to know how compiling is done, but it may be interesting to know the reason for doing it. Each make and style of computer has its own language—that is, its own set of instructions that it can understand. Use of a procedural language allows us to avoid a tower of Babel in which a programmer would have to learn a new language for each machine he wishes to use. A procedural language constitutes a kind of "Esperanto" that enables a programmer to communicate with many different machines in the same language. Moreover, a procedural language is generally much easier to learn to use than machine language. The programmer merely prepares, say, a FORTRAN program on punched cards and feeds it into the computer, which "compiles" a sequence of machine language instructions. This sequence, called a *machine language program,* is then placed in the computer storage. In many systems the programmer may transmit his program to the computer storage by typing it a line at a time, using a typewriter or other keyboard instrument to serve as the input device of the computer system.

## Sequencing of Computer Instructions

Successive SAMOS instructions are placed in consecutively addressed storage locations starting with 0000. After the computer has executed an instruction, the control unit will always take the next instruction from the next address, unless there is a *branching* instruction providing a different address from which to take the next instruction.

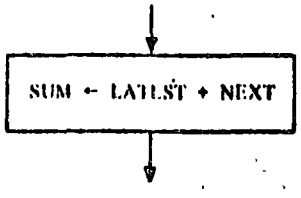



FIGURE 1·30
A flowchart box.

To see how this works, consider the instruction taken from the Fibonacci sequence flowchart (Figure 1·11*b*), shown here in Figure 1·30.

The procedural language equivalent will not look much different. Thus, in FORTRAN this instruction would appear as

```
SUM = LATEST + NEXT
```

and in ALGOL as

```
SUM := LATEST + NEXT
```

and in APL as

```
SUM ← LATEST + NEXT
```

and in BASIC as

```
LET SUM = LATEST + NEXT
```

and in COBOL as

```
COMPUTE SUM EQUALS LATEST
  PLUS NEXT
```

In the SAMOS machine language, a variable cannot be referred to by *name* but only by the *address* in storage associated with the variable. Suppose that NEXT, LATEST, and SUM have been given, respectively, locations 0100, 0101, and 0102. Then in the SAMOS language, the flowchart instruction translates to a sequence of three machine instructions, as shown in Figure 1·31.

```
+  LDA  000  0101
+  ADD  000  0100
+  STO  000  0102
```

**FIGURE 1·31**
**SAMOS instructions for Figure 1-30.**

These instructions have the form of 11-character words, although the first character is unimportant and the fifth, sixth, and seventh are of no interest to us here. The type of *operation* to be performed is coded using the three letters in positions two, three, and four, and the four-digit numeral at the right is the *address* associated with that operation.

The letters LDA stand for "LoaD the Accumulator." The whole instruction

```
+ L D A  0 0 0  0 1 0 1
```

means, "Make a copy of the value stored in address 0101 without altering the original, and store the copy in the accumulator." Clearly, this is the function of the Reader in our conceptual model. We will not go into the details of the electronics involved in carrying out this instruction. It is sufficient to know that when a copy of that instruction is brought to the control unit, certain switches are set by the control unit that allow a pulse current to pass through the cores of the word 0101. The magnetized cores cause a change in the current that, in turn, allows a copy to be made.

The second instruction in Figure 1·31 means, "ADD the value in the word addressed 0100 to the value already in the accumulator and place the result in the accumulator." The third instruction means, "Copy (or STOre) the value in the accumulator *into* the word addressed 0102." Executing a STO instruction is analogous to the work of the Assigner in our conceptual model. Speeds vary from machine to machine, but in modern computers, the time required to carry out such instructions is usually on the order of a millionth of a second.

## A Complete SAMOS Program

We are almost able to translate the entire flowchart for the Fibonacci sequence algorithm (repeated here in Figure 1·32) into SAMOS language. First note, however, that constants never appear explicitly in SAMOS instructions. Instead, an

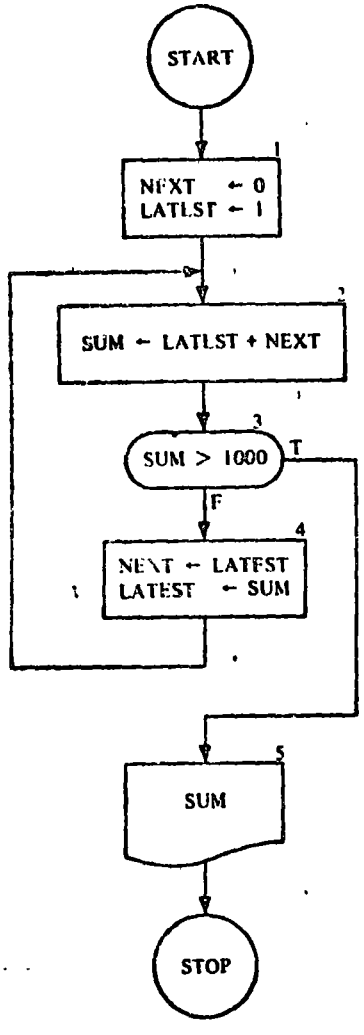



FIGURE 1·32
Fibonacci sequence algorithm.

instruction to fetch a constant must refer to the storage address where the desired value may be found. Of course, this also applies to variables. Thus part of the compiling process involves providing storage addresses for the constants (as well as for the variables) appearing in the program. We allocate the locations 0017, 0018, and 0019 for the constants 0, 1, and 1000 appearing in the flowchart and specify the proper values for these words.

We assume that the storage locations 0100, 0101, and 0102 have been allocated for the variables NEXT, LATEST, and SUM, but that no values have been placed in these words. As execution of the SAMOS program for the Fibonacci algorithm starts, the state of storage is shown in Figure 1·33. This figure

| Storage Location (Address) | ± | Operation Code | | | | Address | | | | Flowchart Equivalent |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 6 7 | 8 | 9 | 10 | 11 | |
| 0 0 0 0 | . | L | D | A | 0 0 0 | 0 | 0 | 1 | 7 | |
| 0 0 0 1 | | S | T | O | 0 0 0 | 0 | 1 | 0 | 0 | |
| 0 0 0 2 | | L | D | A | 0 0 0 | 0 | 0 | 1 | 8 | |
| 0 0 0 3 | | S | T | O | 0 0 0 | 0 | 1 | 0 | 1 | |
| 0 0 0 4 | | L | D | A | 0 0 0 | 0 | 1 | 0 | 1 | |
| 0 0 0 5 | | A | D | D | 0 0 0 | 0 | 1 | 0 | 0 | |
| 0 0 0 6 | | S | T | O | 0 0 0 | 0 | 1 | 0 | 2 | |
| 0 0 0 7 | | L | D | A | 0 0 0 | 0 | 0 | 1 | 9 | |
| 0 0 0 8 | | S | U | B | 0 0 0 | 0 | 1 | 0 | 2 | |
| 0 0 0 9 | | B | M | I | 0 0 0 | 0 | 0 | 1 | 5 | |
| 0 0 1 0 | | L | D | A | 0 0 0 | 0 | 1 | 0 | 1 | |
| 0 0 1 1 | | S | T | O | 0 0 0 | 0 | 1 | 0 | 0 | |
| 0 0 1 2 | | L | D | A | 0 0 0 | 0 | 1 | 0 | 2 | |
| 0 0 1 3 | | S | T | O | 0 0 0 | 0 | 1 | 0 | 1 | |
| 0 0 1 4 | | B | R | U | 0 0 0 | 0 | 0 | 0 | 4 | |
| 0 0 1 5 | | W | W | D | 0 0 0 | 0 | 1 | 0 | 2 | |
| 0 0 1 6 | | H | L | T | 0 0 0 | 0 | 0 | 0 | 0 | |
| 0 0 1 7 | + | 0 | 0 | 0 | 0 0 0 | 0 | 0 | 0 | 0 | The constant 0 |
| 0 0 1 8 | + | 0 | 0 | 0 | 0 0 0 | 0 | 0 | 0 | 1 | The constant 1 |
| 0 0 1 9 | + | 0 | 0 | 0 | 0 0 0 | 1 | 0 | 0 | 0 | The constant 1000 |
| 0 1 0 0 | | | | | | | | | | The variable NEXT |
| 0 1 0 1 | | | | | | | | | | The variable LATEST |
| 0 1 0 2 | | | | | | | | | | The variable SUM |

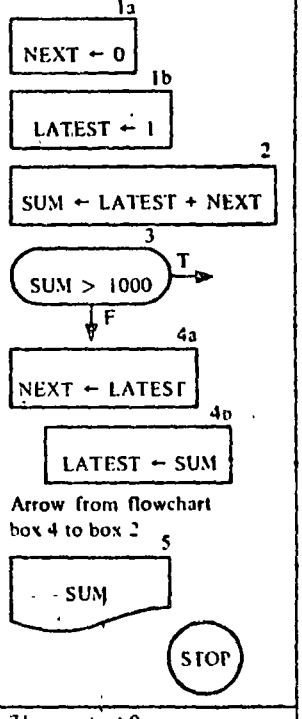



FIGURE 1·33
Fibonacci sequence algorithm. The ± column and columns 5, 6, and 7 may be left empty on lines containing instructions.

also illustrates a "coding form" on which one might have written the SAMOS program (gray-colored information). You will notice several new SAMOS operations not previously seen. These are explained in the following discussion.

**Discussion**

The instructions in storage addresses 0004, 0005, and 0006 have already been discussed. Before looking at the other instructions, remember that the variables are in storage locations 0100 through 0102.

From previous discussions you should see that the instruction found at 0000 will, when executed, copy the value in 0017 (i.e., the number 0) into the accumulator. Next, the instruction in 0001 copies the value in the accumulator into the word at address 0100. Together these steps are equivalent to assigning 0 to the variable NEXT. Similarly, the instructions in addresses 0002 and 0003 are equivalent to assigning the value 1 to the variable LATEST.

Remember that the control unit executes the instructions in order until it comes to a branching instruction. The first branching instruction is found in address 0009, reading

| | B | M | I | 0 | 0 | 0 | 0 | 0 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|

The code BMI stands for "Branch on a MInus." The whole instruction means, "If the value in the accumulator is negative, go to address 0015 for the next instruction; otherwise, go on as usual to the next numbered address (0010)." We will see shortly that the value in the accumulator at this time is just

$$1000 - \text{SUM}$$

so that the value in the accumulator will be negative only in the case that

$$\text{SUM} > 1000$$

is true. In this case, the branching instruction sends us to address 0015, where we see the instruction

| | W | W | D | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|

which means, "Write the WorD in address 0102." This amounts to printing out the value of SUM.

Now why is it that when the instruction in address 0009 is reached, the number in the accumulator is

$$1000 - SUM$$

Well, on looking at the instruction in address 0007, we see that it instructs us to load the accumulator with the contents of address 0019, that is, to put the number 1000 in the accumulator. The next instruction, the one in 0008, tells us to "subtract the contents of address 0102 from the accumulator and put the result in the accumulator." Since the contents of 0102 are just the value of SUM, this amounts to the placing of

$$1000 - SUM$$

in the accumulator.

You should be able to verify for yourself that the instructions in addresses 0010 through 0013 accomplish the assignments indicated in the right-hand column of Figure 1·33.

The instruction in address 0014 needs to be described.

| | B | R | U | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|

BRU stands for "BRanch Unconditionally." The meaning of the entire instruction is, "Go back to address 0004 for the next instruction and continue in order from there." You can see that this corresponds to the arrow from flowchart box 4 leading back to flowchart box 2, where we repeat the summing step.

The instruction in 0016, of course, stands for HaLT and amounts to stopping the computing process.

You can best understand all this by tracing through the SAMOS program by hand, keeping a record of the following details.

1. Which instruction is being executed.
2. The value in the accumulator.
3. The values in the addresses 0100, 0101, and 0102 (the values of NEXT, LATEST, and SUM).

Notice that the instructions in addresses 0000 through 0016 are never altered, nor are the contents of the locations 0017 through 0019 (the constants 0, 1, and 1000).

## EXERCISES 1·6

1. Construct a list of SAMOS instructions for the flowchart in Figure 1·14. You will need two additional types of instructions. The first is

| | OPERATION | | | | | | ADDRESS | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| | R | W | D | 0 | 0 | 0 | 1 | 0 | 0 | 5 |

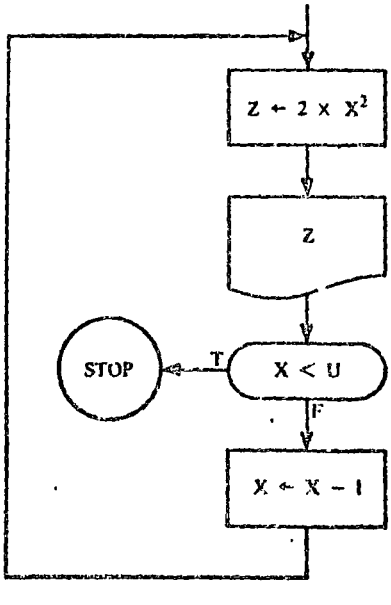
which is an instruction to read a value from a card into the computer word addressed 1005.

The second is

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | M | P | Y | 0 | 0 | 0 | 1 | 0 | 2 | 3 |

which is an instruction to multiply the value in the accumulator by the value in address 1023 and put the result in the accumulator. (Of course, in the address part of these instructions, we may put any address we wish.)

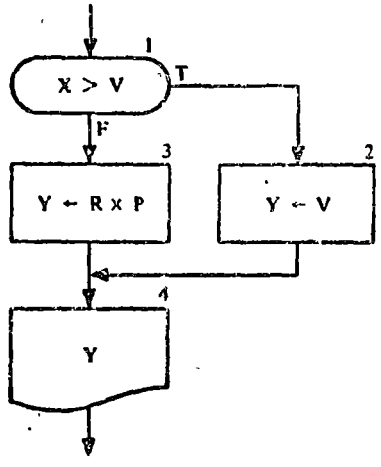
2. This question relates to the flowchart fragment and proposed SAMOS translation of it shown below. For each of your answers the assumed objective is to make the proposed SAMOS fragment consistent with the given flowchart fragment.




| Loc | Opcode | | Addr |
|---|---|---|---|
| 0018 | LDA | 000 | 0500 |
| 0019 | MPY | | 0351 |
| 0020 | MPY | | 0351 |
| 0021 | STO | | 0451 |
| 0022 | WWD | | 0451 |
| 0023 | LDA | | 0351 |
| 0024 | SUB | | 0401 |
| 0025 | | | 0030 |
| 0026 | LDA | | 0351 |
| 0027 | SUB | | 050? |
| 0028 | STO | | 035? |
| 0029 | BRU | | |
| 0030 | | | |

(a) With what memory location must the variable X be associated?
(b) What operand address is needed for the BRU instruction that is shown at location 0029?
(c) What should be the operation code for the instruction at location 0025?
(d) What operation code is needed for the instruction located at 0030?

**3.** This question relates to the flowchart fragment and proposed SAMOS translation of it shown below. For each of your answers, assume the objective is to make the proposed SAMOS fragment consistent with the given flowchart fragment.



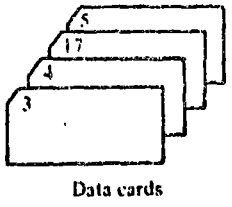

| Loc | Opcode | | Addr |
|---|---|---|---|
| . | | | |
| . | | | |
| . | | | |
| 0017 | LDA | 000 | 0100 |
| 0018 | SUB | 000 | 0200 |
| 0019 | BMI | 000 | 0024 |
| 0020 | LDA | 000 | 0105 |
| 0021 | MPY | 000 | 0107 |
| 0022 | STO | 000 | |
| 0023 | BRU | 000 | |
| 0024 | LDA | 000 | 0100 |
| 0025 | STO | 000 | 0201 |
| 0026 | WWD | 000 | 0201 |
| . | | | |
| . | | | |
| . | | | |

(a) With what location must the variable X be associated?
(b) What is the operand address that should be filled in for the BRU instruction shown at location 0023?
(c) What should be the value of the address field for the STO instruction at location 0022?
(d) If at location 0022 the STO were replaced by a BRU operation code, what then would be the appropriate value for the address field?

**4.** This question relates to the following SAMOS program and the four data cards displayed to the right of it; you are to assume that the given SAMOS program executes with the data cards shown. The DIV (divide) instruction produces an *integer* quotient (see Appendix A).

```
0000    RWD  000  0012
0001    RWD  000  0013
0002    LDA  000  0012
0003    DIV  000  0013
0004    MPY  000  0013
0005    STO  000  0014
0006    LDA  000  0012
0007    SUB  000  0014
0008    STO  000  0014
0009    WWD  000  0014
0010    BRU  000  0000
0011    HLT  000  0000
```

SAMOS program



Data cards

**(a)** Which of the following is a *false* statement?
- (1) The instruction at 0011 will never be reached.
- (2) Only two values will be printed.
- (3) All four data cards will be read.
- (4) The first value printed will be 3.
- (5) Three values will be printed.

**(b)** Which of the following is a *true* statement?
- (1) The program will halt whenever the result of a division is zero.
- (2) The instruction at 0011 would be executed using the given set of data if the instruction at 0010 were revised to BMI 000 0000.
- (3) This program inputs two numbers, selects the larger, and prints its value.
- (4) All three of the above statements are false.

Problems 5 through 7. The following three problems involve programs to be written in SAMOS machine language and run on a computer using a SAMOS simulator. If you do not have a computer available, your final result will be a SAMOS coding form showing your program.

**5.** Draw a flowchart and write and run a SAMOS program to find the areas (to the nearest integer) of circles with each of the following radii: 0, 1, 2, 3, . . . , 10, 11, 12. Use $\pi = 22/7$. The output is to consist of each radius value followed by the associated area, that is,

```
0  <-- First radius
0  <-- First area
1  <-- Second radius
3  <-- Second area
```

This SAMOS program should not execute any *input* steps. Storage locations will be required for instructions and for all constants, including 22, 7, the current radius, 1 (to increment the radius to the next value), and a number (11, 12, or 13, depending on your particular flow chart) to test against to determine when to branch to *halt*.

*Question* Will SAMOS give the same result when you compute $(22/7) \times r^2$ as when you compute $(22 \times r^2)/7$? If not, which gives a better result? Why?

**6.** Draw a flowchart and write and run a SAMOS program to do the following:

For the values from 1 to 10 inclusive (i.e., $1 \leq X \leq 10$), evaluate the following mathematical expression:

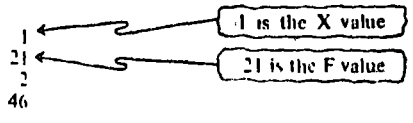
$$F = 5X^2 + 10X + 6$$

Print out the value for X and F after each evaluation.

*Example* The first value of X will be 1. For this value,

$$F = 5(1) + 10(1) + 6$$
$$F = 5 + 10 + 6 = 21$$

Thus the numbers 1 and 21 will be printed out, and F will then be evaluated for X = 2, 3, . . . , 10. The complete output will consist of 20 numbers:




*Note Additional Information for Problem 6*

1. No data cards will be needed for this program.
2. The values from 1 to 10 need not all be stored at the beginning of the program.
3. You must include some way to terminate your program after the final value has been processed and printed.

**7.** Draw a flowchart using variables C, X, TALLY, SUM, and AVG, and write and run a SAMOS program to do the following:

(a) Read a value for the variable C.
(b) Read a value for the variable X.
(c) Check to see whether X equals 9999. If X does not equal 9999 then check whether X equals C. If X equals C, then return to

step (b). If X does not equal C, then add one to a counter called TALLY and add X to the variable SUM and then return to step (b).

If X equals 9999, then no more data cards are to be read. At this point print out the values of TALLY and SUM. Compute AVG, the quotient of SUM and TALLY (AVG = SUM/TALLY). Print the value of AVG.

*Note Additional Information for Problem 7*

1. In your flowchart the average will be a variable AVG. The variable TALLY will hold a count of how many values of X are *not* equal to C. The assignment

TALLY ← TALLY + 1

will be needed. The sum of the values of X not equal to C will be called SUM. What should be the initial values of TALLY and SUM?

2. The value 9999 is called a *sentinel* value. Its purpose is to indicate that all the values of X have been read and processed. (In computer language a sentinel value is said to represent the end of file, i.e., the end of data.) Therefore, your data deck will consist of a value for C, the given values for X, and the value 9999. (See Section 2.2 for additional discussion of sentinels.)

3. In SAMOS the only conditional branch instruction is BMI (Branch On Minus). The programmer faces a problem when he needs to check whether the values of two variables are equal or whether the value of a variable is equal to some constant value. The following is one method of determining whether the values of the variables A and B are equal. First, subtract the value of B from the value of A and, if the result is not negative, subtract the value of A from the value of B. If this result is not negative, we can conclude that the values of A and B are equal.

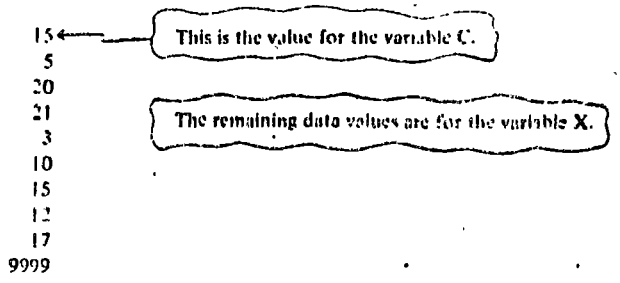
*Examples*

(a) A = 6 and B = 9
A − B = 6 − 9 = −3
Result: A ≠ B since A − B is a negative number.

(b) A = 5 and B = 5
A − B = 5 − 5 = 0
B − A = 5 − 5 = 0
Result: A = B since neither subtraction produced a negative number.

(c) A = 4 and B = 3
A − B = 4 − 3 = 1
B − A = 3 − 4 = −1
Result: A ≠ B since B − A is a negative number.

4. The data for Problem 7 are as follows:




## 1·7 BITOS

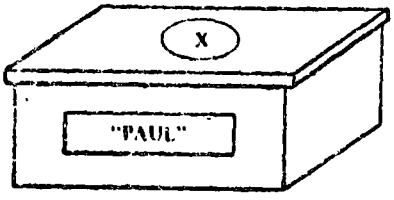
Most of the storage capacity of a 61-bit SAMOS word goes unused when a small integer, for instance, 2, is represented. Conversely, even though a number is known to very great precision, a 61-bit word has a fixed capacity to represent digits. Character strings, such as names and addresses, for instance, vary greatly in length. In general, information comes in many sizes and lengths, and it would be exceedingly convenient to have computer storage responsive to this fact.

The SAMOS language is heavily influenced (i.e., constrained) by its *word-organized* storage system. We briefly mention here another kind of computer storage called BITOS (BIT-Organized SAMOS), whose storage is structured in a more flexible and natural way—natural for the processing of different types of information. The BITOS storage is best thought of as a single sequence of bits instead of a single sequence of words that are, in turn, sequences of bits. For example, a BITOS store roughly equivalent to the SAMOS store contains 610,000 bits whose addresses are 0 through 609,999 respectively. To fetch a unit of information of some known length, one must specify the "bit address" of the beginning of the desired information unit together with its "bit length." Thus,

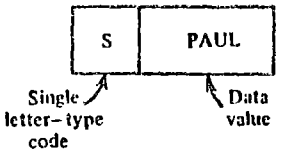
| op code | address | length |
|---|---|---|
| LDA | 24972 | 39 |

is the way one might write out a BITOS instruction to load the accumulator with a data value 39 bits long beginning at bit location 24972.

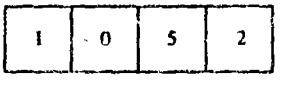
The information containers in a BITOS machine resemble the storage boxes of the conceptual model SIMPLOS in that the capacity of the containers is arbitrary.




There is a second important way in which the containers of SIMPLOS and BITOS resemble one another. In both cases, values stored in these containers are *self-describing*. Note that the SIMPLOS Reader is able to deduce that a storage box contains a character string value as opposed, say, to a numerical value, because he is able to see the quotation marks. The fetching mechanism of BITOS can convey the same type of information to its processing unit because the data object in each container consists of two parts: a code that describes the type or nature of the value and the data value itself. For example, suppose the container associated with X is located at bit address 3001, and suppose character coding for the BITOS store employs the same 6-bit representation used in SAMOS. We might expect to see at that location:




where the type code S stands for string. Then, to represent a string of 4 characters would require 24 bits for the string itself and 6 more bits (for the letter S) to identify the 24 bits as a string. For a string of 91 characters, 546 bits are needed for the string and 6 more for the type code—or 552 bits in all. If a nonnegative integer variable, AGE, never requires more than three digits we can picture the corresponding BITOS container as




for a data value of 52. Here the type code I denotes *integer*.

To recapitulate, in BITOS one defines the size of the container to fit the need. That is, the store is divided up into containers that reflect their actual use. Every reference to a container consists of the bit address of the container and its length. The container itself holds as part of the data object a code that makes the remainder of the information *in* the container self-describing. Each time a new container is needed, a section of the store large enough to hold the required number of bits is "partitioned" for this purpose. When this container

is no longer needed, that section of store and others like it are repartitioned, that is, reused, typically in different container sizes, to suit new needs. To make an analogy with SIMPLOS, imagine that all storage boxes are constructed to fit dimensions of the data values they are to contain. (The Affixer who pastes the sticker on the storage box can also adjust the size of the box if necessary.)

## 1·8 Floating-point representation

Only a few of the ideas about SAMOS and BITOS need to be remembered. One of the important ideas is the sequential manner in which the computer works, that is, the step-by-step way in which the computer performs its tasks. The order in which the tasks are performed is just as important as what is accomplished.

Another property of computers that we must understand is the *finite word length.* We have seen that SAMOS words consist of 10 characters and a sign, so that the largest number representable in this coding system is

+9,999,999,999

a rather large number, but still finite. Although BITOS store may use very "long" containers, they are still finite, so the limitation on what can be represented, although less constricting, still exists in principle. From a practical viewpoint, integer containers, whether in a SAMOS-like or in a BITOS-like store, are sometimes very unsuitable. Consider a variable that, from time to time, has various values assigned to it, sometimes very small integers and at other times very large integers. The storage container for such a variable cannot always be used efficiently if it must be large enough for the largest possible integer value that will be assigned to it.

To cover this situation there are other ways of coding numbers that not only solve this problem but also allow us to work with real numbers as well as integers. One of the most common of these alternate codings is *floating-point* form, which is related to the so-called "scientific notation."

To see how this works, recall that any decimal numeral such as

−382.519

can be expressed as

$$-.382519 \times 10^3$$

in which there is a decimal point (just after the sign, if any) followed by a string of digits (the first not zero) and multiplied by a suitable power of 10. We can code numbers in this way by reserving three character positions for the exponent. The result is shown in Figure 1·34 for −382.519.

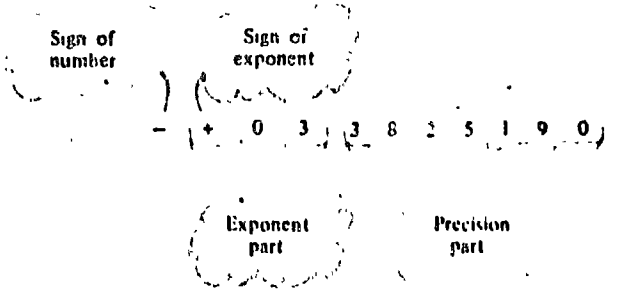



FIGURE 1·34
Anatomy of a floating-point number for a fixed-word size store.

Some examples of how to code numbers including integers in this system are given in Figure 1·35. In this figure, we see that the 8-digit representation of $\pi$, as given in the first column of the third entry, has to be chopped to 7 digits of precision because of space requirements. The same holds true for 1/3 and 11/7 at the bottom of the table. Thus we see that in a computer even a simple fraction such as 1/3 cannot be represented exactly, but only to a close approximation. This characteristic of "finite word length" presents important prob-

| Number | Floating-Point Form | Coding of Floating-Point Form |
|---|---|---|
| 4 | $.4 \times 10^1$ | + + 014000000 |
| −999999000 | $-.999999 \times 10^9$ | − + 099999990 |
| 3.1415926 | $.31415926 \times 10^1$ | + + 013141592 |
| −273.14 | $-.27314 \times 10^3$ | − + 032731400 |
| .0008761 | $.8761 \times 10^{-3}$ | + − 038761000 |
| .73 | $.73 \times 10^0$ | + + 007300000 |
| 1/3 | $.333333333 \times 10^0$ | + + 003333333 |
| 11/7 | $.157142857 \times 10^1$ | + + 011571428 |

FIGURE 1·35
Floating-point coding of numbers in a fixed word-sized store.

lems that will be discussed in various places in this text, especially Chapter 11.

In floating-point form we can represent large numbers, but for computers such as SAMOS with fixed-word size stores the price we pay is giving up three places of precision. Were SAMOS to use floating-point numbers, the largest number representable in floating-point form would be:

| + | + | 9 | 9 | 9 | 0 | 9 | 9 | 9 | 9 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

which represents the number

999,999,900,000,000,000,000,000,000
000,000,000,000,000,000,000,000,000
000,000,000,000,000,000,000,000,000
000,000,000,000,000,000

Similarly, there is a smallest positive number that could be represented:

| + | − | 9 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

.000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 1

which is very small, indeed.

Coding in floating-point form for a BITOS-like machine could be quite similar to the scheme shown in Figure 1·35. On the other hand, since the size of the container may be chosen to fit the particular "needs" of a given variable, the size of the precision part could easily be permitted to vary as required. For that matter, the size of the exponent part could also be expanded or contracted to fit the need.

In summary, both SAMOS-like and BITOS-like machines are often built to operate on numbers coded in floating-point form. However, in our discussions of SAMOS, in particular in the description given in the Appendix, the machine is initially described as if it were not capable of dealing with numbers coded in floating-point form, but only with numbers coded as integers.

EXERCISE 1·8

1. Several million pocket-sized electronic calculators are now being produced annually. They are becoming relatively accessible to the average student. Many of these calculators, such as the one shown in Figure 1·36, use floating point arithmetic.

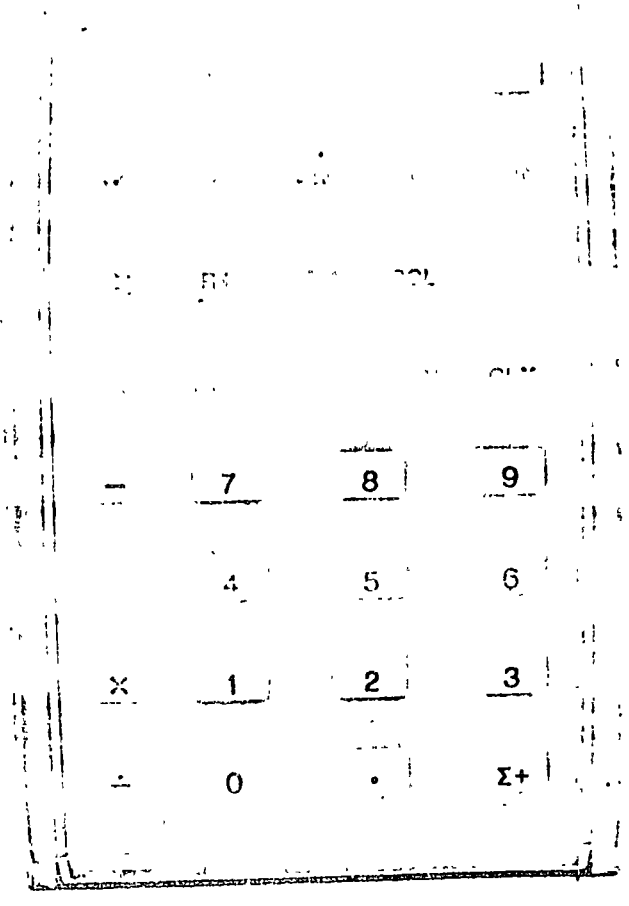



FIGURE 1·36
A popular hand-held calculator.

(a) Locate an electronic calculator and compare the method it uses to represent floating-point numbers with the method used in SAMOS.

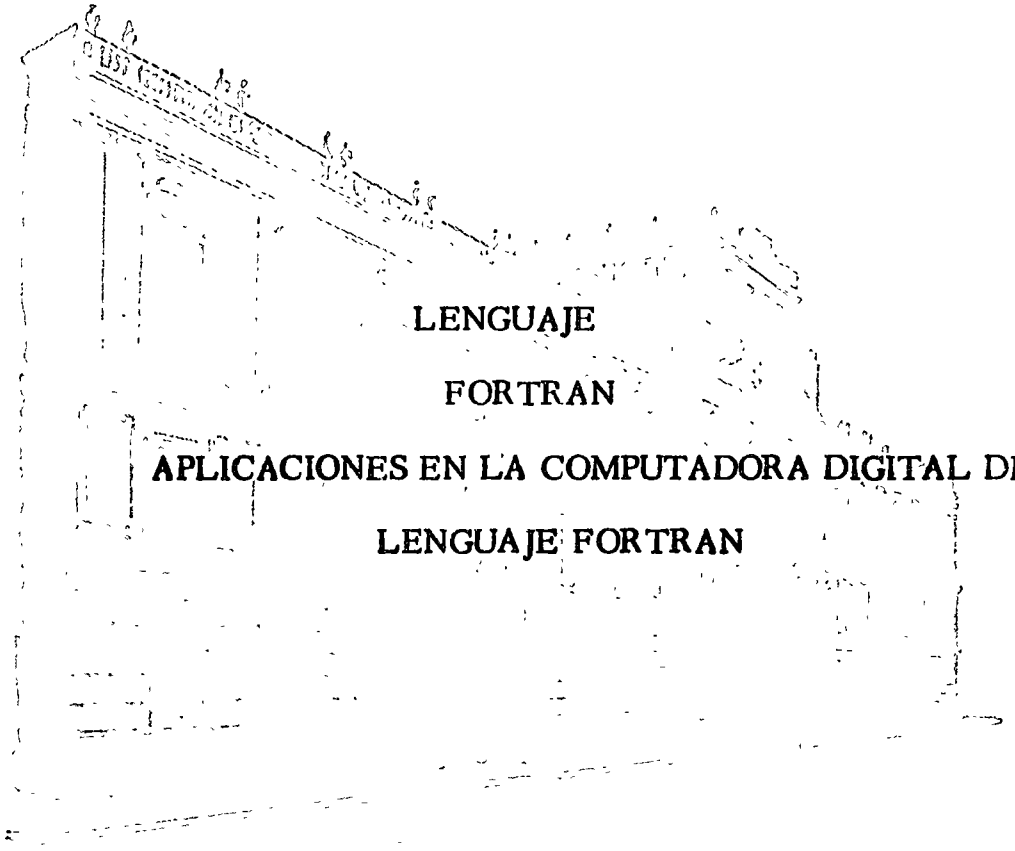
(b) Which form do you prefer and why?
(c) Run several simple computations on the pocket calculator, such as $a + b$, to obtain c

or $a \times b$, to obtain d, etc.

where a and b are keyed in manually as real numbers in ordinary decimal notation, for example, $-382.519$. Determine under what conditions results c and d are displayed in floating-point form.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA



## LENGUAJE

## FORTRAN

## APLICACIONES EN LA COMPUTADORA DIGITAL DEL

## LENGUAJE FORTRAN

ING. HERIBERTO OLGUIN ROMO
ING. RICARDO CIRIA MARCE

Febrero-Marzo 1977

Palacio de Minería Calle de Tacuba 5, primer piso. México 1, D. F. Tels.: 521-40-23 521-73-35 5123-123

H E R I B E R T O  O L G U I N  R O M O

R I C A R D O  C I R I A  M E R C E

# F Ø R T R A N

## ELEMENTOS DE UN SUPERLENGUAJE DE PROGRAMACION: FØRTRAN



..2

## 1.- Introducción al lenguaje FØRTRAN

El lenguaje FØRTRAN, cuyo nombre corresponde a las primeras letras de las palabras inglesas FORmula (fórmula) y TRANslation (traducción), es un lenguaje de programación orientado a problemas matemáticos y se emplea en casi todas las computadoras del mundo. Debido a su parecido con el lenguaje aritmético común, el FØRTRAN simplifica la preparación de problemas que pueden resolverse mediante una computadora. Los datos e instrucciones se pueden organizar mediante una secuencia de enunciados fortran; estos constituyen el llamado Programa Fuente.

Todas las computadoras que "entienden" el lenguaje FØRTRAN, tienen lo que se llama un Compilador Fortran, llamado también traductor o interprete, el cual analiza los enunciados fortran y los traduce a un Programa Objeto, el cual queda en Lenguaje de Máquina.

Un programa escrito en lenguaje FØRTRAN se puede procesar en cualquier máquina que tenga un Compilador Fortran. Esto nos indica que el lenguaje es independiente para cada máquina, o sea que el compilador se debe preparar en cada caso teniendo en cuenta la máquina que ha de usarse en particular; puesto que las máquinas difieren en su organización interna, se ha desarrollado un número de "dialectos" del Lenguaje FØRTRAN, cada uno de los cuales es apropiado para una clase de máquinas. Las diferencias entre los

varios dialectos son mínimas y se ajustan el uno al otro facilmente.

## 1.1 El alfabeto

El alfabeto FØRTRAN esta constituido de caracteres que son símbolos familiares de escritura y de teclados de máquinas de escribir, asi como de dispositivos especiales de perforación; dichos caracteres son:

Alfabéticos: A B C D E F G H
I J K L M N
* Ø P Q R S T U V W X Y Z

Numéricos: 0 1 2 3 4 5 6 7 8 9

Símbolos: + - * / = . , ( ) ' @

De este alfabeto se construyen todos nuestros símbolos, expresiones y enunciados que se utilizan en el lenguaje FØRTRAN.

## 2.- Números

Los números pueden representarse en diferentes formas, las cuales se asemejan a los símbolos de la aritmética general; pero debido a la estructura interna de las computadoras se establecen las convenciones de: Punto Fijo y Punto Flotante que proporcionan facilidades para su manejo en FØRTRAN. Los símbolos de punto fijo

* La letra O la expresaremos como Ø para diferenciarla del N° cero.

se usarán solamente con números enteros y los cálculos asociados se denominarán aritmética de los enteros o modo entero; mientras que la aritmética de los números reales se hará en la forma de punto flotante y se llamará aritmética de los reales o modo real. Debido a que también es necesario distinguir las constantes (números que no cambian durante toda la ejecución de un programa) de las variables (números que pueden cambiar), surgen cuatro clases de símbolos para los números.

## 2.1 Constantes enteras.

Dependiendo del tipo de computadora se podrán representar por un cierto número de dígitos, asi para IBM-1130 se representan mediante cinco dígitos sin el punto decimal. Si el entero es negativo, los dígitos deberán ser precedidos del signo menos; si el entero es positivo el signo es opcional.

Ejem. Símbolos para constantes enteras pueden ser entre otras:

1976 +1 0 +1976 -1976

Símbolos que no se aceptan para constantes enteras:

7483282 (más de cinco dígitos)

1976. (el punto decimal no se permite)

## 2.2 Constantes reales

Dependiendo del tipo de computadora, las constantes reales se podrán representar por varios dígitos, pero en el caso de la

IBM-1130 sólo se admiten siete dígitos con punto decimal pudiendose colocar al principio de los dígitos, al final o entre dos dígitos cualesquiera. Cuando aparece un punto en una constante su tratamiento será de punto flotante. Si la constante real es precedida de un signo menos, se indicará que es negativa, si es positiva el signo es opcional.

Ejem. Símbolos para constantes reales pueden ser entre otras:

| | | | |
|---|---|---|---|
| 1976. | -.00001976 | +12,345 | -12.345 |
| -.007 | .007 | 5,348 | 0.3 |

Símbolos que no se aceptan para constantes reales:

123456789.32 (más de siete dígitos significativos)

5343 (falta el punto decimal)

Para representar las constantes reales existe también la llamada forma exponencial; esta la podemos representar mediante una letra E y una constante entera de uno o dos dígitos, positiva o negativa. Esta constante entera es un exponente del número diez; el signo menos es para los exponentes negativos y para los positivos, el signo es opcional. En FØRTRAN, la presencia del exponente hace que el uso del punto decimal sea opcional.

Ejem.

| Forma exponencial | Forma no exponencial |
|---|---|
| 1.328E2 | 132.8 |
| 1.328E02 | 132.8 |

| | |
|---|---|
| 1.328E00 | 132.8 |
| -4.724E-03 | -.00472 |
| +7.61E3 | 7610. |
| -6432E-3 | -6.432 |

## 2.3 Variables enteras

Estas se representan por combinaciones de una a cinco letras y digitos (IBM-1130), no se permiten otros caracteres y el primer caracter deberá ser una de las letras I, J, K, L, M ó N. El primer caracter de una variable es el que indica si es entera o real. Durante la ejecución de un programa, las variables enteras deberán restringirse a valores enteros.

Ejem. Símbolos para variables enteras pueden ser, entre otros:

NUMCT KILO N1 N2 M10 KONT

IJALC JCLAV MARY KONT1 L1976

Símbolos no aceptables para variables enteras:

CUENT (el primer caracter debe ser I, J, K, L, M ó N).

KONTADOR (demasiados caracteres)

12.34 (sólo se aceptan letras y números)

## 2.4 Variables reales

Estas se representan por combinaciones de una a cinco letras y dígitos (IBM-1130), no se permiten otros caracteres y el primer

caracter tiene que ser necesariamente una letra diferente a I, J, K, L, M ó N. Durante la ejecución de un programa dichas variables se deben restringir a valores reales.

Ejem. Símbolos para variables reales pueden ser, entre otros:

FUERZ VELOC ACEL1 CUENT A1 A2

ALFA VIELA RA42 X1 PROD SUMA

Símbolos no aceptables para variables reales:

A3.8 (el punto no es letra o número)

CORRIEN (demasiados caracteres)

3 BASO (el primer caracter debe ser una letra)

MUMCT (el primer caracter no puede ser M)

3.- peraciones aritméticas!

Las operaciones aritméticas y los símbolos que se utilizan en FØRTRAN son:

| | Ejem. | Algebra | FØRTRAN |
|---|---|---|---|
| Adición | + | a + b | A + B |
| Sustracción | - | a - b | A - B |
| Multiplicación | * | a b | A * B |
| División | / | $\frac{a}{b}$ | A / B |
| Exponenciación | ** | $a^2$ | A * A |
| | | $a^2$ | A** 2 |

4.- Expresiones aritméticas

En base a lo expuesto anteriormente podemos ahora formular

expresiones aritméticas en lenguaje FØRTRAN y nos daremos cuenta que son muy similares a las expresiones aritméticas del algebra común.

| Expresiones FØRTRAN | Expresiones Comunes |
|---|---|
| A**2-B**2 | $a^2-b^2$ |
| B**2-4.*A*C | $b^2-4ac$ |
| (A+B)/2. | $\frac{1}{2}(a+b)$ |
| 2*K-J+N | 2k-j+n |
| C+B-3.*A | c+b-3a |

4.1 Reglas para las expresiones aritméticas

Las reglas a las que debemos sujetar las expresiones aritméticas son necesarias debido a la estructura de las computadoras y al observarlas tendremos un ahorro en el tiempo de ejecución de un programa.

Regla 1 Si nos fijamos en las expresiones FØRTRAN anteriores nos damos cuenta que: Todas las constantes y variables en una expresión deben estar en el mismo modo, esto es, todas deben ser enteras o todas deben ser reales. (Como toda regla existe su excepción que mencionaremos más adelante).

Es necesario consultar los manuales de cada máquina, ya que como hemos mencionado anteriormente dependerá esta regla del tipo de computadora. Por lo pronto la consideraremos como se ha indicado.

Regla 2 A**I, I**J y A**B son exponenciaciones permitidas. En el caso A**I se mezclan los modos y es la excepción a la Regla 1, pero sabemos que esta exponenciación significa multiplicaciones sucesivas (así B**3= B*B*B), mientras que las potencias no enteras implican cálculos más sofisticados. Nos damos cuenta que I**A, no es forma de exponenciación permitida (en algunas máquinas sí se permite).

Regla 3 Deberá tenerse en cuenta que las operaciones se ejecutarán con las siguientes prioridades:

1) Las operaciones indicadas dentro de los paréntesis más internos se ejecutan en primer lugar.
2) Exponenciación.
3) Multiplicación y división.
4) Adición y sustracción.

Entre las operaciones de igual prioridad, el orden de ejecución es de izquierda a derecha.

Ejem. Si A=5., B=8. y C=2.

A+B-3.*C se calculará en el siguiente orden:

3.*2.=6. 5.+8.=13. 13.-6.=7.

B**2-4.*A*C se calcula en el siguiente orden:

8.**2=64. 4.*5.=20. 20.*2.=40.

64.-40.=24.

Si A=5., B=8., C=2. y D=1.6

Entonces (A+B)/C se calcula en el siguiente orden:

5.+8.=13.  13./2.=6.5

Mientras que A+B/C se calcula en el siguiente orden:

8./2.=4.  5.+4.=9.

Ahora si deseamos calcular (A+C)**2 conducirá a:

5.+2.=7.  7.**2=49.

Mientras que A+C**2 conducirá a.

2.**2=4.  5.+4.=9

Ahora si: (A*B)/(C*D)=40./3.2=12.5

Entonces: A*B/C*D=40./C*D=20.*D=32.

Finalmente si tenemos paréntesis dentro de otros paréntesis se tiene:

(A*(B+C))**2=(A*10.)**2=50.**2=2500.

B+C tiene la más alta prioridad por encontrarse en el paréntesis más interno.

(A*B+C)**2=(40.+2)**2=42.**2=1764.

A*(B+C)**2=A*10.**2=A*100.=500.

A*B+C**2=A*B+4.=40.+4.=44

Debemos tener cuidado en expresar lo que deseamos realizar.

Regla 4 No deberemos colocar un signo de operación antes de un signo más o menos, esto es, no deberemos poner dos signos de operación juntos.

Ejem. A*-B I+-J M-+N A/-B

Estas expresiones deberán sustituirse por:

A*(-B) I+(-J) M-(+N) A/(-B)

4.2 Funciones predefinidas en lenguaje FØRTRAN

Estas funciones predefinidas que proporciona el lenguaje FØRTRAN son de tipo de biblioteca. Para utilizarlas usaremos el nombre de la función seguido de un argumento que deberá estar entre paréntesis. Dichos argumentos pueden ser variables simples ó con subíndices, constantes, expresiones aritméticas u otras funciones predefinidas en FØRTRAN.

Para IBM - 1130 tenemos:

| NOMBRE | FUNCION EJECUTADA | NUM. DE ARGUMENTOS | TIPO DE ARGUMENTO(S) | TIPO DE FUNCION |
|---|---|---|---|---|
| SIN | Seno trigonométrico (argumento en radianes) | 1 | Real | Real |
| COS | Coseno trigonométrico (argumento en radianes) | 1 | Real | Real |
| ALOG | Logaritmo natural | 1 | Real | Real |
| EXP | Argumento de potencia del número e. | 1 | Real | Real |
| SQRT | Raíz cuadrada | 1 | Real | Real |
| ATAN | Arco tangente | 1 | Real | Real |

del número e

| | | | | |
|---|---|---|---|---|
| ABS | Valor absoluto | 1 | Real | Real |
| IABS | Valor absoluto | 1 | Entero | Entero |
| FLOAT | Convertir argumento de entero a real | 1 | Entero | Real |
| IFIX | Convertir argumento de real a entero | 1 | Real | Entero |
| SIGN | Transferencia de signo (Arg.$_1$ recibe signo de Arg.$_2$) | 2 | Real | Real |
| ISIGN | Transferencia de signo (Arg.$_1$ recibe signo de Arg.$_2$) | 2 | Entero | Entero |
| TANH | Tangente Hiperbólica | 1 | Real | Real |

Ejem. SQRT (B**2-4.*4.*A*C) indica que a lo que se encuentra entre paréntesis se le sacará la raíz cuadrada. SIN (BETA) indica que se obtendrá el seno trigonométrico de el valor de la variable BETA.

5.- Enunciados

Los enunciados son las unidades basicas con las cuales se construyen los programas FØRTRAN. Podemos clasificarlos de acuerdo a su función en grupos como:

1.- Aritméticos de asignación

2.- De control

3.- De entrada y salida

4.- De especificación

5.1 Los enunciados aritméticos de asignación

Se forman con las expresiones presentadas anteriormente y

nos indican los cálculos particulares que deben hacerse. Su forma es:

Variable = Expresión aritmética

El significado del signo = es el de asignación, esto es, que deberá calcularse el valor de la expresión a la derecha del signo = y su valor se asignará a la variable que se encuentre a la izquierda del signo, la cual tiene una localidad en la memoria de la computadora.

Ejem. Si A=5., B=8., C=2. y D=1.6

X=(A+B)/C se le asignará a la X el valor 6.5

ALO=(A+B)**2 se le asignará a ALO el valor 169.

RAI=SQRT(B*C) se le asignará a RAI el valor 4.

Algo diferente al algebra normal es el enunciado A=A+3. el cual no debe alarmarnos ya que indica que a la localidad de memoria con el nombre A se le asignará el nuevo valor A+3. esto es:

Si A=5. y A=A+3. entonces:

A=5.+3. A=8. o sea que la variable A se le asigna el valor de 8. y el valor anterior que fué 5. se pierde.

## 5.2 Los enunciados de control

Debido a que los enunciados de un programa FØRTRAN se ejecutan en el orden que aparecen y que en muchas ocasiones queremos transferir la ejecución a otros enunciados si se satisface una

cierta condición, FØRTRAN nos permite numerar dichos enunciados. Un número de enunciado debe ser una constante entera de uno a cinco caracteres.sin el signo más o menos; el número se coloca a la izquierda del enunciado.

Ejem.
```
    3 CONT = CONT+1.
   24 RAIZ = SQRT (A**2+B**2)
```

5.2.1 El enunciado GØ TØ

Este toma la forma GØ TØ N en donde N es un número de enunciado.

El GØ TØ produce un salto incondicional; así GØ TØ 3 envia la ejecución al enunciado número 3 que puede ser la instrucción de conteo del ejemplo anterior. GØ TØ 24 pasa el control al enunciado 24 que puede ser el del ejemplo anterior.

Ejem. Supongamos que unos de los enunciados de un programa son:

```
      I = 1              Esto nos representa la suma de
      ISUM = 0           los números enteros, desde luego
    1 ISUM = ISUM+I      es necesario ponerle otros enuncia-
      I = I+1            dos pero por el momento nos aclara
      GØ TØ 1            lo indicado.
```

5.2.2 El enunciado IF

Debido a que las computadoras estan diseñadas a base de circuitos lógicos y el pensamiento del ser humano debe

ser de este tipo, nos concretaremos el IF lógico, además de que el alumno ya tiene elementos de algunos operadores de relación comc OR, AND y NOT.

El IF lógico es de la forma:

IF (L) S

L= expresión lógica que puede tener dos valores: Verdadero o Falso.

S= cualquier enunciado FØRTRAN diferente de: un DØ, un enunciado de especificación o de otro IF lógico.

Si L es falso (.FALSE.) entonces se ignora S y la computación continúa al siguiente enunciado. Si L es verdadero (.TRUE.) el enunciado S se ejecuta en seguida.

Resulta interesante hacer notar que si L es relativamente complicada, éste IF puede ser el equivalente de varios IF aritméticos.

Para formar las expresiones lógicas (L) utilizaremos los operadores de comparación y los de relación.

Operadores de comparación:

| Símbolo Matemático | Significado | Símbolo FØRTRAN | Significado Inglés |
|---|---|---|---|
| < | Menor que | .LT. | Less than |
| > | Mayor que | .GT. | Greater than |
| ≤ | Menor o igual a | .LE. | Less or equal |
| ≥ | Mayor o igual a | .GE. | Greater or equal |

| | | | |
|---|---|---|---|
| = | Igual a | .EQ. | Equal |
| ≠ | Diferente a ó No igual a | .NE. | Not equal |

Operadores de relación:

| | | | |
|---|---|---|---|
| ∪ | Unión | .OR. | ó ("o inclusive) |
| ∩ | Intersección | .AND. | y ("al mismo tiempo) |
| — | Complemento | .NOT. | no |

Para valuar una expresión lógica se hará con las siguientes prioridades:

1.- Expresiones entre paréntesis

2.- Operadores aritméticos

3.- Operadores de comparación (.LT., .GT., .LE., .GE., .EQ. y .NE.)

4.- .NOT.

5.- .AND.

6.- .OR.

En caso de igual jerarquía la evaluación será de ezquierda a derecha.

Ejem. (1) X=5. y=0.5

IF (X.GT.3..AND. Y .LE.2.) Z=X**3+X*Y

Significa que si X>3. y (al mismo tiempo) y≤2. se asignará a Z el valor que se obtenga al calcular $X^3+XY$, esto es Z=125.+2.5=127.5

(2) IF (A.LE.X.AND.B.GE. Y .OR.C.GT.Z) GØ TØ 12

Significa que si A≤X y (al mismo tiempo) B≥Y e

verdadero ó C>Z es verdadero ó ambos, entonces se transfiere el control al enunciado 12.

```
(3)     I = 1
        ISUM = 0                        Esto nos indica
    1   ISUM = ISUM+1                   que sólo sumare-
        I = I+1                         mos los números
        IF (I.LE.100) GØ TØ 1           enteros del 1 al 100
        STOP
```

5.2.3 El enunciado DØ

Este toma la forma:

```
DØ K I = L, M, N
DØ K I = L, M
```

La segunda forma sólo se aplica cuando N=1, lo que es bastante frecuente.

K representa un número de enunciado

I representa una variable entera

L, M, N son variables enteras ó constantes sin signo.

El DØ produce la ejecución repetida de todos los enunciados que le siguen, hasta el enunciado número K. La primera vez que se ejecutan estos enunciados la variable I es igual a L, en cada paso subsiguiente I se incrementa en la cantidad N, hasta hacerse mayor ó igual a M en el paso final; en este momento se termina el llamado

lazo DØ y el control pasa al enunciado que está a continuación del enunciado K. Así, L es el valor inicial de la variable I y M su valor final. I se llama el índice del enunciado DØ y su valor corriente se puede usar en cálculos durante la ejecución del lazo. Todos los enunciados que le siguen al DØ hasta el número K inclusive constituyen el rango del DØ. También es posible que la variable I no se encuentre en ninguno de los enunciados del rango del DØ y esto nos indica que se realice la ejecución de todos los enunciados del rango del DØ M entre N veces (la parte entera de este cociente M/N). Deberemos tomar en cuenta que: el índice I se incrementa secuencial y automáticamente durante la ejecución del lazo y que se puede, en estos momentos, tratar como cualquier variable entera; el índice I queda indefinido después de terminado el lazo del DØ y puede utilizarse para cualquier uso general. El enunciado K no debe ser un enunciado de especificación ni una transferencia de control esto incluye cosas como GØ TØ, IF y DØ, así como FORMAT, END y algunos otros. Debemos considerar que no se puede desde ningún punto del programa llegar a un enunciado dentro del rango de un DØ. Y que la entrada a un DØ deberá hacerse a través del enunciado DØ. Y por último es muy frecuente que un DØ esté completamente dentro de otro.

Ilustrando gráficamente tenemos:

| Correcto | | Incorrecto | |
|---|---|---|---|
| DØ | DØ | DØ | DØ |
| DØ | DØ | DØ | DØ |
| DØ | | | DØ |

Ejem. Utilizaremos un DØ para sumar los números enteros del 1 al 100, ejemplo que ya hemos visto anteriormente.

```
      ISUM = 0
      DØ 1 I = 1,100
    1 ISUM = ISUM+1
      STOP
```

Nos damos cuenta que el DØ tiene la misma función que un IF, un GØ TØ y un contador; como podrá observarse con el ejemplo anterior.

3.2.4 El enunciado STØP

Este aparece simplemente como STØP y es el que nos indica que ha terminado la ejecución y en el caso de IBM - 1130 la computadora se detiene y el operador tendrá que hacer que continúe trabajando. Debido a ello se recomienda que se utilice el enunciado CALL EXIT, el cual pasa el control a un programa monitor que hace que la computadora continúe ejecutando los otros programas que siguen a continuación.

Tanto el STØP como el CALL EXIT podrán aparecer después de cualquier enunciado.

## 5.3 Los enunciados de entrada y salida

Estos, como su nombre lo indica, sirven para introducir y sacar información de la computadora.

### 5.3.1 El enunciado READ

Este enunciado tiene la forma READ (I, N) LISTA I y N son enteros sin signo y LISTA representa una lista de nombres de variables para las cuales se leerán valores. I designa el tipo de periférico de entrada que se utilice (lectora de tarjetas, consola, etc.). N es el número de un enunciado FØRMAT asociado al READ.

Ejem. El enunciado READ (2, 101) J, B, H

Producirá la lectura de tres números: un entero y dos reales y se almacenarán en las localidades de la memoria de la computadora designadas con las variables J, B y H en su orden. Las comas que separan éstos nombres de variables en el READ son indispensables, 2 es la unidad de entrada y 101 un FØRMAT.

### 5.3.2 El enunciado WRITE

Este tiene la forma WRITE (I, N) LISTA I y N son enteros sin signo y LISTA representa una lista de variables para las cuales se imprimen valores. I designa el tipo de periférico de salida que se utilice (impresora, cinta, etc.). N es el número de un enunciado FØRMAT aso-

ciado al WRITE.

Ejem. El enunciado WRITE (3, 108) L, X, Y

Producirá que se impriman los valores de las variables L, X y Y que se encuentren en las localidades de memoria con esos nombres, en el formato especificado por el enunciado número 108 y por la unidad de salida número 3; las comas que separan éstos nombres de variables en el WRITE son indispensables.

3.4 Los enunciados de especificación

Este tipo de enunciados no inician por si mismos los cálculos, no producen transferencia de control ni estimulan el flujo de información, pero proveen al compilador FØRTRAN de los detalles esenciales para la traducción del programa fuente en FØRTRAN al programa objeto en lenguaje de máquina ó para la conversión de datos a la entrada o la salida.

Si queremos introducir datos a la computadora lo podemos hacer mediante un enunciado que esté dentro del programa, como A = 3.1416, ésto es lo que podríamos llamar inicializar una variable; y el programa se compilaría cada vez que quisiéramos darle un valor diferente a A, lo cual resulta muy costoso, ya que las compilaciones son laboriosas. Para evitar esto se usa el enunciado READ y los valores que se le den a A podrán estar en tarjetas de datos, los cuales son independientes del progra-

ma fuente.

5.4.1 El enunciado FØRMAT

Este tiene la forma: N FØRMAT ( , , , ...) en la cual N es el número del enunciado FØRMAT y corresponde al N de los enunciados READ y WRITE. Los espacios entre las comas están disponibles para las especificaciones del tipo que se describen más adelante, siendo el número de espacios uno o más, de acuerdo a las necesidades del programador.

5.4.1.1 La especificación I:Iw

Aquí I indica un valor entero y W es un entero que indica el número de columnas o ancho de campo, que ocupa ese valor en la tarjeta de entrada o en el papel de impresión. El número w deberá incluír un lugar para el signo de ese valor, siendo + opcional.

| Ejem. | Valor de los datos de entrada o salida: | 1130 | +1620 | -370 | 0 | -14 |
|---|---|---|---|---|---|---|
| | Especificación: | I4 | I5 | I4 | I1 | I3 |

5.4.1.2 La especificación F:Fw.d

Aquí F indica un valor real, w indica el número de columnas que ocupará el valor en la tarjeta de entrada o en el papel de impresión; d indica el número de cifras que se encontrarán des-

pués del punto decimal. w deberá incluir un lugar para el signo y otro para el punto decimal.

Ejem. Valor de los datos de entrada ó salida: 32.787 -.007 1130. +3.70

Especificación: F6.3 F5.3 F5.0 F5.2

5.4.1.3 La especificación E:Ew.d

Aquí E indica un valor real en forma exponencial y w indica la anchura de campo para ese valor y debe de incluir el signo, si lo hay, el punto decimal, el lugar para la letra E, un lugar para el signo del exponente, si es negativo, y dos lugares para el exponente; d indica el número de dígitos a la derecha del punto decimal.

Ejem. Valor de los datos de entrada o salida: .1403E04 -.7E-02 .1442E+04

Especificación: E8.4 E7.1 E9.4

Es conveniente que cuando deseemos sacar información de la computadora, tomemos en cuenta para el ancho del campo lo siguiente:

1.- El signo, aún cuando el + generalmente no se imprime.

2.- El punto decimal para las especificaciones F y E.

3.- Por lo menos un dígito a la izquierda

del punto decimal, puesto que muchas máquinas imprimirán allí un cero si otro dígito no ocurre.

4.- Suficientes lugares para todos los dígitos significativos deseados, debido a que para los dígitos que no se les deja espacio se truncan o redondean.

5.- Cuatro lugares para el exponente de la especificación E.

6.- El primer lugar se deja en blanco para el control de carro.

5.4.2 El enunciado END

Este se lee simplemente END e informa al compilador que el programa fuente ha terminado y debe ser el último enunciado de cualquier programa FØRTRAN.

6.- Arreglos

Frecuentemente tratamos con un grupo de variables que forman ó pertenecen a una clase o colección. Cuando las variables forman un conjunto ordenado, pueden relacionarse unas con otras por la notación de subíndices; entonces designamos esa colección como arreglo y las variables que pertenecen a ésta serie son los elementos del arreglo. A veces se emplea como sinónimo de

arreglo el nombre de matríz y, en consecuencia, hablamos de elementos de la matríz.

6.1 Variables con subíndices

Un conjunto de números que pueda arreglarse en un renglón ó columna se considera como un arreglo lineal ó unidimensional, y ésta serie puede llamarse vector. Identificamos los elementos de un vector renglón ó columna por un sólo subíndice.

Ejem. La columna de números del vector llamado A, consiste de los elementos A, hasta An inclusive y se representa como sigue:

| Notación asoctumbrada | Notación FØRTRAN |
|---|---|
| $A_1$ | A (1) |
| $A_2$ | A (2) |
| $A_3$ | A (3) |
| ⋮ | ⋮ |
| $A_I$ | A (I) |
| ⋮ | ⋮ |
| An | A (N) |

Cada una de estas A(I), en donde I varía de 1 a N, son el nombre de una variable, el conjunto de todas ellas es lo que llamamos arreglo.

Si se usan dos subíndices para identificar los elementos de un arreglo se considera éste como un arreglo bidimensional. Los cuadros de un tablero de ajedrez, pueden considerarse como un arreglo bidiminsional. Y si llamamos a cualquiera de los cua-

dros con la variable CTAJ tendremos 64 variables; pero como el tablero tiene 8 renglones y 8 columnas, podemos referirnos al cuadro que se encuentra en el renglón 3 y la columna 5 con la variable CTAJ-(3,5).

Dependiendo del tipo de computadora será el número de subíndices que podremos asignarle a un arreglo; en IBM - 1130 sólo se admiten arreglos con un máximo de tres subíndices.

Las variables que se utilicen para designar arreglos deberán observar las reglas que se dieron anteriormente al hablar de variables enteras y reales considerando que para los cinco caracteres alfanuméricos son independientes de los índices que se encuentran entre paréntesis.

6.1.1 Reglas para los subíndices.

Regla 1 Un subíndice debe ser un entero, puede ser constante, variable ó una de las expresiones aritméticas siguientes:

A * V + b  A * V - b

en donde v es una variable entera y a y b son constantes enteras sin signo.

Ejem. Algunos subíndices pueden ser:

1  1972  10*KONT  2*I  J

1976*N-8  2*I-4  2*I+3

No se pueden usar como subíndice:

1+I  -I  2-10*CONT  -1932  -KILO

Regla 2 Un subíndice sólo debe tomar valores positivos.

Regla 3 Un subíndice en sí no debe ser una variable con subíndices. Así X(I(2)) no es permitido.

Regla 4 Un símbolo que representa un arreglo, una variable con subíndice, no debe usarse sin subíndices para representar otra variable diferente en el mismo programa. Esto es A(I) y A no deben referirse a variables diferentes. Como siempre hay una excepción que por anora no tocaremos.

Ejem. Los símbolos para variables reales con subíndices podrían incluir:

X(I) SUM(K+2) A(I, 2*J+1) B(INT) C(I,J)

Para variables enteras con subíndices podemos tener:

INT(M,N) I(J) ICTA(J,2*I)

## 6.2 El enunciado DIMENSIØN

Siempre que en un programa utilicemos variables con subíndices deberemos poner como primer enunciado el DIMENSIØN, el cual indica al compilador qué tanto espacio de memoria se debe reservar para las variables con subíndices. Su forma es:

DIMENSIØN u, v, w, ...

Donde u, v, w, ... son nombres de variables, cada una de las cuales va seguida por el máximo número de elementos en el

arreglo correspondiente. Deberán observarse las siguientes reglas:

Regla 1 Cada variable con subíndices se debe mencionar en un enunciado DIMENSIØN antes de su primer uso en el programa.

Regla 2 Los símbolos representados anteriormente por u, v, w, .... deben tener la forma:

nombre de variable (máximo número de elementos)

el número entre paréntesis debe ser una constante entera sin signo.

Ejem. DIMENSION A(20), B(4,8), CARR(5,3,4)

Esto indica que el compilador reservará 20 localidades para el arreglo A, sus veinte variables serán A(1), A(2), ..., A(20) al mismo tiempo se reservarán 32 (4x8) localidades para las variables B(1,1), B(1,2), B(1,3), ...., B(1,8), B(2,1), B(2,2), ..., B(2,8), B(3,1), B(3,2), ..., B(3,8), B(4,1), B(4,2) ..., B(4,8) y por último se reservarán 60 (5x3x4) localidades para las variables del arreglo CAR, con tres subíndices cada una.

Regla 3 El arreglo que se use en particular, dentro del programa podrá tener menos elementos que los especificados en la magnitud del enunciado DIMENSION, pero no más.

Regla 4 La variable tal como aparece en el enunciado DIMENSIØN debe tener exactamente el mismo número de subíndices que en cualquier otra parte del programa.

## 7 - SUBPROGRAMAS.

Los subprogramas, también llamados subrutinas, son programas que pueden ser puestos en uso por otros programas cuando sea necesario.

Las funciones de biblioteca ó funciones del sistema constituyen una variedad dè subprogramas.

### 7.1- FUNCIONES

Cuando el valor de una variable depende de una ó más variables ó constantes y además de una serie de cálculos, y dicha variable ha de calcularse repetidamente y en diferentes puntos de un programa, es posible definirla como -- una Función. En otras palabras, Además de las funciones con que cuenta la biblioteca del sistema, el usuario puede escribir sus propias funciones para uso específico de su programa.

Tomemos un ejemplo para visualizar lo anterior:

Supongamos que para un programa en especial, en el cual trabajamos con grados en lugar de radianes, deseamos calcular continuamente SENØ (X), sin el uso - de funciones sería necesario transformar el argumento deseado de grados a radianes y después llamar a la función del sistema SIN (X). A continuación presentamos una función que calculará SENO (X), (X en grados) :

```
FUNCTION SENØ  ( X )
 X = X * 3.14 15 92/180.
 SENØ = SIN (X)
 RETURN
 END
```

que es llamada desde el programa como:

```
GRAD= SENØ (GRADØS)
```

En base a éste ejemplo podemos generalizar el uso de la proposición FUNCTION .

a) Debe ser codificada en forma independiente del programa que la usará, es decir, no debe aparecer "dentro" del programa.
b) Debe empezar con la palabra FUNCTIØN

FUNCTION nombre (parámetro )

c) A continuación se escribe el nombre con que será llamada.

d) Después, entre paréntesis y separados por comas, aparecen los argu - mentos.

7.1.1 EJEMPLOS.-

```
      FUNCTION RAIZ 1 (A,B,C )
      RAIZ1= (-B + SQRT (B **2 - 4. * A * C )) / ( 2,* A )
      RETURN
    END
    FUNCTIØN  RAIZ2 ( A, B, C )
      RAIZ2 = ( -B - SQRT (B**2.4.* A * C )) / (2.* A )
      RETURN
    END
```

---

```
C       EC. SEGUNDØ GRADØ
        READ ( 2,100) A, B, C
100     FORMAT ( 3F10.5)
        X1 = RAIZ1 (A,B,C)
        X2 = RAIZ2 (A,B,C)
        WRITE (3,200) A,B,C, X1,X2
200     FORMAT ( 5 (;F10,5')
        CALL EXIT
        END
```

Este ejemplo es solamente para mostrar el uso de la proposición FUNCTIØN y no contempla algunas situaciones como raíces complejas,

7.2 SUBRUTINAS

Como es fácil notar, la proposición FUNCTIØN nos "regresa" un sólo va - lor y lo hace a través de su nombre. En muchos casos es conveniente ó - necesario que se nos regrese más de un valor, para éstos casos usamos la proposición o enunciado:

SUBRØUTINE.

Una subrutina es un subprograma que puede "recibir" cualquier número - de parámetros ( desde cero hasta un número determinado por el tipo de compilador) y puede "regresar" diferentes valores calculados.

Veamos algunos ejemplos:

Supongamos que al imprimir resultados de un cierto programa tenemos que escribir algún título usando los primeros renglones de la hoja. En tal caso podemos hacer uso de una subrutina como sigue:

```
      SUBRØUTINE  ENCA
      WRITE (3,200)
200   FØRMAT--(/,1X,- 'REPORTE- SEMANAL' , / )
      RETURN
    END
```

Como vemos no hemos pasado ningún parámetro ó valor a la subrutina. Para que se ejecute ésta se debe hacer uso de la proposición CALL, de la siguiente forma:

```
CALL ENCA
```

dentro del programa y en el lugar donde deseemos que ocurra la impresión.

Discutamos ahora un ejemplo muy simple para ejemplificar el uso de parámetros. Hagamos una subrutina que "reciba" como entrada dos números, los sume y el resultado lo "regrese" en otra variable. Sean A y B los numeros a sumar, y C la variable en donde se pondrá el resultado.

```
SUBROUTINE  SUMA  (A,B,C )
    C = A + B
    RETURN
  END
```

Es importante detenerse a ver el significado de los parámetros para las subrutinas:

La subrutina anterior SUMA puede ser llamada de diversas formas:

```
CALL SUMA (AA,BB,CC)
CALL SUMA (4, 7, X )
etc.
```

Como vemos, las variables A,B y C que aparecen en la subrutina son variables

mudos o dormidos y solo tienen sentido dentro de la subrutina. Veamos lo - anterior:

Supóngase el siguiente programa:

```
      X1= 3.
      X2= 4.
      CALL SUMA ( X1,X2,X3)
      SUM= X3
      WRITE (3,200) X1,X2,X3, SUM
200   FØRMAT(4 F10.5)
      CALL EXIT
      END
```

Se propone como ejercicio al lector que haga las veces de la máquina y escriba lo que ésta imprimiría.

La máquina imprimirá :

3.0 4.0 7.0 7.0

Una de las facilidades más utiles en subrutinas es la de pasar arreglos como parámetros, ej:

```
      SUBRØUTINE MAXIM  (A, MAX )
      DIMENSION        A ( 10)
      - - - - -
      - - - - -
      - - - - -
      - - - - -
      - - - - -
      RETURN
  END
```

Supóngase que ésta subrutina encuentra el elemento del arreglo A (10) con mayor valor y lo regresa a través de la variable MAX. Es importante notar que si pasamos como parámetro uno ó más arreglos hay que dimensionarlos otra vez dentro de la subrutina, lo cual se puede hacer de al menos dos formas: 1) poniendo la dimensión que aparece en el programa que lo llama;

2) Poniéndole dimensión 1 (UNO)

.Ejemplo:

```
DIMENSION  A (10) ,  B  (20)
- - - - - -
- - - - - -
- - - - - -
- - - - - -

CALL ØRDEN   (A)
CALL MAXIM   (B)
CALL MAXIM   (A)
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -

CALL EXIT
END
```

---

Caso 1:

```
   SUBRØUTINE ØRDEN  (X)
   DIMENSION       X (10)
   - - - -
   - - - -
   - - - -
   - - - -

   RETURN
END
```

---

Caso 2:

```
   SUBRØUTINE MAXIM  (Y)
   DIMENSION    Y    (1)
   - - - -
   - - - -
   - - - -

   RETURN
END
```

### 7.2.1 COMMON.-

Como es posible visualizar en los párrafos anteriores, las variables usadas en las subrutinas, o mejor dicho, dentro de las subrutinas, son totalmente independientes a las variables usadas en el programa principal. Muchas veces es conveniente que tanto las subrutinas como el programa que las llama tengan variables en COMUN. Para lograr ésto existe la declaración

COMMON

La forma general de ésta proposición es:

COMMON lista de variables

donde "lista de variables" es un conjunto de variables y/o arreglos separados por comas a las cuales queremos adjudicarles la propiedad anterior, es decir, sean comunes a varios subprogramas.

Ej.

```
COMMON    A,B,  X (10), AB (30)
```

Esta declaración debe aparecer al principio de cualquier programa o subrutina en que se desee usar. Veamos un ejemplo:

```
C     SUMA DE DOS NUMEROS
      COMMON  A, B, C
      A= 3
      B= 7
      CALL SUMA
      Z = C
      WRITE (3,200) A, B, C, Z
200   FØRMAT( 4 F10.5 )
      CALL EXIT
      END
```

```
SUBROUTINE SUMA
COMMON  A, B, C
C = A + B
RETURN
END
```

Este programa debe imprimir :

3.0  7.0  10.0  10.0

Una propiedad importante del COMMON es que si un arreglo es especificado en COMMON que dá automáticamente dimensionado, es decir, no hay que especifi - car dicho arreglo a través de la declaración DIMENSION .

En las siguientes páginas se muestran veintiún programas, que incluyen sus - diagramas de flujo, codificaciones, datos y resultados; el objeto es que el - lector pueda complementar la parte teórica con la práctica, amén de que deberá hacer los propios y procesarlos en una computadora a su alcance.

REFERENCIAS BIBLIOGRAFICAS

1.- J.K. Hughes : Programación del Sistema IBM-1130
Limusa-Wiley-1969

2.- D.D. McCracken :Fortran IV
Limusa-Wiley-1964

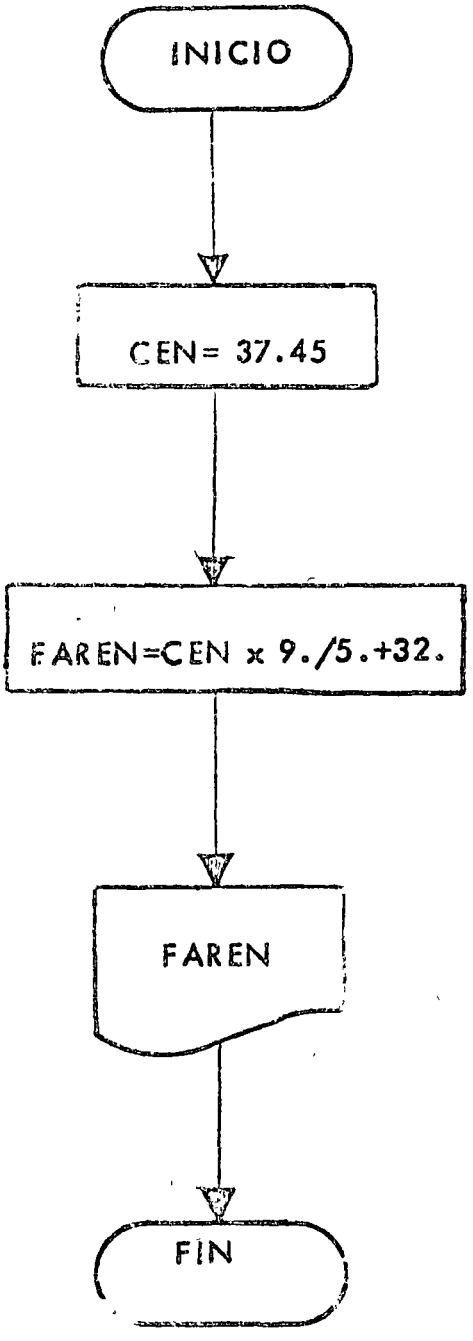
3.- E.I. Organick :Fortran IV
Fondo Educativo Interamericano, S. A.
1966

4.- Francis Scheid :Introducción a la Ciencia de las Computadoras.
Serie de Compendios Schaum.
McGraw-Hill-1970

5.- W.Schick y Ch. J. Merz, Jr. :Fortran para Ingeniería.
McGraw-Hill - 1972

6.- R.E. Smith y D.E. Johnson :Fortran, Texto Programado.
Limusa Wiley - 1971

"CONVERSION DE GRADOS CENTIGRADOS A GRADOS FARENHEIT"

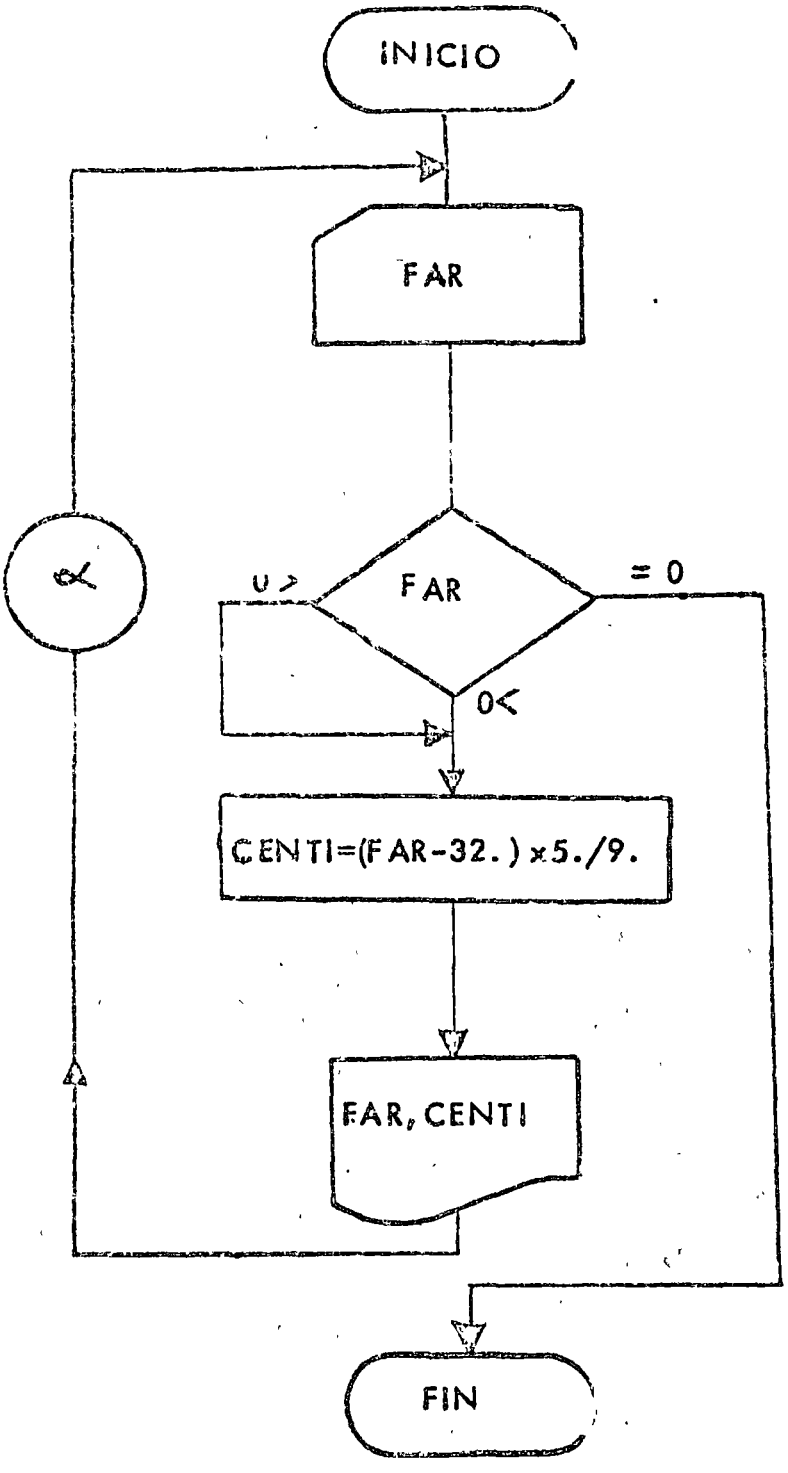


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------U N O----------
C       CONVERSION DE GRADOS CENTIGRADOS A
C       GRADOS FARENHEIT
  100 FORMAT(F10.4)
      IMP=3
      CEN=37.45
      FAREN=CEN*9./5.+32.
      WRITE(IMP,100)FAREN
      CALL EXIT
      END
// XEQ
/*
```

## R E S U L T A D O S

99.8700

"CONVERSION DE GRADOS FARENHEIT A GRADOS CENTIGRADOS"

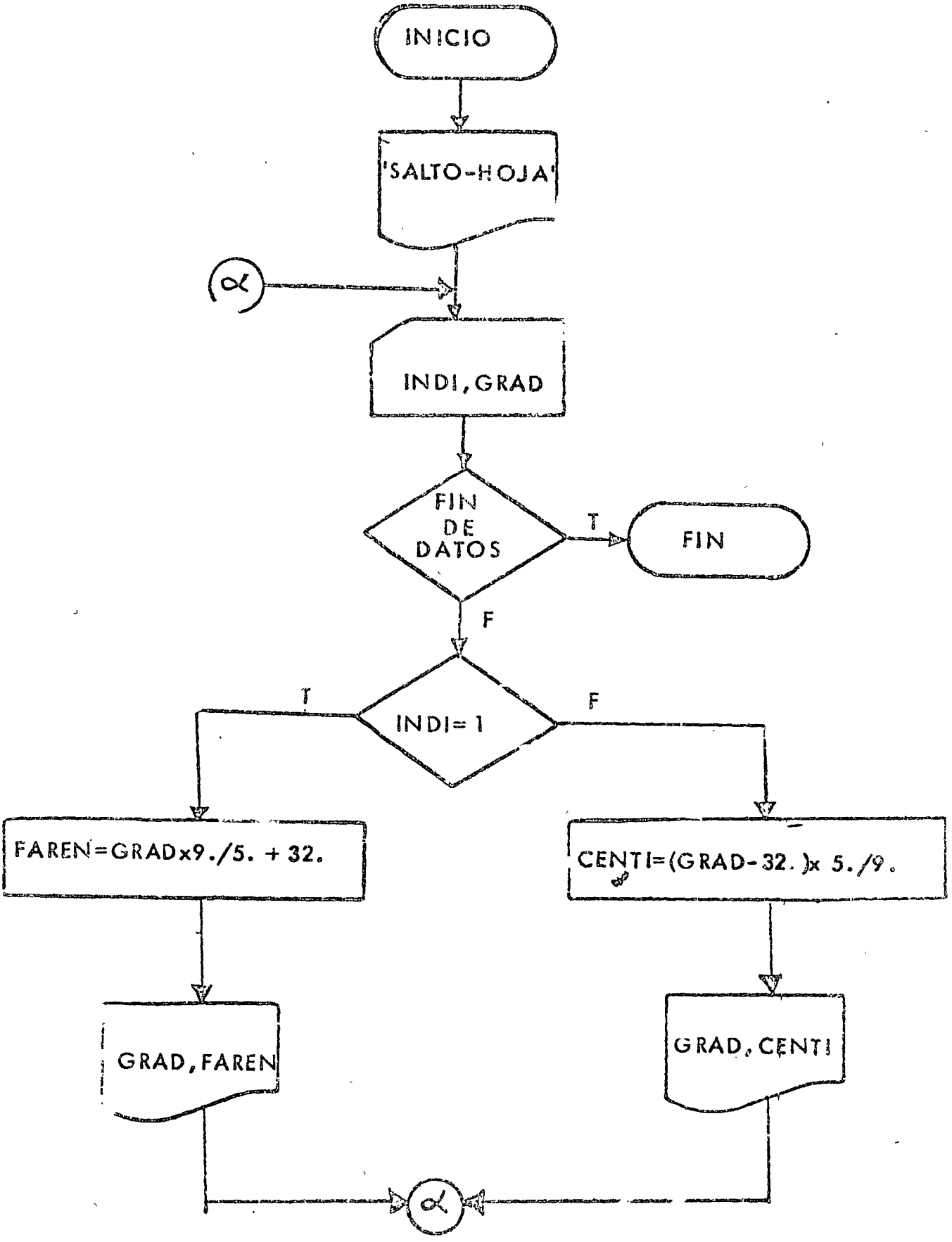


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------D O S----------
C      CONVERSION DE GRADOS FARENHEIT A
C                    GRADOS CENTIGRADOS
  100 FORMAT(F10.4)
  101 FORMAT(F10.4,22H GRADOS FARENHEIT SON ,F10.4,20H GRADOS CENTIGRADO
     1S.)
      LEE=2
      IMP=3
  200 READ(LEE,100)FAR
C      FAR IGUAL CERO INDICA TERMINO DE DATOS.
      IF(FAR)210,220,210
  210   CENTI=(FAR-32.)*5./9.
        WRITE(IMP,101)FAR,CENTI
        GO TO 200
  220 CALL EXIT
      END
// XEQ
   1260000
126.
  -14.
    18.26
0.0
/*
```

## RESULTADOS

```
 126.0000 GRADOS FARENHEIT SON    52.2222 GRADOS CENTIGRADOS.
 126.0000 GRADOS FARENHEIT SON    52.2222 GRADOS CENTIGRADOS.
 -14.0000 GRADOS FARENHEIT SON   -25.5556 GRADOS CENTIGRADOS.
  18.2600 GRADOS FARENHEIT SON    -7.6333 GRADOS CENTIGRADOS
```

"CONVERSION ENTRE GRADOS FARENHEIT Y GRADOS CENTIGRADOS"

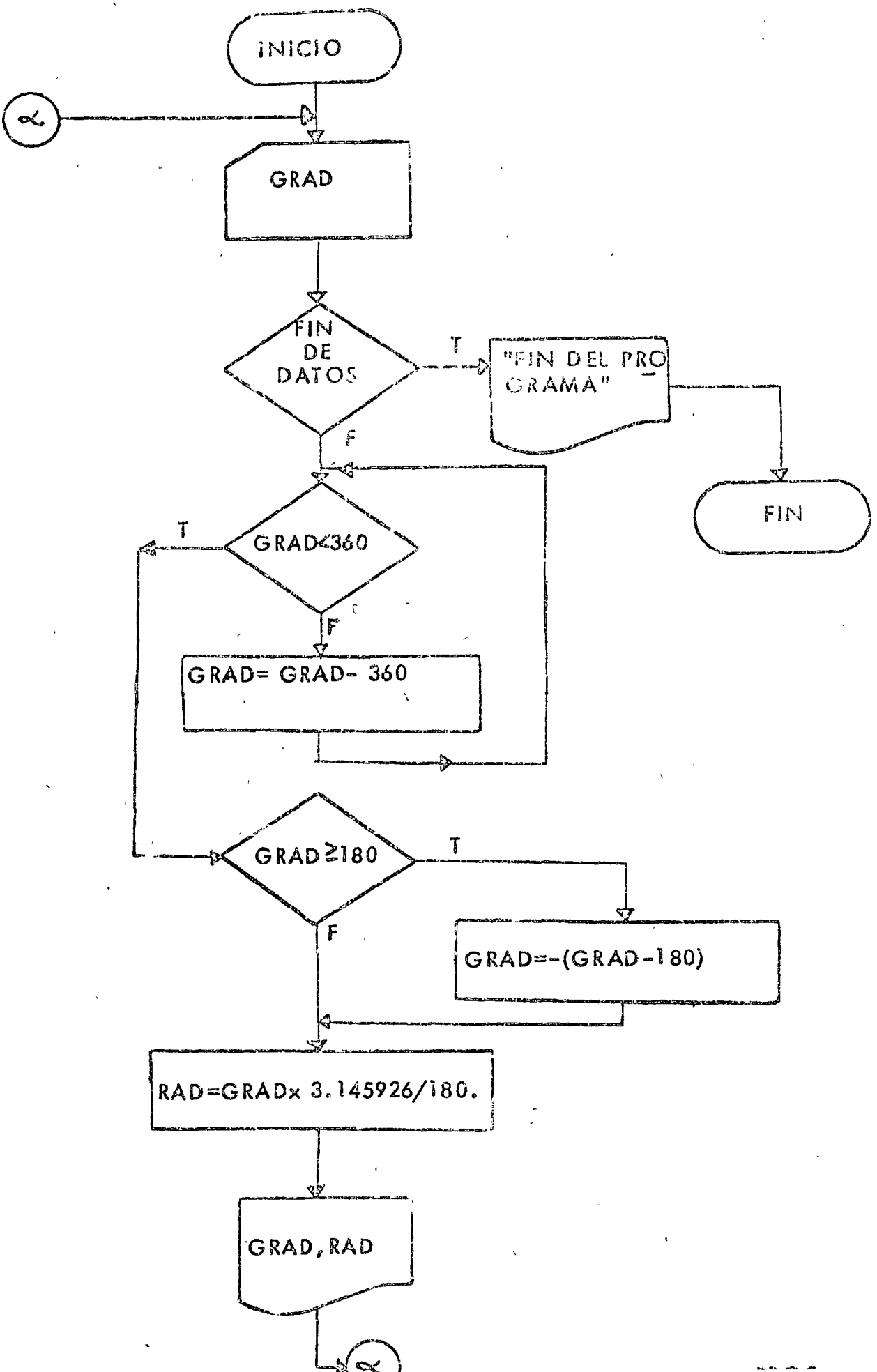


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------T R E S---------
C      CONVERSION ENTRE GRADOS FARENHEIT
C                 Y GRADOS CENTIGRADOS
  100 FORMAT(1H1)
  101 FORMAT(I1,F10.3)
  102 FORMAT(F10.2,15H FARENHEIT SON ,F11.3,13H CENTIGRADOS.)
  103 FORMAT(F10.2,17H CENTIGRADOS SON ,F9.3,11H FARENHEIT.)
      LEE=2
      IMP=3
      WRITE(IMP,100)
  200 READ(LEE,101,END=220)INDI,GRAD
      IF(INDI.EQ.1)GO TO 210
C     INDI DIFERENTE DE 1 DATO EN GRADOS FARENHEIT.
C     SE CONVIERTE A CENTIGRADOS.
        CENTI=(GRAD-32.)*5./9.
        WRITE(IMP,102)GRAD,CENTI
        GO TO 200
  210 CONTINUE
C       EL DATO ES EN GRADO CENTIGRADO.
C       SE CONVIERTE A FARENHEIT.
        FAREN=GRAD*9./5.+32.
        WRITE(IMP,103)GRAD,FAREN
        GO TO 200
  220 CALL EXIT
      END
// XEQ
1      12000
0     11.48
1   0.
2     32.00
1   -16.
1       18.
/*
```

## RESULTADOS

```
   12.00 CENTIGRADOS SON      53.600 FARENHEIT.
   11.48 FARENHEIT SON       -11.400 CENTIGRADOS.
    0.00 CENTIGRADOS SON      32.000 FARENHEIT.
   32.00 FARENHEIT SON         0.000 CENTIGRADOS.
  -16.00 CENTIGRADOS SON       3.200 FARENHEIT.
   18.00 CENTIGRADOS SON      64.400 FARENHEIT.
```

"CONVERSION DE GRADOS A RADIANES"

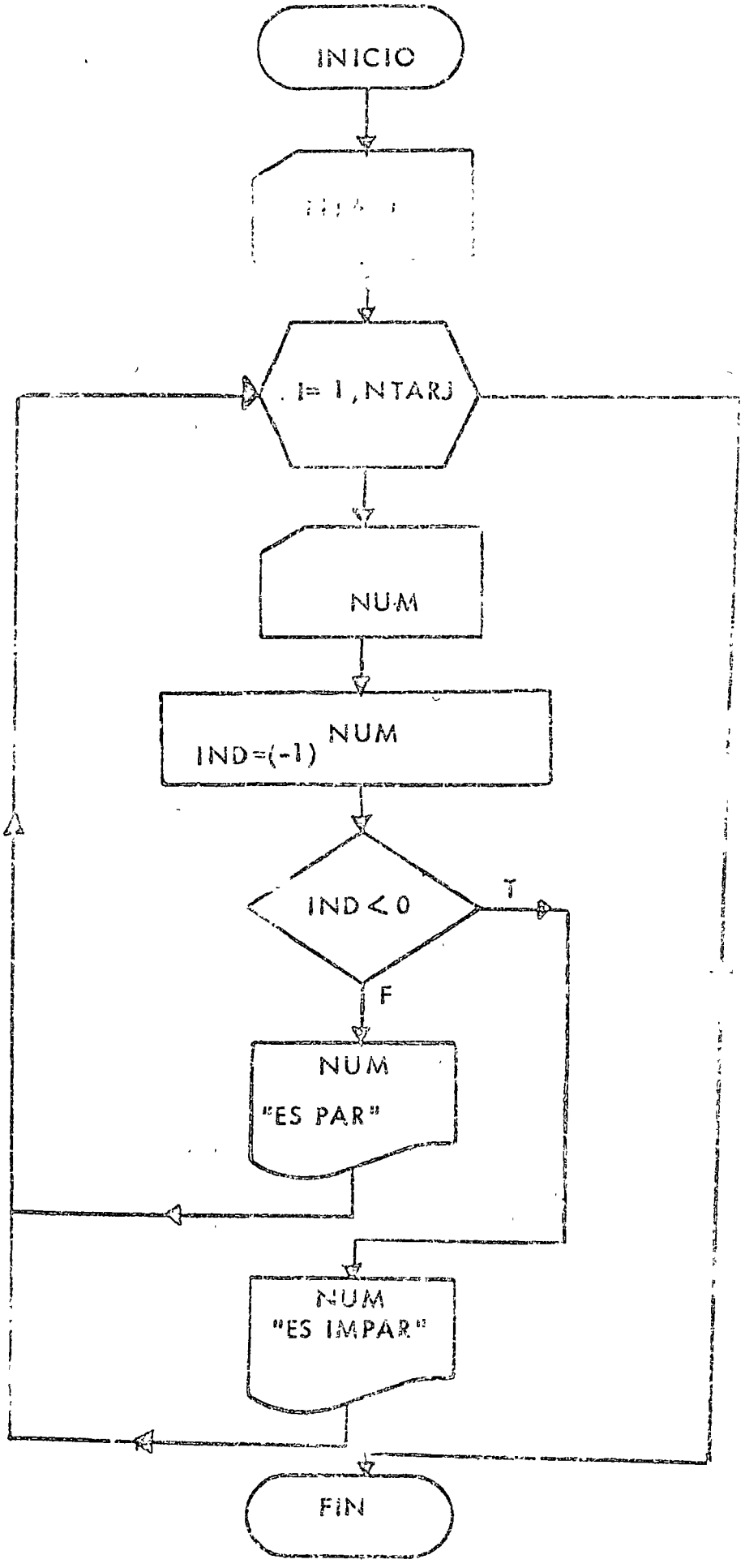


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------C U A T R O-------
C      CONVERSION DE GRADOS A RADIANES
  101 FORMAT(F8.3)
  102 FORMAT(F9.3,12H GRADOS SON ,F6.3,10H RADIANES.)
  103 FORMAT(///,10X,16HFIN DEL PROGRAMA)
      LEE=2
      IMP=3
  200 READ(LEE,101,END=230)GRAD
  210 IF(GRAD.LT.360)GO TO 220
C       EL DATO ES IGUAL O SOBREPASA LOS 360 GRADOS(SE AJUSTA).
        GRAD=GRAD-360.
        GO TO 210
  220 CONTINUE
C     SE TRABAJA ENTRE +180 Y -180 GRADOS.
      IF(GRAD.GE.180.)GRAD=-(GRAD-180.)
      RAD=GRAD*3.1415926/180.
      WRITE(IMP,102)GRAD,RAD
      GO TO 200
  230 WRITE(IMP,103)
      CALL EXIT
      END
// XEQ
   90000
-90.
  3600.
  -380.
  0.0
 185.27
 132.4
-79.9
/*
```

## RESULTADOS

```
      90.000 GRADOS SON  1.571 RADIANES.
     -90.000 GRADOS SON -1.571 RADIANES.
       0.000 GRADOS SON  0.000 RADIANES.
    -380.000 GRADOS SON -6.632 RADIANES.
       0.000 GRADOS SON  0.000 RADIANES.
      -5.270 GRADOS SON -0.092 RADIANES.
     132.400 GRADOS SON  2.311 RADIANES.
     -79.900 GRADOS SON -1.395 RADIANES.

            FIN DEL PROGRAMA
```

"DETERMINACION DE NUMEROS PARES E IMPARES"

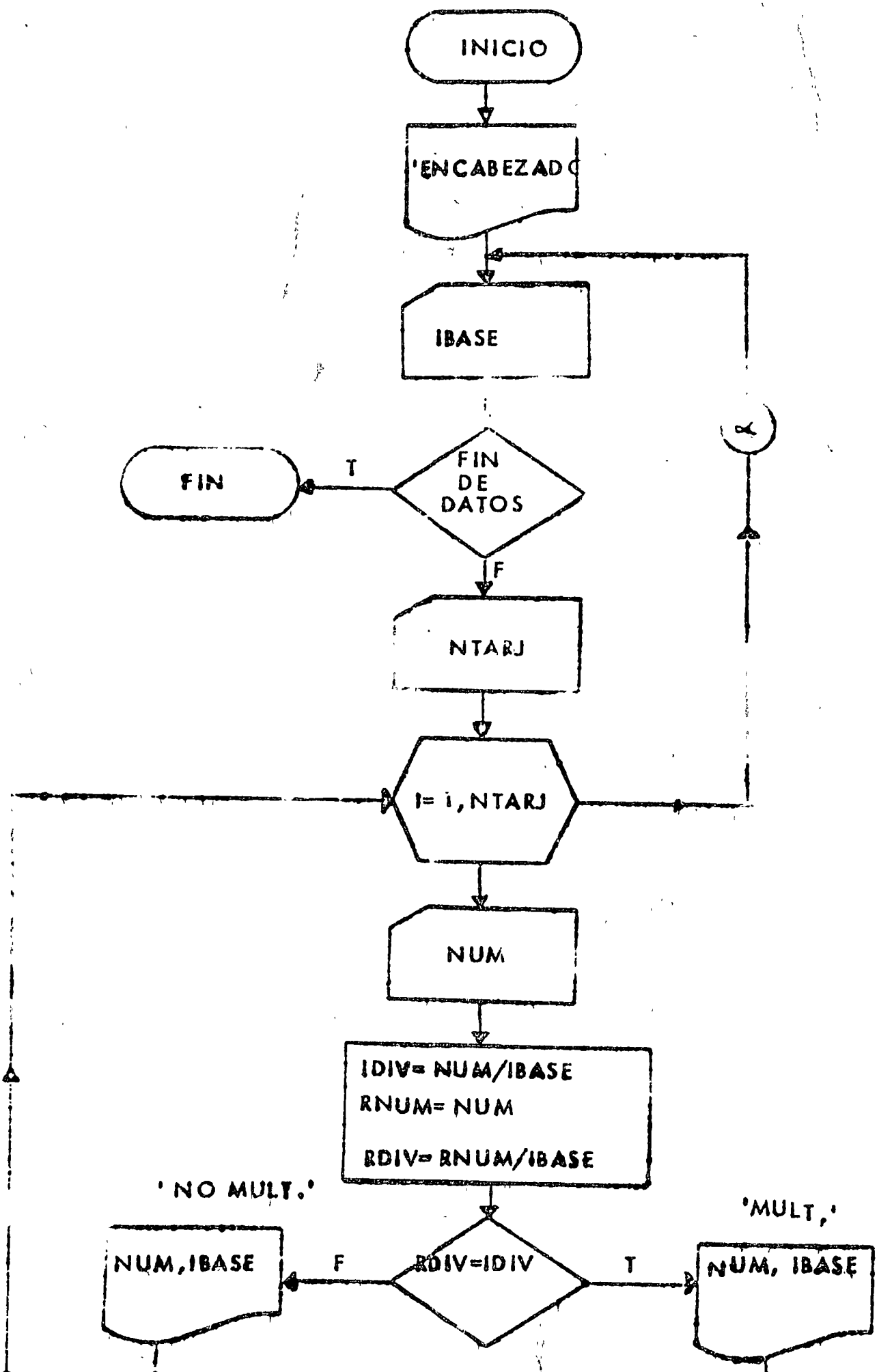


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------C I N C O-------
C      DETERMINACION DE NUMEROS PARES
C      E IMPARES
  100 FORMAT(I3)
  101 FORMAT(I4,8H ES PAR.)
  102 FORMAT(I4,10H ES IMPAR.)
      LEE=2
      IMP=3
      READ(LEE,100)NTARJ
C     NTARJ INDICA NO. DE TARJETAS CON DATOS.
      DO 202 I=1,NTARJ
        READ(LEE,100)NUM
        IND=(-1)**NUM
        IF(IND.LT.0)GO TO 200
C         EL NUMERO ES PAR.
          WRITE(IMP,101)NUM
          GO TO 201
  200   CONTINUE
C         EL NUMERO ES IMPAR.
          WRITE(IMP,102)NUM
  201   CONTINUE
  202 CONTINUE
      CALL EXIT
      END
// XEQ
005
 17
  1
 14
291
  8
/*
```

## RESULTADOS

```
 17 ES IMPAR.
  1 ES IMPAR.
 14 ES PAR.
291 ES IMPAR.
  8 ES PAR.
```

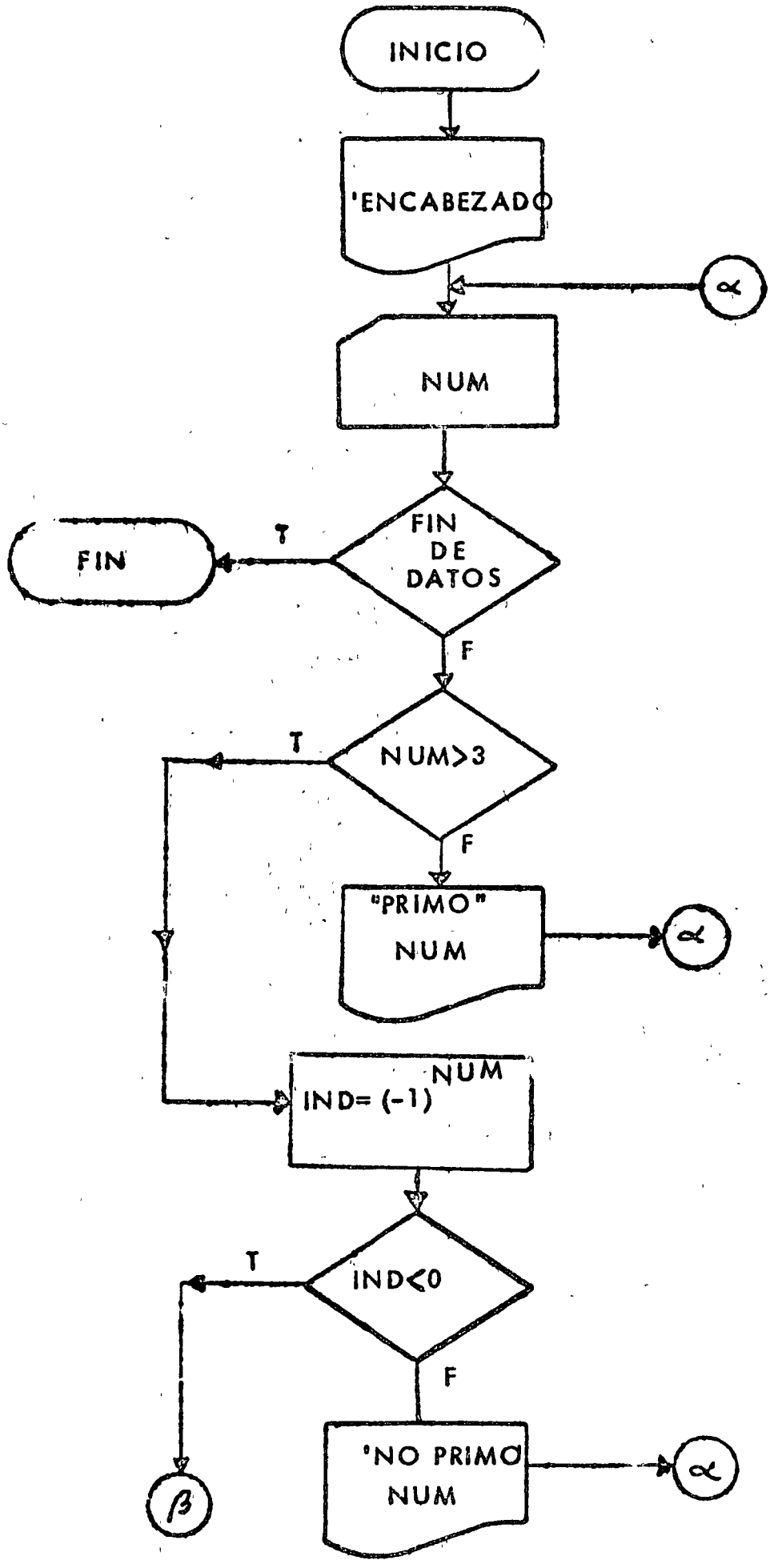
"DETERMINACION DE MULTIPLOS DE UN NUMERO"



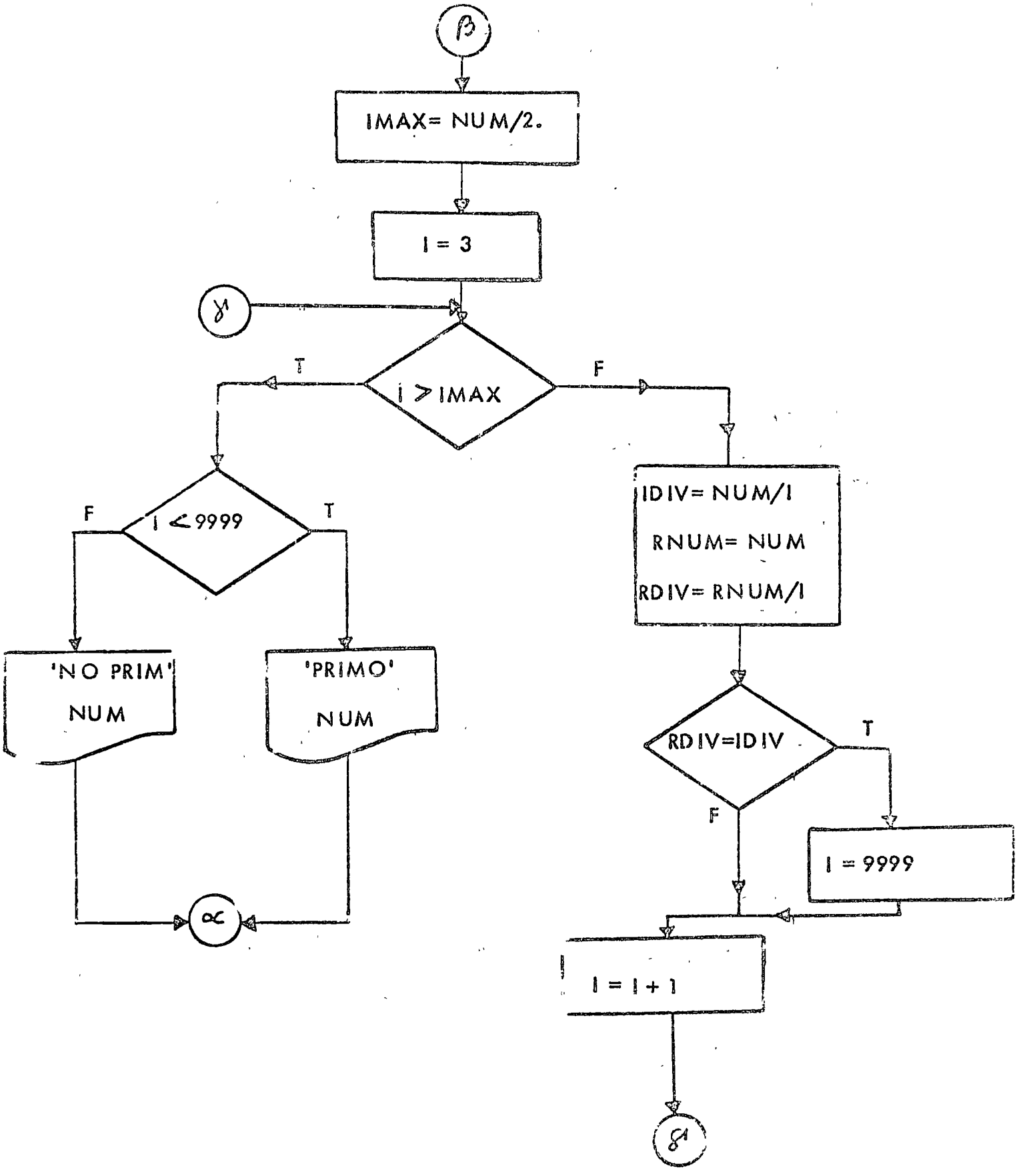
```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------S E I S---------
C      DETERMINACION DE MULTIPLOS
C      DE UN NUMERO
  100 FORMAT(I3)
  101 FORMAT(34H1NUMERO-MULTIPLO DE-NO MULTIPLO DE)
  102 FORMAT(2X,I3,7X,I3)
  103 FORMAT(2X,I3,21X,I3)
      LEE=2
      IMP=3
      WRITE(IMP,101)
  200 READ(LEE,100,END=240)IBASE
      READ(LEE,100)NTARJ
      DO 230 I=1,NTARJ
        READ(LEE,100)NUM
        IDIV=NUM/IBASE
        RNUM=NUM
        RDIV=RNUM/IBASE
        IF(RDIV.EQ.IDIV)GO TO 210
C         NUM NO ES MULTIPLO DE IBASE.
        WRITE(IMP,103)NUM,IBASE
        GO TO 220
  210   CONTINUE
C         NUM SI ES MULTIPLO DE IBASE.
        WRITE(IMP,102)NUM,IBASE
  220   CONTINUE
  230

  240 CALL EXIT
      END
// XEQ
002
005
 17
001
 14
291
 8
003
002
 9
 11
005
 01
 09
```

R E S U L T A D O S

| NUMERO-MULTIPLO DE-NO MULTIPLO DE | |
|---|---|
| 17 | 2 |
| 1 | 2 |
| 14 | 2 |
| 291 | 2 |
| 8 | 2 |
| 9 | 3 |
| 11 | 3 |
| 9 | 5 |


INICIO
'ENCABEZADO
α
NUM
FIN DE DATOS
T
FIN
F
NUM>3
T
F
"PRIMO"
NUM
α
IND= (-1) NUM
IND<0
T
F
'NO PRIMO
NUM
α
β

PROG - 7

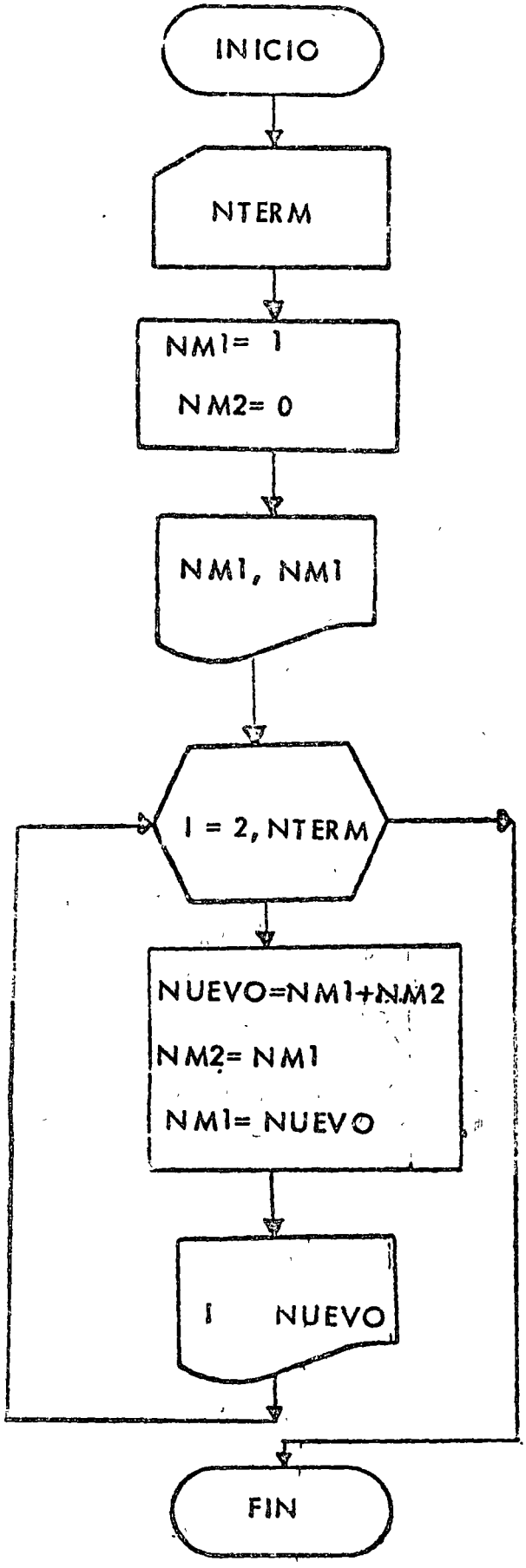

B
IMAX= NUM/2.
I = 3
γ
T
I > IMAX
F
F
I < 9999
T
'NO PRIM'
NUM
'PRIMO'
NUM
α
IDIV= NUM/I
RNUM= NUM
RDIV= RNUM/I
RDIV=IDIV
T
F
I = 9999
I = I + 1
δ

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------S 1 E T E-------
C      NUMEROS PRIMOS
  100 FORMAT(I3)
  101 FORMAT(19H1PRIMOS  NO PRIMOS)
  102 FORMAT(2X,I3)
  103 FORMAT(14X,I3)
      LEE=2
      IMP=3
      WRITE(IMP,101)
  200 READ(LEE,100,END=290)NUM
      IF(NUM.GT.3)GO TO 210
C       NUM ES MENOR O IGUAL A 3(TODO NUMERO NATURAL MENOR O IGUAL A 3 ES
        WRITE(IMP,102)NUM
        GO TO 280
  210 CONTINUE
C      NUM ES MAYOR QUE 3.
        IND=(-1)**NUM
        IF(IND.LT.0)GO TO 220
C         NUM ES PAR(TODO NUMERO PAR MAYOR QUE 3 NO ES PRIMO.).
          WRITE(IMP,103)NUM
          GO TO 270
  220   CONTINUE
C         NUM ES IMPAR(SE INICIA PROCESO DE PRIMO O NO-PRIMO).
          IMAX=NUM/2
          I=3
  230     IF(I.GT.IMAX)GO TO 240
C           SE OBTIENEN LAS DIVISIONES ENTERA Y REAL DE NUM/I
C           CON I DE 3 HASTA NUM/2.
            IDIV=NUM/I
            RNUM=NUM
            RDIV=RNUM/I
            IF(RDIV.EQ.IDIV)I=9999
C           I=9999 INDICA QUE NUM ES PRIMO.
            I=I+1
            GO TO 230
  240     CONTINUE
          IF(I.LT.9999)GO TO 25
C           NUM NO ES PRIMO.
            WRITE(IMP,103)NUM
            GO TO 260
  250     CONTINUE
C           NUM ES PRIMO.
            WRITE(IMP,102)NUM
  260     CONTINUE
  270   CONTINUE
  280 CONTINUE
      GO TO 200
  290 CALL EXIT
      END
// XEQ
  1
  2
  3
  4
  5
  6
  7
```

```
 9
10
11
12
13
15
17
19
21
/*
```

RESULTADOS

| PRIMOS | NO PRIMOS |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| | 4 |
| 5 | |
| | 6 |
| 7 | |
| | 8 |
| | 9 |
| | 10 |
| 11 | |
| | 12 |
| 13 | |
| | 15 |
| 17 | |
| 19 | |
| | 21 |


INICIO
NTERM
NM1= 1
NM2= 0
NM1, NM1
I = 2, NTERM
NUEVO=NM1+NM2
NM2= NM1
NM1= NUEVO
I NUEVO
FIN

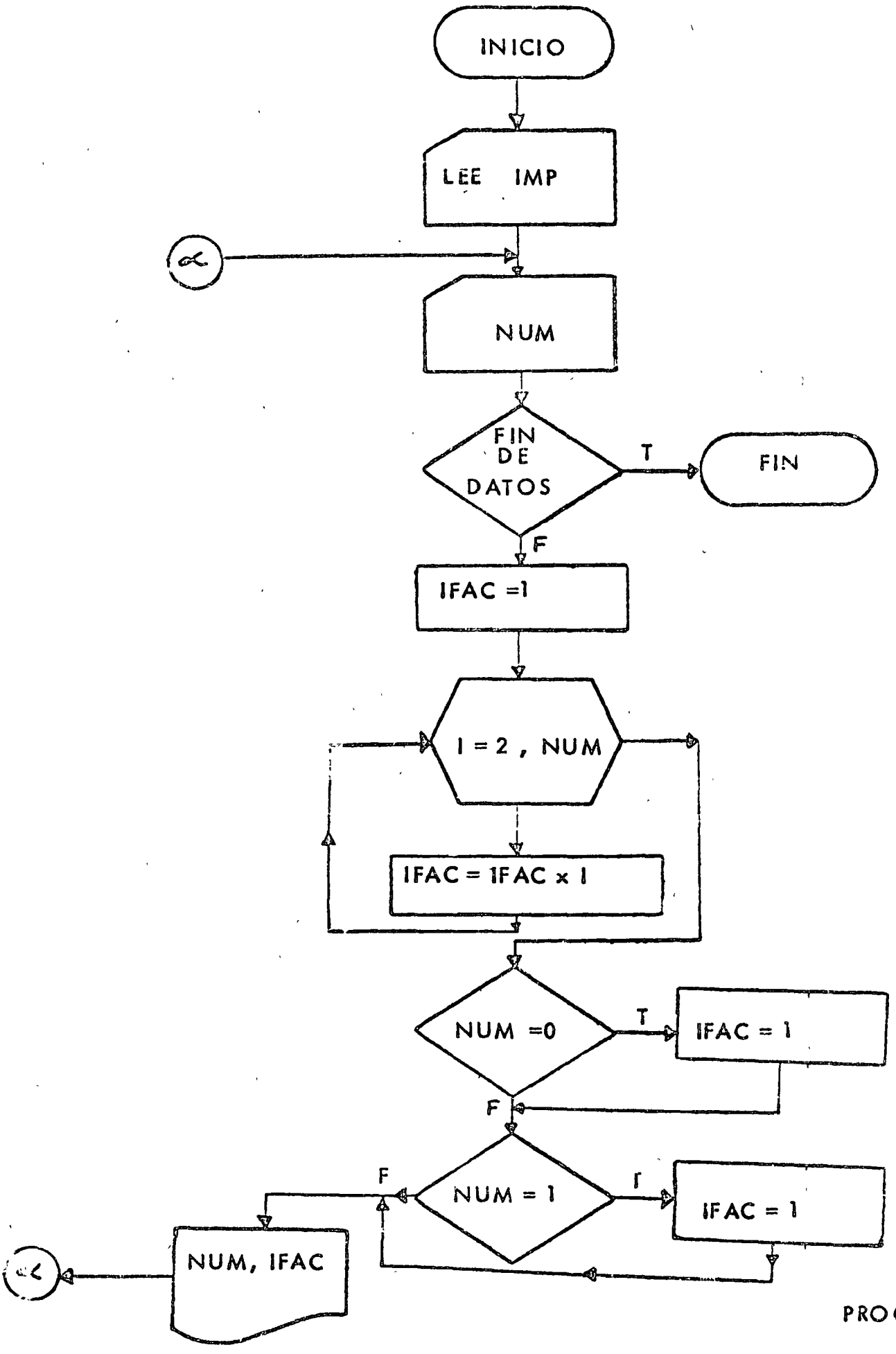
PROG - 5

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------0 C H 0--------
C      SERIE DE FIBONACCI
  100 FORMAT(I3)
  101 FORMAT(I4,3X,I5)
      LEE=2
      IMP=3
      READ(LEE,100)NTERM
C     NTERM REPRESENTA EL NUMERO DESEADO DE TERMINOS
      NM1=1
      NM2=0
      WRITE(IMP,101)NM1,NM1
      DO 200 I=2,NTERM
        NUEVO=NM1+NM2
        NM2=NM1
        NM1=NUEVO
C       SE IMPRIME LA POSICION Y EL VALOR DEL TERMINO
        WRITE(IMP,101)I,NUEVO
  200 CONTINUE
      CALL EXIT
      END
// XEQ
015
/*
```

## R E S U L T A D O S

| | |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 5 |
| 6 | 8 |
| 7 | 13 |
| 8 | 21 |
| 9 | 34 |
| 10 | 55 |
| 11 | 89 |
| 12 | 144 |
| 13 | 233 |
| 14 | 377 |
| 15 | 610 |

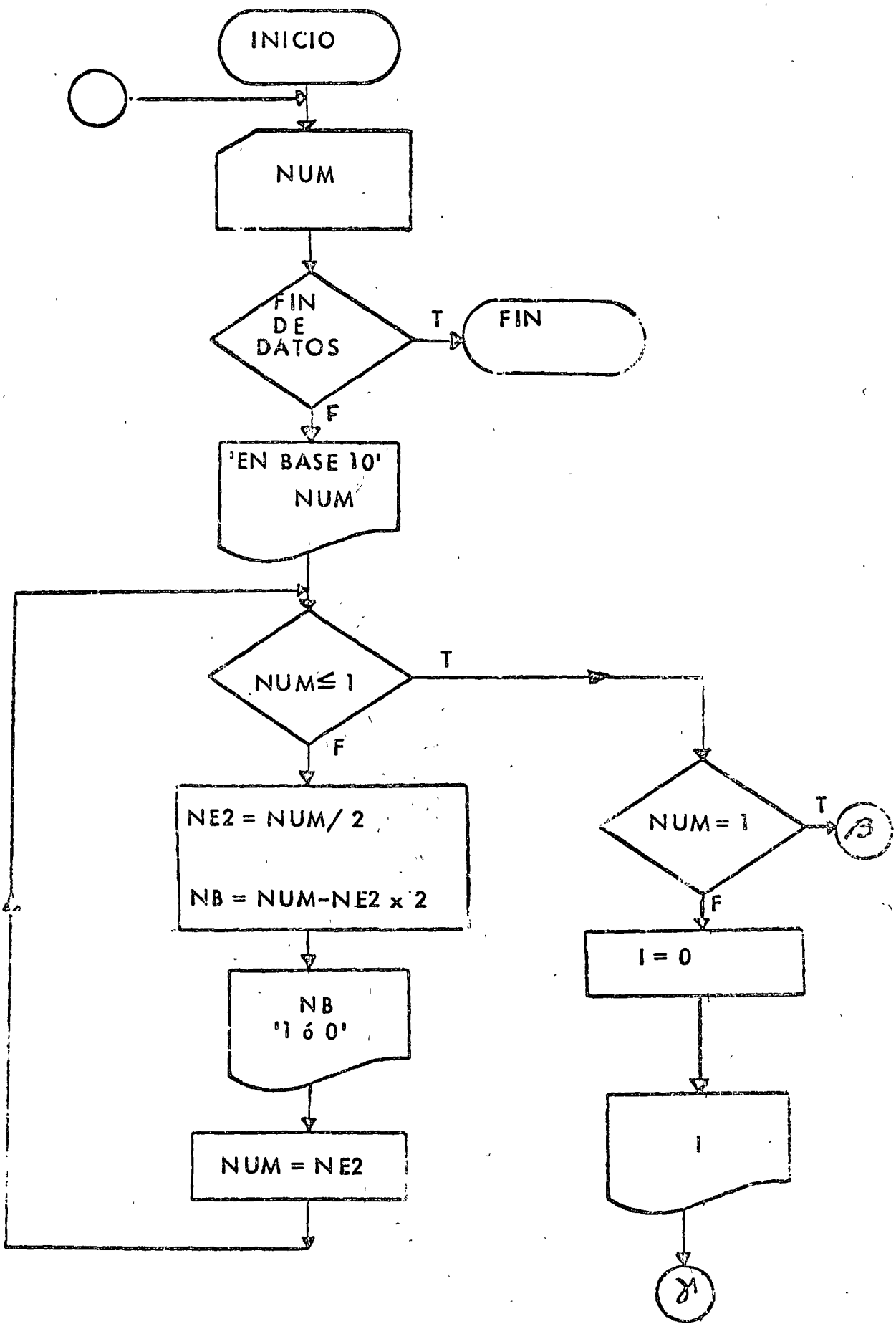
" FACTORIAL "


INICIO
LEE IMP
∝
NUM
FIN DE DATOS
T
FIN
F
IFAC =1
I = 2 , NUM
IFAC = IFAC x I
NUM =0
T
IFAC = 1
F
NUM = 1
F
T
IFAC = 1
NUM, IFAC
∝

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------N U E V E-------
C       FACTORIAL
  100 FORMAT(2I1)
  101 FORMAT(I2)
  102 FORMAT(I3,3X,I5.
C       SE LEEN LAS UNIDADES LOGICAS DE LECTURA E IMPRESIONN
      READ(2,100)LEE,IMP
  200 READ(LEE,101,END=220)NUM
      IFAC=1
      DO 210 I=2,NUM
        IFAC=IFAC*I
  210 CONTINUE
      IF(NUM.EQ.0)IFAC=1
      IF(NUM.EQ.1)IFAC=1
C       SE IMPRIME EL NUMERO Y SU FACTORIAL
      WRITE(IMP,102)NUM,IFAC
      GO TO 200
  220 CALL EXIT
      END
// XEQ
23
01
02
03
04
05
 0
 *
```

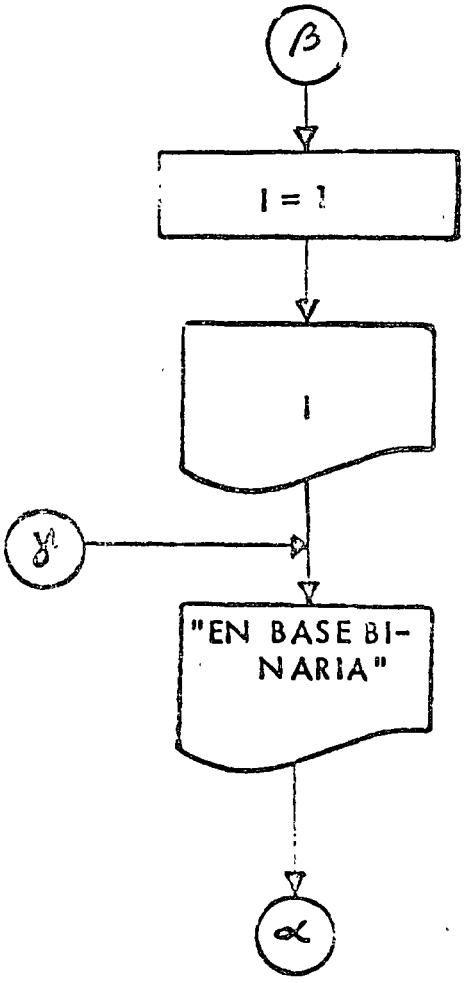
RESULTADOS

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |
| 5 | 120 |
| 0 | 1 |


INICIO
NUM
FIN DE DATOS
T
FIN
F
'EN BASE 10' NUM
NUM≤ 1
T
F
NE2 = NUM/ 2
NB = NUM-NE2 x 2
NB '1 ó 0'
NUM = NE2
NUM = 1
T
B
F
I = 0
I
A

PROG - 10

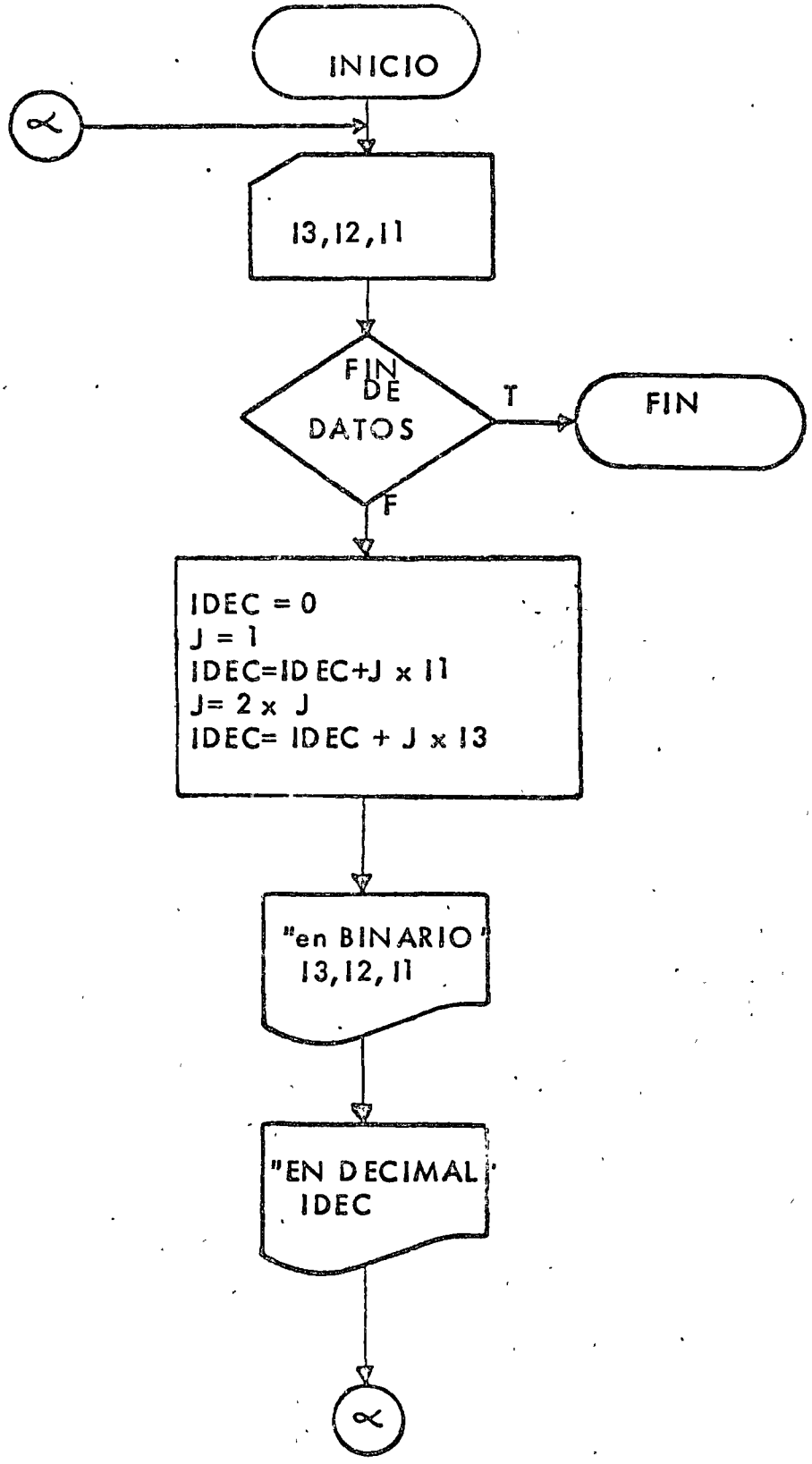
```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------D I E Z---------
C      CAMBIO DE BASE 8 DECIMAL A BINARIA
  100 FORMAT(I5)
  101 FORMAT(1X,I5,19H EN BASE DECIMAL ES)
  102 FORMAT(1X,I1)
  103 FORMAT(17H EN BASE BINARIA.)
      LEE=2
      IMP=3
  200 READ(LEE,100,END=250)NUM
      WRITE(IMP,101)NUM
  210 IF(NUM.LE.1)GO TO 220
C       NUM ES MAYOR QUE UNO, SE SIGUE DESCOMPONIENDO
        NE2=NUM/2
        NB=NUM-NE2*2
C       NB ES UNO O CERO
        WRITE(IMP,102)NB
        NUM=NE2
        GO TO 210
  220 CONTINUE
      IF(NUM.EQ.1)GO TO 230
C       SE IMPRIME EL ULTIMO CERO EN LA REPRESENTACION BINARIA
        I=0
        WRITE(IMP,102)I
        GO TO 240
  230 CONTINUE
C       SE IMPRIME EL ULTIMO 1 EN LA REPRESENTACION BINARIA
        I=1
        WRITE(IMP,102)I
  240 CONTINUE
      WRITE(IMP,103)
      GO TO 200
  250 CALL EXIT
      END
// XEQ
   12
   63
   10
00001
  011
 -131
    0
    1
   13
/*
```


β
I = 1
I
γ
"EN BASE BI-
NARIA"
α

# RESULTADOS

```
   12 EN BASE DECIMAL ES
0
0
1
1
EN BASE BINARIA.
  163 EN BASE DECIMAL ES
1
1
0
0
0
1
0
1
EN BASE BINARIA.
   10 EN BASE DECIMAL ES
0
1
0
1
EN BASE BINARIA.
    1 EN BASE DECIMAL ES
1
EN BASE BINARIA.
   11 EN BASE DECIMAL ES
1
1
0
1
EN BASE BINARIA.
 -131 EN BASE DECIMAL ES
0
EN BASE BINARIA.
    0 EN BASE DECIMAL ES
0
EN BASE BINARIA.
    1 EN BASE DECIMAL ES
1
EN BASE BINARIA.
   13 EN BASE DECIMAL ES
1
0
1
1
EN BASE BINARIA.
```

" CAMBIO DE BASE , BINARIO A DECIMAL"

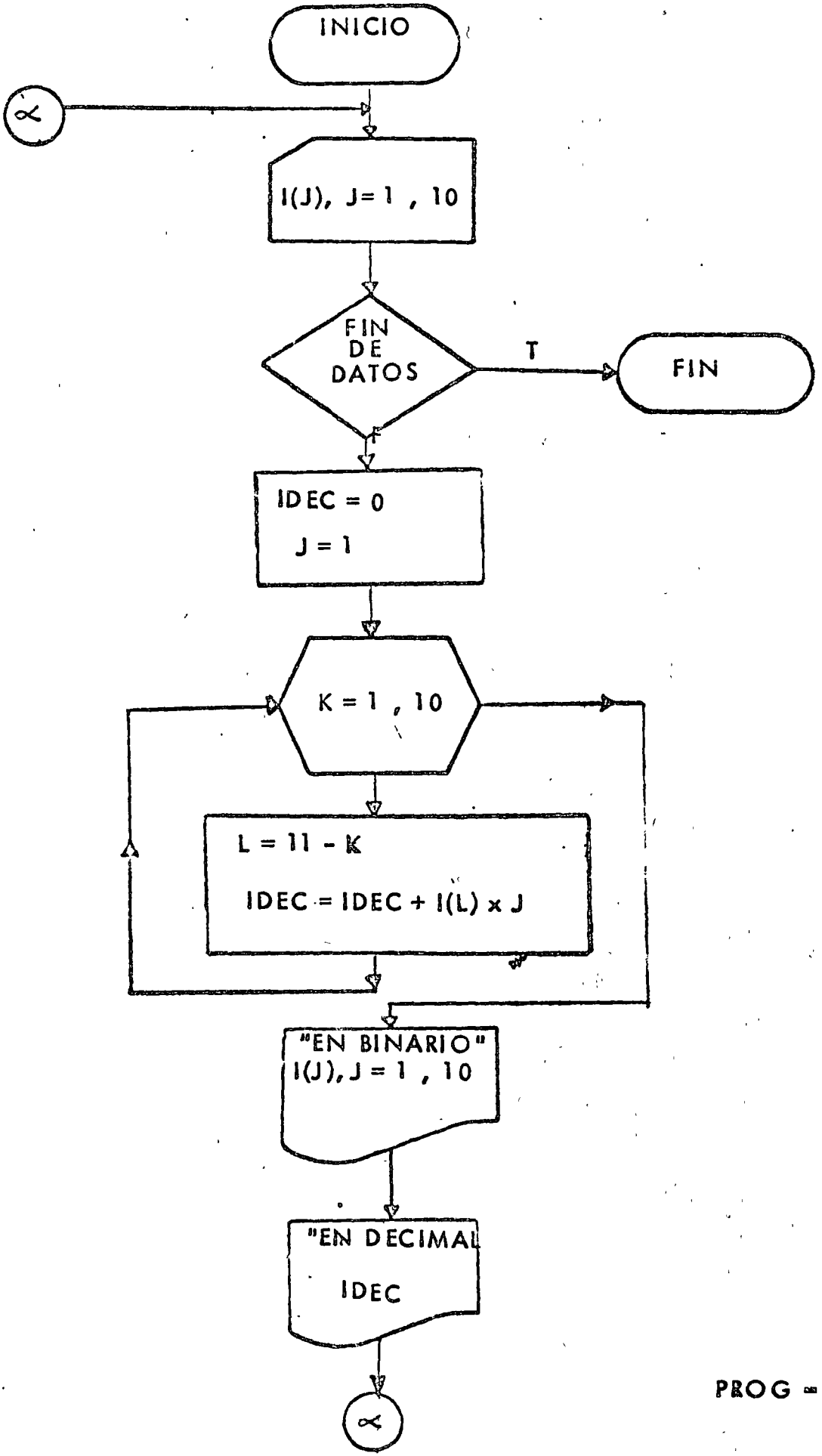


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------O N C E---------
C      CAMBIO DE BASE 8 BINARIA A DECIMAL
  100 FORMAT(3I1)
  101 FORMAT(1X, 3I1,19H EN BASE BINARIA ES)
  102 FORMAT(1X,I5,17H EN BASE DECIMAL.)
      LEE=2
      IMP=3
  200 READ(2,100,END=210)I3,I2,I1
      IDEC=0
C     J CONTIENE LAS POTENCIAS DE 2.
      J=1
      IDEC=IDEC+J*I1
      J=2*J
      IDEC=IDEC+J*I2
      J=2*J
      IDEC=IDEC+J*I3
      WRITE(IMP,101)I3,I2,I1
C     IDEC CONTIENE LA REPRESENTACION DECIMAL DEL NUMERO BINARIO.
      WRITE(IMP,102)IDEC
      GO TO 200
  210 CALL EXIT
      END
// XEQ
001
010
011
100
101
110
111
/*
```

## RESULTADOS

```
001 EN BASE BINARIA ES
      1 EN BASE DECIMAL.
010 EN BASE BINARIA ES
      2 EN BASE DECIMAL.
011 EN BASE BINARIA ES
      3 EN BASE DECIMAL.
100 EN BASE BINARIA ES
      4 EN BASE DECIMAL.
101 EN BASE BINARIA ES
      5 EN BASE DECIMAL.
110 EN BASE BINARIA ES
      6 EN BASE DECIMAL.
111 EN BASE BINARIA ES
      7 EN BASE DECIMAL.
```

"CAMBIO DE BASE, BINARIA A DECIMAL USANDO ARREGLOS"

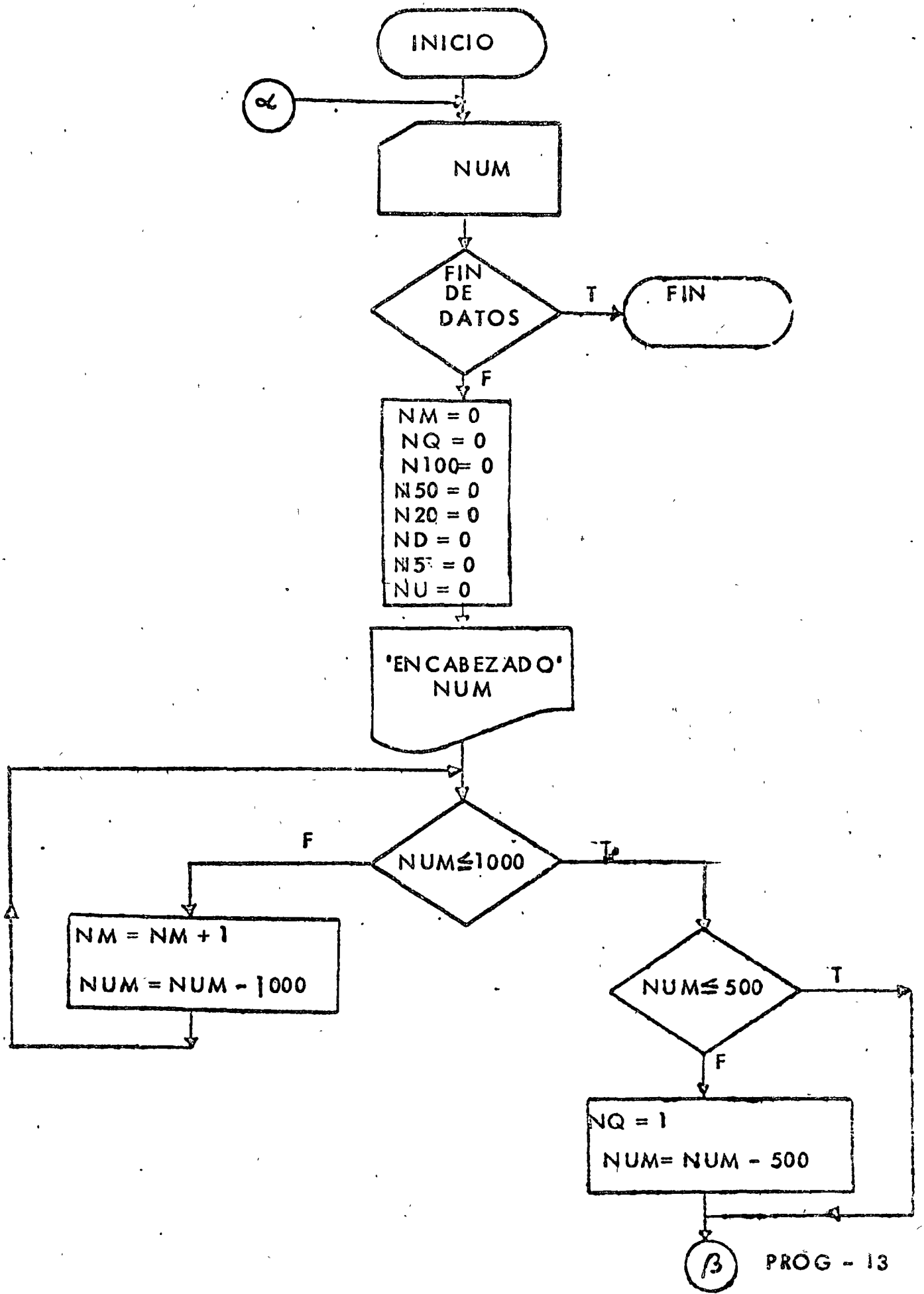


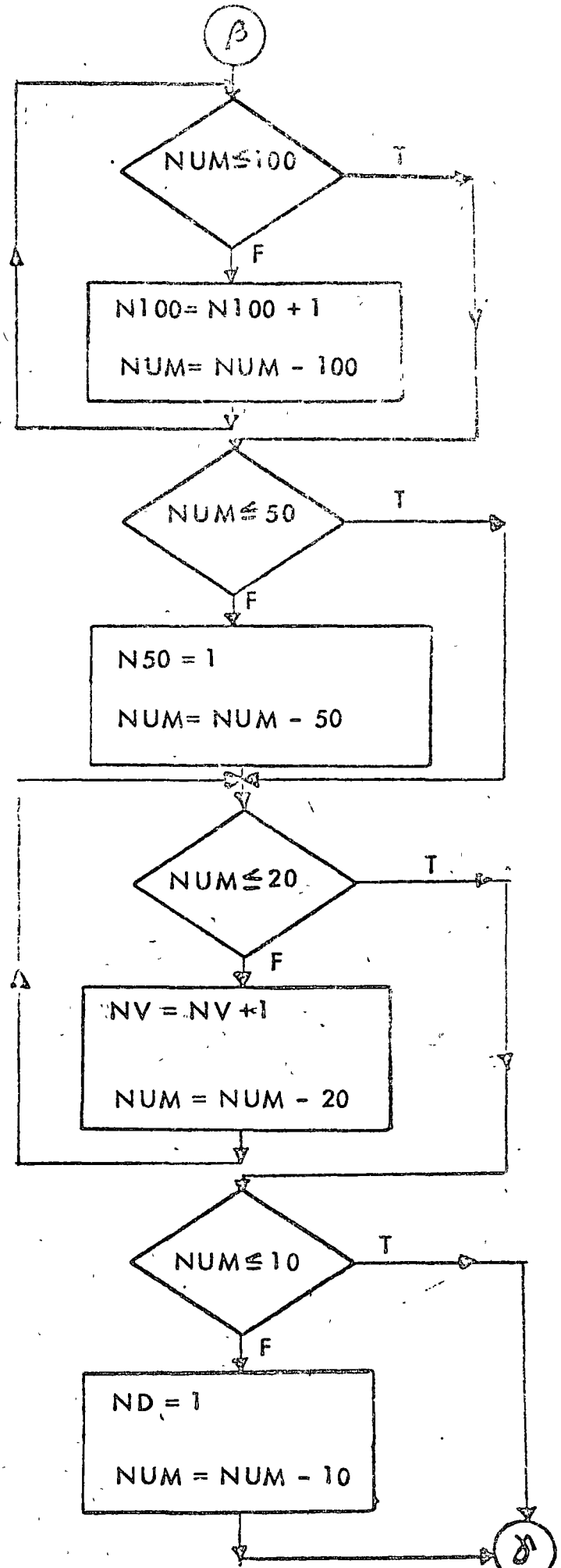

PROG - 12

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------D O C E--------
C      CAMBIO DE BASE 8 BINARIA A DECIMAL
C      USANDO ARREGLOS
       DIMENSION I(10)
  100  FORMAT(10I1)
  101  FORMAT(1X,10I1,19H EN BASE BINARIA ES)
  102  FORMAT(1X,I5,17H EN BASE DECIMAL.)
       LEE=2
       IMP=3
  200  READ(2,100,END=220)I
       IDEC=0
       J=1
       DO 210 K=1,10
C       SE ANALIZA EL VECTOR I DE DERECHA A IZQUIERDA
        L=11-K
        IDEC=IDEC+I(L)*J
C       J CONTIENE LAS POTENCIAS DE 2
        J=J*2
  210  CONTINUE
       WRITE(IMP,101)(I(J),J=1,10)
       WRITE(IMP,102)IDEC
       GO TO 200
  220  CALL EXIT
       END
// XEQ
         1
        11
1000100010
1
1        1
   1001001
        00
/*
```

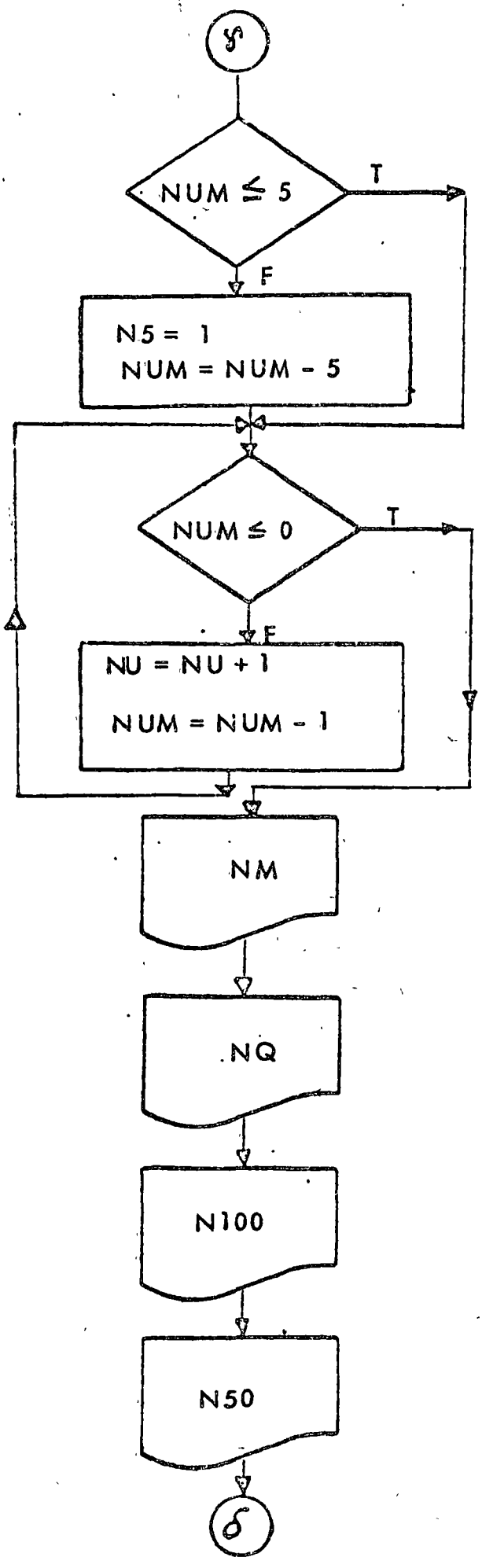
## RESULTADOS

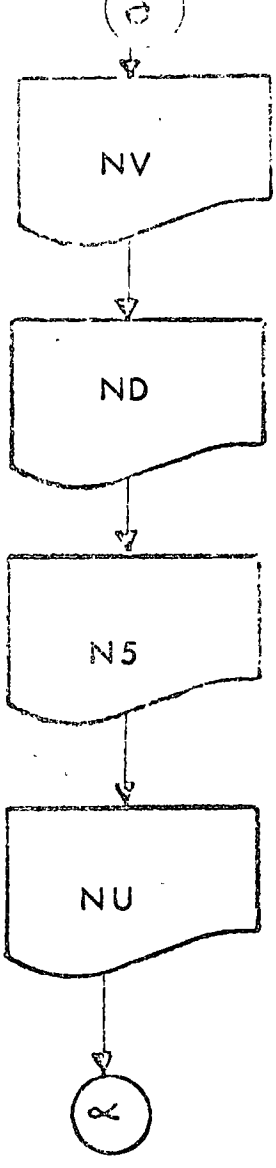
```
0000000001 EN BASE BINARIA ES
     1 EN BASE DECIMAL.
0000000110 EN BASE BINARIA ES
     6 EN BASE DECIMAL.
1000100010 EN BASE BINARIA ES
   546 EN BASE DECIMAL.
1000000000 EN BASE BINARIA ES
   512 EN BASE DECIMAL.
1000000001 EN BASE BINARIA ES
   513 EN BASE DECIMAL.
0001001001 EN BASE BINARIA ES
    73 EN BASE DECIMAL.
0000000000 EN BASE BINARIA ES
     0 EN BASE DECIMAL.
```


INICIO
α
NUM
FIN DE DATOS
T
FIN
F
NM = 0
NQ = 0
N100 = 0
N50 = 0
N20 = 0
ND = 0
N5 = 0
NU = 0
'ENCABEZADO' NUM
F
NUM≤1000
T
NM = NM + 1
NUM = NUM - 1000
NUM≤500
T
F
NQ = 1
NUM= NUM - 500
β


β
NUM≤100
T
F
N100= N100 + 1
NUM= NUM - 100
NUM≤ 50
T
F
N50 = 1
NUM= NUM - 50
NUM≤20
T
F
NV = NV +1
NUM = NUM - 20
NUM≤10
T
F
ND = 1
NUM = NUM - 10
δ

PROG - 13


NUM ≤ 5
T
F
N5 = 1
NUM = NUM - 5
NUM ≤ 0
T
F
NU = NU + 1
NUM = NUM - 1
NM
NQ
N100
N50

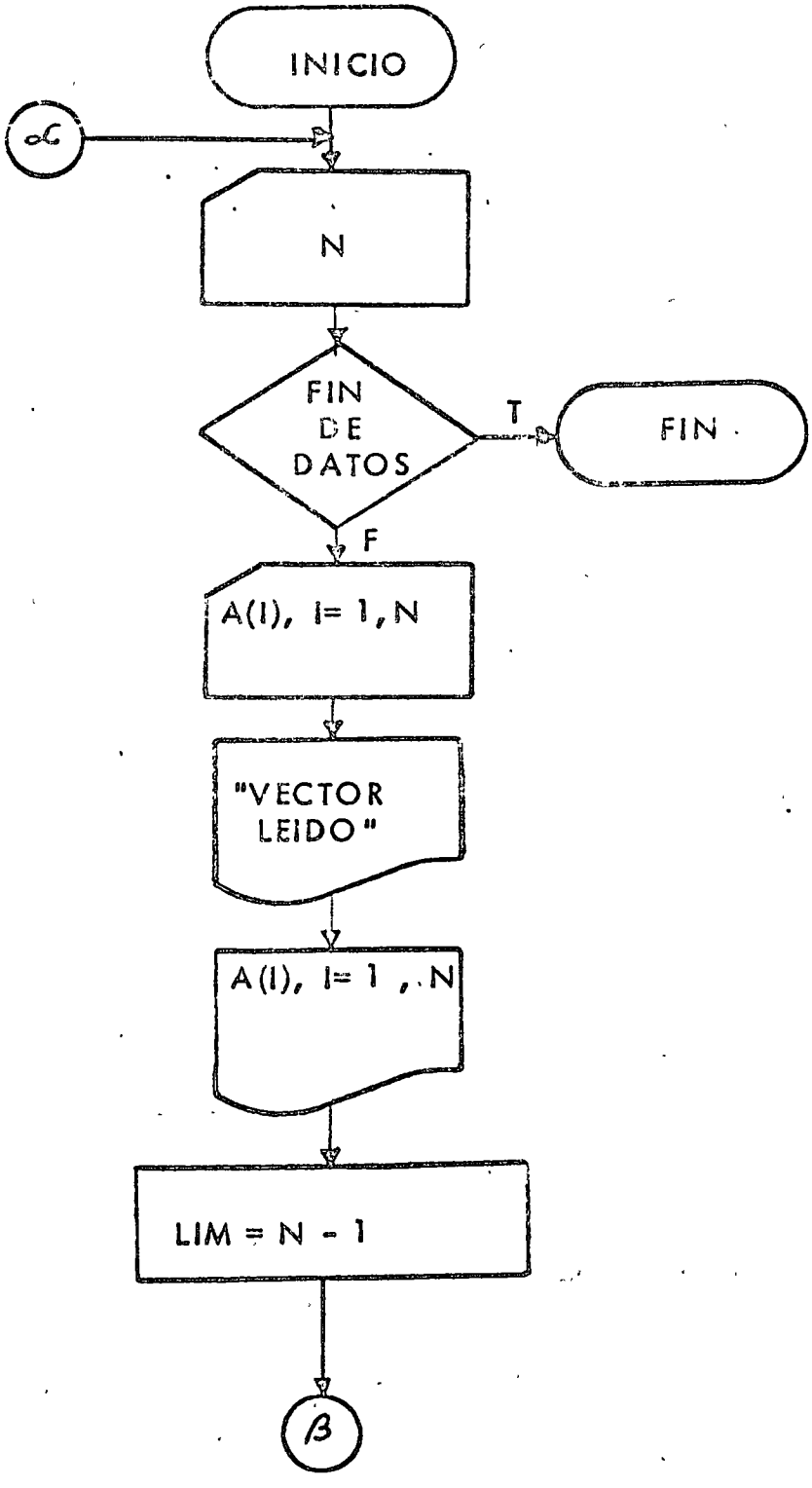

(5)
NV
ND
N5
NU
∝

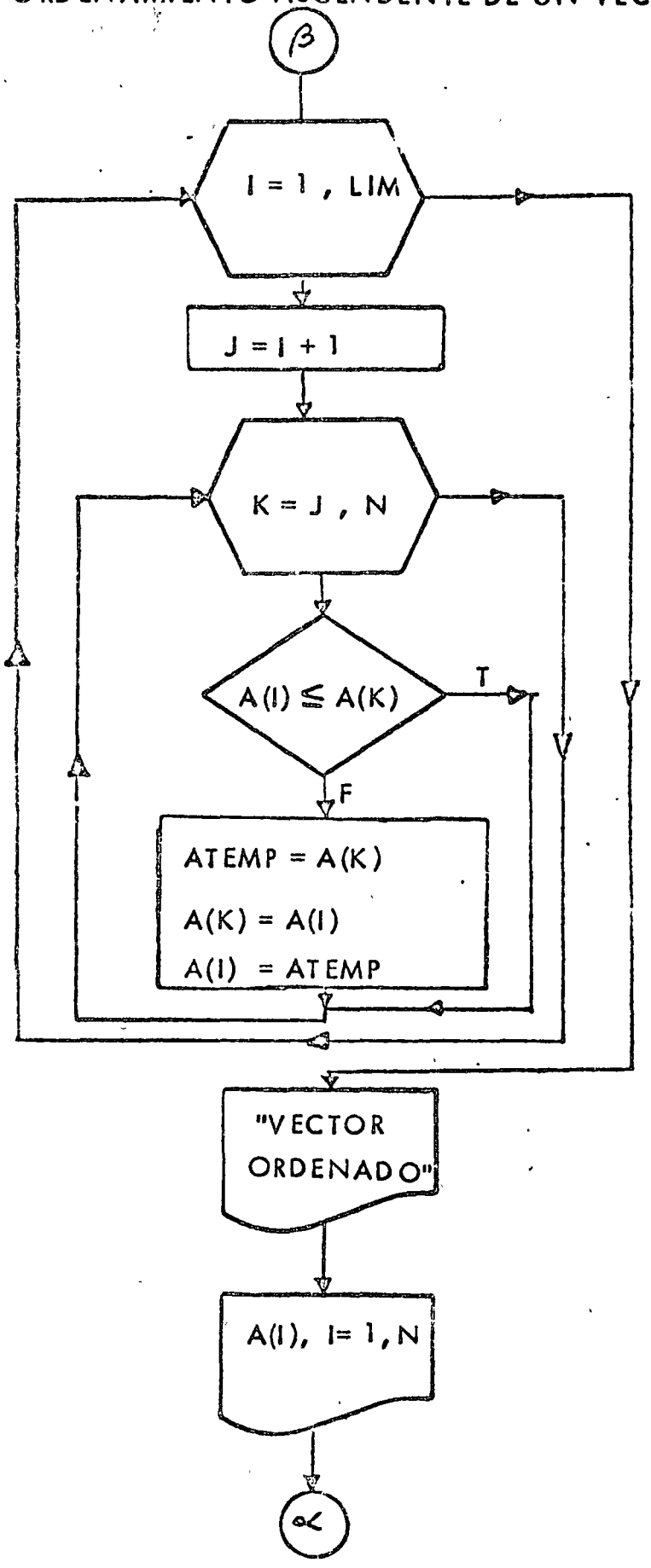
```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------T R E C E-------
C      CALCULO DEL NUMERO DE BILLETES
  100 FORMAT(I4)
  101 FORMAT(/,11H EL NUMERO=,I4,24H,00=PUEDE DESGLOSARSE EN)
  102 FORMAT(I5, 7H DE MIL)
  103 FORMAT(I5,14H DE QUINIENTOS)
  104 FORMAT(I5, 8H DE CIEN)
  105 FORMAT(I5,13H DE CINCUENTA)
  106 FORMAT(I5,10H DE VEINTE)
  107 FORMAT(I5, 8H DE DIEZ)
  108 FORMAT(I5, 9H DE CINCO)
  109 FORMAT(I5, 7H DE UNO)
      LEE=2
      IMP=3
  200 READ(LEE,100,END=330)NUM
      NM=0
      NQ=0
      N100=0
      N50=0
      NV=0
      ND=0
      N5=0
      NU=0
      WRITE(IMP,101)NUM
  210 IF(NUM.LE.1000)GO TO 220
C       NUM ES MAYOR QUE  1000
        NM=NM+1
        NUM=NUM-1000
        GO TO 210
  220 CONTINUE
      IF(NUM.LE.500)GO TO 230
C       NUM ES MAYOR QUE 500 (MAXIMO UN BILLETE DE 500)
        NQ=1
        NUM=NUM-500
  230 CONTINUE
  240 IF(NUM.LE.100)GO TO 250
C       NUM ES MAYOR QUE 100
        N100=N100+1
        NUM=NUM-100
        GO TO 240
  250 CONTINUE
      IF(NUM.LE.50)GO TO 260
C       NUM ES MAYOR QUE 50 (MAXIMO UN BILLETE DE 50)
        N50=1
        NUM=NUM-50
  260 CONTINUE
  270 IF(NUM.LE.20)GO TO 280
C       NUM ES MAYOR QUE 20
        NV=NV+1
        NUM=NUM-20
        GO TO 270
  280 CONTINUE
      IF(NUM.LE.10)GO TO 290
C       NUM ES MAYOR QUE 10 (MAXIMO UN BILLETE DE 10)
        ND=1
        NUM=NUM-10
  290 CONTINUE
      IF(NUM.LE.5)GO TO 300
```

```
C       NUM ES MAYOR QUE 5 (MAXIMO UN BILLETE DE 5)
        NS=1
        NUM=NUM-5
  300 CONTINUE
  310 IF(NUM.LE.0)GO TO 320
C       NUM ES MAYOR QUE 1
        NU=NU+1
        NUM=NUM-1
        GO TO 310
  320 CONTINUE
      WRITE(IMP,102)NM
      WRITE(IMP,103)NQ
      WRITE(IMP,104)N100
      WRITE(IMP,105)N50
      WRITE(IMP,106)NV
      WRITE(IMP,107)ND
      WRITE(IMP,108)N5
      WRITE(IMP,109)NU
      GO TO 200
  330 CALL EXIT
      END
// XEQ
9000
1314
6893
1000
 500
  13
/*
```

# RESULTADOS

```
EL NUMERO=9000.00=PUEDE DESGLOSARSE EN
 8 DE MIL
 1 DE QUINIENTOS
 4 DE CIEN
 1 DE CINCUENTA
 2 DE VEINTE
 0 DE DIEZ
 1 DE CINCO
 5 DE UNO

EL NUMERO=1314.00=PUEDE DESGLOSARSE EN
 1 DE MIL
 0 DE QUINIENTOS
 3 DE CIEN
 0 DE CINCUENTA
 0 DE VEINTE
 1 DE DIEZ
 0 DE CINCO
 4 DE UNO

EL NUMERO=6893.00=PUEDE DESGLOSARSE EN
 6 DE MIL
 1 DE QUINIENTOS
 3 DE CIEN
 1 DE CINCUENTA
 2 DE VEINTE
 0 DE DIEZ
 0 DE CINCO
 3 DE UNO

EL NUMERO=1000.00=PUEDE DESGLOSARSE EN
 0 DE MIL
 1 DE QUINIENTOS
 4 DE CIEN
 1 DE CINCUENTA
 2 DE VEINTE
 0 DE DIEZ
 1 DE CINCO
 5 DE UNO

EL NUMERO= 500.00=PUEDE DESGLOSARSE EN
 0 DE MIL
 0 DE QUINIENTOS
 4 DE CIEN
 1 DE CINCUENTA
 2 DE VEINTE
 0 DE DIEZ
 1 DE CINCO
 5 DE UNO

EL NUMERO=  13.00=PUEDE DESGLOSARSE EN
 0 DE MIL
 0 DE QUINIENTOS
 0 DE CIEN
 0 DE CINCUENTA
 0 DE VEINTE
 1 DE DIEZ
 0 DE CINCO
 3 DE UNO
```


INICIO
α
N
FIN DE DATOS
T
FIN
F
A(I), I= 1, N
"VECTOR LEIDO"
A(I), I= 1 , N
LIM = N - 1
β

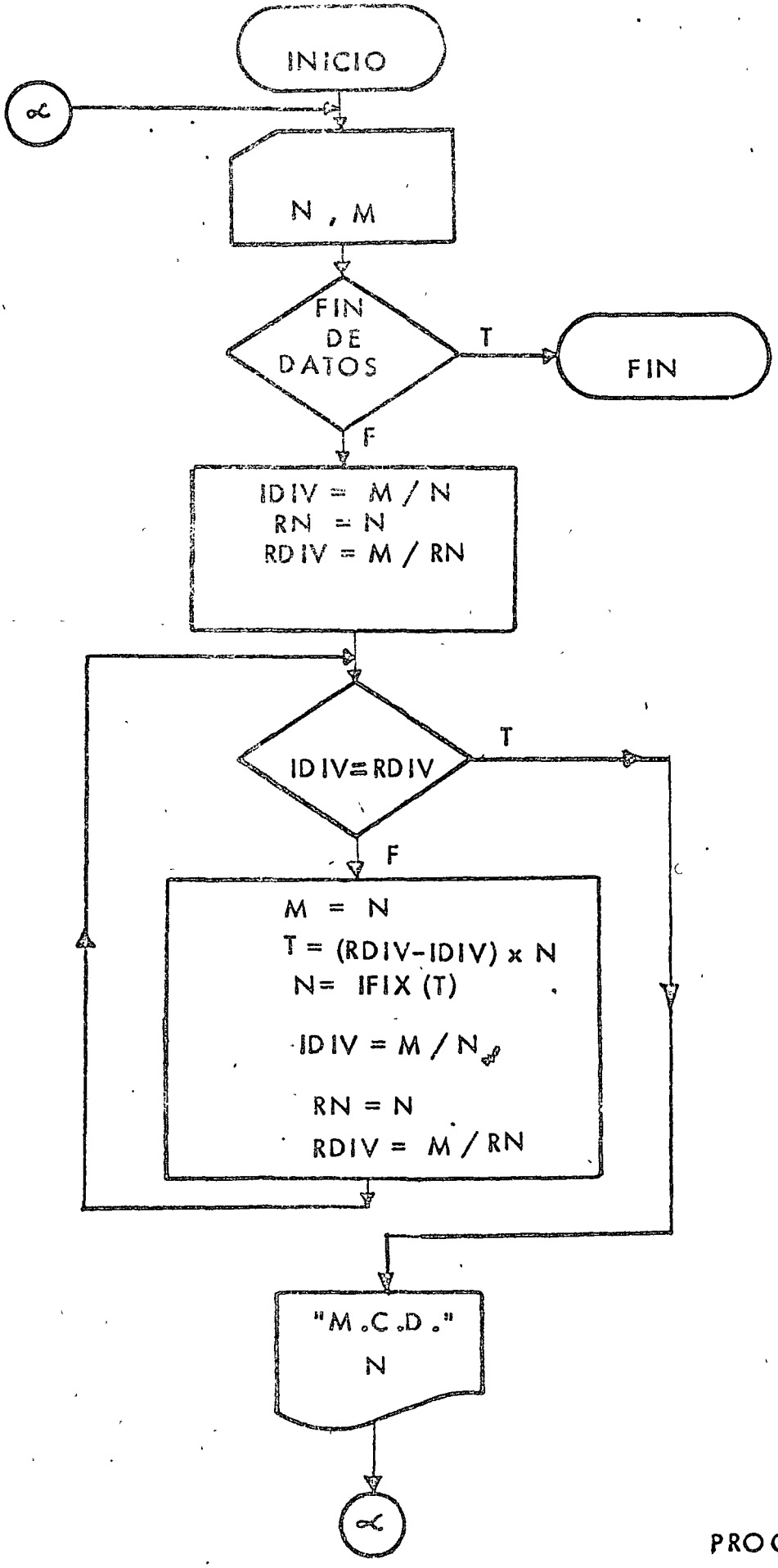

β
I = 1 , LIM
J = I + 1
K = J , N
A(I) ≤ A(K)
T
F
ATEMP = A(K)
A(K) = A(I)
A(I) = ATEMP
"VECTOR ORDENADO"
A(I), I= 1, N
α

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------C A T O R C E---
C      ORDENAMIENTO ASCENDENTE DE UN VECTOR
       DIMENSION A(100)
  100 FORMAT(I2)
  101 FORMAT(8F10.0)
  102 FORMAT(13H VECTOR LEIDO,//)
  103 FORMAT(10(1X,F11.4))
  104 FORMAT(16H VECTOR ORDENADO,//)
       LEE=2
       IMP=3
  200 READ(LEE,100,END=240)N
C      N REPRESENTA EL NUMERO DE ELEMENTOS A ORDENAR
       READ(LEE,101)(A(I),I=1,N)
       WRITE(IMP,102)
       WRITE(IMP,103)(A(I),I=1,N)
       LIM=N-1
       DO 230 I=1,LIM
         J=I+1
C        SE ASUME QUE A(I) ES EL MENOR
         DO 220 K=J,N
           IF(A(I).LE.A(K))GO TO 210
C            A(I) FUE MAYOR QUE A(K)
             ATEMP=A(K)
             A(K)=A(I)
             A(I)=ATEMP
  210      CONTINUE
  220    CONTINUE
C        AHORA SE TIENE EN A(I) EL MENOR
  230  CONTINUE
       WRITE(IMP,104)
       WRITE(IMP,103)(A(I),I=1,N)
       GO TO 200
  240  CALL EXIT
       END
// XEQ
04
-4.          1.          -3.          17.
03
 0.        -287.          32.
04
-28.        -32.          11.          0.
/*
```

# RESULTADOS

```
VECTOR LEIDO
    -4.0000      1.0000     -3.0000     17.0000
VECTOR ORDENADO
    -4.0000     -3.0000      1.0000     17.0000
VECTOR LEIDO
     0.0000   -287.0000     32.0000
VECTOR ORDENADO
  -287.0000      0.0000     32.0000
VECTOR LEIDO
   -28.0000    -32.0000     11.0000      0.0000
VECTOR ORDENADO
   -32.0000    -28.0000      0.0000     11.0000
```

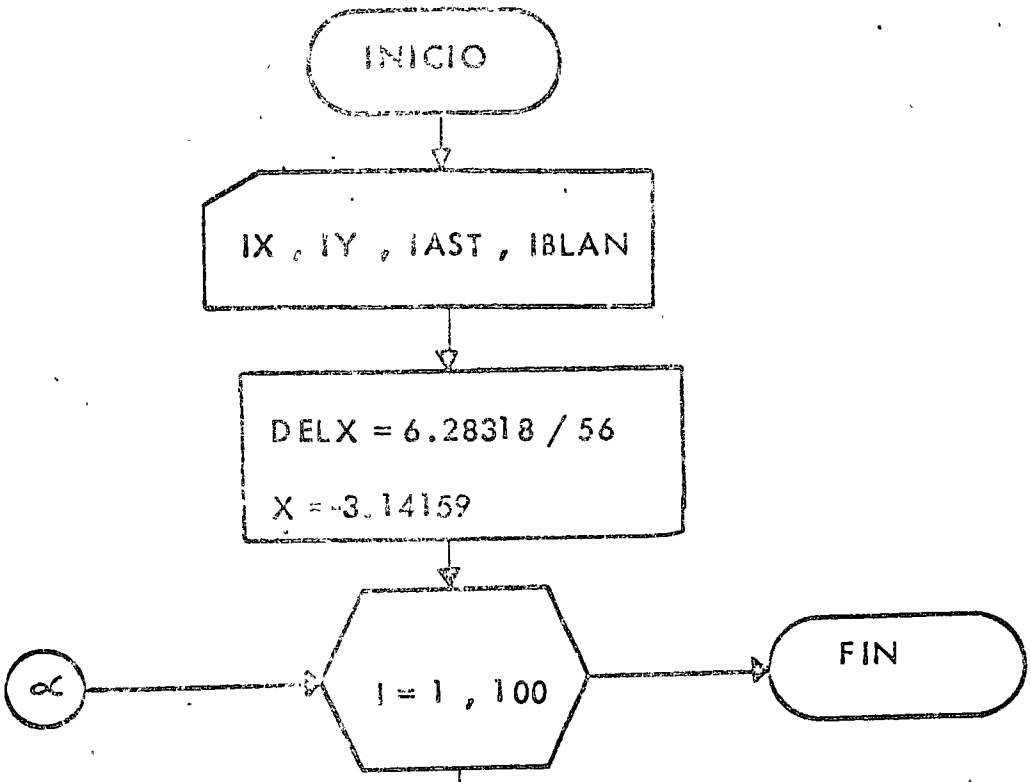
" MAXIMO COMUN MULTIPLO ALGORITMO DE EUCLIDES "

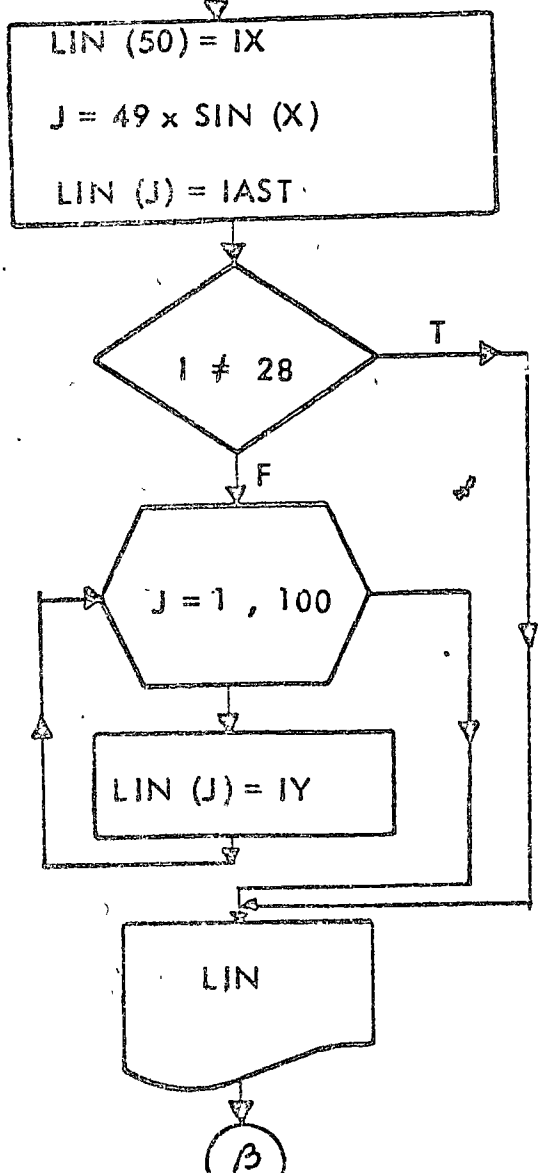


```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------Q U I N C E-----
C       MAXIMO COMUN MULTIPLO
C       ALGORITMO DE EUCLIDES
  100 FORMAT(2I3)
  101 FORMAT(3H N=,I3,3H M=,I3)
  102 FORMAT(8H M.C.D.=,I3,//)
      LEE=2
      IMP=3
  200 READ(LEE,100,END=230)N,M
C       SE CALCULA EL RESIDUO
      IDIV=M/N
      RN=N
      RDIV=M/RN
  210 IF(IDIV.EQ.RDIV)GO TO 220
        M=N
        T=(RDIV-IDIV)*N
        N=IFIX(T)
        IDIV=M/N
        RN=N
        RDIV=M/RN
        GO TO 210
  220 CONTINUE
C     N REPRESENTA EL MAXIMO COMUN DIVISOR
      WRITE(IMP,102)N
      GO TO 200
  230 CALL EXIT
      END
// XEQ
  5  7
  3  6
  8 16
 11 98
 16 24
  8 12
/*
```

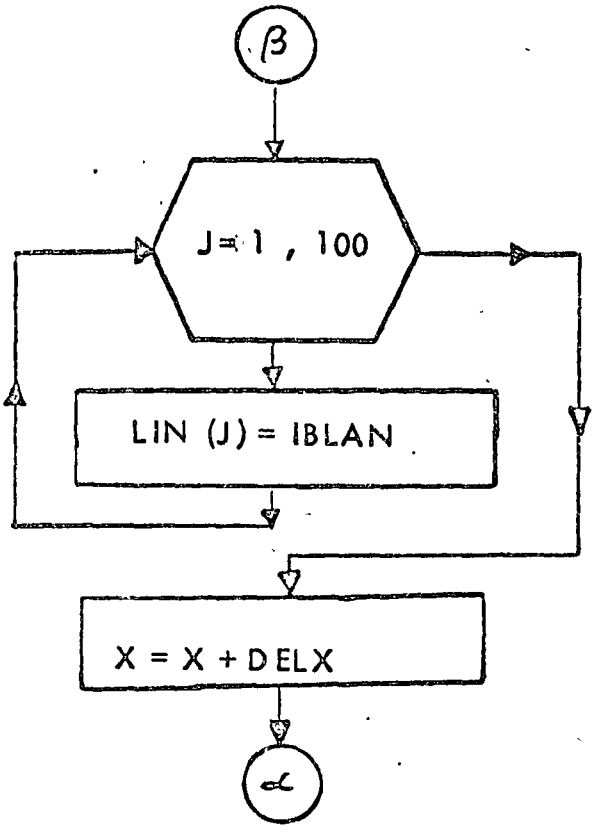
## RESULTADOS

```
M.C.D.=  1
M.C.D.=  3
M.C.D.=  8
M.C.D.=  1
M.C.D.=  8
M.C.D.=  4
```


INICIO
IX , IY , IAST , IBLAN
DELX = 6.28318 / 56
X = -3.14159
α
I = 1 , 100
FIN


LIN (50) = IX
J = 49 x SIN (X)
LIN (J) = IAST
I ≠ 28
T
F
J = 1 , 100
LIN (J) = IY
LIN
β

PROG - 16


β
J=1 , 100
LIN (J) = IBLAN
X = X + DELX
α

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------D I E C I S E I S--------
C      GRAFICA DE SEN(X)
       DIMENSION LIN(100)
  100  FORMAT(4A1)
  101  FORMAT(10X,100A1)
       LEE=2
       IMP=3
       READ(LEE,100)IX,IY,IAST,IBLAN
       DELX=6.28318/56
       X =-3.14159
       DO 200 I=1,100
         LIN(I)=IBLAN
  200  CONTINUE
       DO 240 I=1,56
         LIN(50)=IX
         J=49*SIN(X)+50
         LIN(J)=IAST
         IF(I.NE.28)GO TO 220
           DO 210 J=1,100
C            I FUE IGUAL A 28, SE IMPRIME EL EJE Y
             LIN(J)=IY
  210      CONTINUE
  220    CONTINUE
         WRITE(IMP,101)LIN
         DO 230 J=1,100
           LIN(J)=IBLAN
  230    CONTINUE
         X=X+DELX
  240  CONTINUE
       CALL EXIT
       END
// XEQ
XY*
/*
```

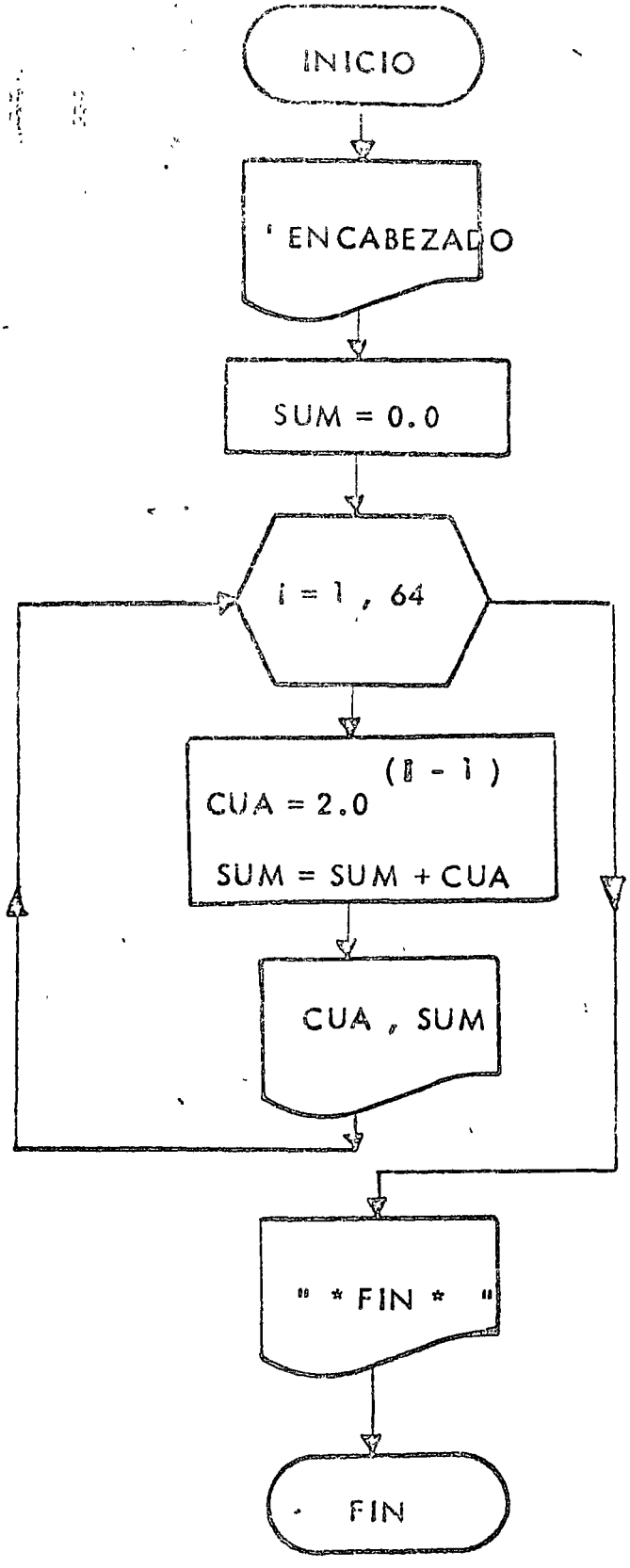
# RESULTADOS



PROG - 16

R/6

" No. DE GRANOS DE MAIZ GANADOS POR EL INVENTOR DEL AJEDREZ "



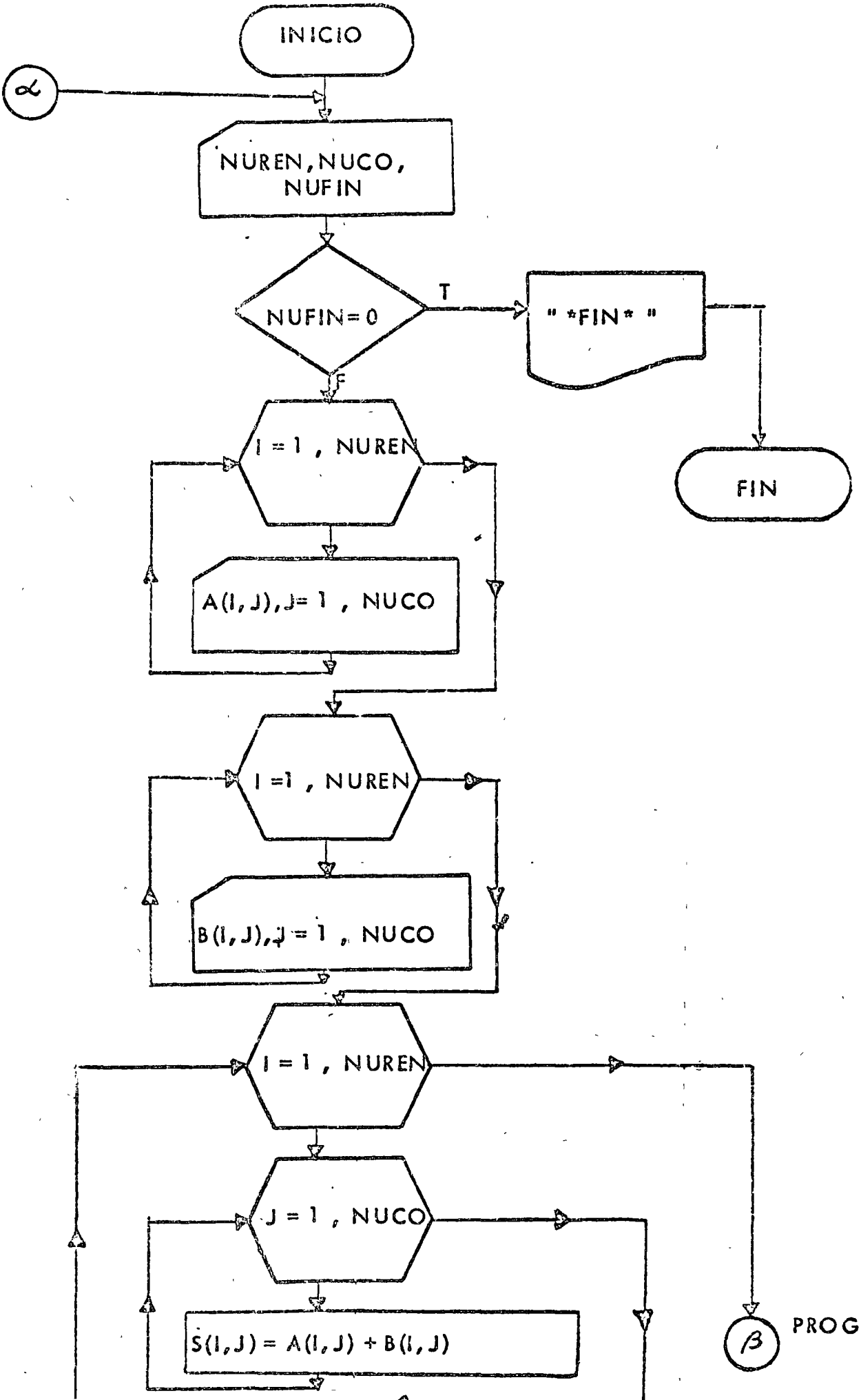
```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*IOCS(CARD,1132 PRINTER)
*ONE WORD INTEGERS
C--------------D I E C I S I E T E-----
C      ESTE PROGRAMA CALCULA EL NUMERO DE GRANOS DE MAIZ QUE COBRO EL
C      INVENTOR DE AJEDRES
C      FILES
           LEE=2
           IMP=3
C      FORMATOS
  100 FORMAT(10X,6HCUADRO,9X,4HSUMA,//)
  101 FORMAT(3X,I2,2E15.7)
  102 FORMAT(///)
  103 FORMAT(53X,11H**********)
  104 FORMAT(53X,11H*   FIN   *)
      WRITE(IMP,100)
      SUM=0.0
      DO 200 I=1,64
       CUA=2.0**(I-1)
       SUM=SUM+CUA
       WRITE(IMP,101)I,CUA,SUM
  200 CONTINUE
      WRITE(IMP,102)
      WRITE(IMP,103)
      WRITE(IMP,104)
      WRITE(IMP,103)
      CALL EXIT
      END
// XEQ
/*
```

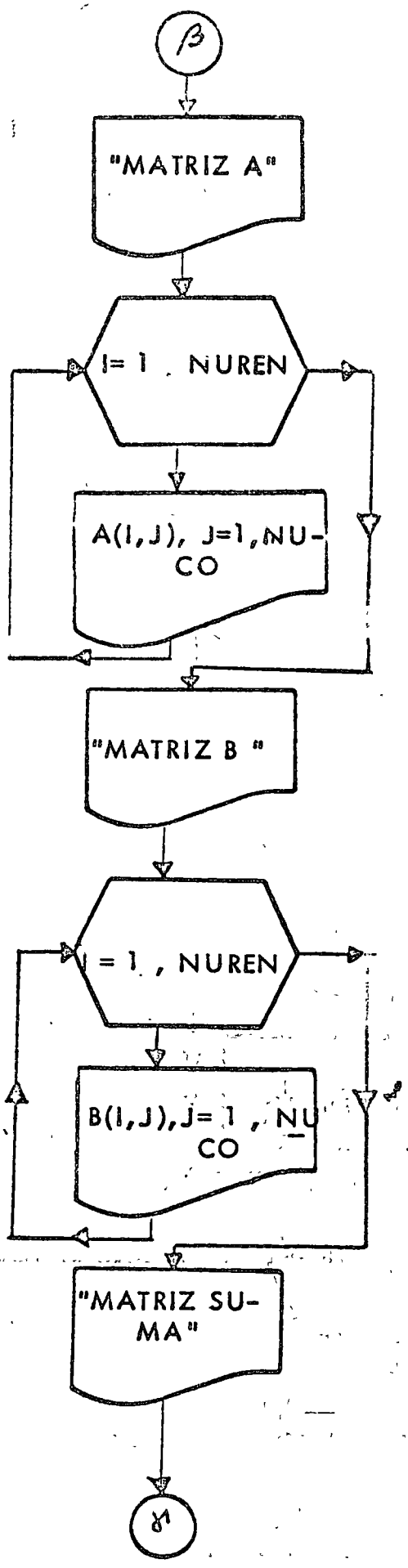
RESULTADOS

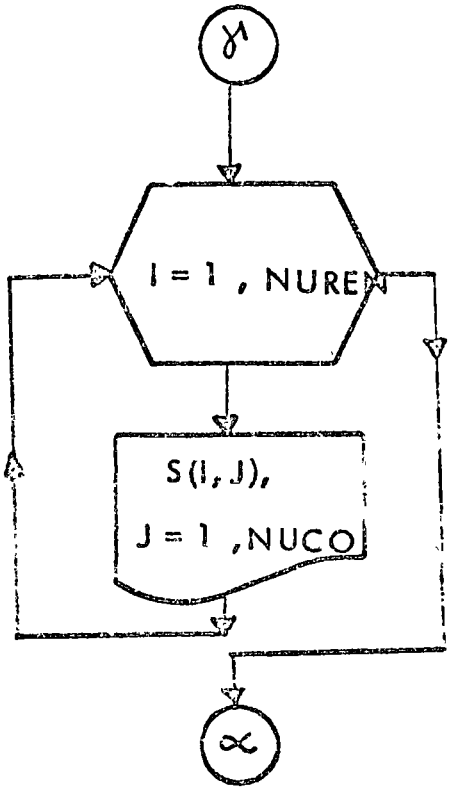
| | CUADRO | SUMA |
|---|---|---|
| 1 | .1000000E+01 | .1000000E+01 |
| 2 | .2000000E+01 | .3000000E+01 |
| 3 | .4000000E+01 | .7000000E+01 |
| 4 | .8000000E+01 | .1500000E+02 |
| 5 | .1600000E+02 | .3100000E+02 |
| 6 | .3200000E+02 | .6300000E+02 |
| 7 | .6400000E+02 | .1270000E+03 |
| 8 | .1280000E+03 | .2550000E+03 |
| 9 | .2560000E+03 | .5110000E+03 |
| 10 | .5120000E+03 | .1023000E+04 |
| 11 | .1024000E+04 | .2047000E+04 |
| 12 | .2048000E+04 | .4095000E+04 |
| 13 | .4096000E+04 | .8191000E+04 |
| 14 | .8192000E+04 | .1638300E+05 |
| 15 | .1638400E+05 | .3276700E+05 |
| 16 | .3276800E+05 | .6553500E+05 |
| 17 | .6553600E+05 | .1310710E+06 |
| 18 | .1310720E+06 | .2621430E+06 |
| 19 | .2621440E+06 | .5242870E+06 |
| 20 | .5242880E+06 | .1048575E+07 |
| 21 | .1048576E+07 | .2097151E+07 |
| 22 | .2097152E+07 | .4194303E+07 |
| 23 | .4194304E+07 | .8388607E+07 |
| 24 | .8388608E+07 | .1677722E+08 |
| 25 | .1677722E+08 | .3355443E+08 |
| 26 | .3355443E+08 | .6710886E+08 |
| 27 | .6710886E+08 | .1342177E+09 |
| 28 | .1342177E+09 | .2684355E+09 |
| 29 | .2684355E+09 | .5368709E+09 |
| 30 | .5368709E+09 | .1073742E+10 |
| 31 | .1073742E+10 | .2147484E+10 |
| 32 | .2147484E+10 | .4294967E+10 |
| 33 | .4294967E+10 | .8589935E+10 |
| 34 | .8589935E+10 | .1717987E+11 |
| 35 | .1717987E+11 | .3435974E+11 |
| 36 | .3435974E+11 | .6871948E+11 |
| 37 | .6871948E+11 | .1374390E+12 |
| 38 | .1374390E+12 | .2748779E+12 |
| 39 | .2748779E+12 | .5497558E+12 |
| 40 | .5497558E+12 | .1099512E+13 |
| 41 | .1099512E+13 | .2199023E+13 |
| 42 | .2199023E+13 | .4398047E+13 |
| 43 | .4398047E+13 | .8796093E+13 |
| 44 | .8796093E+13 | .1759219E+14 |
| 45 | .1759219E+14 | .3518437E+14 |
| 46 | .3518437E+14 | .7036874E+14 |
| 47 | .7036874E+14 | .1407375E+15 |
| 48 | .1407375E+15 | .2814750E+15 |
| 49 | .2814750E+15 | .5629500E+15 |
| 50 | .5629500E+15 | .1125900E+16 |
| 51 | .1125900E+16 | .2251800E+16 |
| 52 | .2251800E+16 | .4503600E+16 |
| 53 | .4503600E+16 | .9007199E+16 |
| 54 | .9007199E+16 | .1801440E+17 |
| 55 | .1801440E+17 | .3602880E+17 |
| 56 | .3602880E+17 | .7205759E+17 |
| 57 | .7205759E+17 | .1441152E+18 |
| 58 | .1441152E+18 | .2882304E+18 |
| 59 | .2882304E+18 | .5764608E+18 |
| 60 | .5764608E+18 | .1152922E+19 |
| 61 | .1152922E+19 | .2305843E+19 |
| 62 | .2305843E+19 | .4611686E+19 |
| 63 | .4611686E+19 | .9223372E+19 |
| 64 | .9223372E+19 | .1844674E+20 |

PROG - 17

************
* FIN *
************


INICIO
α
NUREN, NUCO, NUFIN
NUFIN= 0
T
F
" *FIN* "
FIN
I = 1 , NUREN
A(I, J), J= 1 , NUCO
I =1 , NUREN
B(I, J), J = 1 , NUCO
I = 1 , NUREN
J = 1 , NUCO
S(I, J) = A(I, J) + B(I, J)
β


B
"MATRIZ A"
I= 1 , NUREN
A(I,J), J=1,NU-
CO
"MATRIZ B "
I = 1 , NUREN
B(I,J),J= 1 , NU
CO
"MATRIZ SU-
MA"
A


γ
I = 1 , NUREN
S(I, J),
J = 1 ,NUCO
∝

```
// JOB T
// FOR
*IOCS(CARD,1132 PRINTER)
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
C----------D I E C I O C H O---------
C      SUMA DE DOS MATRICES: A Y B
C      EL PROGRAMA ESTA HECHO PARA SUMAR DOS MATRICES DE 10X 10 MAXIMO.
C      SE RESERVAN LUGARES EN LA MEMORIA PARA LAS MATRICES A SUMAR Y PARA
C      LA MATRIZ SUMA.
       DIMENSION A(10,10),B(10,10),S(10,10)
C      FILES
           LEE=2
           IMP=3
C      FORMATOS
  100  FORMAT(3I2)
  101  FORMAT(10F8.3)
  102  FORMAT(///,5X,9HMATRIZ A:,//)
  103  FORMAT(5X,10(F8.3,2X),/)
  104  FORMAT(///,5X,9HMATRIZ B:,//)
  105  FORMAT(///,5X,18HLA MATRIZ SUMA ES:,//)
  106  FORMAT(53X,11H**********)
  107  FORMAT(53X,11H*   FIN   *)
C      LECTURA DEL NUMERO DE RENGLONES DE LAS MATRICES (NUREN) Y DEL NUME
C      RO DE COLUMNAS (NUCO),Y DE UN DETECTOR (NUFIN)
  199  READ(LEE,100)NUREN,NUCO,NUFIN
C      ANALISIS DE NUFIN. SI VALE CERO YA NO SE EJECUTA EL PROGRAMA,DE LO
C      CONTRARIO SI.
       IF(NUFIN.EQ.0)GO TO 1000
C       LECTURA POR RENGLONES DE LA MATRIZ A.
        DO 200 I=1,NUREN
         READ(LEE,101)(A(I,J),J=1,NUCO)
  200   CONTINUE
C       LECTURA POR RENGLONES DE LA MATRIZ B.
        DO 201 I=1,NUREN
         READ(LEE,101)(B(I,J),J=1,NUCO)
  201   CONTINUE
C       SE HARA LA SUMA ELEMENTO A ELEMENTO
        DO 203 I=1,NUREN
         DO 202 J=1,NUCO
           S(I,J)=A(I,J)+B(I,J)
  202    CONTINUE
  203   CONTINUE
C       IMPRESION DE LA MATRIZ A POR RENGLONES
        WRITE(IMP,102)
        DO 204 I=1,NUREN
         WRITE(IMP,103)(A(I,J),J=1,NUCO)
  204   CONTINUE
C       IMPRESION DE LA MATRIZ B POR RENGLONES
        WRITE(IMP,104)
        DO 205 I=1,NUREN
         WRITE(IMP,103)(B(I,J),J=1,NUCO)
  205   CONTINUE
C       IMPRESION DE LA MATRIZ S POR RENGLONES
        WRITE(IMP,105)
        DO 206 I=1,NUREN
        WRITE(IMP,103)(S(I,J),J=1,NUCO)
  206   CONTINUE
        GO TO 199
 1000  CONTINUE
       WRITE(IMP,106)
       WRITE(IMP,107)
       WRITE(IMP,106)
```

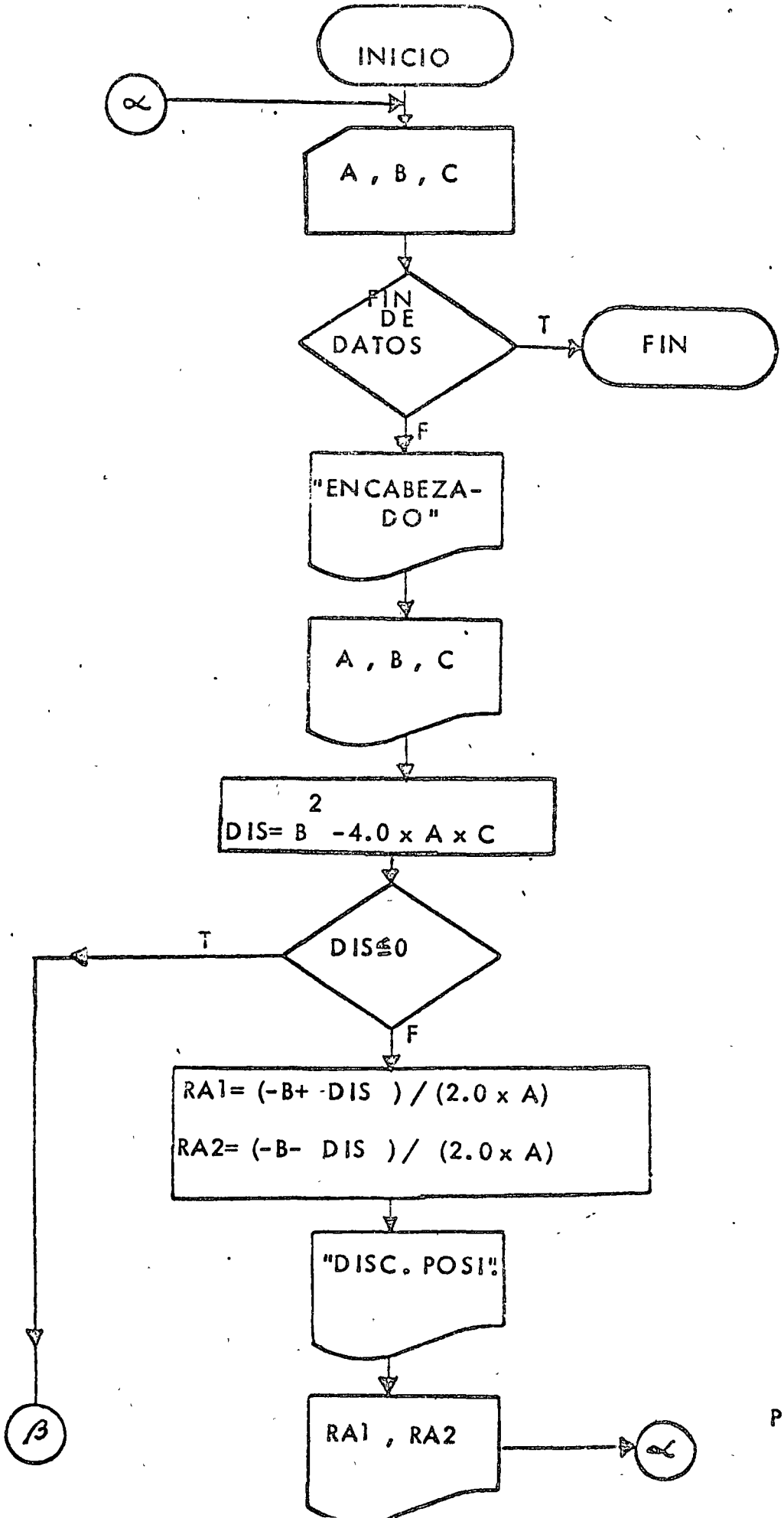
```
      CALL EXIT
      END
// XEQ
 2 2 1
   5.0      0.0
  12.2      3.4
  -4.7      2.1
   0.0     -1.2
 2 2 1
  90.15   -87.02
   5.478    12.22
 -90.15     87.02
  -5.478  -12.22
 2 2 0
/*
```

## RESULTADOS

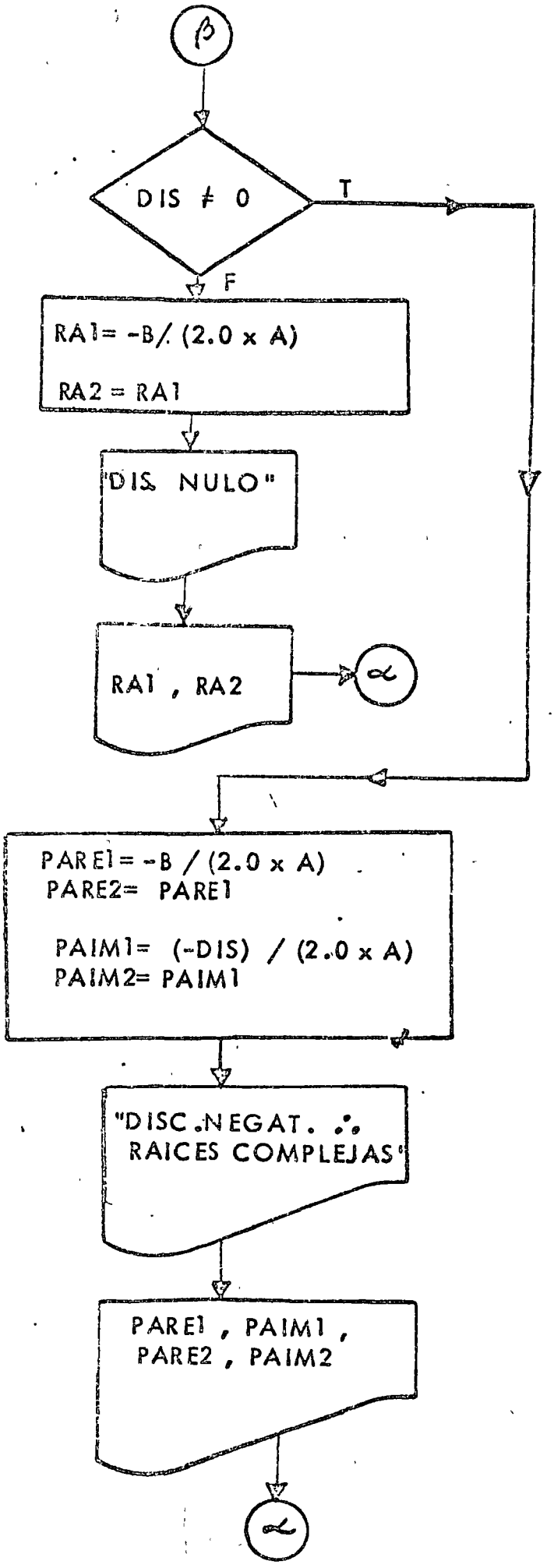
```
 MATRIZ A:

    5.000      0.000
   12.200      3.400

 MATRIZ B:

   -4.700      2.100
    0.000     -1.200

 LA MATRIZ SUMA ES:

    0.300      2.100
   12.200      2.200

 MATRIZ A:

   90.150    -87.020
    5.478     12.220

 MATRIZ B:

  -90.150     87.020
   -5.478    -12.220

 LA MATRIZ SUMA ES:

    0.000      0.000
   -0.000      0.000
```

PROG - 18

* FIN *


INICIO
α
A , B , C
FIN DE DATOS
T
FIN
F
"ENCABEZA-DO"
A , B , C
DIS= B 2 -4.0 x A x C
DIS≤0
T
F
RA1= (-B+ -DIS ) / (2.0 x A)
RA2= (-B- DIS ) / (2.0 x A)
"DISC. POSI".
RA1 , RA2
α
β

PROG - 19


B
DIS ≠ 0
T
F
RA1= -B/ (2.0 x A)
RA2 = RA1
"DIS. NULO"
RA1 , RA2
α
PARE1= -B / (2.0 x A)
PARE2= PARE1
PAIM1= (-DIS) / (2.0 x A)
PAIM2= PAIM1
"DISC.NEGAT. ∴ RAICES COMPLEJAS"
PARE1 , PAIM1 , PARE2 , PAIM2
α

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------D I E C I N U E V E-------
C     SOLUCION DE ECUACIONES CUADRATICAS.
  100 FORMAT(3F11.5)
  101 FORMAT(///,2X,36HLOS COEFICIENTES DE LA ECUACION SON,,//)
  102 FORMAT(2X,3HA= ,F11.5,2X,3HB= ,F11.5,2X,3HC= ,F11.5,//)
  103 FORMAT(2X,52HEL DISCRIMINANTE ES POSITIVO,POR TANTO RAICES REALES,
     1//)
  104 FORMAT(5X,3HX1=,F11.5,10X,3HX2=,F11.5,///)
  105 FORMAT(2X,49HEL DISCRIMINANTE ES NULO,POR TANTO RAICES IGUALES,//)
  106 FORMAT(5X,3HX1=,F11.5,10X,3HX2=,F11.5,///)
  107 FORMAT(2X,55HEL DISCRIMINANTE ES NEGATIVO,POR TANTO RAICES COMPLEJ
     1AS,//)
  108 FORMAT(5X,3HX1=,F11.5,2H +,F11.5,4H IMA,10X,3HX2=,F11.5,2H -,F11.5
     1,4H IMA,///)
      LEE=2
      IMP=3
C     LEE LOS COEFICIENTES
  200 READ(LEE,100,END=240)A,B,C
C     IMPRIME LA ECUACION
      WRITE(IMP,101)
      WRITE(IMP,102)A,B,C
C     CALCULO DEL DISCRIMINANTE
      DIS=B**2-4.0*A*C
      IF(DIS.LE. 0.0)GO TO 210
C        RAICES REALES DIFERENTES
         RA1=(-B+SQRT(DIS))/(2.0*A)
         RA2=(-B-SQRT(DIS))/(2.0*A)
         WRITE(IMP,103)
         WRITE(IMP,104)RA1,RA2
         GO TO 230
  210 CONTINUE
         IF(DIS.NE.0.0)GO TO 220
C           RAICES REALES IGUALES
            RA1=-B/(2.0*A)
            RA2=RA1
            WRITE(IMP,105)
            WRITE(IMP,106)RA1,RA2
            GO TO 230
  220    CONTINUE
C           RAICES COMPLEJAS
            PARE1=-B/(2.0*A)
            PARE2=PARE1
            PAIM1=SQRT(-DIS)/(2.0*A)
            PAIM2=PAIM1
            WRITE(IMP,107)
            WRITE(IMP,108)PARE1,PAIM1,PARE2,PAIM2
            GO TO 230
C        ENDIF
C     ENDIF
  230 CONTINUE
      GO TO 200
  240 CONTINUE
      CALL EXIT
      END
// XEQ
   1.0        2.0        3.0    1.
   5.0       10.0        5.0    1.
   1.0       10.0       27.0    1.
```

3.0 20.0 1.0 1.

/*

## RESULTADOS

LOS COEFICIENTES DE LA ECUACION SON.

A= 1.00000 B= 2.00000 C= 3.00000

EL DISCRIMINANTE ES NEGATIVO.POR TANTO RAICES COMPLEJAS

X1= -1.00000 + 1.41421 IMA X2= -1.00000 - 1.41421 IMA

LOS COEFICIENTES DE LA ECUACION SON.

A= 5.00000 B= 10.00000 C= 5.00000

EL DISCRIMINANTE ES NULO.POR TANTO RAICES IGUALES

X1= -1.00000 X2= -1.00000

LOS COEFICIENTES DE LA ECUACION SON.

A= 1.00000 B= 10.00000 C= 27.00000

EL DISCRIMINANTE ES NEGATIVO.POR TANTO RAICES COMPLEJAS

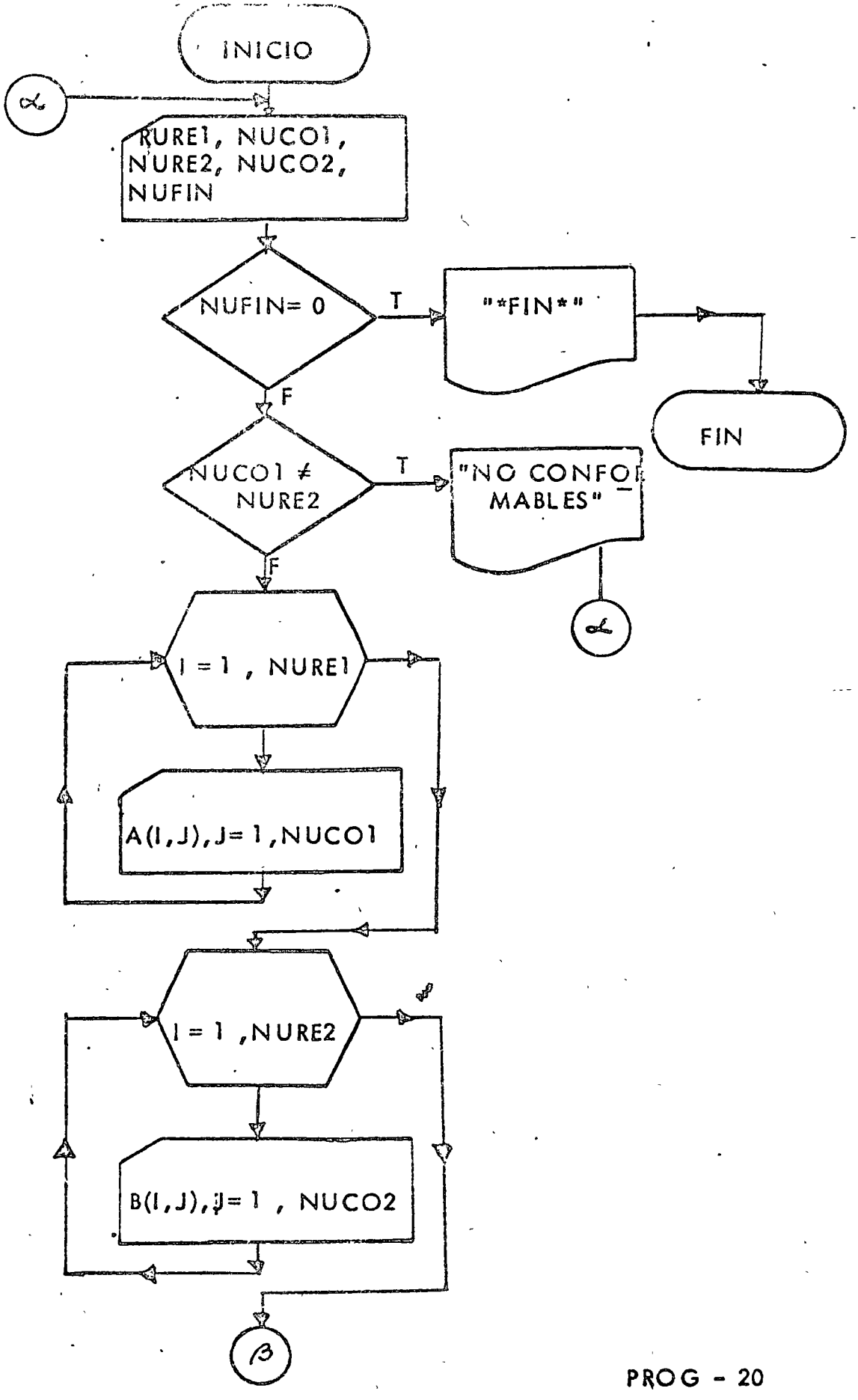
X1= -5.00000 + 1.41421 IMA X2= -5.00000 - 1.41421 IMA

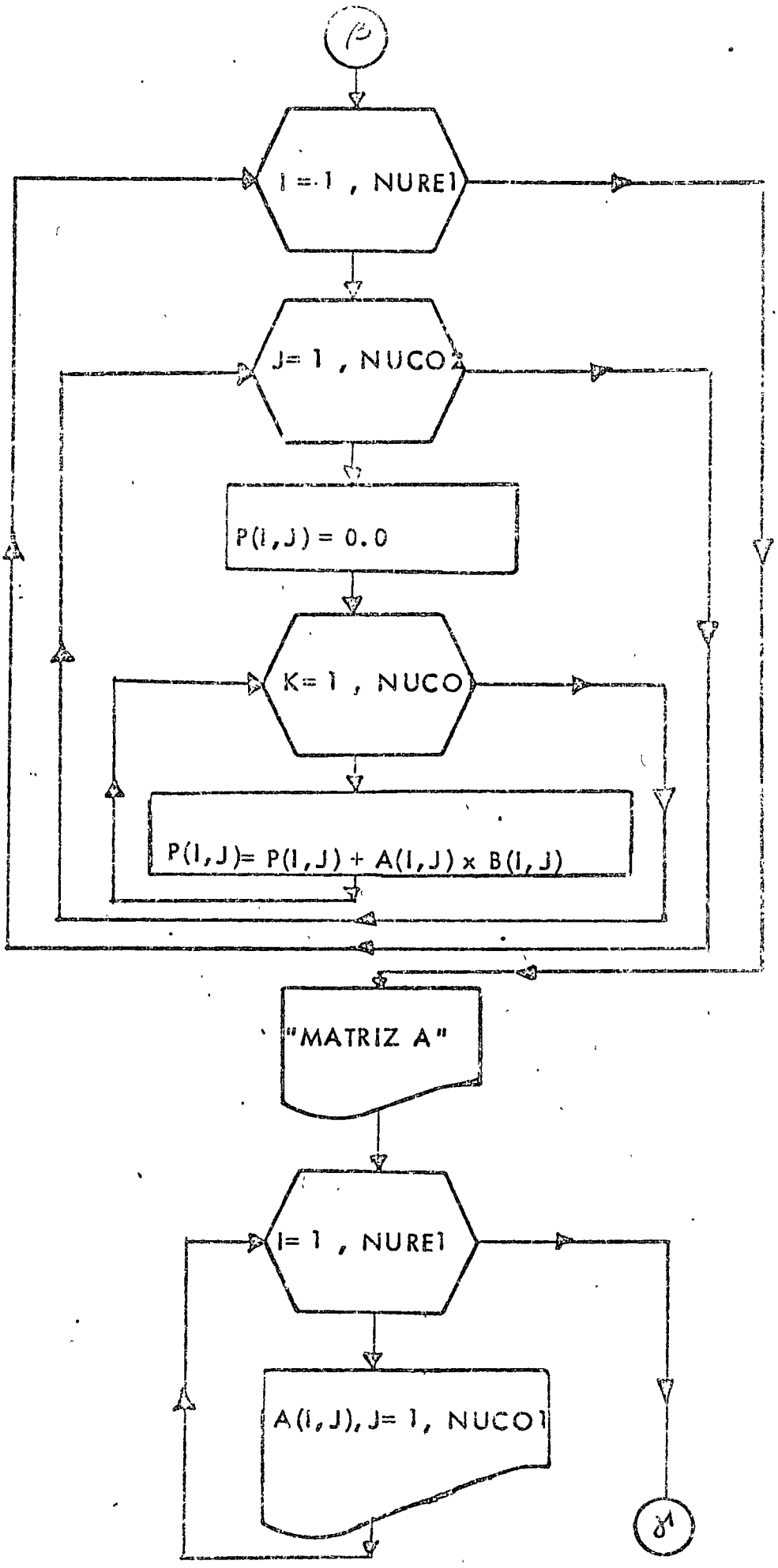
LOS COEFICIENTES DE LA ECUACION SON.

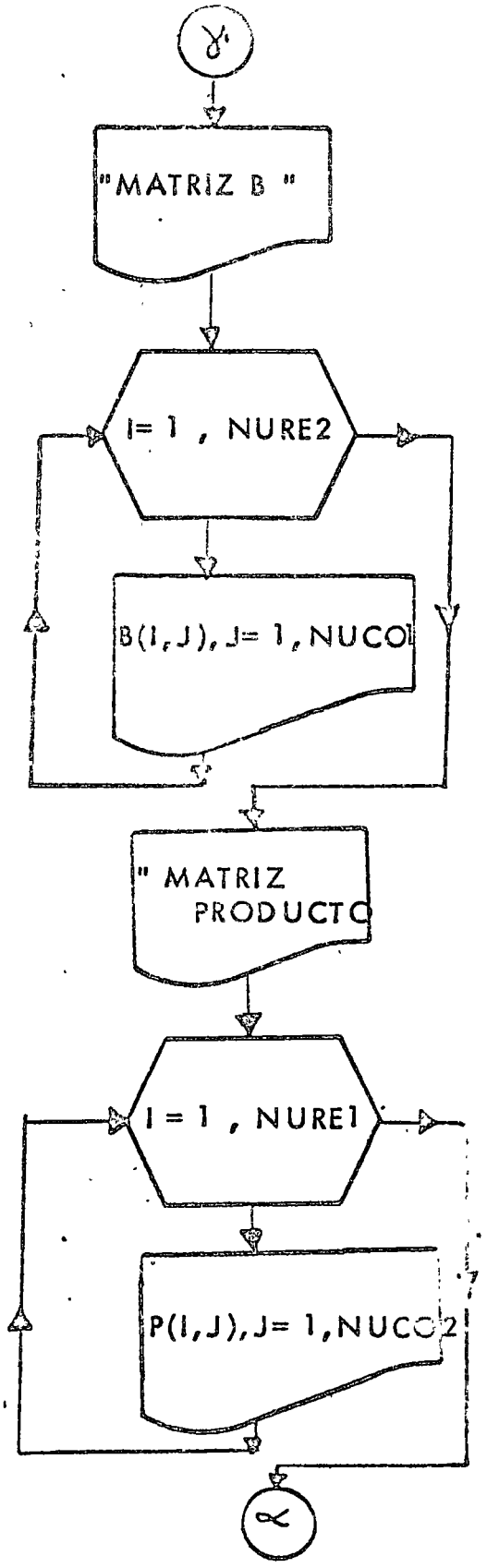
A= 3.00000 B= 20.00000 C= 1.00000

EL DISCRIMINANTE ES POSITIVO.POR TANTO RAICES REALES

X1= -0.05038 X2= -6.61629


INICIO
α
RURE1, NUCO1,
NURE2, NUCO2,
NUFIN
NUFIN= 0
T
"*FIN*"
FIN
F
NUCO1 ≠
NURE2
T
"NO CONFOR
MABLES"
α
F
I = 1 , NURE1
A(I,J),J= 1,NUCO1
I = 1 ,NURE2
B(I,J),J= 1 , NUCO2
β


P
I =1 , NURE1
J= 1 , NUCO2
P(I,J) = 0.0
K= 1 , NUCO
P(I,J)= P(I,J) + A(I,J) x B(I,J)
"MATRIZ A"
I= 1 , NURE1
A(I,J), J= 1, NUCO1
8


"MATRIZ B "
I= 1 , NURE2
B(I,J), J= 1, NUCO
" MATRIZ PRODUCTO
I = 1 , NURE1
P(I,J), J= 1, NUCO2

```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C--------------V E I N T E --------------
C      EL PROGRAMA REALIZA EL PRODUCTO DE DOS MATRICES DE 10 X 10 MAXIMO.
C      UNA ES LA MATRIZ A(NURE1,NUCO1).
C      LA OTRA ES LA MATRIZ B(NURE2,NUCO2).
C      SE RESERVAN LUGARES EN LA MEMORIA PARA LAS MATRICES QUE SE VAN A
C      MULTIPLICAR Y PARA LA MATRIZ PRODUCTO.
       DIMENSION A(10,10),B(10,10),P(10,10)
C      FILES
            LEE=2
            IMP=3
C      FORMATOS
  100   FORMAT(5I2)
  101   FORMAT(10F8.3)
  102   FORMAT(///,5X,9HMATRIZ A:,//)
  103   FORMAT(5X,10(F8.3,2X),/)
  104   FORMAT(///,5X,9HMATRIZ B:,//)
  105   FORMAT(///,5X,22HLA MATRIZ PRODUCTO ES:,//)
  106   FORMAT(5X,10E15.7,/)
  107   FORMAT(///,5X,76HEL PRODUCTO NO SE PUEDE LLEVAR A CABO YA QUE LAS
     1MATRICES NO SON CONFORMABLES.,///)
  108   FORMAT(///)
  109   FORMAT(53X,11H**********)
  110   FORMAT(53X,11H*   FIN   *)
C      LECTURA DE LOS NUMEROS DE RENGLONES Y DE COLUMNAS DE CADA MATRIZ
C      Y DEL DETECTOR NUFIN.
  199 READ(LEE,100)NURE1,NUCO1,NURE2,NUCO2,NUFIN
C      ANALISIS DEL VALOR DE NUFIN. SI VALE CERO EL PROGRAMA NO SE LLEVA
C      A CABO, DE LO CONTRARIO SI.
       IF(NUFIN.EQ.0)GO TO 1000
C        SE VE SI LAS MATRICES SON CONFORMABLES.
         IF(NUCO1.NE.NURE2)GO TO 900
C          LECTURA POR RENGLONES DE LA MATRIZ A.
           DO 200 I=1,NURE1
             READ(LEE,101)(A(I,J),J=1,NUCO1)
  200      CONTINUE
C          LECTURA POR RENGLONES DE LA MATRIZ B.
           DO 201 I=1,NURE2
             READ(LEE,101)(B(I,J),J=1,NUCO2)
  201      CONTINUE
C          SE REALIZA EL PRODUCTO
           DO 204 I=1,NURE1
             DO 203 J=1,NUCO2
               P(I,J)=0.0
               DO 202 K=1,NUCO1
                 P(I,J)=P(I,J)+A(I,K)*B(K,J)
  202          CONTINUE
  203        CONTINUE
  204      CONTINUE
C          IMPRESION DE LA MATRIZ A POR RENGLONES
           WRITE(IMP,102)
           DO 205 I=1,NURE1
             WRITE(IMP,103)(A(I,J),J=1,NUCO1)
  205      CONTINUE
C          IMPRESION DE LA MATRIZ B POR RENGLONES.
           WRITE(IMP,104)
           DO 206 I=1,NURE2
             WRITE(IMP,103)(B(I,J),J=1,NUCO1)
  206      CONTINUE
```

```
C           IMPRESION DE LA MATRIZ P POR RENGLONES.
            WRITE(IMP,105)
            DO 207 I=1,NUREI
              WRITE(IMP,106)(P(I,J),J=1,NUCO2)
  207       CONTINUE
            GO TO 199
  900     CONTINUE
          WRITE(IMP,107)
          GO TO 199
 1000 CONTINUE
      WRITE(IMP,108)
      WRITE(IMP,109)
      WRITE(IMP,110)
      WRITE(IMP,109)
      CALL EXIT
      END
// XEQ
 2 2 2 210
  15.54  -42.07
  -1.22    0.0
1950.75  -12.0
   0.001   5.0
 2 2 2 210
   0.2    98.75
 -12.5    32.52
1000.01    0.0
  -1.52   15.51
 2 3 2 210
 2 2 2 200
/*
```

# RESULTADOS

MATRIZ A:

```
 15.540   -42.070
  0.000    -1.220
```

MATRIZ B:

```
1950.750   -12.000
   0.001     5.000
```

LA MATRIZ PRODUCTO ES:

```
 .3031461E+05   -.3968300E+03
-.1220000E-02   -.6100000E+01
```

MATRIZ A:

```
  0.200    98.750
  0.000   -12.500
```

MATRIZ B:

```
1000.010     0.000
  -1.520    15.510
```

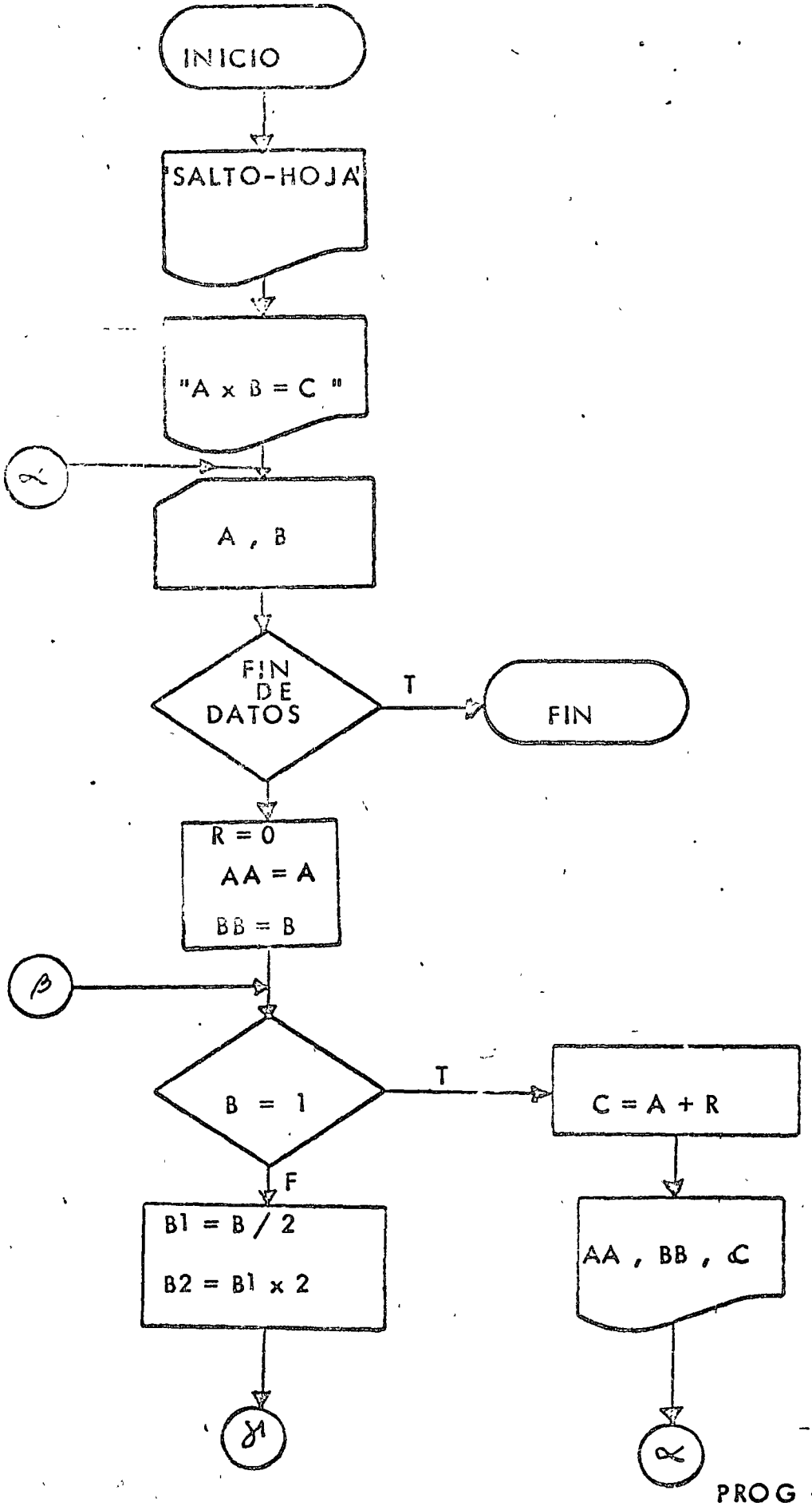
LA MATRIZ PRODUCTO ES:

```
 .4990200E+02    .1531613E+04
 .1900000E+02   -.1938750E+03
```

EL PRODUCTO NO SE PUEDE LLEVAR A CABO YA QUE LASMATRICES NO SON CONFORMABLES

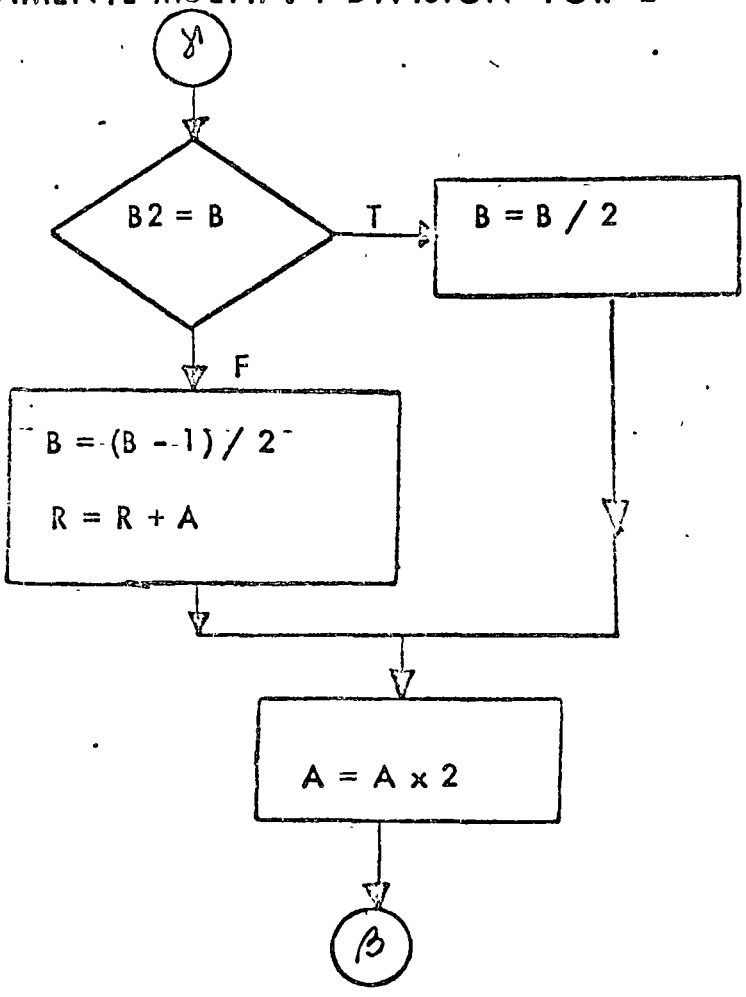
```
**********
*  FIN   *
**********
```

"MULTIPLICACION DE DOS NUMEROS UTILIZANDO EXCLUSIVAMENTE MULTIPLICACION Y DIVISION POR 2"




PROG - 21

" MULTIPLICACION DE DOS NUMEROS UTILIZANDO EXCLUSIVAMENTE MULTIP. Y DIVISION POR 2 "



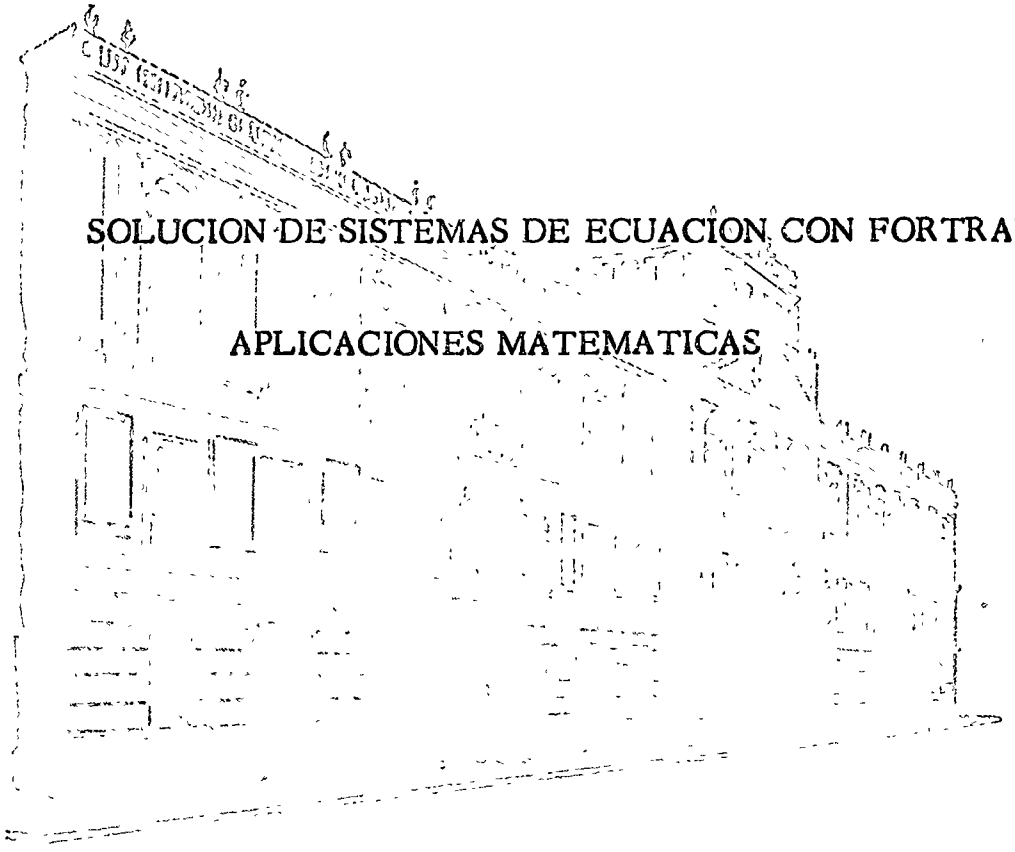
```
// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-------------V E I N T I U N O --------
C      MULTIPLICACION DE DOS NUMEROS
C           UTILIZANDO SOLO MULTIPLICACIONES Y
C           DIVISIONES POR 2
       INTEGER A,B,C,R,AA,BB
       WRITE (3,101)
  101  FORMAT (1H1)
       WRITE (3,102)
  102  FORMAT (9X,1HA,3X,1HX,4X,1HB,4X,1H=,4X,1HC)
  200  READ (2,100,END=260)A,B
  100  FORMAT (2I3)
       R=0
       AA=A
       BB=B
  210  IF(B.EQ.1) GO TO 240
            B1=B/2
            B2=B1*2
            IF(B2.EQ.B) GO TO 220
C             ES IMPAR
              B=(B-1)/2
              R=R+A
              GO TO 230
  220       CONTINUE
C             ES PAR
            B=B/2
  230       CONTINUE
            A=A*2
            GO TO 210
  240  CONTINUE
       C=A+R
       WRITE (3,103) AA,BB,C
  103  FORMAT (3I10)
       GO TO 200
  260  CALL EXIT
       END
// XEQ
 60 80
 19 17
 68 35
 40 11
 77 99
/*
```

## RESULTADOS

| A | X | B | = | C |
|---|---|---|---|---|
| 60 | | 80 | | 4800 |
| 19 | | 17 | | 323 |
| 68 | | 35 | | 2380 |
| 40 | | 11 | | 440 |
| 77 | | 99 | | 7623 |

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA



# SOLUCION DE SISTEMAS DE ECUACION CON FORTRAN

## APLICACIONES MATEMATICAS

ING. ARMANDO TORRES FENTANES

Febrero-Marzo, 1977

# 2. ALGEBRA MATRICIAL

## 2.1 Introducción

Una matriz es un arreglo rectangular de elementos distribuidos en "m" renglones y "n" columnas, si a la matriz se le denota por la letra $\underline{A}$, entonces al elemento del "i-ésimo" renglón y de la "j-ésima" columna se le representará por el símbolo $a_{ij}$ Generalmente una matriz se representa mediante paréntesis cuadrados como se muestra a continuación:

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad (2.1)$$

Los elementos que componen una matriz pueden ser de diversos tipos: números reales, números complejos, funciones en el dominio del tiempo, etc..

Al ser una matriz un arreglo ordenado de elementos, permite que al aplicar cierta metodología a dicho arreglo se obtenga una serie de resultados que responden a las interrogantes por las que se originó el arreglo; entre algunos de los procesos en los que se utilizan arreglos matriciales se tiene: jerarquización de actividades, almacenamiento de datos, inventarios, representación de sistemas dinámicos, sistemas de ecuaciones, etc..

Existen ciertas distribuciones típicas de los elementos de una matriz y de acuerdo a ellas se clasifica a las matrices en diferentes tipos, entre los que se tienen:

Matriz Cuadrada

Es una matriz en la que el número de renglones es igual al número de columnas, es decir, m=n. Por ejemplo:

$$\underline{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad (2.2)$$

Matriz Nula

Es una matriz de orden cualquiera, en la que todos los elementos son nulos; por ejemplo:

$$\underline{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad (2.3)$$

Se acostumbra denotarla por el símbolo $\underline{0}$.

Matriz Identidad

Es una matriz cuadrada en la cual los elementos de la diagonal principal son unitarios y el resto son nulos, es decir:

$$\delta_{ij} = \begin{cases} 0 & , i \neq j \\ 1 & , i = j \end{cases}$$

Se suele denotarla como $\underline{I}_n$ donde "n" indica el orden de la matriz y al símbolo $\delta_{ij}$ se le conoce como delta de Kronecker. Por ejemplo:

$$\underline{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.4)$$

Matriz Diagonal

Es una matriz cuadrada en la que los elementos que no pertenecen a la diagonal principal son nulos, es decir:

$$a_{ij} = 0 \quad si \; i \neq j \qquad (2.5)$$

Un ejemplo de este tipo de matriz sería:

$$\underline{A} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & sent \end{bmatrix} \qquad (2.6)$$

Matriz Transpuesta

Es una matriz cuadrada que se obtiene a partir de una matriz dada $\underline{A}$ intercambiando renglones con columnas. Se le denota con el símbolo $\underline{A}^T$ y se cumple que:

$$a^T_{ij} = a_{ji} \qquad (2.7)$$

Matriz Simétrica

Es una matriz cuadrada $\underline{B}$ para la que se cumple:

$$\underline{B} = \underline{B}^T \qquad (2.8)$$

o equivalentemente:

$$b_{ij} = b_{ji} \qquad (2.9)$$

Entre las matrices se definen dos operaciones básicas:
-suma o resta de matrices,
-multiplicación matricial.

## 2.2 Suma Matricial

### 2.2.1 Objeto

Obtener la suma de dos matrices de igual orden, o sea:

$$\underline{C} = \underline{A} + \underline{B} \qquad (2.10)$$

### 2.2.2 Método

Para poder efectuar la suma de dos matrices $(\underline{A}+\underline{B})$ se requiere que sean conformables para la suma, lo cual implica que el orden de las dos matrices es igual. En otras palabras:
si $\underline{A}$ es de orden $(mxn)$
y $\underline{B}$ es de orden $(rxs)$
la suma $\underline{C}=\underline{A}+\underline{B}$ es posible solo si $m=r$ y $n=s$.

Los elementos de la matriz suma están dados por la siguiente relación:

$$c_{ij} = a_{ij} + b_{ij} \qquad (2.11)$$

El restar dos matrices equivale a cambiar el signo de todos los elementos de una de ellas y efectuar la suma, esto es:

$$\underline{W} = \underline{X} - \underline{Y} = \underline{X} + (-\underline{Y}) \qquad (2.12)$$

### 2.2.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE SUMAT(A,B,C,N,M), esta subrutina efectua la suma matricial, el programa principal lee e imprime resultados.

b) Descripción de las variables:

Para la subrutina SUMAT:

| | |
|---|---|
| N | cantidad de renglones de cada una de las matrices que se desea sumar. |
| M | cantidad de columnas de cada una de las matrices que se desea sumar. |
| A(I,J) | matriz sumando de orden NxM |
| B(I,J) | matriz sumando de orden NxM |
| C(I,J) | matriz suma |

Para el programa principal:

| | |
|---|---|
| N | cantidad de renglones de las matrices |
| M | cantidad de columnas de las matrices |
| A(I,J) | matriz sumando de orden NxM |
| B(I,J) | matriz sumando de orden NxM |
| C(I,J) | matriz suma |

c) Dimensiones:

La proposición DIMENSION deberá ser modificada tanto to en el programa principal como en la subrutina cuando:

$N > 10$ y/o $M > 10$

d) Formatos para los datos de entrada:

| SEC. TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (2I5) | N, M |
| 2 | (8F10.0) | A(I,J), se dan los elementos de la matriz renglón por renglón. Emplear tantas tarjetas como se requiera. |
| 3 | (8F10.0) | B(I,J), igual que para la matriz A. |
| . . . | | |
| otros paquetes de datos (opcional) | | |
| n | | TARJETA EN BLANCO, al finalizar toda la información. |

e) Diagrama de bloques

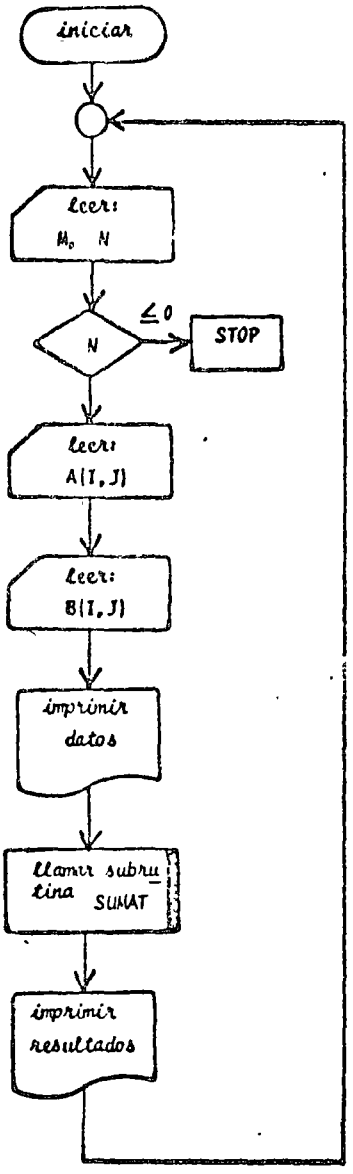



Fig. 2.1 Diagrama de bloques para el programa principal

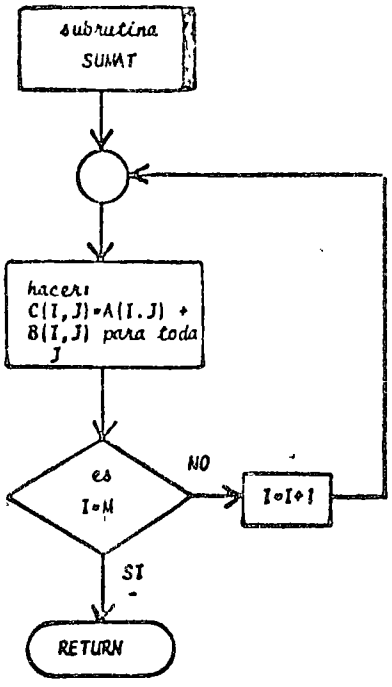



Fig.2.2 Diagrama de bloques para la subrutina SUMAT

6) Listado:

```
C     PROGRAMA PARA EFECTUAR LA SUMA DE DOS MATRICES
C     EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C     M=CANTIDAD DE RENGLONES DE LAS MATRICES
C     N=CANTIDAD DE COLUMNAS DE LAS MATRICES
C     A Y B=MATRICES QUE SERAN SUMADAS
C     C=MATRIZ SUMA
      DIMENSION A(10,10),B(10,10),C(10,10)
C     LECTURA DE DATOS
C
    1 READ(5,100) M,N
      IF(N) 2,2,3
    2 CALL EXIT
    3 DO 4 I=1,M
    4 READ(5,150) (A(I,J),J=1,N)
      DO 5 I=1,M
    5 READ(5,150) (B(I,J),J=1,N)
C
C     IMPRESION DE DATOS
C
      WRITE(6,200)
      DO 6 I=1,M
    6 WRITE(6,250) (A(I,J),J=1,N)
      WRITE(6,300)
      DO 7 I=1,M
    7 WRITE(6,250) (B(I,J),J=1,N)
C
C     LLAMADO DE SUBRUTINA PARA EFECTUAR LA SUMA
C
      CALL SUMAT(A,B,C,M,N)
C
C     IMPRESION DE RESULTADOS
C
      WRITE(6,350)
      DO 8 I=1,M
    8 WRITE(6,250) (C(I,J),J=1,N)
      GO TO 1
C
C     FORMATOS DE LECTURA E IMPRESION
C
  100 FORMAT(2I5)
  150 FORMAT(7F10.0)
  200 FORMAT(1H1,5(/),10X,'LAS MATRICES POR SUMAR SON',3(/),10X,'MATRIZ
     1A',/)
  250 FORMAT(/,3X,10(E10.3,2X))
  300 FORMAT(3(/),10X,'MATRIZ B',/)
  350 FORMAT(6(/),10X,'MATRIZ SUMA',/)
      END
```

*Fig. 2.3 Listado del programa principal*

```
      SUBROUTINE SUMAT(A,B,C,M,N)
C     SUBRUTINA PARA SUMAR MATRICES
C     SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     A Y B=MATRICES QUE SE DESEA SUMAR
C     M=CANTIDAD DE RENGLONES DE LAS MATRICES
C     N=CANTIDAD DE COLUMNAS DE LAS MATRICES
      DIMENSION A(10,10),B(10,10),C(10,10)
      DO 1 I=1,M
      DO 1 J=1,N
    1 C(I,J)=A(I,J) + B(I,J)
      RETURN
      END
```

*Fig. 2.4 Listado de la subrutina SUMAT*

2.2.4 Ejemplo

En una tienda de artículos eléctricos se venden resistencias eléctricas de 1/4, 1/2 y 1 Watt de potencia en seis diferentes valores resistivos.

Si las existencias un viernes por la tarde son:

| | 1/4 | 1/2 | 1 |
|---|---|---|---|
| 100 Ω | 200 | 380 | 275 |
| 150 Ω | 400 | 250 | 275 |
| 1.0 K | 500 | 175 | 325 |
| 1.5 K | 800 | 225 | 150 |
| 10.0 K | 600 | 380 | 180 |
| 15.0 K | 550 | 250 | 220 |

y el sábado se recibe una remesa con las siguientes características:

| | 1/4 | 1/2 | 1 |
|---|---|---|---|
| 100 Ω | 80 | 90 | 50 |
| 150 Ω | 90 | 100 | 55 |
| 1.0 K | 75 | 90 | 60 |
| 1.5 K | 65 | 95 | 55 |
| 10.0 K | 80 | 100 | 60 |
| 15.0 K | 75 | 110 | 60 |

Determine las resistencias que tendrá en inventario el establecimiento el lunes por la mañana dado que ni el sábado ni el domingo hubo ventas.

* SOLUCION

TABLA 2.1 Datos para el problema del ejemplo 2.2.4

N=6

M=3

$$\underline{A} = \begin{bmatrix} 200 & 380 & 275 \\ 400 & 250 & 275 \\ 500 & 175 & 325 \\ 800 & 225 & 150 \\ 600 & 380 & 180 \\ 550 & 250 & 220 \end{bmatrix}$$

$$\underline{B} = \begin{bmatrix} 80 & 90 & 50 \\ 90 & 100 & 55 \\ 75 & 90 & 60 \\ 65 & 95 & 55 \\ 80 & 100 & 60 \\ 75 & 110 & 60 \end{bmatrix}$$

TABLA 2.2 Resultados del problema del ejemplo 2.2.4

LAS MATRICES POR SUMAR SON

MATRIZ A

| | | |
|---|---|---|
| .200E+03 | .340E+03 | .275E+03 |
| .400E+03 | .250E+03 | .275E+03 |
| .500E+03 | .175E+03 | .325E+03 |
| .800E+03 | .225E+03 | .150E+03 |
| .600E+03 | .360E+03 | .180E+03 |
| .550E+03 | .250E+03 | .220E+03 |

MATRIZ B

| | | |
|---|---|---|
| .800E+02 | .900E+02 | .500E+02 |
| .900E+02 | .100E+03 | .550E+02 |
| .750E+02 | .900E+02 | .600E+02 |
| .650E+02 | .950E+02 | .550E+02 |
| .800E+02 | .100E+03 | .600E+02 |
| .750E+02 | .110E+03 | .600E+02 |

MATRIZ SUMA

| | | |
|---|---|---|
| .280E+03 | .470E+03 | .325E+03 |
| .490E+03 | .350E+03 | .330E+03 |
| .575E+03 | .265E+03 | .385E+03 |
| .865E+03 | .320E+03 | .205E+03 |
| .680E+03 | .480E+03 | .240E+03 |
| .625E+03 | .360E+03 | .280E+03 |

## 2.3 Multiplicación Matricial

### 2.3.1 Objeto

Dadas dos matrices $\underline{A}$ y $\underline{B}$, obtener el producto matricial $\underline{C}$ de la forma:

$$\underline{C} = \underline{A} \times \underline{B} \qquad (2.13)$$

### 2.3.2 Método

Para efectuar el producto entre dos matrices ($\underline{A}x\underline{B}$) se requiere que las matrices sean conformables para la multiplicación, lo que equivale a que el número de columnas de la matriz premultiplicadora ($\underline{A}$) sea igual al número de renglones de la postmultiplicadora ($\underline{B}$), es decir:
si $\underline{A}$ es de orden (mxn)
y $\underline{B}$ es de orden (rxs)
el producto matricial $\underline{AB}$ será posible solo si n=r y el orden de la matriz producto sera (mxs).

Si la matriz $\underline{C}$ representa la matriz resultante del producto matricial $\underline{AB}$, entonces el elemento $c_{ij}$ está dado por:

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj} \quad , \quad \begin{matrix} i=1,\ldots,m \\ j=1,\ldots,s \end{matrix} \qquad (2.14)$$

Es importante hacer notar que el producto matricial no es conmutativo, esto es:

$$\underline{A} \times \underline{B} \neq \underline{B} \times \underline{A}$$

### 2.3.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE MULTMA(A,B,N,M,L,X), esta subrutina efectua el producto matricial $\underline{A}x\underline{B}$. El programa principal se emplea para la lectura de datos e impresión de resultados.

b) Descripción de las variables:

Para la subrutina MULTMA:

A(I,J) matriz premultiplicadora de orden NxM

B(I,J) matriz postmultiplicadora de orden MxL
X(I,J) matriz producto de orden NxL
Para el programa principal:
A(I,J) matriz premultiplicadora de orden NxM
B(I,J) matriz postmultiplicadora de orden MxL
X(I,J) matriz producto de orden NxL

c) Dimensiones:

La proposición DIMENSION deberá ser modificada tanto en el programa principal como en la subrutina cuando:
$N > 10$ y/o $M > 10$ y/o $L > 10$

d) Formatos para los datos de entrada:

| SEC. TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (3I5) | N, M, L |
| 2 | (8F10.0) | A(I,J), los elementos de la matriz se dan renglón por renglón. Emplear la cantidad de tarjetas que sea necesaria. |
| 3 | (8F10.0) | B(I,J), igual que en el caso anterior. |
| . | | |
| . | | |
| . | | |
| otros paquetes de datos (opcional) | | |
| n | | TARJETA EN BLANCO, al finalizar toda la información |

e) Diagrama de bloques:

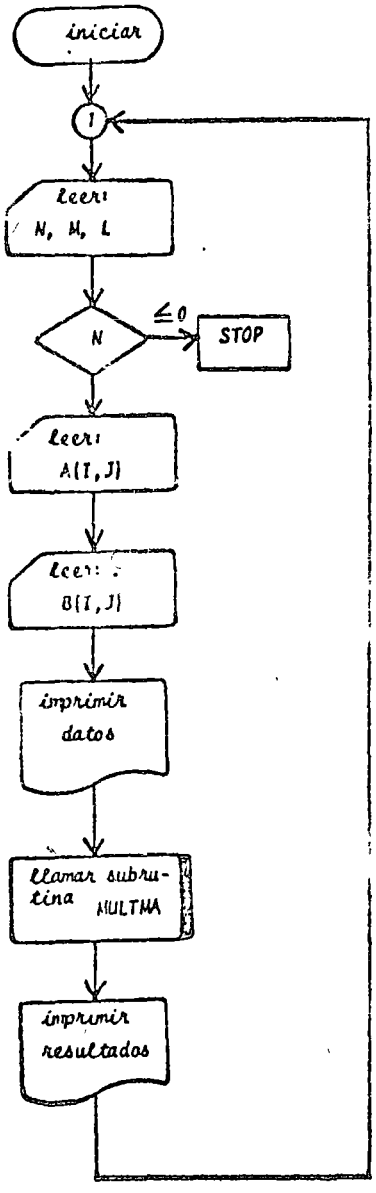



Fig. 2.5 Diagrama de bloques para el programa principal

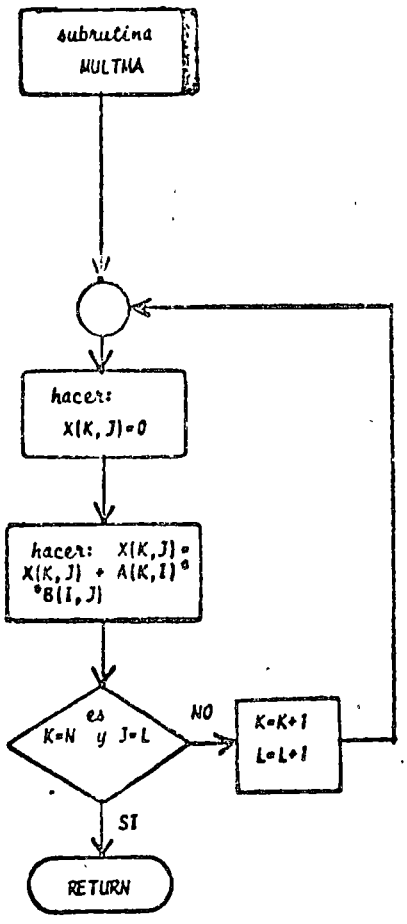



Fig. 2.6 Diagrama de bloques para la subrutina MULTMA

6) *Listado:*

```
C     PROGRAMA PARA EFECTUAR PRODUCTOS MATRICIALES
C     SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     A=MATRIZ PREMULTIPLICADORA DE ORDEN (NXM)
C     B=MATRIZ POSTMULTIPLICADORA DE ORDEN (MXL)
C     X=MATRIZ PRODUCTO DE ORDEN (NXL)

      DIMENSION A(10,10),B(10,10),X(10,10)
      IR=5
      IW=6
C     LECTURA DE DATOS
    1 READ(IR,10) N,M,L
      IF(N) 2,2,3
    2 CALL EXIT
    3 DO 4 I=1,N
    4 READ(IR,11) (A(I,J),J=1,M)
      DO 5 I=1,M
    5 READ(IR,11) (B(I,J),J=1,L)
C     IMPRESION DE DATOS
      WRITE(IW,12)
      DO 6 I=1,N
    6 WRITE(IW,13) (A(I,J),J=1,M)
      WRITE(IW,14)
      DO 7 I=1,M
    7 WRITE(IW,13) (B(I,J),J=1,L)
      WRITE(IW,15)
C     LLAMADO DE SUBRUTINA PARA EFECTUAR PRODUCTO MATRICIAL
      CALL MULTMA(A,B,N,M,L,X)
C     IMPRESION DE RESULTADOS
      DO 8 I=1,N
    8 WRITE(IW,13) (X(I,J),J=1,L)
      GO TO 1
C     FORMATOS DE LECTURA E IMPRESION
   10 FORMAT (3I5)
   11 FORMAT(8F10.0)
   12  FORMAT(4(/),5X,'MATRIZ A',/)
   13 FORMAT(/,2X,10(E10.3,1X))
   14 FORMAT(4(/),5X,'MATRIZ B',/)
   15 FORMAT(4(/),5X,'MATRIZ PRODUCTO',/)
      END
```

*Fig.2.7 Listado del programa principal*

```
      SUBROUTINE MULTMA(A,B,N,M,L,X)
C
C     SUBRUTINA PARA MULTIPLICAR DOS MATRICES
C     EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C     A=MATRIZ PREMULTIPLICADORA DE ORDEN (N,M)
C     B=MATRIZ POSTMULTIPLICADORA DE ORDEN (M,L)
C     X=MATRIZ PRODUCTO
C
      DIMENSION A(10,10),B(10,10),X(10,10)
      DO 1 J=1,L
      DO 1 K=1,N
      X(K,J)=0.0
      DO 1 I=1,M
    1 X(K,J)=X(K,J) + A(K,I)*B(I,J)
      RETURN
      END
```

*Fig. 2.8 Listado de la subrutina MULTMA*

2.3.4 Ejemplo

Cuatro componentes de un automóvil requieren como materia prima de hule, aluminio y acero. Las unidades que se requieren de cada material para formar una unidad de cada componente del automóvil se proporcionan a continuación:

| | hule | aluminio | acero |
|---|---|---|---|
| comp. 1 | 8 | 5 | 3 |
| comp. 2 | 3 | 4 | 5 |
| comp. 3 | 20 | 2 | 4 |
| comp. 4 | 1 | 8 | 10 |

si los costos unitarios de cada material son:

| | $ |
|---|---|
| hule | 25.00 |
| aluminio | 30.00 |
| acero | 40.00 |

Determine el costo total de cada componente debido a la materia prima de que está compuesto.

*SOLUCION

TABLA 2.3 Datos para el problema del ejemplo 2.3.4

N=4

M=3

L=1

$$\underline{A} = \begin{bmatrix} 8 & 5 & 3 \\ 3 & 4 & 5 \\ 20 & 2 & 4 \\ 1 & 8 & 10 \end{bmatrix}$$

$$\underline{B} = \begin{bmatrix} 25 \\ 30 \\ 40 \end{bmatrix}$$

TABLA 2.4 Resultados del problema del ejemplo 2.3.4

MATRIZ A

| | | |
|---|---|---|
| .800E+01 | .500E+01 | .100E+01 |
| .300E+01 | .400E+01 | .500E+01 |
| .200E+02 | .200E+01 | .400E+01 |
| .100E+01 | .600E+01 | .100E+02 |

MATRIZ B

.250E+02
.300E+02
.400E+02

MATRIZ PRODUCTO

.470E+03
.395E+03
.720E+03
.665E+03

## 2.4 Inversión de Matrices

### 2.4.1 Objeto

Dada una matriz cuadrada $\underline{A}$ obtener su matriz inversa $\underline{A}^{-1}$.

### 2.4.2 Método

La matriz inversa de una matriz cuadrada $\underline{A}$ es otra matriz cuadrada que se representa por $\underline{A}^{-1}$ y que cumple la siguiente propiedad si la matriz $\underline{A}$ es de orden $(n \times n)$:

$$\underline{A}\,\underline{A}^{-1} = \underline{I}_n = \underline{A}^{-1}\,\underline{A} \qquad (2.15)$$

Se define a la matriz inversa como:

$$\underline{A}^{-1} = \frac{\underline{A}^{+}}{|\underline{A}|} \qquad (2.16)$$

donde $\underline{A}^{+}$ se conoce como la matriz adjunta de la matriz $\underline{A}$ y $|\underline{A}|$ representa el determinante de la matriz $\underline{A}$.

De la ecuación (2.16) se infiere que para que exista la inversa de una matriz se requiere que $|\underline{A}| \neq 0$, es decir, que la matriz sea no singular.

Para la obtención numérica de la matriz inversa es necesario acudir al método de Gauss-Jordan modificado. Esto se hace debido a que para obtener $\underline{A}^{-1}$ en una computadora digital mediante la ecuación (2.16) se requiere una gran cantidad de operaciones y consecuentemente de tiempo. Para obtener la inversa de una matriz (10x10) se requieren más de 340 millones de operaciones con el método directo.

El método de Gauss-Jordan es un método de eliminación sistemática mediante el cual se transforma la matriz original $\underline{A}$ en una matriz identidad $\underline{I}_n$ y al mismo tiempo esta última se transforma en la matriz inversa $\underline{A}^{-1}$, es decir, partiendo del arreglo:

$$\left[\underline{A}_n \;\vdots\; \underline{I}_n\right] \qquad (2.17)$$

y aplicando algunas de las siguientes transformaciones al arreglo (2.17):
-intercambio de renglones,
-multiplicación de un renglón por un escalar $\lambda \neq 0$,
-suma de equimúltiplos de un renglón a otro renglón.
se llega al siguiente arreglo:

$$\left[\underline{I}_n \;\vdots\; \underline{A}_n^{-1}\right] \qquad (2.18)$$

El método parte de la suposición de que $\underline{A}$ es una matriz no singular, lo cual implica que sus columnas son vectores linealmente independientes, en caso de no serlo el método lo puede detectar; en dicha situación se presenta que todos los elementos de un renglón de la matriz $\underline{A}$ o de sus matrices transformadas, son nulos.

A fin de minimizar los errores de redondeo, la eliminación de elementos se efectua pivoteando sobre los mayores elementos que quedan en la matriz $\underline{A}$ o en las matrices obtenidas a partir de esta última por transformación; debe tenerse cuidado de no emplear como pivotes elementos de renglones que ya hayan sido utilizados como pivotes.

2.4.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE MATINV(A,N,EPS,DET), obtiene la matriz inversa de la matriz $\underline{A}$. El programa principal se emplea para la lectura de datos e impresión de resultados.

b) Descripción de las variables:

Para la subrutina MATINV:

| | |
|---|---|
| A(I,J) | matriz de la que se buscará la inversa, durante el proceso se convierte en la matriz inversa. |
| N | orden de la matriz $\underline{A}$ |
| EPS | criterio para determinar si el determinante de la matriz es nulo |
| DET | parámetro que indica si el determinante de la matriz es nulo |
| C(I,J) | matriz identidad que se emplea para obtener la matriz inversa |
| MVR(I) y MVC(I) | contadores que indican cuáles renglones y cuáles columnas de la matriz $\underline{A}$ ya fueron empleados como pivotes |

| | |
|---|---|
| RAMAX | mayor elemento de la matriz A o de sus transformaciones que se emplea como elemento pivote |
| TEMP | variable de localización temporal |

Para el programa principal:

| | |
|---|---|
| A(I,J) | matriz de la que se busca la inversa, durante el proceso se convierte en la matriz inversa |
| N | orden de la matriz A |
| EPS | criterio para determinar si el determinante de la matriz A es nulo. |
| DET | variable que indica si el determinante de A es o no nulo |

c) Dimensiones:

La proposición DIMENSION del programa principal y de la subrutina deberá ser modificada cuando:

$N > 10$

d) Formatos para los datos de entrada:

| SEC.TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (I5) | N |
| 2 | (8F10.0) | A(I,J), se proporcionan los elementos de la matriz renglón por renglón. Emplear tantas tarjetas como se requieran. |
| . | | |
| . | | |
| . | | |
| otros paquetes de datos (opcional) | | |
| n | | TARJETA EN BLANCO, al finalizar toda la información. |

e) Diagrama de bloques:

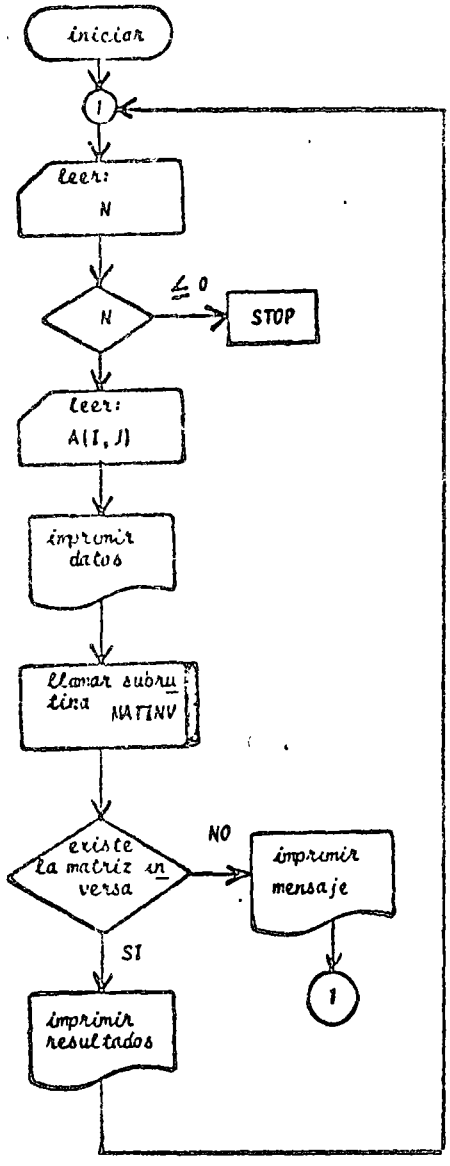



Fig. 2.9 Diagrama de bloques del programa principal

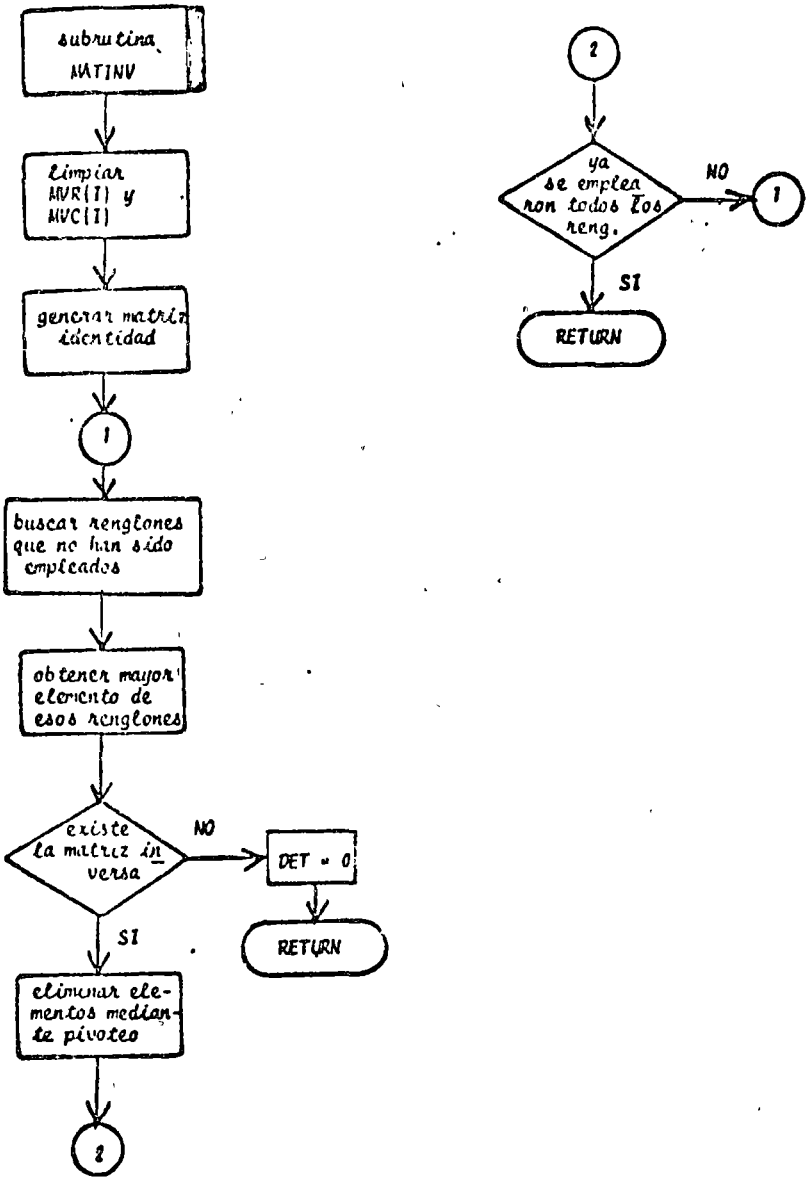



Fig. 2.10 Diagrama de bloques de la subrutina MATINV

6) *Listado:*

```
C     PROGRAMA PARA INVERTIR MATRICES POR EL METODO DE GAUSS-JORDAN
C     SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     N=ORDEN DE LA MATRIZ A
C     A=MATRIZ DE LA QUE SE BUSCA SU INVERSA
C     EPS=CRITERIO PARA DETERMINAR SI EXISTE O NO LA INVERSA DE LA MATRIZ
C     DET=PARAMETRO QUE INDICA SI EXISTE O NO LA INVERSA DE LA MATRIZ
      DIMENSION A(10,10),C(10,10)
      IR=5
      IW=6
      EPS=0.000001
C     LECTURA DE DATOS
    1 READ(IR,19) N
      IF(N) 2,2,3
    2 CALL EXIT
    3 DO 4 I=1,N
    4 READ(IR,20) (A(I,J),J=1,N)
C     IMPRESION DE DATOS
      WRITE(IW,21)
      DO 5 I=1,N
    5 WRITE(IW,22) (A(I,J),J=1,N)
C     LLAMADO DE SUBRUTINA PARA OBTENER LA MATRIZ INVERSA
      CALL MATINV(A,N,EPS,DET)
      IF(DET.GT.EPS) GO TO 7
      WRITE(IW,23)
      GO TO 1
    7 WRITE(IW,24)
      DO 8 I=1,N
    8 WRITE(IW,22) (A(I,J),J=1,N)
      GO TO 1
C     FORMATOS DE LECTURA E IMPRESION
   19 FORMAT(I5)
   20 FORMAT(8F10.0)
   21 FORMAT(4(/),5X,'MATRIZ A',//)
   22 FORMAT(/,2X,10(E10.3,1X))
   23 FORMAT(4(/),5X,'NO EXISTE LA MATRIZ INVERSA')
   24 FORMAT(4(/),5X,'INVERSA DE LA MATRIZ A')
      END
```

*Fig. 2.11 Listado del programa principal*

```
      SUBROUTINE MATINV(A,N,EPS,DET)
C
C     SUBRUTINA PARA OBTENER LA INVERSA DE UNA MATRIZ
C     EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C     A=MATRIZ A LA QUE SE BUSCARA SU INVERSA Y QUE DURANTE EL PROCESO
C       SE CONVIERTE EN LA MATRIZ INVERSA
C     N=ORDEN DE LA MATRIZ
C     EPS=CRITERIO PARA DETERMINAR SI EL DETERMINANTE DE LA MATRIZ ES
C       NULO
C     DET=VALOR ABSOLUTO DEL DETERMINANTE DE LA MATRIZ
C     C=MATRIZ IDENTIDAD QUE SE UTILIZA PARA OBTENER LA MATRIZ INVERSA
C       POR EL METODO DE GAUSS-JORDAN MODIFICADO
C     MVR Y MVC=CONTADORES QUE INDICAN CUALES RENGLONES Y COLUMNAS YA
C       FUERON UTILIZADOS COMO PIVOTES
C
      DIMENSION A(10,10),C(10,10),MVR(10),MVC(10)
C
C     OBTENCION DE LA MATRIZ IDENTIDAD Y ACTUALIZACION DE VALORES PARA
C     INICIAR EL PROCESO
C
      DO 1 I=1,N
      MVR(I)=0
    1 MVC(I)=0
      DO 4 I=1,N
      DO 3 J=1,N
      IF(I.EQ.J) GO TO 2
      C(I,J)=0.0
      GO TO 3
    2 C(I,J)=1.0
    3 CONTINUE
    4 CONTINUE
C
C     OBTENCION DE LA MATRIZ INVERSA
C
      DO 12 K=1,N
      RAMAX=0.0
      LC=0
      LR=0
      DO 6 I=1,N
      IF(MVR(I).EQ.1) GO TO 6
      DO 5 J=1,N
      IF(MVC(J).EQ.1) GO TO 5
      IF(ABS(RAMAX).GE.ABS(A(I,J))) GO TO 5
      RAMAX=A(I,J)
      LR=I
      LC=J
    5 CONTINUE
    6 CONTINUE
      DET=ABS(RAMAX)
      IF(DET.LE.EPS) GO TO 14
      IF(LR.EQ.LC) GO TO 8
      DO 7 I=1,N
      TEMP=A(LR,I)
      A(LR,I)=A(LC,I)
      A(LC,I)=TEMP
      TEMP=C(LR,I)
      C(LR,I)=C(LC,I)
    7 C(LC,I)=TEMP
    8 DO 9 I=1,N
      A(LC,I)=A(LC,I)/RAMAX
    9 C(LC,I)=C(LC,I)/RAMAX
      DO 11 I=1,N
      IF(I.EQ.LC) GO TO 11
      TEMP=A(I,LC)
      DO 10 J=1,N
      A(I,J)=A(I,J) - TEMP*A(LC,J)
   10 C(I,J)=C(I,J) - TEMP*C(LC,J)
   11 CONTINUE
      MVR(LC)=LC
      MVC(LC)=LC
   12 CONTINUE
      DO 13 I=1,N
      DO 13 J=1,N
   13 A(I,J)=C(I,J)
   14 RETURN
      END
```

Fig. 2.12 Listado de la subrutina MATINV

2.4.4 Ejemplo

Obtener la inversa de la matriz:

$$\underline{A} = \begin{bmatrix} 10 & 2 & 3 & -1 \\ 1 & -20 & -1 & 3 \\ 1 & 1 & -10 & 2 \\ 2 & -1 & -1 & 30 \end{bmatrix}$$

*SOLUCION

TABLA 2.5 Datos para el problema del ejemplo 2.4.4

N=4

$$\underline{A} = \begin{bmatrix} 10 & 2 & 3 & -1 \\ 1 & -20 & -1 & 3 \\ 1 & 1 & -10 & 2 \\ 2 & -1 & -1 & 30 \end{bmatrix}$$

TABLA 2.6 Resultados del problema del ejemplo 2.4.4

MATRIZ A

| | | | |
|---|---|---|---|
| .100E+02 | .200E+01 | .300E+01 | -.100E+01 |
| .100E+01 | -.200E+02 | -.100E+01 | .300E+01 |
| .100E+01 | .100E+01 | -.100E+02 | .200E+01 |
| .200E+01 | -.100E+01 | -.100E+01 | .300E+02 |

INVERSA DE LA MATRIZ A

| | | | |
|---|---|---|---|
| .961E-01 | .110E-01 | .277E-01 | .253E-03 |
| .347E-02 | -.490E-01 | .553E-02 | .471E-02 |
| .876E-02 | -.437E-02 | -.977E-01 | .724E-02 |
| -.600E-02 | -.253E-02 | -.492E-02 | .337E-01 |

## 2.5 Bibliografía


1. CARNAHAN B., LUTHER H., WILKES J.,"Applied Numerical Methods". New York: John Wiley and Sons Inc.,1969. pp.210-218,282-296.
2. HADLEY G., "Algebra Lineal". Bogotá: Fondo Educativo Interamericano, 1969. pp.60-131.
3. HAMMING Richard, "Numerical Methods for Scientists and Engineers". New York: Mc Graw Hill Book Co.,1962. pp.366-367.
4. JOHNSTON J., BALEY PRICE G., VAN VLECK F.,"Linear Equations and Matrices".Reading Mass.: Addison-Wesley Co., 1966. pp.95-157.
5. KAPLAN Lewis, "Calculus and Linear Algebra Vol.2". New York: John Wiley and Sons Inc., 1971. pp.718-803.
6. KUO S. Shan, "Computer Applications of Numerical Methods". Reading Mass.: Addison-Wesley Co., 1972. pp.176-179, 189-194.

# 3. SOLUCION DE SISTEMAS DE ECUACIONES LINEALES

## 3.1 Introducción

Por sistemas de ecuaciones lineales se entiende un grupo de ecuaciones que presentan la siguiente estructura:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \quad \vdots \\ &\vdots \quad \vdots \\ &\vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned}\right\} \qquad (3.1)$$

donde $a_{ij}$ y $b_i$ son constantes y las incógnitas del sistema son los valores $x_i$, donde $1 \leq i \leq n$.

Dichos sistemas se pueden representar en la forma:

$$\underline{A}\ \underline{X} = \underline{B} \qquad (3.2)$$

donde $\underline{A}$ se conoce como la matriz de coeficientes del sistema, $\underline{B}$ como vector de términos independientes y $\underline{X}$ como vector de incógnitas.

Si el vector de términos independientes es diferente de cero se habla de sistemas de ecuaciones no homogéneas y en caso contrario de sistemas homogéneos.

Antes de proceder a resolver un sistema de ecuaciones es necesario determinar si dicho sistema tiene solución y en caso de tenerla, cuántas posibles soluciones tiene. En base a lo anterior se tiene la siguiente clasificación:

- Sistema de ecuaciones lineales
  - no homogéneo
    - compatible
      - determinado
      - indeterminado
    - incompatible
  - homogéneo
    - compatible
      - determinado (Sol. trivial)
      - indeterminado

*Sistema compatible es aquél que sí tiene solución y para que esto se cumpla se requiere:*

$$\text{rango}\left[A\right] = \text{rango}\left[A\,|\,B\right] \quad * \qquad (3.3)$$

*donde a la matriz* $\left[A\,|\,B\right]$ *se le conoce como la matriz ampliada del sistema.*

*Sistema incompatible es aquél que no tiene solución y se cumple que:*

$$\text{rango}\left[A\right] < \text{rango}\left[A\,|\,B\right] \qquad (3.4)$$

*Sistema determinado es un sistema compatible que presenta solución única y se verifica que:*

$$\text{rango}\left[A\right] = \text{número de incógnitas} \qquad (3.5)$$

*Cuando se presenta esta situación en sistemas homogéneos se habla de solución trivial, ya que* $\underline{X} = \underline{0}$.

*Un sistema compatible que presenta infinidad de soluciones se conoce como sistema indeterminado y se caracteriza por:*

$$\text{rango}\left[A\right] < \text{número de incógnitas} \qquad (3.6)$$

*Para la solución de sistemas de ecuaciones lineales existen diversos métodos de los cuales solo se tratarán: Método de Gauss-Jordan modificado y el Método de Gauss-Seidel.*

## 3.2 Método de Gauss-Jordan modificado.

### 3.2.1 Objeto

*Obtener la solución de sistemas de ecuaciones lineales de la forma:*

$$\underline{A}\ \underline{X} = \underline{B} \qquad (3.7)$$

### 3.2.2 Método

*Dado el sistema de ecuaciones:*

$$\underline{A}\ \underline{X} = \underline{B} \qquad (3.8)$$

---

* *rango* $[\underline{A}]$ *es la cantidad de vectores linealmente independientes del conjunto de vectores columna que forman la matriz* $\underline{A}$.

el método consiste en trabajar con la matriz de coeficientes y el vector de términos independientes, es decir, con la matriz ampliada del sistema:

$$\left[\underline{A} \mid \underline{B}\right] \qquad (3.9)$$

A dicha matriz se le aplican una serie de transformaciones que conducen a obtener otra matriz ampliada equivalente:

$$\left[\underline{I}_n \mid \underline{C}\right] \qquad (3.10)$$

donde $\underline{C}$ representa la solución de cada una de las incógnitas del sistema.

El proceso equivale a premultiplicar la ecuación (3.9) - por $\underline{A}^{-1}$, es decir, el método de la matriz inversa, solo que este método consiste en una eliminación sistemática de valores.

La transformación de la matriz (3.9) en la matriz (3.10) se efectúa basándose en tres operaciones que no alteran el sistema de ecuaciones sino que proporcionan sistemas de ecuaciones equivalentes, ellas son:

- intercambio de dos renglones, lo cual equivale a intercambiar dos ecuaciones.
- multiplicación de un renglón por un escalar diferente de cero, lo cual equivale a multiplicar ambos miembros de una ecuación por la misma constante.
- suma de equimúltiplos de un renglón a otro renglón, es decir, multiplicar una ecuación por una constante "K" y sumarla a otra ecuación.

Para aplicar las operaciones anteriores se procede en la siguiente forma:

(1) Seleccionar un renglón pivote y un elemento pivote dentro de dicho renglón.
(2) Normalizar el elemento pivote, es decir, convertirlo en unitario.
(3) Cancelar elementos que se encuentren en la columna arriba y/o abajo del elemento pivote mediante la suma de - equimúltiplos.
(4) Regresar al paso (1) y así sucesivamente hasta que se - transforma la matriz de coeficientes $\underline{A}$ en una matriz --

identidad $I_n$.

Debido a que durante el proceso se presentan errores por redondeo, la forma óptima de escoger los elementos pivote es - seleccionando el mayor elemento que quede en la matriz A o en sus transformaciones. Hay que tener presente que los elementos de un renglón que ya fue seleccionado como línea pivote no se pue- den usar como pivotes, aún cuando el mayor elemento quede colo- cado en dicho renglón.

Al seleccionar los pivotes en la forma antes mencionada el error se reduce al mínimo y, debido a que puede quedar una - matriz no identidad al término de las iteraciones, es necesario efectuar un intercambio de líneas hasta obtener $I_n$.

Cabe mencionar que el presente método es un método directo de solución que no requiere que se determine con anterioridad si el sistema es compatible y determinado, el método durante el proceso proporciona dicha información.

Si el sistema es compatible y determinado, el procedimien- to descrito se puede llevar a cabo sin contratiempos hasta lle- gar a $\left[\begin{array}{c|c} I_n & C \end{array}\right]$.

Si el sistema es compatible pero indeterminado, la matriz ampliada adquirirá la configuración:

$$\left[\begin{array}{ccc|c} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \qquad (3.11)$$

es decir, un renglón será nulo; en esta situación se obtienen - las ecuacionesindependientes que restan en el sistema y se apli- ca la metodología correspondiente a sistemas indeterminados.

Si el sistema es incompatible, se presentará lo siguiente:

$$\left[\begin{array}{ccc|c} 1 & 1 & 2 & 1 \\ 0 & 2 & 3 & 2 \\ \hline 0 & 0 & 0 & \lambda \neq 0 \end{array}\right] \qquad (3.12)$$

o sea, $0 = \lambda \neq 0$, lo cual es una contradicción.

### 3.2.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE GAUTOR (A, B, N, EPS, DET), esta subrutina obtiene la solución del sistema de ecuaciones por el método de Gauss-Jordan modificado, el programa principal solo sirve para entrada y salida de datos.

b) Descripción de las variables:

Para la subrutina GAUTOR:

| | |
|---|---|
| A(I,J) | matriz de coeficientes del sistema de ecuaciones. |
| B(I) | vector de términos independientes del sistema de ecuaciones, durante el procéso se transforma en la solución. |
| N | orden del sistema de ecuaciones. |
| RAMAX | mayor elemento de la matriz A que se emplea como pivote. |
| MVR(I) y MVC(I). | contadores que indican qué renglón y columnas ya fueron empleados. |
| EPS | criterio para determinar si el determinante de la matriz A es nulo. |
| DET | parámetro que indica si el determinante de A es nulo. |
| LR y LC | indicadores del renglón y columna que se utilizan. |
| TEMP | variable de localización temporal. |

Para el programa principal:

| | |
|---|---|
| A(I,J) | matriz de coeficientes del sistema de ecuaciones. |
| B(I) | vector de términos independientes. |
| N | orden del sistema de ecuaciones. |
| EPS | criterio para determinar si el determinante de A es nulo. |
| DET | parámetro que indica si el determinante de A es nulo. |

c) Dimensiones:

La proposición DIMENSION del programa principal y de la subrutina se deberán modificar en el caso de que: $N > 10$

d) Formatos para los datos de entrada:

| SEC. TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (I5) | N |
| 2 | (8F10.0) | A(I,J), se dan los elementos de A renglón por renglón, empleando tantas tarjetas como sean necesarias para cada renglón. |
| . | | |
| . | | |
| . | | |
| 3 | (8F10.0) | B(I), el vector de términos independientes se da en una tarjeta o más según la cantidad de elementos. |
| otros paquetes de datos (opcional) | | |
| n | | TARJETA EN BLANCO, al finalizar toda la información. |

e) Diagrama de bloques:

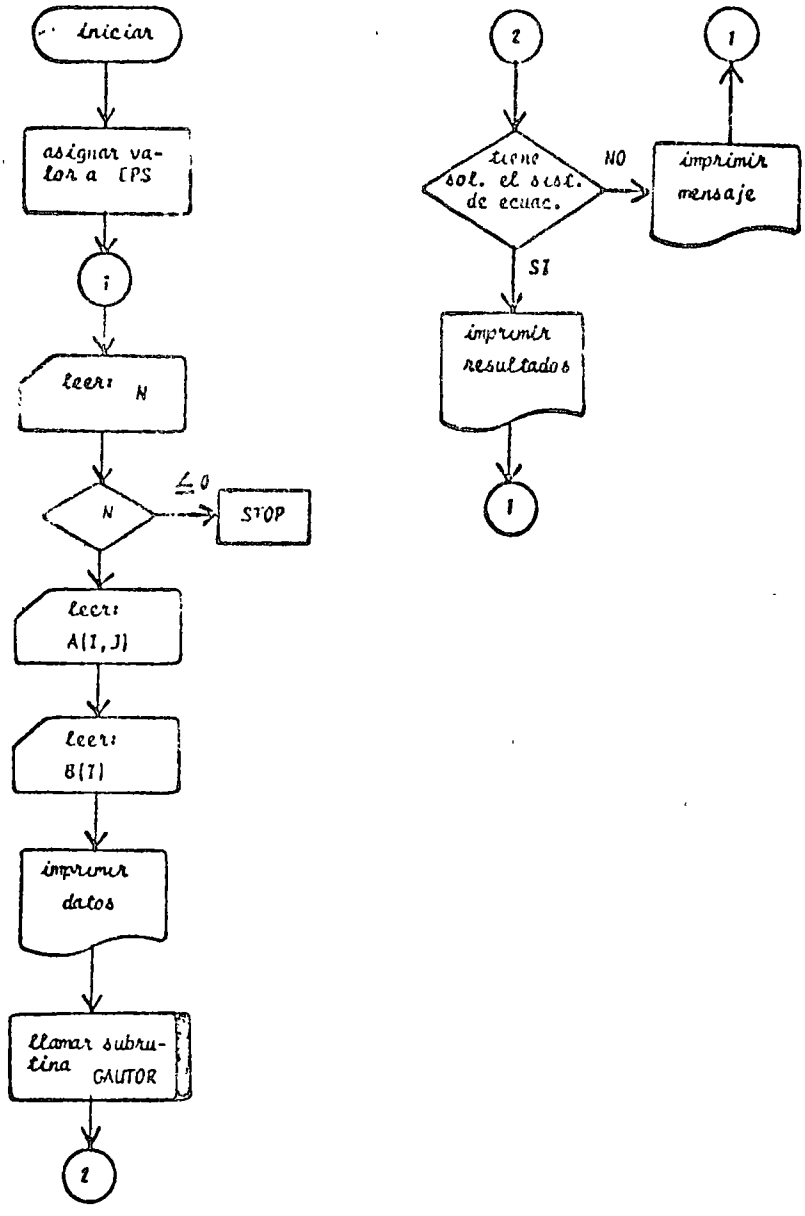



Fig. 3.1 Diagrama de bloques del programa principal.

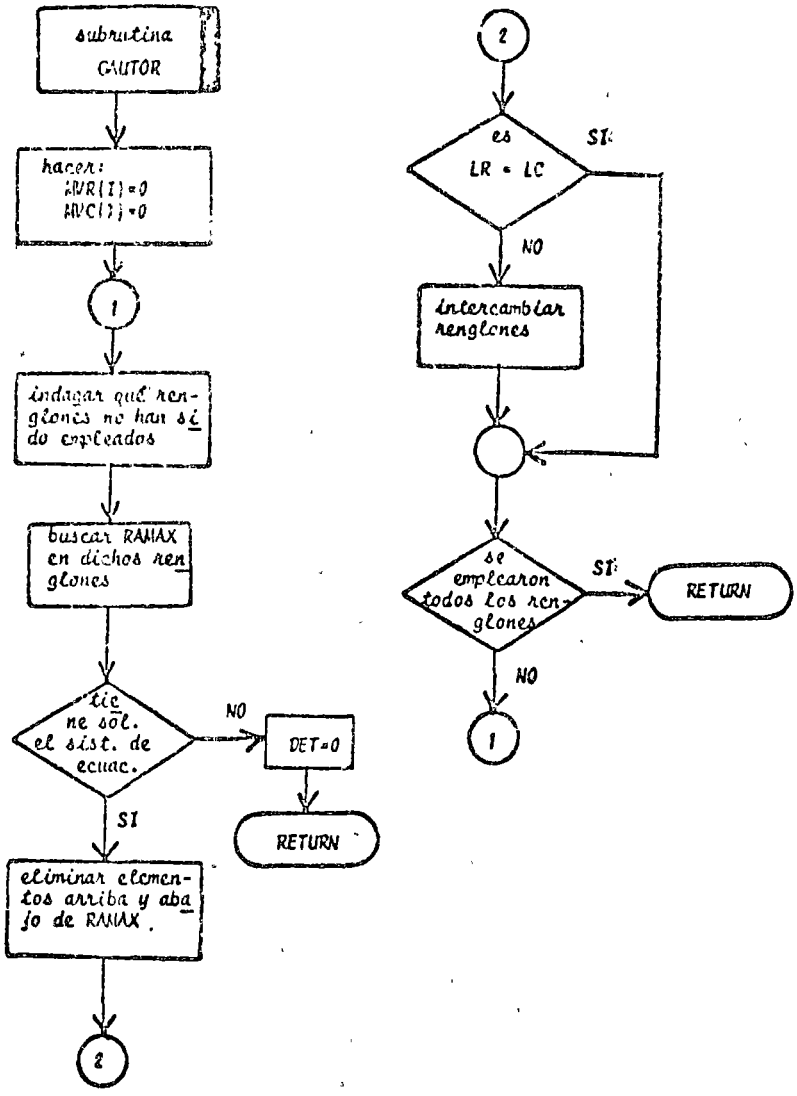



Fig. 3.2 Diagrama de bloques de la subrutina GAUTOR.

6) Listado:

```
C     PROGRAMA PARA RESOLVER SISTEMAS DE ECUACIONES LINEALES POR EL METO
C     DO DE GAUSS-JORDAN
C     SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     N=ORDEN DEL SISTEMA DE ECUACIONES
C     A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C     B=VECTOR DE TERMINOS INDEPENDIENTES, SE CONVIERTE EN LA SOLUCION
C     CRITERIO PARA DETERMINAR SI EL DETERMINANTE DE A ES DIFERENTE DE 0
C     DET=VARIABLE QUE INDICA SI EL SISTEMA TIENE O NO SOLUCION

      DIMENSION A(10,10),B(10)
      IR=5
      IW=6
      EPS=0.000001
C     LECTURA DE DATOS
    1 READ(IR,20) N
      IF(N) 2,2,3
    2 CALL EXIT
    3 DO 4 I=1,N
    4 READ(IR,21) (A(I,J),J=1,N)
      READ(IR,21) (B(I),I=1,N)
C     IMPRESION DE DATOS
      WRITE(IW,22)
      DO 5 I=1,N
    5 WRITE(IW,23) (A(I,J),J=1,N),B(I)
C     LLAMADO DE SUBRUTINA PARA RESOLVER EL SISTEMA DE ECUACIONES
      CALL GAUTOR(A,B,N,EPS,DET)
      IF(DET.LE.EPS) GO TO 7
C     IMPRESION DE RESULTADOS
      WRITE(IW,24)
      DO 6 I=1,N
    6 WRITE(IW,25) I,B(I)
      GO TO 1
    7 WRITE(IW,26)
      GO TO 1
C     FORMATOS DE LECTURA E IMPRESION
   20 FORMAT(I5)
   21 FORMAT(8F10.0)
   22 FORMAT(1H(/),5X,'EL SISTEMA DE ECUACIONES ES'//)
   23 FORMAT(/,2X,10(E10.3,1X))
   24 FORMAT(1H(/),5X,'LA SOLUCION DEL SISTEMA DE ECUACIONES ES'///,5X,'I
     1',5X,'X(I)',/)
   25 FORMAT(/,5X,I2,4X,E12.5)
   26 FORMAT(1H(/),9X,'EL SISTEMA DE ECUACIONES NO TIENE SOLUCION')
      END
```

*Fig. 3.3 Listado del programa principal*

```
      SUBROUTINE GAUTOR(A,B,N,EPS,DET)
C
C     SUBRUTINA PARA RESOLVER UN SISTEMA DE ECUACIONES POR EL METODO DE
C     GAUSS-JORDAN MODIFICADO
C     EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C     A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C     B=VECTOR DE TERMINOS INDEPENDIENTES QUE DURANTE EL PROCESO SE
C       TRANSFORMA EN LA SOLUCION DEL SISTEMA DE ECUACIONES
C     N=ORDEN DEL SISTEMA DE ECUACIONES
C     RAMAX=MAYOR ELEMENTO DE LA MATRIZ A QUE SE USA COMO PIVOTE
C     HVR Y HVC=CONTADORES QUE INDICAN QUE RENGLON Y QUE COLUMNA YA FUE-
C       RON UTILIZADOS
C     EPS=CRITERIO PARA DETERMINAR SI EL DETERMINANTE DE LA MATRIZ A ES
C       NULO
C     DET=VALOR ABSOLUTO DEL DETERMINANTE DE LA MATRIZ A
C
      DIMENSION A(10,10),B(10),HVR(10),HVC(10)
C
C     ACTUALIZACION DE VALORES PARA INICIAR EL PROCESO
C
      DO 1 I=1,N
      HVR(I)=0
    1 HVC(I)=0
C
C     SOLUCION DEL SISTEMA DE ECUACIONES
C
      DO 9 K=1,N
      RAMAX=0.0
      LC=0
      LR=0
      DO 3 I=1,N
      IF(HVR(I).EQ.1) GO TO 3
      DO 2 J=1,N
      IF(HVC(J).EQ.1) GO TO 2
      IF(ABS(RAMAX).GE.ABS(A(I,J))) GO TO 2
      RAMAX=A(I,J)
      LR=I
      LC=J
    2 CONTINUE
    3 CONTINUE
      DET=ABS(RAMAX)
      IF(DET.LE.EPS) GO TO 10
      IF(LR.EQ.LC) GO TO 5
      DO 4 I=1,N
      TEMP=A(LR,I)
      A(LR,I)=A(LC,I)
    4 A(LC,I)=TEMP
      TEMP=B(LR)
      B(LR)=B(LC)
      B(LC)=TEMP
    5 DO 6 I=1,N
    6 A(LC,I)=A(LC,I)/RAMAX
      B(LC)=B(LC)/RAMAX
      DO 8 I=1,N
      IF(I.EQ.LC) GO TO 8
      TEMP=A(I,LC)
      B(I)=B(I) - TEMP*B(LC)
      DO 7 J=1,N
    7 A(I,J)=A(I,J) - TEMP*A(LC,J)
    8 CONTINUE
      HVR(LC)=LC
      HVC(LC)=LC
    9 CONTINUE
   10 RETURN
      END
```

*Fig. 3.4 Listado de la subrutina GAUTOR*

### 3.2.4 Ejemplo

Empleando las leyes de Kirchhoff (ver referencia 2), se obtuvieron las siguientes ecuaciones lineales para el circuito mostrado en la figura 3.5:

$$
\begin{aligned}
i_8 - i_4 - I_A &= 0 \\
i_4 + i_5 + I_A - i_1 - i_3 &= 0 \\
i_1 - i_2 - I_B &= 0 \\
i_2 + I_B + i_3 + i_6 - i_7 &= 0 \\
I_C - i_8 - i_5 - i_6 - i_9 &= 0 \\
R_1 i_1 + R_2 i_2 - R_3 i_3 &= 0 \\
R_4 i_4 - R_5 i_5 + R_8 i_8 &= 0 \\
R_5 i_5 + R_3 i_3 - R_6 i_6 &= 0 \\
R_6 i_6 + R_7 i_7 - R_9 i_9 &= 0
\end{aligned}
$$

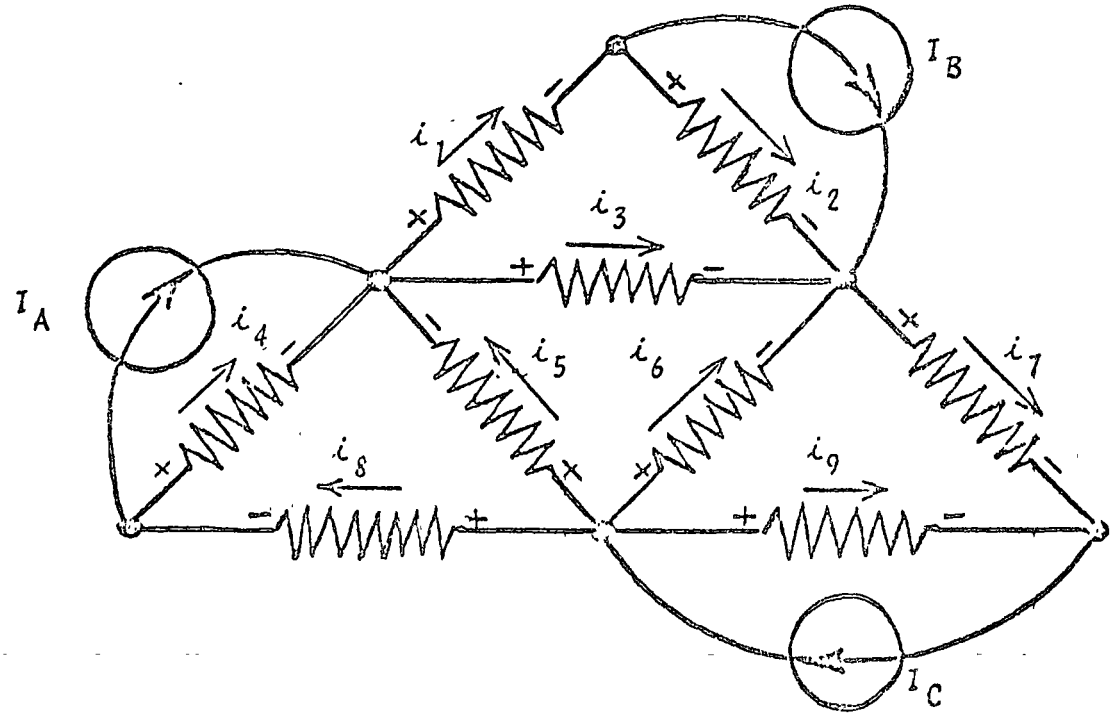



Fig. 3.5 Circuito del ejemplo 3.2.4

Si el valor de las fuentes es $I_A = 2A$, $I_B = 6A$, $I_C = 4A$ y el de las resistencias:

$R_1 = R_2 = 2\ \Omega$

$R_4 = R_8 = 3\ \Omega$

$R_5 = R_6 = 5\ \Omega$

$R_7 = R_9 = 4\ \Omega$

$R_3 = 6\ \Omega$

Obtenga las corrientes de rama $i_1$, $i_2$, $i_3$, $i_4$, $i_5$, $i_6$, $-i_7$, $i_8$, $i_9$.

* SOLUCION

TABLA 3.1 Datos para el problema del ejemplo 3.2.4

$N = 9$

$$A = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 2 & 2 & -6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & -5 & 0 & 0 & 3 & 0 \\ 0 & 0 & 6 & 0 & 5 & -5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 4 & 0 & -4 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 \\ -2 \\ 6 \\ -6 \\ 4 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

TABLA 3.2 Resultados del problema del ejemplo 3.2.4

EL SISTEMA DE ECUACIONES ES

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0. | 0. | 0. | -.100E+01 | 0. | 0. | 0. | .100E+01 | 0. | .200E+01 |
| -.100E+01 | 0. | -.100E+01 | .100E+01 | .100E+01 | 0. | 0. | 0. | 0. | -.200E+01 |
| .100E+01 | -.100E+01 | 0. | 0. | 0. | 0. | 0. | 0. | 0. | .600E+01 |
| 0. | .100E+01 | .100E+01 | 0. | 0. | .100E+01 | -.100E+01 | 0. | 0. | -.600E+01 |
| 0. | 0. | 0. | 0. | .100E+01 | .100E+01 | 0. | .100E+01 | .100E+01 | .400E+01 |
| .200E+01 | .200E+01 | -.600E+01 | 0. | 0. | 0. | 0. | 0. | 0. | 0. |
| 0. | 0. | 0. | .300E+01 | -.500E+01 | 0. | 0. | .300E+01 | 0. | 0. |
| 0. | 0. | .000E+01 | 0. | .500E+01 | -.500E+01 | 0. | 0. | 0. | 0. |
| 0. | 0. | 0. | 0. | 0. | .500E+01 | .400E+01 | 0. | -.400E+01 | 0. |

LA SOLUCION DEL SISTEMA DE ECUACIONES ES

| I | X(I) |
|---|---|
| 1 | .23781E+01 |
| 2 | -.36239E+01 |
| 3 | -.01596E+00 |
| 4 | -.56359E+00 |
| 5 | .52370E+00 |
| 6 | .24948E+01 |
| 7 | .19807E+01 |
| 8 | .14364E+01 |
| 9 | .20151E+01 |

## 3.3 Método de Gauss-Seidel

### 3.3.1 Objeto

Obtener la solución de sistemas de ecuaciones lineales -- con la configuración:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{m2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned} \qquad (3.13)$$

empleando el método de Gauss-Seidel.

### 3.3.2 Método

El método de Gauss-Seidel es un método de tipo iterativo que sirve para la solución de sistemas de ecuaciones lineales del tipo:

$$\underline{A}\ \underline{X} = \underline{B} \qquad (3.14)$$

cuando los valores numéricos de los elementos de la diagonal - principal son mayores que los demás de su correspondiente renglón.

Para asegurar la convergencia del método se requiere que:

a) los elementos no nulos de la matriz de coeficientes ($\underline{A}$) se acumulen en la diagonal principal.

b) los elementos de la diagonal principal de la matriz de coeficientes ($\underline{A}$) sean mayores en valor absoluto que la sumatoria de los valores absolutos de los elementos -- restantes del renglón correspondiente, es decir:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad , \quad i = 1, 2, \ldots, n \qquad (3.15)$$

Para aplicar el método se procede a despejar una incógni

ta de cada ecuación del arreglo (3.13), es decir, despejar la incógnita $x_i$ de la "i-ésima" ecuación, o sea:

$$\left.\begin{aligned} x_1 &= \frac{1}{a_{11}}\left[b_1 - a_{12}x_2 - a_{13}x_3 - \dots - a_{1n}x_n\right] \\ x_2 &= \frac{1}{a_{22}}\left[b_2 - a_{21}x_1 - a_{23}x_3 - \dots - a_{2n}x_n\right] \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}\left[b_n - a_{n1}x_1 - a_{n2}x_2 - \dots - a_{n,n-1}x_{n-1}\right] \end{aligned}\right\} \quad (3.16)$$

y se establecen las siguientes ecuaciones iterativas:

$$\left.\begin{aligned} x_1^{(k+1)} &= \frac{1}{a_{11}}\left[b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \dots - a_{1n}x_n^{(k)}\right] \\ x_2^{(k+1)} &= \frac{1}{a_{22}}\left[b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \dots - a_{2n}x_n\right] \\ &\vdots \\ x_n^{(k+1)} &= \frac{1}{a_{nn}}\left[b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \dots - a_{n,n-1}x_{n-1}^{(k+1)}\right] \end{aligned}\right\} \quad (3.17)$$

donde $x_i^{(k+1)}$ indica el valor de la "i-ésima" incógnita en la iteración "k + 1"

Para arrancar el método se establece una solución inicial $\underline{X}_0$:

$$\underline{X}_0 = \begin{bmatrix} x_1^\circ \\ x_2^\circ \\ \vdots \\ x_n^\circ \end{bmatrix} \quad (3.18)$$

dichos valores se sustituyen en el lado derecho de la ecuación (3.17) para obtener la siguiente solución aproximada:

$$\underline{X}_1 = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ . \\ . \\ . \\ x_n^{(1)} \end{bmatrix} \qquad (3.19)$$

y así sucesivamente hasta que

$$\left| \underline{X}_{n+1} - \underline{X}_n \right| < \underline{\varepsilon} \qquad (3.20)$$

Para poder emplear este método es necesario verificar con anterioridad que el sistema sea compatible y determinado; además de que cumpla con las condiciones de convergencia del método. Afortunadamente la mayoría de los problemas de tipo ingenieril cumplen los requisitos mencionados.

Ciertos sistemas que a primera vista no cumplen los requisitos del método pueden llenar los requisitos mediante un simple intercambio en la posición de las ecuaciones.

3.3.3 Descripción del programa

a) Subrutinas requeridas:

Ninguna.

b) Descripción de las variables.

| | |
|---|---|
| A(I,J) | matriz de coeficientes del sistema |
| B(I) | vector de términos independientes |
| N | orden del sistema de ecuaciones |
| X(I) | valor inicial de las incógnitas del sistema y variable de localización temporal |
| Y(I) | valor de las incógnitas en la iteración "n" |
| XN(I) | valor de las incógnitas en la iteración "n + 1" |

| | |
|---|---|
| M | máximo número de iteraciones a efectuar |
| E | criterio de convergencia |
| NCON | contador de iteraciones efectuadas |
| SUM | sumador |

c) Dimensiones:

La proposición DIMENSION deberá modificarse cuando se presente el caso de que $N > 20$.

d) Formatos para los datos de entrada:

| SEC. TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (2I5,F10.0) | N, M, E |
| 2 | (10F8.0) | A(I,J), los elementos de la matriz A se dan renglón por renglón empleando la cantidad de tarjetas necesaria para cada renglón. |
| 3 | (10F8.0) | B(I) el vector de términos independientes se da en una tarjeta o más según el orden del sistema. |
| 4 | (10F8.0) | X(I), la solución para arrancar el método se da en una tarjeta o más según sea el tamaño de N. |
| otros paquetes de datos (opcional) | | |
| n | | TARJETA EN BLANCO, al finalizar toda la información. |

e) Diagrama de bloques:

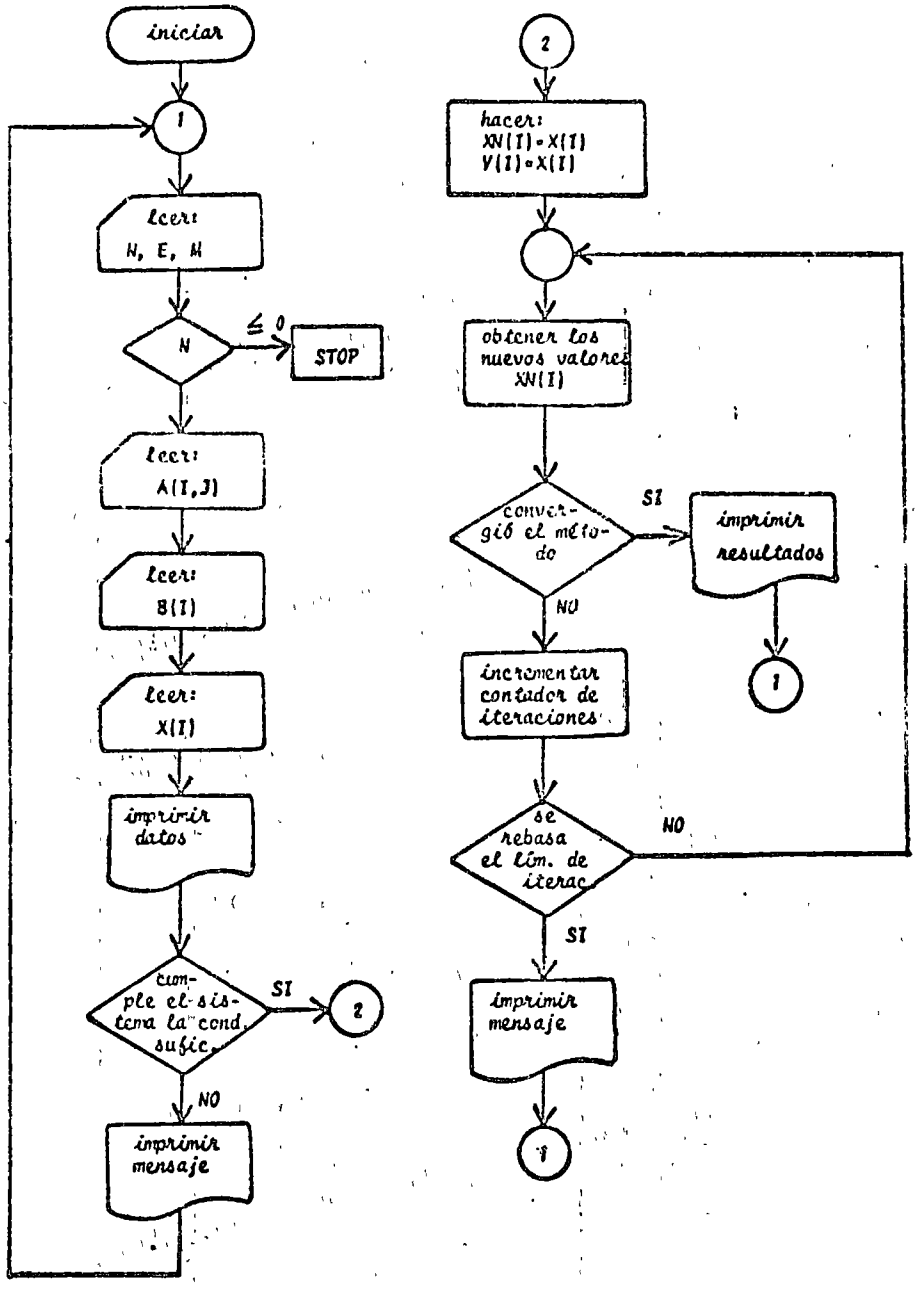



Fig. 3.6 Diagrama de bloques para el programa

6) Listado:

```
C     PROGRAMA PARA RESOLVER SISTEMAS DE ECUACIONES POR EL METODO DE
C     GAUSS-SEIDEL
C     SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C     B=VECTOR DE TERMINOS INDEPENDIENTES
C     X=VALOR INICIAL DE LA SOLUCION DEL SISTEMA
C     XN=SOLUCION DEL SISTEMA DE ECUACIONES EN LA SIGUIENTE ITERACION
C     N=ORDEN DEL SISTEMA
C     Y=VALOR DE LA SOLUCION DEL SISTEMA DE ECUACIONES EN LA ITERACION
C         ANTERIOR
C     M=MAXIMO NUMERO DE ITERACIONES
C     E=CRITERIO DE CONVERGENCIA

      DIMENSION A(20,20),B(20),X(20),Y(20),XN(20)
C     LECTURA DE DATOS
    1 READ(5,200) N,M,E
      IF(N) 2,2,3
    2 CALL EXIT
    3 DO 4 I=1,N
    4 READ(5,300) (A(I,J),J=1,N)
      READ(5,300) (B(I),I=1,N)
      READ(5,300) (X(I),I=1,N)
C     IMPRESION DE DATOS
      WRITE(6,400)
      DO 5 I=1,N
    5 WRITE(6,500) (A(I,J),J=1,N),B(I)
      WRITE(6,600) (X(I),I=1,N)
C     SE INDAGA SI EL SISTEMA CUMPLE LA CONDICION SUFICIENTE DE CONVER-
C     GENCIA
      DO 7 I=1,N
      DO 6 J=1,N
      IF(ABS(A(I,I)) - ABS(A(I,J))) 8,6,6
    6 CONTINUE
    7 CONTINUE
      GO TO 9
    8 WRITE(6,700) I,J,I,I
      GO TO 1
C     OBTENCION DEL VALOR DE LAS INCOGNITAS
    9 NCON=1
      DO 10 I=1,N
      XN(I)=X(I)
   10 Y(I)=X(I)
   11 DO 14 K=1,N
      SUM=0.
      DO 13 I=1,N
      IF(K-I) 12,13,12
   12 SUM=SUM + A(K,I)*XN(I)
   13 CONTINUE
      XN(K)=(B(K)-SUM)/A(K,K)
   14 CONTINUE
      DO 15 I=1,N
C     SE VERIFICA SI YA CONVERGIO EL METODO
      IF(ABS(XN(I)-Y(I))-E) 15,16,16
   15 CONTINUE
C     IMPRESION DE RESULTADOS
      WRITE(6,800) (XN(I),I=1,N)
      WRITE(6,950) NCON
      GO TO 1
   16 NCON=NCON + 1
      IF(NCON-M) 18,17,17
   17 WRITE(6,900) (XN(I),I=1,N)
      WRITE(6,950) NCON
      GO TO 1
   18 DO 19 I=1,N
   19 Y(I)=XN(I)
      GO TO 11
C     FORMATOS-DE LECTURA E IMPRESION
  200 FORMAT (2I5,F10.0)
  300 FORMAT (10F8.0)
  400 FORMAT(1H1,9(/),15X,'MATRIZ AMPLIADA')
  500 FORMAT (/,15X,10(F8.3,5X))
  600 FORMAT(3(/),15X,'PRIMERA APROXIMACION DE LA SOLUCION',///,10X,10(F
     16.2,5X))
  700 FORMAT(3(/),15X,'EL METODO PUEDE NO CONVERGER DADO QUE',///,15X,'A(
     2I2,',',I2,') ES MAYOR QUE A(',I2,',',I2,')')
  800 FORMAT(3(/),15X,'LA SOLUCION DEL SISTEMA ES',///,5X,9(E12.5,2X))
  900 FORMAT(3(/),15X,'NO SE LLEGA A LA SOLUCION',///,15X,'LA ULTIMA APR
     1OXIMACION ENCONTRADA FUE',///,5X,9(E12.5,2X))
  950 FORMAT(3(/),15X,'ITERACIONES REALIZADAS= ',I4)
      END
```

*Fig. 3.7 Listado del programa*

3.3.4 Ejemplo

Para el circuito eléctrico de la fig. 3.8 se sabe que --- $I_1 = 1A$ e $I_2 = 2A$, $R_1 = R_2 = R_3 = R_4 = R_5 = R_6 = 1\ \Omega$.

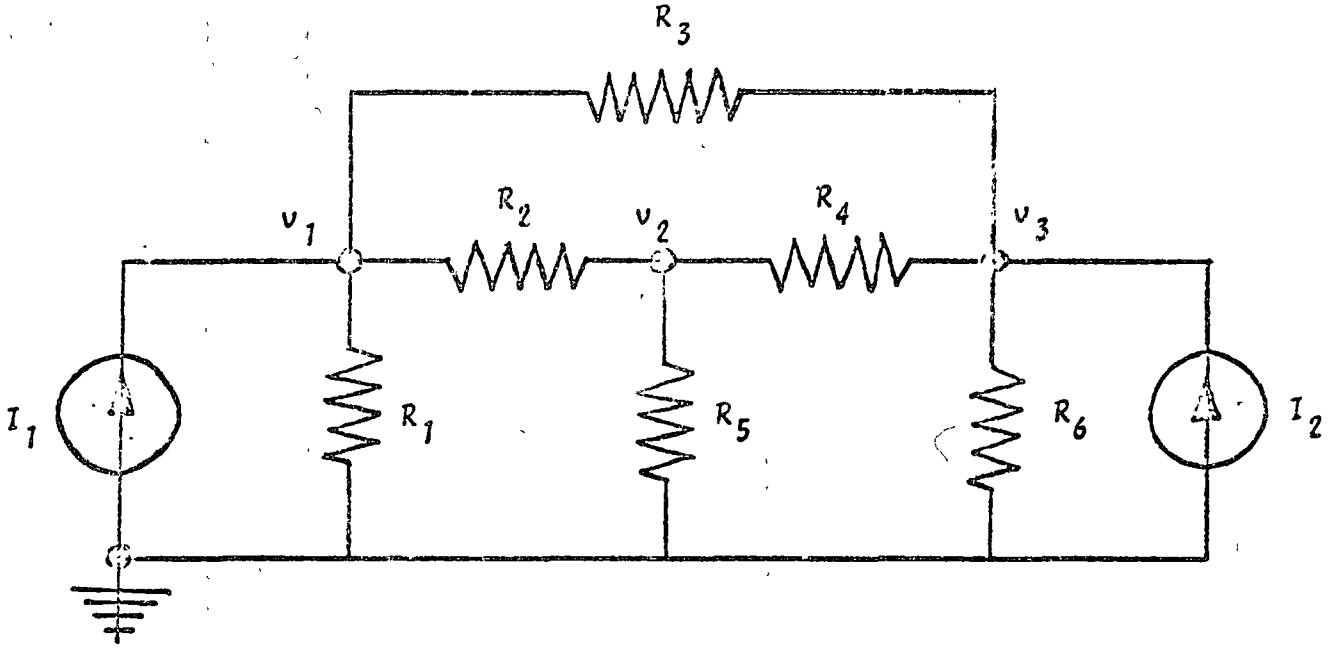



Fig. 3.8 Circuito eléctrico del problema del ejemplo 3.3.4

Se desea obtener el voltaje de los nodos $V_1$, $V_2$ y $V_3$. Aplicando análisis nodal al circuito se obtiene:

$$3V_1 - V_2 - V_3 = 1$$
$$-V_1 + 3V_2 - V_3 = 0$$
$$-V_1 - V_2 + 3V_3 = 2$$

arreglo que es un sistema de ecuaciones lineales con todas las características propias para aplicar el método de Gauss-Seidel.

Se seleccionará como solución inicial al siguiente vector:

$$\begin{bmatrix} V_1^o \\ V_2^o \\ V_3^o \end{bmatrix} \qquad \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

* SOLUCION

TABL 3.3 Datos del problema del ejemplo 3.3.4

$N = 3$

$M = 50$

$EPS = 0.0001$

$$A = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

$$\underline{X} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

TABLA 3.4 Resultados del problema del ejemplo 3.3.4

MATRIZ AMPLIADA

| | | | |
|---|---|---|---|
| 3.000 | -1.000 | -1.000 | 1.000 |
| -1.000 | 3.000 | -1.000 | 0.000 |
| -1.000 | -1.000 | 3.000 | 2.000 |

PRIMERA APROXIMACION DE LA SOLUCION

0.50 0.50 0.50

LA SOLUCION DEL SISTEMA ES

.10000E+01 .75000E+00 .12500E+01

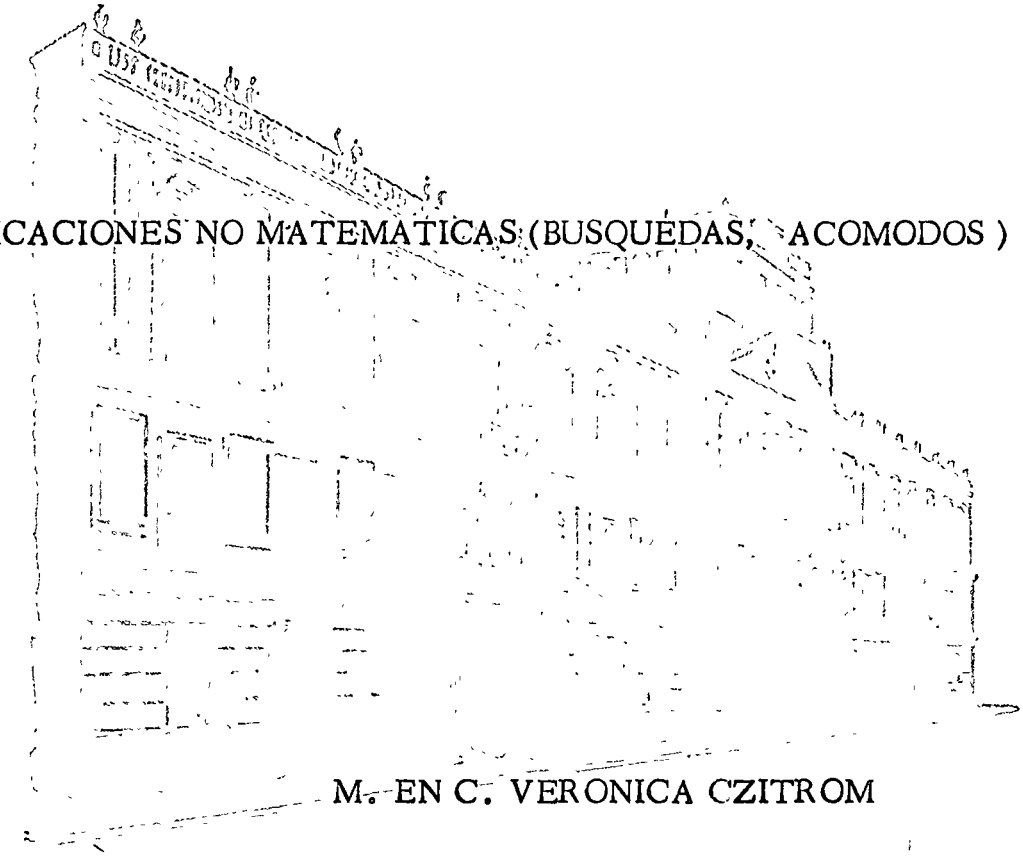
ITERACIONES REALIZADAS= 10

## 3.4 Bibliografía


1. CARNAHAN B., LUTHER H., WILKES J., "Applied Numerical Methods". New York: John Wiley and Sons Inc., 1969. pp. 269-307.
2. GEREZ G. Víctor, MURRAY LASSO M.A., "Teoría de Sistemas y Circuitos". México: Representaciones y Servicios de Ingeniería, S.A., 1972. pp. 99-123.
3. HADLEY G. "Algebra Lineal". Bogotá: Fondo Educativo - Interamericano, 1969. pp. 162-187.
4. HAMMING Richard, "Numerical Methods for Scientists -- and Engineers". New York: Mc Graw Hill Book Co., 1962. pp. 360-365.
5. JAMES M., SMITH G., WOLFORD J., "Applied Numerical -- Methods for Digital Computation with FORTRAN". ---- Scranton Penn: International Text Book Co., 1967. pp. 184-230.
6. JOHN STON J., BALEY PRICE G., VAN VLECK F., "Linear - Ecuations and Matrices", Reading Mass.: Addison---- Wesley Co., 1966. pp. 1-94.
7. KUO S. Shan, "Computer Applications of Numerical ---- Methods". Reading Mass.: Addison Wesley Co., 1972. pp. 179-212.
8. CLIVERA S. Antonio, "Apuntes de Métodos Numéricos". - México.: Facultad de Ingeniería, UNAM. 1972. pp.. 4.1-4.34.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA



## APLICACIONES NO MATEMATICAS (BUSQUEDAS, ACOMODOS )

M. EN C. VERONICA CZITROM

Febrero-Marzo 1977

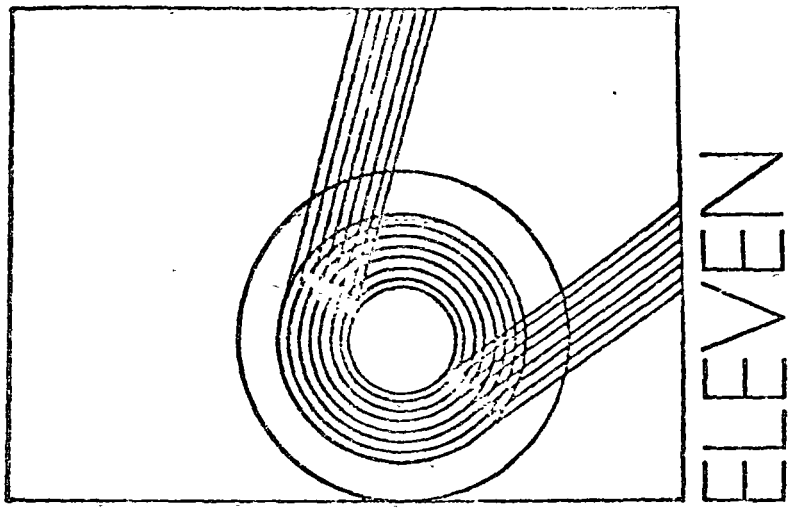
IF ((A LE B) OR (B GT C)) GO TO 43

cause a transfer of control to the statement number 43?

(*a*) A = 25.0 B = 30.0 C = 15.5
(*b*) A = 25.0 B = 23.0 C = 15.5
(*c*) A = 25.0, B = 23.0, C = 25.5
(*d*) A = 6.0, B = 26.0 C = 27.5

25 Modify the program in Fig. 10.18 so that razors are sent to families where the answer to question 3 is No and there are two or more males in the family. Test your program.

26 Modify the program in Fig. 10.18 so that razors are sent if the answer to question 3 is Yes and four times the number of males over 16 plus two times the number of women over 16 is greater than 15.



# ELEVEN

## NONNUMERICAL ALGORITHMS, FILES, AND DATA STRUCTURES

The use of computers extends through almost every part of our modern lives. The largest usage of computers is, however, in the worlds of business and government. Keeping records (bookkeeping) is always an important part of running a business, and this extends to the business of running the government. For that matter, one of the largest uses of computers by the military is in logistics, which is the control of supplies. Computers also perform such functions as inventory control in factories and warehouses, processing checks and calculating balances in banks, and managing airline and train reservations, and they take part in just about every aspect of business record maintenance. Computers also play major roles in process control and the automation of manufacturing, scheduling of production in factories, monitoring patients after surgery, scheduling aircraft maintenance, and assisting management in making decisions by gathering and calculating statistics on items of interest.

Important applications in science include work in psychology, biology, medicine, and the social sciences in general. Probably no other part of the computer business is growing as fast as the work in these areas, and the potential gain for mankind is truly staggering.

Just as business management finds it advantageous to be able to

process data and gather statistics in business transactions. Medical researchers find an important application in gathering statistics on the effects of drugs when used in treating patients. Similarly, biologists and chemists now use the data processing capabilities of computers to gather statistics for their work and to model systems of interest, molecular structures for instance.

There is an important difference in the use of the computer in business data processing and in most scientific applications. This difference arises from the fact that business data processing generally involves maintaining large files of data while performing relatively few calculations on individual data items. In scientific computing the computer is generally called on to perform many calculations on a relatively small number of input data. Nevertheless, the same computers are used in both application areas (although some computers are better suited to scientific applications and others to business data processing).

The Fortran language was not specifically designed for business data processing applications, although it is often used for this purpose, particularly for smaller systems. The Cobol language was designed especially for data handling and has been the most widely used language of this type, to date. (An official ANSI Cobol is used by the government and many businesses.) Cobol has very limited computational abilities, however, and is rarely used in science and in other applications where much numerical work is called for. A new language, PL/1, developed by IBM, has both business data processing features and a Fortran-like ability to perform numerical calculations (as well as some features for character string handling). The language PL/1 is gaining in popularity, and some manufacturers other than IBM are now providing PL/1 compilers. (PL/1 is a very large language, and a complete compiler is of some size.)

## 11.1 FILE MAINTENANCE

Since large businesses and governments tend to have large files to maintain, large memories and sophisticated filing techniques are called for. Similarly, scientists who gather data sometimes establish large files of data. As an extreme example, consider the work in automated libraries and information retrieval systems which maintain bibliographic files containing hundreds of millions of items. In these systems it is possible to search millions of abstracts for key items in minutes. It was, in fact, a need for special types of files with extraordinary maintenance procedures which led workers in the field of artificial intelligence to develop certain of the most interesting data structures, which will be described.

When discussing file maintenance, it is useful to define certain terms more carefully. An *item* is an individual piece of information. A *record* is composed of all the items in a file relating to the same object or individual. A collection of related items is called a *file*.



| | Automobile | | | Owner | | | |
|---|---|---|---|---|---|---|---|
| | License | Make | Year | Name | Street Address | Town | State |
| Entries in file | 119-606 | Plymouth | 1955 | J. F. Jones | 19 Carey Ave. | Woburn | Mass. |
| | 192-731 | Mercury | 1971 | F. P. Jackson | 18 Knoll St. | Belmont | Mass. |
| | 473-842 | Corvair | 1973 | J. P. Fremon | 163 Kenwood St. | Concord | Mass. |
| | 972-167 | Cadillac | 1974 | F. M. Mayo | 462 April Ave. | Bedford | Mass. |

*Note* Each row is a complete record consisting of seven items (1) the license number (2) the make of the car (3) the year of manufacture (4) the name of the owner (5) the owner's street address, (6) the town in which the owner resides, (7) the state in which the owner resides.

**Figure 11.1** A section of a file

Examples will help clarify these definitions. An *item* might be a name, such as "John M. Jones," or an age, such as "29," or a marital status, such as "M" or "S," or an address such as "39 Rhodes Avenue, Newton, Iowa." A *record* would then be the set of all items for John M. Jones which would be John M. Jones, 29, M, 39 Rhodes Avenue, Newton, Iowa. A collection of records such as the above would constitute a file. Figure 11.1 shows a small portion of a typical file for a department of motor vehicles.

Large files can consist of from hundreds of thousands to millions of records, each record containing several items. Consider the files of the Bureau of Census or Internal Revenue Service or the files of any major insurance company. Because of the size of these files, they must be maintained on such storage media as magnetic tape or cards or even, for instance, on microfilm. The maintenance of these files requires considerable work, since they must be continually updated. Further, in order for the files to be really useful, it must be possible to acquire data of a specified class from the files with minimal search time.

In order to maintain large files and to search them effectively for specified records or items in a particular class, these must be carefully designed with regard to their organization.

The term *data structure* refers to the method used for organizing data and the resulting interrelations between the data items and their addresses or identifiers so that an efficient computer implementation results. Data structures form an important area in computer science, as do the algorithms for maintaining and using them.

## 11.2 OPERATIONS ON FILES

Large files must be continually maintained. This primarily consists of adding new records to the file, deleting old records, and modifying records already in the file. In performing these operations and in locating and

processing data in the files, it is generally efficient to maintain the records in the file in some prescribed order rather than simply adding new items at the end, closing up "holes" when items are deleted, etc.

In order to see the need for ordering the items in a file, consider the way we find a name in a telephone book. If the names in a telephone book were not ordered, we would have to start at page 1 and search the book a name at a time. Since however, we know last names are arranged in alphabetic order, we can guess at the location of a name, open the book to that point, and see if we have made a good guess or if we need to move forward or backward in the book. Contrast the small amount of hunting necessary to locate a name in a telephone book with the effort required to find a name in a novel or some other book where it is necessary to search at random.

Arranging a given file in a prescribed order is called *sorting* the file. Generally some particular item in each record is chosen (such as last name, social security number, part number), and the file is sorted by arranging the records in the file so that the selected items are in the prescribed order. The selected item is called a *key*, and the file is said to be *sorted on the key*.

For practical purposes, sorting in a computer generally consists of arranging the records by ordering the keys in ascending (or sometimes descending) numerical order. When alphanumeric characters are used in the key, each character will, in actual practice, consist of several binary digits, and the complete set of binary digits in a given key can be considered a binary number. By then arranging the binary numbers for the keys in numerically ascending order, we can "sort on the key." [The order in which an alphanumeric code causes the characters in the code to be arranged when they are in ascending numerical (binary) order is called the *collating sequence* for the code.]

Since sorting is such an important function in file maintenance, many algorithms have been invented for it, and several of them are examined in the following sections.

Another important operation is that of *searching*. When a particular record or a set of records with some specific characteristic in a file is required, searching the file is necessary. The simplest form of a search is the *linear search* where the records are examined one at a time in order. This is time-consuming for most memories but is natural for tape memories. If a file has been sorted, the most efficient *binary search* can be used. Some aspects of the search problem are examined in following sections.

A most interesting and important aspect of files involves the ways in which the data are stored in the file and the overall organization of the file structure. There are, in fact, many ways to organize a file in a memory, and some are examined in the final sections of this chapter.



## 11.3 SORTING AND MERGING

An important operation in maintaining business and other information system files consists of sorting a file on a selected key. As was mentioned previously, the sorting places the records in the file in an order so that the key items in the records are in nondescending order. If no two records in the file have the same value for the key (for instance, if the key was a social security number in a personnel file), the records would be arranged with the keys in ascending order. In an actual file consisting of many records each containing several items, sorting the file would involve moving the entire records around (or at least pointers to the records—as will be discussed later). The actual sorting is generally done on an array of the values of the key, however, as this is more efficient, and the records can be moved after the new arrangement for the keys has been found. In fact, for most large files it is necessary to keep the file on some mass storage device such as magnetic tape, and to sort the file in stages, moving portions of the file from and into the mass storage devices. Our concern will be with the sorting process only, excellent descriptions of file maintenance procedures for mass storage devices will be found in several of the references.

Chapters 3 and 7 have already presented an algorithm and subprogram for sorting an array so that the elements are in nondescending order. The existence of this algorithm is assurance that we can sort an array into nondescending order; the remaining questions concern the efficiency of the sorting technique.

In order to examine sorting techniques we limit the problem to that of sorting an array M with integer values M(1),M(2), . . .,M(N) so that M(I) is less than or equal to M(J) when I is less than J.

Perhaps the most natural or intuitive way to sort such an array is to search through the set of values in the array and find the smallest value, call this value M(K), then place this value in M(1) by exchanging this M(K) with M(1). Next examine M(2),M(3), . . . ,M(N) and find the smallest element, call this element M(J), and place this element in M(2) by exchanging M(2) and M(J).

These steps are repeated for M(3), M(4), and so on until M(N − 1) and M(N) are finally compared and arranged.

The above description details the procedure generally used by people when arranging a bridge hand or when sorting a deck of index cards into order. A flow chart for the algorithm is shown in Fig. 11.2, and a Fortran subprogram is in Fig. 11.3.

Examination of the flow chart and the program indicates that there is an outer and inner loop in the algorithm. For an array with N elements, the outer loop is performed N − 1 times, the first time the smallest of element M(1), M(2), . . . ,M(N) is found and moved into M(1), the final time the smallest of M(N − 1) and M(N) is found and moved into M(N − 1).

The inner loop in the algorithm sequences through the set of elements

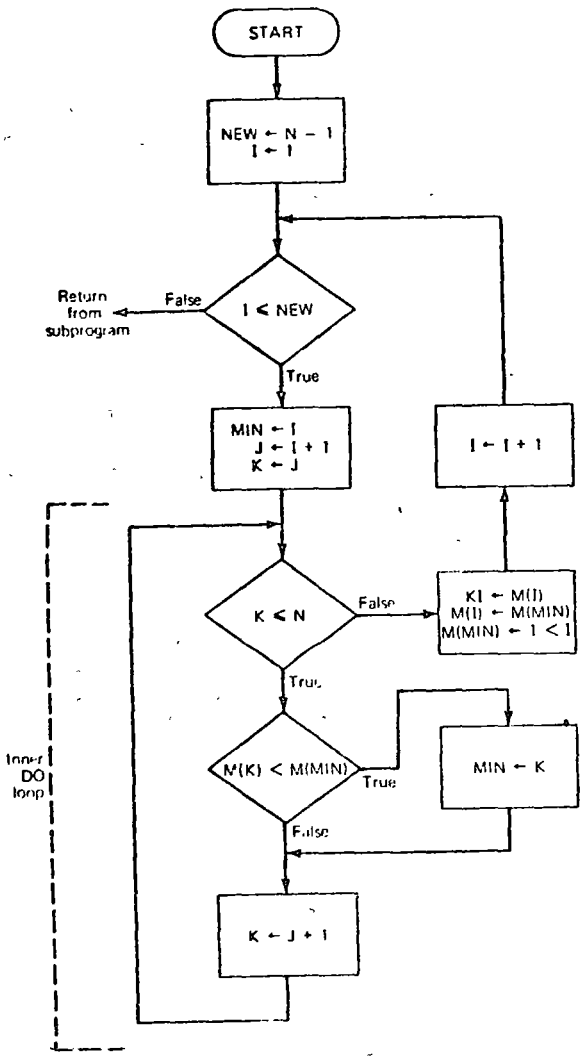



Figure 11.2 Flow chart of a replacement sort. An array called M of N elements is sorted.

under consideration, determining the smallest of them, and moving this element into the correct position.

In order to evaluate the efficiency of this algorithm, we count the number of passes which are made through the inner loop. Let us set N equal to 10



... we have an array of 10 elements. First the ... examines M(1) M(2) ... M(10), finding the smallest and placing it in position M(1). This requires nine passes through the inner "comparison" loop. Then the algorithm examines M(2) M(3) ... M(10) finding the smallest element which involves eight passes through the inner loop. We now see the basic pattern: on the third pass through the outer loop seven passes will be made through the inner loop, on the fourth pass through the outer loop five passes will be made through the inner loop, and on the final ninth pass through the outer loop one pass will be made through the inner loop. Thus, the algorithm requires $9 + 8 + 7 + 6 + \cdots + 2 + 1$ passes through the inner loop that is, 45 passes.

The general case is as follows. To sort an array of $N$ elements, the algorithm makes $N - 1$ passes through the outer loop. The first pass requires $N - 1$ passes through the inner loop, the second pass through the outer loop requires $N - 2$ passes through the inner loop, and this pattern continues until on the final or $(N - 1)$st pass through the outer loop when one pass is required through the inner loop. Thus, we arrive at the following sum for the number of passes through the inner loop: $(N - 1) + (N - 2) + \cdots + 2 + 1$.

Then let us call $P$ the value of the above sum. Now,† $P = N(N - 1)/2$ or, written another way,

$$P = \frac{N^2 - N}{2}$$

This sum grows very quickly as $N$ becomes larger. For instance, for $N = 100$ the value of $P$ is 4,950, but for $N = 1{,}000$ the value of $P$ is 499,500.

†This can be shown as follows. The average value in the sum $(N - 1) + (N - 2) + \cdots + 1$ is $[(N - 1) + 1]/2$, which is also $N/2$, and there are $N - 1$ terms, so the sum has value $N/2 \cdot (N - 1)$ or $[N(N - 1)]/2$.

```
      SUBROUTINE SORT1(M,N)
      DIMENSION M(100)
        NEW=N-1
      DO 3 I=1,NEW
C     SELECT THE ELEMENT M(I) AS A POSSIBLE SMALLEST
      MIN=I
C     SEARCH ARRAY FOR SMALLEST VALUE
      J=I+1
      DO 5 K=J,N
      IF (M(K) .LT. M(MIN)) MIN=K
    5 CONTINUE
C     EXCHANGE THE SMALLEST WITH M(I)
      KI=M(I)
      M(I)=M(MIN)
    3 M(MIN)=KI
      RETURN
      END
```

Figure 11.3 Subprogram for replacement sort. M is the array to be sorted and N is the number of elements in the array M.

Another sorting algorithm which resembles the above but which can be more efficient on the average is the "bubble sort." When the bubble sort is used comparisons are made only between *adjacent* elements in the array, and the elements are exchanged if they are out of order. In the first pass of a bubble sort to sort an array M of N elements, M(1) is compared with M(2) and if M(2) is smaller, they are exchanged. Then M(2) is compared with M(3), and the elements are swapped if they are out of order. This is continued until M(N − 1) is finally compared with M(N), and a swap made if necessary. During the second pass the same general procedure is followed except only adjacent items from M(1) to M(N − 1) are considered, on the third pass M(1) to M(N − 2) is considered, and on the final pass only M(1) and M(2) are compared. When this sort is used, larger items float down and small items "bubble up," hence the name. A subprogram to implement this sort is shown in Fig. 11.4.

The subprogram in Fig. 11.4 has an additional feature, which can improve its efficiency. Each time a pass is made through the section of the array being examined by the outer loop, a test is made, and if no exchanges are made the program discontinues its sort, for the array is already in nondescending order. The test is made by setting the variable of "flag" J to 0 each time the outer loop is entered and then setting J equal to 1 if an exchange is made in the inner loop. A test at the end of the outer loop determines if an exchange has been made, if not, the program returns to the calling program; if an exchange is made, the program continues.

In the worst case (when the array is originally in descending order) it can be shown that this program requires $N(N-1)/2$ passes through the inner loop. However, in the best case (if the array is already in the correct

```
      SUBROUTINE SORT2(M,N)
      DIMENSION M(100)
      NEW=N-1
      DO 10 I=1,NEW
      J=N-I
C     SET A FLAG TO SEE IF EXCHANGES OCCUR
      IFLAG=0
      DO 5 K=1,J
      K1=K+1
      IF (M(K)  .LE. M(K1)) GO TO 5
C     EXCHANGE THE TWO ELEMENTS AND SET THE FLAG
      KSWAP=M(K1)
      M(K1)=M(K)
      M(K)=KSWAP
      IFLAG=1
    5 CONTINUE
C     SEE IF EXCHANGES HAVE BEEN MADE BY TESTING IFLAG
      IF (IFLAG .EQ. 0) GO TO 15
   10 CONTINUE
   15 CONTINUE
      RETURN
      END
```

Figure 11.4 Subprogram for bubble sort



order) the program requires only $N - 1$ passes through the inner loop. In general this program exploits the tendency of lists to be already somewhat ordered, thus, as a result, for large values of $N$ the program tends to be more efficient than the previous algorithm.

The sorting of files is so important in business and information processing that many studies have been made of sorting algorithms and many algorithms have been invented. In programming languages such as Cobol there is a SORT operation on a file which is a subprogram in the system just as SINE, COSINE, SQUARE ROOT, and other functions are provided in Fortran.

The studies of file-sorting techniques show that the number of operations required grows approximately as $N^2$ divided by some constant for "interchange sorting algorithms" such as those mentioned. More sophisticated sorting techniques such as "radix sorts" and "merge sorts" require $NK$ times some constant number of operations or $N^{1.5}$ times some constant operations, where $N$ is the number of elements and $K$ is the number of bits in each element. These sorting techniques require more complicated programs, with the result that they are generally not used for small numbers of elements.

More detailed introductions can be found in the book by Flores and the paper by W. A. Martin listed in the Bibliography.

For a really comprehensive treatment of sorting techniques the reader is referred to volume 3 in the series of books by Knuth which devotes nearly 400 pages to this subject. It is very difficult to evaluate mathematically many of the more complicated sorting techniques, and so quite often the algorithms are programmed, sample arrays (or lists) are sorted, and the results evaluated.[1] The book by Rich analyzes a number of the best sorting techniques by programming and evaluating the time required, storage required, and other factors for a number of sample problems.

Another important operation in maintaining files is that of *merging*. Two sets are said to be merged when they are combined into a single set, however, if the two sets have been ordered by sorting in ascending (or nondescending) order, the resulting set of items must also be sorted in ascending (or nondescending) order.

In order to study the merging operation, we shall again restrict ourselves to internal merging rather than merging files on tapes (the files could be copied into the core memory, if they are not too large). Let us consider two arrays of integers M1 and M2, with M1 having K1 elements and M2 having K2 elements. The arrays M1 and M2 are assumed to be sorted in nondescending order, and we wish to develop an array M3 with K1 + K2 elements which is sorted in nondescending order.

The first step in the algorithm is to compare M1(1) with M2(1) and place

[1] Dorn's book describes such sorts (giving Fortran programs) as the "monkey puzzle sort" and the "tour-

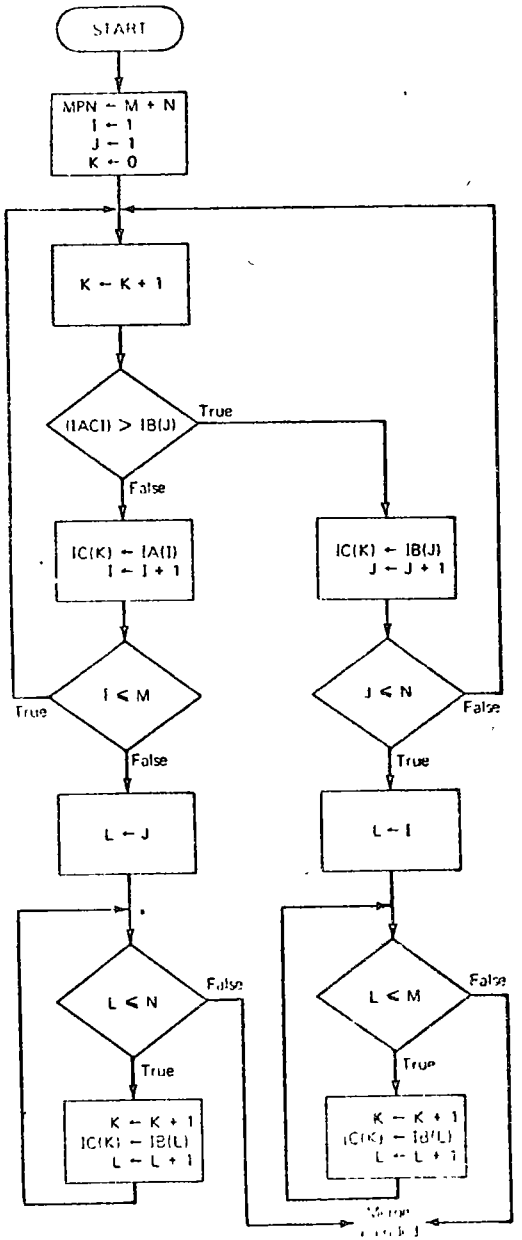



Figure 11 5 Flow chart of merge algorithm



```
      SUBROUTINE MERGE(IA,IB,IC,M,N,MN)
      DIMENSION IA(100),IB(100),IC(100)
      MN=M+N
      I=1
      J=1
      K=0
 5    K=K+1
      IF (IA(I) .GT. IB(J)) GO TO 10
      IC(K)=IA(I)
      I=I+1
      IF (I .LE. M) GO TO 5
C     ARRAY IA IS EXHAUSTED SO LOAD THE REST OF IB
      DO 15 L=J,N
      K=K+1
 15   IC(K)=IB(L)
      GO TO 20
 10   IC(K)=IB(J)
      J=J+1
      IF (J .LE. N) GO TO 5
C     ARRAY IB IS EXHAUSTED SO THE REMAINDER OF IA IS LOADED
      DO 25 L=I,M
      K=K+1
 25   IC(K)=IA(L)
 20   CONTINUE
      RETURN
      END
```

Figure 11 6 Subprogram for merge algorithm

the smaller of the two in M3(1). If the item M1(1) is selected we then consider M1(2) and M2(1), placing the smaller in M3(2), if M2(1) was selected in the preceding step, we consider M1(1) and M2(2), placing the smaller in M3(2) This basic process continues with M1(I) being compared with M2(J) at each step until we reach the end of either M1 or M2 [that is, we select either M1(K1) or M2(K2)] When this occurs, the remaining elements in the other array are simply copied into the remainder of M3.

Figure 11 5 shows a flow chart for this algorithm and Fig 11 6 shows a Fortran subprogram to merge two arrays (Languages such as Cobol provide MERGE subprograms in their basic library and a programmer can simply write MERGE A AND B rather than writing his own subprogram)

The exercises at the end of the chapter investigate the efficiency of the above program as well as show a short program for merging arrays with a special STOP element in the last position.

A *merge sort* is a sorting algorithm which breaks sets of elements into subsets, sorts the subsets, and then merges these sorted subsets By dividing the original set of elements into an appropriate number of subsets, so that the sorting is efficient, and then depending on the natural efficiency of merging, an efficient sorting algorithm can be obtained The exercises develop this procedure Knuth's book as well as several of the others in the Bibliography also treat this sorting procedure in detail.

## 11.4 SEARCHING A FILE

Probably the most frequently performed operation in business or information systems is that of searching the files for elements which satisfy some specified condition. The condition specified ranges from equality—for instance, "Find the record of the person with social security number 972-85-36."—to "Produce a list of the parts in our inventory which cost more than $45." In any case, a file must be searched, and it is important that the file be organized so that it can be efficiently searched. It is also important to have an efficient search algorithm so that too much computer time is not expended on searching.

The organizing of files and of search procedures are important topics in systems design for data processing systems. If large files must be maintained in mass storage devices and if these files must be regularly maintained by adding new records and modifying and deleting old records, and if it is further necessary to search the files frequently for records which satisfy some specified criteria, then a file organization must be found which is not too expensive to update, which does not require too much storage space, and which can be conveniently searched. Designing a good data structure for the file and programming efficient file maintenance and search algorithms are interesting and still developing aspects of data processing systems design.

We shall first examine a particular aspect of file searching and maintenance, that of finding a specified element in an array. This will include showing how the search algorithm can be greatly speeded up if the array is sorted.

If a table or array is searched for a given item, the search *succeeds* if the item is found and *fails* if it is not found. If we store our table in an array which we call ITEM, at a given time ITEM will contain N elements. If these elements are not ordered, in order to find a given item in the array, the most natural search algorithm is to examine ITEM(1), ITEM(2), ITEM(3) and so on up to ITEM(N), each time seeing if the value is the desired one.

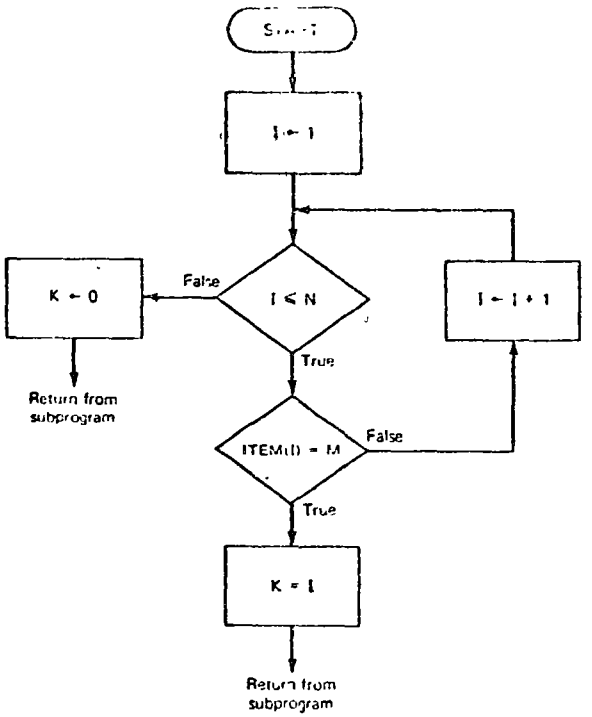
In order to convert this procedure to flow chart form, we let the values in ITEM be integers and call the value to be found M. A flow chart of the resulting search algorithm is shown in Fig. 11.7*a*. An integer variable K is set to 0 if the search fails; otherwise it will have a value such that ITEM(K) = M.

A Fortran subprogram implementing this algorithm is shown in Fig. 11.7*b*.

The question now arising is, "How efficient is this algorithm?" It is clear that if the desired value is not in the array, N steps or passes through the search loop will be required. If, however, the item is in the array (and we are not required to find duplicate values in the array), on the average *N*/2 passes through the array will be required.

Now let us assume the array has been sorted so that the elements are in numerically ascending (or nondescending) order. We now sequence through the array, starting with ITEM(1) and proceeding through ITEM(2), ITEM(3), etc., in turn as before. However, at each step we also test to see if






Note. This algorithm searches an array ITEM of N elements to find an element M. If for some I does ITEM (I) = M, then I is placed in the variable K; if for no I does ITEM (I) = M, then K is given the value 0.

(a)

```
      SUBROUTINE FIND1(M,ITEM,N,K)
      DIMENSION ITEM(100)
      DO 5 I=1,N
      IF (ITEM(I)-M) 5,10,5
    5 CONTINUE
      K=0
      RETURN
   10 K=I
      RETURN
      END
```

(b)

Figure 11.7 Searching a linear array. (a) Linear search algorithm. (b) Subprogram to search a linear array.

again element ITEM(I) is greater than MATCH, and if it is, the search is immediately terminated, since the element is not in the array. This will improve the efficiency of the search procedure since the entire pass through the array will not be required in cases where an element is not in the array. (Also, if duplicates are required, the entire array need not be searched.) A program for this algorithm is shown in Fig. 11.8.

An even more efficient algorithm for searching a sorted array is the *binary search algorithm*. It closely follows the procedure used by most people in trying to locate a name in a dictionary, index card file, or other alphabetically ordered file. The file is opened to its middle, dividing it in half, the middle entry is examined to see in which half the desired item would be, and that half is again opened to the middle, and an examination then indicates in which half the item lies, this process continues until the particular item is located.

In order to make the above procedure more precise, we assume a table of N items, and we shall refer to these as ITEM(1), ITEM(2), . . . , ITEM(N). If N is even, we can divide the table into two equal sets of $N/2$ items and then determine in which half the value to be located might lie. If N is odd, the table is "divided" into two sets, one with INT(N/2) and one with INT(N/2) + 1 items, where INT is the "integer" function which takes a number X into the largest integer which is not greater than X. [In Fortran IFIX(X) is a "compiler supplied" function which takes a real value X into the largest integer not greater than X. If we write I = N/2, however, I will have the value we call INT(N/2) since the value of I will automatically be truncated, not rounded, to an integer value.]

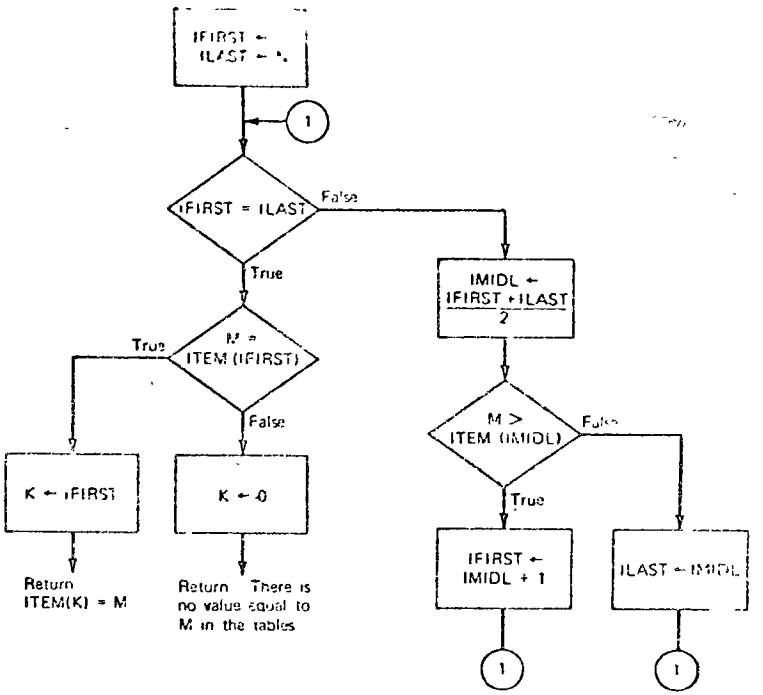
The largest element in the "lower half" of the table with the lowest values is now examined to see in which part of the table the desired item lies, this is done by seeing if the desired element has a value greater than the largest element in the "lower half." If so, the upper half is selected; if not, the lower half is selected. This step continues until either the item is found or the discovery is made that no such item is in the table. Figure 11.9 shows a flow chart of this algorithm and Fig. 11.10, a Fortran subprogram.

In both the flow chart and the program, an array called ITEM containing N elements is searched for an item called M. An integer variable K is set to 0 if the value M is not in the array. The variable K is set so that ITEM(K) = M if M is in the array.

```
      SUBROUTINE FIND2(M,ITEM,N,K)
      DIMENSION ITEM(100)
      DO 5 I=1,N
      IF (ITEM(I)-M) 5,10,5
    5 CONTINUE
      K=0
      RETURN
   10 K=I
      RETURN
      END
```

Figure 11.8 Linear search of a sorted array






Note: This algorithm searches a table called ITEM with N entries. The algorithm searches for an element called M, and if this element is in the table there sits an integer value in K such that ITEM(K) = M. If no element in the table called ITEM is equal to M, then the value 0 is placed in K.

Figure 11.9 Binary search algorithm

```
      SUBROUTINE FIND3(M,ITEM,N,K)
      DIMENSION ITEM(100)
      IFIRST=1
      ILAST=N
      GO TO 5
   10 IMIDL=(IFIRST+ILAST)/2
      IF (M .GT. ITEM(IMIDL)) GO TO 15
C     ITEM IS IN LOWER HALF OF TABLE
      ILAST=IMIDL
      GO TO 5
C     ITEM IS IN UPPER HALF OF TABLE
   15 IFIRST=IMIDL+1
    5 IF (IFIRST .NE. ILAST) GO TO 10
      IF (M .EQ. ITEM(IFIRST)) GO TO 20
      K=0
      RETURN
   20 K=IFIRST
      RETURN
      END
```

Figure 11.10 Subprogram for binary search algorithm

How efficient is the binary search algorithm? The efficiency of the algorithm is most easily evaluated for tables with $N$ entries, where $N = 2^I$ for some $I$ (that is, tables with 2,4,8,16,32, etc., entries), for then the table is reduced to one-half its original size the first step, one-fourth its original size the second step, and so on until it is reduced to a single element, which either is or is not the item desired. Since our rule is that after $I$ steps the table has been reduced to a subtable of size $2^{-I} \times N$, and since $N = 2^I$ for some $I$, then when $2^{-I} \times 2^I = 1$, each of the two parts will contain a single element and one of these must be the desired element. Thus, for a table with $N$ elements, where $N = 2^I$ for some integer $I$, exactly $I$ passes through the basic loop are required to find an element using the program or flow chart which has been shown. This value is less than $N$ for $N$ greater than 2 and is much less than $N$ for large $N$.

For any given $N$ which is not a power of 2 (that is, not equal to $2^I$ for some $I$), we can determine the number of passes through the loop as follows. Let $K$ be the smallest integer such that $2^K \geq N$, then $K$ passes through the loop are required.†

Can we improve on this algorithm? It might seem that before each pass through the loop it would be a good idea to test whether the largest element in the lower half is indeed the item desired and to terminate the search if it is. This would mean that sometimes the search could be terminated early. For instance, if we had a table with four elements which were the integers 1,3,7,9 and we search it for the integer 3, then the table is divided into two parts, 1,3 and 7,9. The greatest integer in the lower half, which is 3, is selected, and, instead of seeing if the desired item 3 is greater than 3, we also test to see if it is equal to 3, and stop if it is. In this case it is the desired element, and only one pass through the major loop has been made, whereas the previous algorithm required two.

It can be shown, however, that on the average the number of passes through the major loop is decreased by only 1. That is, if $L$ passes are required on the average for the first binary search algorithm, then $L - 1$ passes are required for the second algorithm. If $L$ is large, the additional time required to test for equality, which lengthens the basic loop, will probably exceed the time required for a single pass through the loop, and the original procedure will run faster on the average.

The above analysis is typical of that performed on algorithms for system use. In many cases, it is impossible to completely and precisely analyze subtle points, and it is either necessary to test the algorithms with a number

†In general, if we define $\log_2(N)$ to be a function with value $X$ such that $2^X = N$, then $\log_2(N)$ is called the *binary logarithm* of $N$, and if $X = \log_2(N)$, $X$ will be some positive real number for each positive integer $N$. Now in the general case it can be shown that the number of passes required is exactly CEIL ($\log_2(N)$), where CEIL has value the smallest integer larger than or equal to $\log_2(N)$. This shows that the number of passes through the loop increases at a logarithmic rate.



of test cases and compare results or to generate simulated data, try the algorithms on the simulated data, and compare results.

## 11.5 ARRANGING ARRAYS AND LINEAR LISTS IN MEMORY

The structure of the data which are processed in business systems can be quite complicated. Further, in fields such as information retrieval, the quantity of data used is quite large, and the structure of the data is quite complex. The manner of organizing the data in the computer's memory can be very important when large volumes of data which are interrelated in a complicated way must be processed.

The simplest and most direct way to organize data in a computer memory is by placing them in a one-dimensional array. Consider a table or array of data consisting of N items. By using the name TABLE, the items can be referred to as TABLE(1), TABLE(2), through TABLE(N). If the memory of the computer is word-organized and each item in the list can fit in a single word in memory, then by assigning a starting address in memory for the table, say B, we can find a given item TABLE(I) with index I by looking at address B + I − 1 in the memory. This organization is shown in Fig. 11.11a.

***Example*** We assign the starting location B to location 4000 in memory. Now if the items in the array are referred to as TABLE(1), TABLE(2), through TABLE(100), the values in TABLE will be stored in locations 4000 through 4099 in memory, and TABLE(1) will be at location B + 1 − 1, which will be 4000 + 1 − 1 or 4000, TABLE(2) will be at 4000 + 2 − 1, which is 4001; TABLE(50) will be at the address 4000 + 50 − 1, etc.

The basic idea of placing the words in an array or linear list in consecutive addresses in memory can be extended to arrays with two (or more)

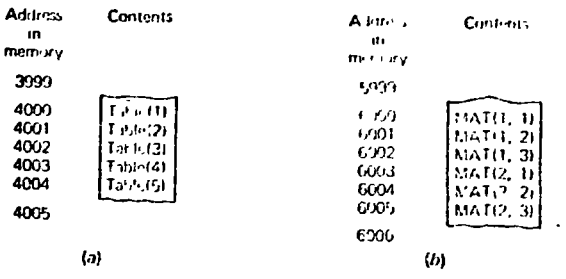



**Figure 11.11** Organization of one- and two-dimensional arrays in memory. (a) Storing an array with five elements of consecutive addresses in memory. (b) Storing a 2-row, 3-column array in memory (by row).

dimensions or indexes Consider an array called MAT with four rows and three columns If each entry in MAT requires a single word in memory, then 12 words will be required If we choose address 200 to be the address of the MAT(1,1), then 211 will contain the last entry MAT(4,3) There are two ways to "lay in" the array, however, by *rows* or by *columns* Let us use rows,[†] the elements will then appear in memory in this order MAT(1,1), MAT(1,2),MAT(1 3) MAT(2,1),MAT(2,2),MAT(2,3),MAT(3,1), and so on until MAT(4,2),MAT(4 3) If MAT(1,1) is at address 200,MAT(1,2) will be at 201, MAT(1,3) at 202, etc To find the address of an element MAT(I,J) in memory, we calculate as follows. MAT(I,J) is at memory address 200 + 3(I − 1) + (J − 1) Thus, to find MAT(3,2), we form 200 + 3(3 − 1) + (2 − 1), which is 207

The general rule is: For an array with M rows and N columns with one word per element in memory, and where the first element is to go in address B, the element in the *I*th row and the *J*th column goes at address B + N(I − 1) + (J − 1) This organization is shown in Fig 11 11*b*.

Similar rules can be derived for arrays with more than two indexes and for arrays where elements require more than a single word in memory. Notice the ease with which an element can be located in an array stored in the above manner. Arrays are very desirable data structures

The tables or arrays which have been considered above have contained items of fixed size Many data are alphanumeric, consisting of names, addresses, written text, etc The individual items in this kind of data have mixed lengths For instance, suppose we wish to store a list of names These names will be of differing numbers of alphanumeric characters, which obviously complicates the problem.

A simple solution is to allocate a number of locations in memory sufficient to hold the longest name and then to allocate each name that amount of space.

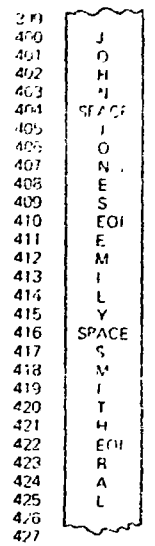
*Example* A list of 200 names is to be kept in memory The computer, an IBM System/370, stores one character in a single address (a byte-per-address) The longest name has 20 characters in it The list is to begin at location 1000 We allocate 20 × 200 addresses in the memory so the list will start at 1000, and the last character will be at location 4999 If we call the list NAME, the *k*th element in NAME [that is, NAME (*k*)] will begin at 1000 + 20*k* − 1 and end at 1000 + 20*k* + 19

If the names are of widely varying lengths, the above technique causes an inefficient use of memory Fortunately, several alternatives are possible One is to have a special END OF ITEM character and to place this character after each name[‡] This is shown in Fig 11.12. Another is to keep a table of

[†]It should be noted that by official decree Fortran arrays are stored by listing columns (rather than rows) in consecutive order This is discussed in the exercises at the end of the chapter

[‡]This is a special character In this case it is used for end of name The special character's name just happens to be END OF ITEM






Note EOI is the END OF ITEM character
The memory stores one character at each address

Figure 11 12 Storing character strings using END OF ITEM characters

the addresses at which each name begins This is shown in Fig 11 13*a* Still another technique is to have a starting address and a table containing the number of characters in each name as shown in Fig 11 13*b* (We then find the location of element I by adding the lengths of the names preceding I at the starting address in the table)

Which technique should be used depends on the characteristics of the data, the computer word length and the premium on memory space and search time The last technique (that in Fig 11 13*b*) uses less memory, in general, but requires more time to find an item

Judgment as to which technique to use must be made dependent on the particular file characteristics For example, an inventory control system file might consist of a set of records, where each record consists of several items Figure 11.1 shows a section of such a file, where each record consists of the license number, make, year, name of owner, street address of owner, town in which owner resides, and state in which owner resides Since these particular items are either numbers or alphanumeric strings of limited variations in length, it would be practical to store these records in a fixed-length format with a fixed number of computer words in memory for each record

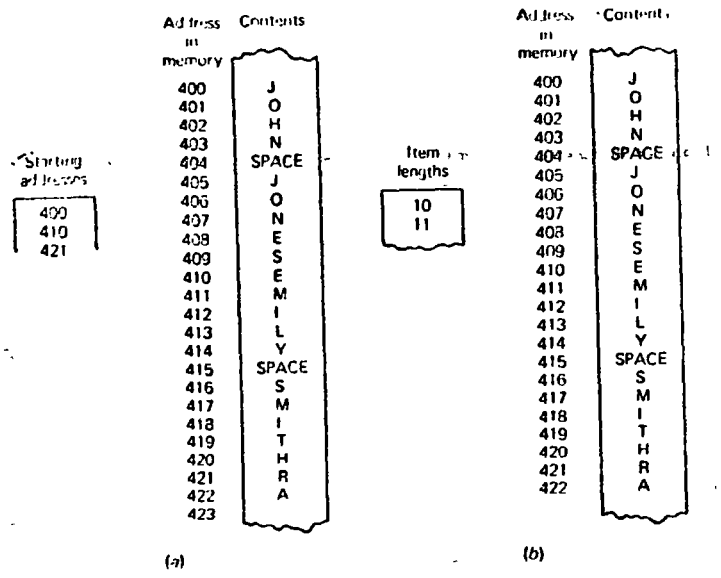



Figure 11.13 Techniques for storing lists with items of different lengths. (a) Table with the address of first character in each item (b) Table with length of each item

When lists of records with each record containing several items are sorted, they are sorted on a key which consists of one of the items. Thus, the file shown in Fig 11.1 might well be sorted with the license number as the key. Then, to locate a particular record given the license number, a binary search would be performed on the list If the list were sorted on license number and we wished to search the file for some other attribute, a sequential search would be necessary (Sometimes tables or "dictionaries" which list attributes-versus-key or location in memory are used to cut down the search time Files organized in this way are called *inverted files* Refer to the Bibliography for works giving details.) For example, if the file were sorted using license numbers as a key and we wished to obtain the names of all car owners with Cadillacs from some particular town, it would be necessary to search through the file sequentially, looking at each record in the file to see if the owner were from the town and owned a Cadillac.

If a file consisted of a set of records, with each record containing sentences in the English language (as might be the case in an information retrieval system), the varying lengths of these sentences would make a storage technique involving use of an END OF ITEM character or one of the table techniques for keeping track of the ends of the sentences more efficient.



## 11.6 STACKS, DEQUES, AND QUEUES

A useful data structure in computers, particularly in systems programs is called a *stack* The name *stack* is derived from the fact that the data items are arranged in a stack in memory which resembles a stack of plates in a cafeteria In other words, the items are stacked one on the other With a stack the assumption is made that only the last item placed on the stack is immediately available. This shows the resemblance to a stack of plates Only the top plate is immediately available Similarly, items can be placed only on the top of the stack (This "last-in first-out" principle leads to stacks sometimes being called *LIFO lists*) Placing an item on top of a given stack (that is, adding an element to the stack) is called *pushing*, and removing an item from a stack is called *popping* As an example, suppose we have the set of items 1,2,3,4,5,6 arranged in order, a stack called STACK, and two operations on the stack PUSH and POP. If each item is placed in order in a new ordered set when it is "popped" or removed from the stack, then the sequence of operations PUSH, PUSH, PUSH, POP, POP, PUSH POP, PUSH, POP, PUSH, POP, POP will result in 3 2 4,5,6 1 being the order in the new ordered set. Figure 11.14 shows examples of pushing and popping

Stacks are very useful in managing complicated sequences of interrupt or other unpredictable operations Placing information concerning program state on a stack where it can be later found is a convenient way to manage task sequencing in system programming Stacks are so convenient for systems' use that several computers have automatic stacking of interrupt information.

A *queue* is similar to a stack except that new items are added to the top of the list but items are removed from the bottom Thus, a queue is operated on a first-in first-out (sometimes called FIFO) basis just like a waiting line

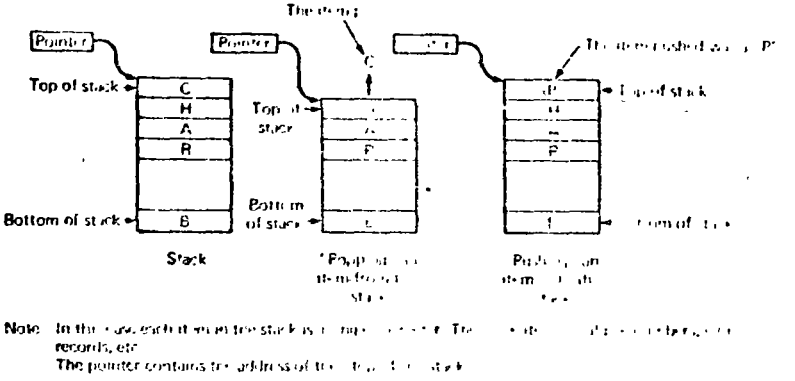



Figure 11.14 Stack operations

or queue for a theatre Queues are useful in managing tasks when it is desirable to service requests on a basis where "those who have waited longest" get attention first.

A *deque* is simply a linear list where it is possible to add or remove items from both ends

Some applications for stacks are developed in the exercises at the end of the chapter. The algorithms in Sec 11 7 use stacks

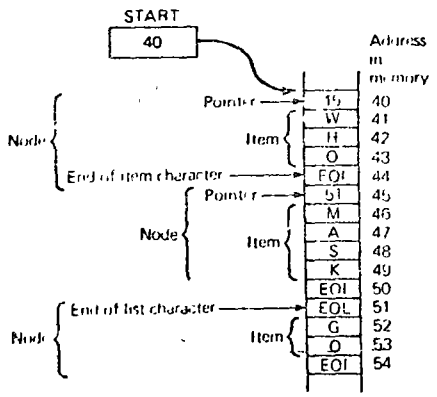
## 11.7 LINKED LISTS

The most natural and efficient method for maintaining data in a computer memory is the one-dimensional array, which is often called a *linear list* by those working with data structures There are problems, however, with maintaining linear lists when many additions and deletions are necessary and the lists must be maintained in order.

For instance, in some systems where textual data must be frequently searched and strings of characters must be added, modified, and deleted, a data structure called a *linked list* has been found to be useful Insertions and deletions can be easily made in a linked list, searches are moderately efficient, but the data structure requires more storage space than in linear lists

A linked list essentially consists of a set of *nodes* Each node has two parts: a *pointer* and a *data item*† The pointer gives the (beginning) address of another node while the data item contains the actual data at the node.

†The term *data item* is used here instead of simply *item* to help differentiate the data part from the pointer part




Note The address in START gives the address of the first item in the list
Each pointer 'points' to or gives the address of the next node in the list

Figure 11 15 Linked list arrangement in memory



This is shown in Fig 11 15 In actual practice the pointer will probably not contain the full address of the next node, but will simply contain an index into an array containing the actual addresses or perhaps an amount to be added to some beginning address to form the actual address

In order to manage a linked list two other things are necessary (1) The address of the first node must be stored somewhere so we can enter the linked list (2) A special value or symbol which we shall call EOL (for END OF LIST) must be placed in the pointer of the last node in the linked list (This could be a negative number or 0 if addresses or indexes are used as pointers, since either value would indicate that this was not a valid pointer to another node)

In order to represent a linked list diagramatically, we use the technique shown in Fig 11 16 Here each node is shown as a rectangle containing two parts, the data item and pointer The START box gives the address of the first node (a special value in the START box will indicate that there are no nodes in the list) An arrow on the drawing indicates to which node the pointer points

The addition or deletion of nodes is straightforward for a linked list, and herein lies their primary advantage. To add a node, we simply place the node in memory at some convenient unused address Then the pointer in the node which is to precede the new node is changed to give the beginning address in the new node, and the pointer in the new node is set to the address of the node which is to follow it

The deletion of a node is also straightforward The pointer in the node preceding it in the linked list is simply changed to the address of the node which follows the node to be deleted Examples of adding and deleting entries are shown in Fig 11.17

A problem arises after many changes have been made to a linked list If a number of nodes have been deleted there will be "holes" in the actual memory space used because entries which have been deleted will still reside in memory even though they are effectively "dead" In order to use this space, sometimes "garbage collection" programs are written to collect

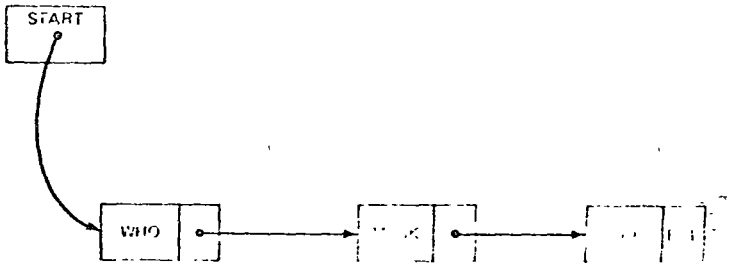



Figure 11 16 Symbolic representation of a linked list

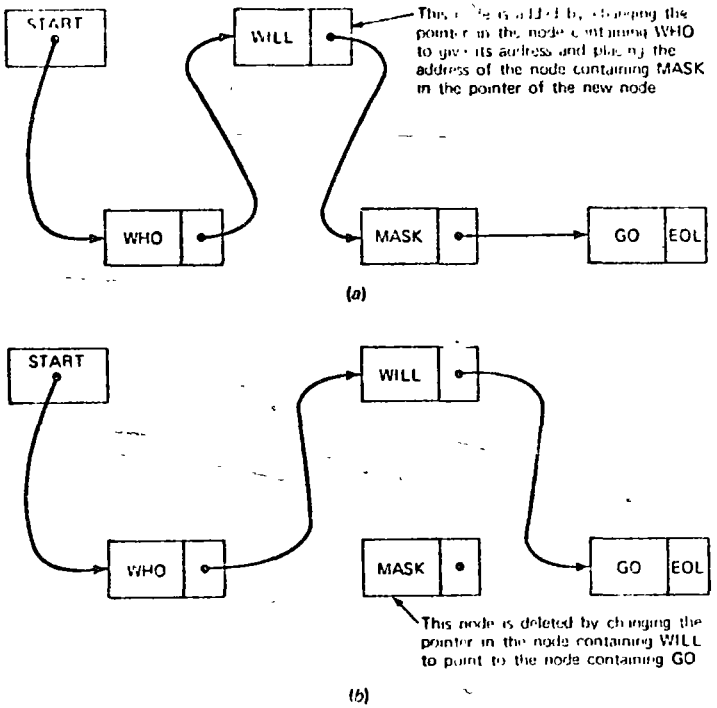



**Figure 11.17** Adding and deleting nodes in a linked list (a) The linked list in Fig 11.16 with an added node (b) The linked list in (a) with a node deleted

lists of "free space" which can be used for new entries Similarly, some systems maintain lists of free space continuously so that memory is not wasted A particularly advantageous strategy consists of maintaining the list of free space in a stack and pushing or popping nodes from the stack as nodes are released from or added to the linked list, respectively More details of these procedures can be found in the exercises and still further details in the works in the Bibliography.

Note that a linked list must be searched sequentially, even though it is maintained in sorted order, because the entries are not actually in order in the memory (Separate tables can be kept to facilitate searching, but this involves using even more space in memory)

## 11.6 TREES

As can be deduced from the preceding descriptions of data structures and the algorithms for searching and for deleting and inserting new items, the



... structure employed has a profound effect on ... efficiency of the algorithm used and the amount of memory required ... particular class of data structures called *trees* provides for reasonably efficient search algorithms and insertion and deletion algorithms, with the penalty of additional memory required for implementing the structure

The term *tree* derives from the name of the familiar "woody perennial plant,"[1] but usage in computer science appears to derive from a mathematical structure in graph theory A *directed graph* is a collection of *nodes* and *branches* with each branch connecting two nodes Each branch indicates a direction (an arrowhead is used at one end of the branch in our figures)[2] Figure 11.18 shows a drawing of a graph with several nodes and branches. The position of the arrows on the branches is important, for if two nodes are connected by a branch, the node at the end of the branch with no arrowhead is called the *predecessor* of the node at the end of the branch to which the arrow points Further, if node $A$ is the predecessor of node $B$, then $B$ is called the *successor* of $A$

In order for a set of nodes and branches composing a directed graph to qualify as a *tree*, two other properties are required

1 There is a single node which has no predecessor This node is called the *root*
2 Every node excepting the root is connected to the root by a unique *path* where a path consists of the branches connecting a sequence of nodes $N_1, N_2, \ldots, N_M$ where $N_i$ is the predecessor of $N_{i+1}$ for all $i$ from 1 to $M-1$ (Notice that there can be only one path between two nodes if the path is unique)

[1] This is from the definition in "Webster's New Collegiate Dictionary" G & C Merriam Co Springfield Mass

[2] The graphs in Fig 11.19 are all *directed* graphs Undirected graphs (or just plain graphs) have no direction to the branches Consider a road map The cities are like the nodes and the roads are like branches for an undirected graph since we can drive in either direction

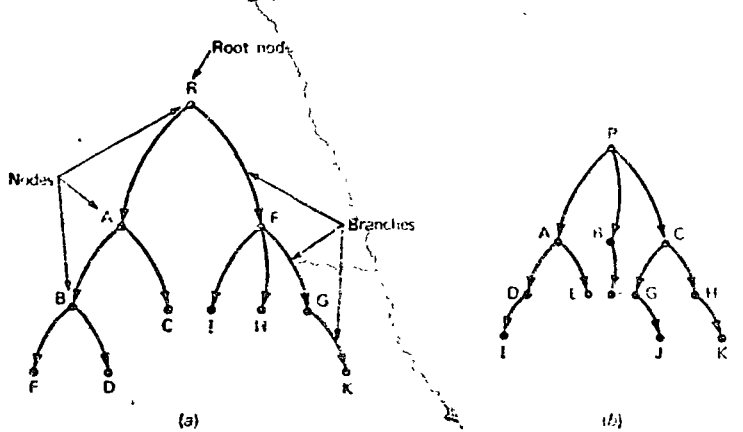



**Figure 11.18** Trees

Figure 11.18 shows several trees For the tree in Fig 11 18*a*, a path from the root node *R* to the node *D* follows the nodes *R,A,B,D* Does this satisfy requirement 2 above? Yes, for *R* is the predecessor of *A*, *A* is the predecessor of *B*, etc

There is a considerable literature in mathematics concerning trees Several of the works in the Bibliography cover this material

In computer usage, a most important kind of tree is called a *binary tree* It has the characteristic that each node has at most two successors Figure 11.19 shows several binary trees

One reason that the binary tree is so useful as a data structure for computers relates to the means of storing a tree structure in a computer memory. Let us assume that we are to store a number of data items in a memory using a binary tree structure First we shall represent the data structure to be used as the graph of a binary tree. Each data item will be stored at a node on the binary tree. For convenience of description we shall assume the data items to be simply English letters, and we can then use each letter as the name of a node Figure 11.19 shows this arrangement

We can now store this tree in memory by associating with each node on the graph several words in memory which contain (1) the data item at the node and (2) *two* pointers, one to the leftmost successor node and the other to the rightmost successor node on the graph A START pointer which gives the address of the root node will be required so we can enter the tree, and a special pointer value will also be required which can be placed in any pointer and which indicates that no (leftmost or rightmost, whichever the case may be) successor exists An example of this arrangement is shown in Fig 11 20[†]

[†]As before, the pointers may be simply indexes into an array, values to be added to some base value, etc., thus saving memory space

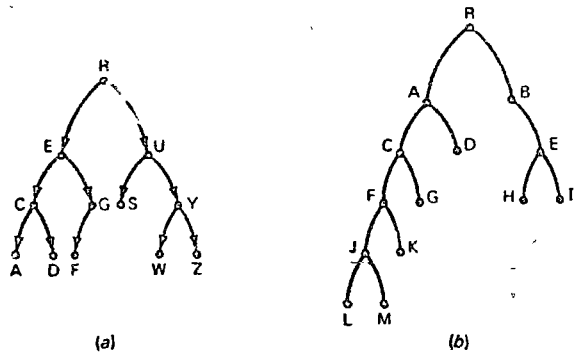



Figure 11.19 Binary trees



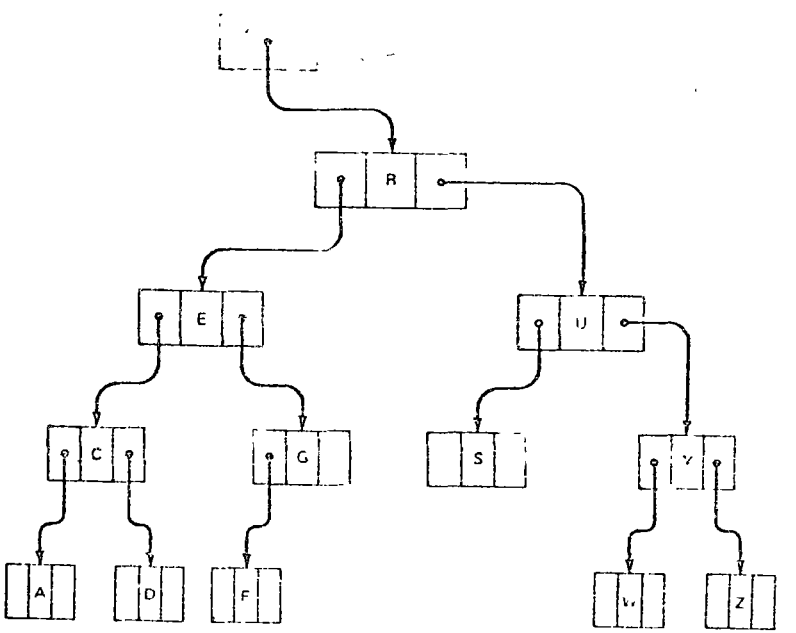



Figure 11 20 Data structure for binary tree in Fig 11 19*a*

As may be surmised, the storing of two pointers with each data item is somewhat uneconomical in the use of memory The compensating factors lie in the ease with which items can be entered into or deleted from the binary tree and the facility with which the tree can be searched (provided the nodes are correctly arranged)

Notice that the tree in Fig 11 20 has the nodes ordered[†] in a specific manner The leftmost successor node always precedes its predecessor in alphabetical order, while the rightmost successor always follows its predecessor in alphabetic order When a tree is arranged in this way, it is considered to be sorted

An important use of trees in the data structures for files involves a tree where each data entry is a single symbol This structure is call a *symbol tree* Figure 11 21 shows a graph of a symbol tree for several English words At each leaf in the tree the address of the record for the word which terminates in the leaf is stored Thus, to find the record corresponding to a given

[†]A *leaf* in a symbol tree is a node at the end of a wo d

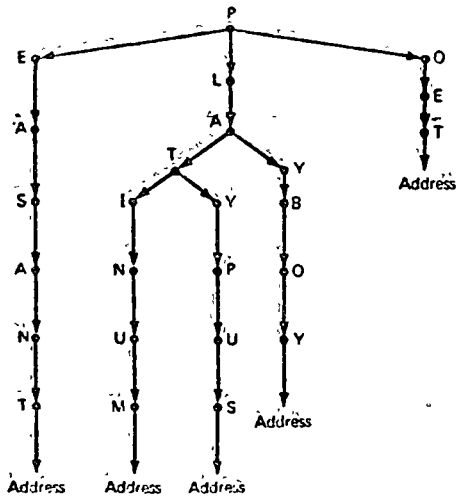



Figure 11.21 Symbol tree

word, we search the tree for the word, and if it is found, the record for the word would be pointed to by the pointer in the last symbol in the word.†

An important point can be noted here. The tree in Fig. 11.21 has several nodes with more than two successor nodes and is therefore not a binary tree. The graph can be stored as a binary tree, however, as shown in Fig. 11.22. (Notice the method for handling nodes with more than two successors.)

(The trees in Figs. 11.21 and 11.22 could be used in a directory for an *inverted file*, see questions 6 and 7.)

In order to search trees such as those in Figs. 11.21 and 11.22, one can use a version of the binary search algorithm. The search is almost as efficient as the binary search for linear lists or arrays. The insertion or deletion of a name of word in these files is straightforward and efficient characteristics that compensate for the additional search time for many files of interest.

In order to be searched efficiently, a binary tree must be maintained in what is called *balanced form*. The exercises that follow investigate this subject along with algorithms for balancing trees, searching trees, and inserting and deleting nodes from trees.

†A possible use is to have the words be those in a dictionary and the records be their definitions. Or the words might be the Dewey decimal notation for books in a library and the records the description of the books.



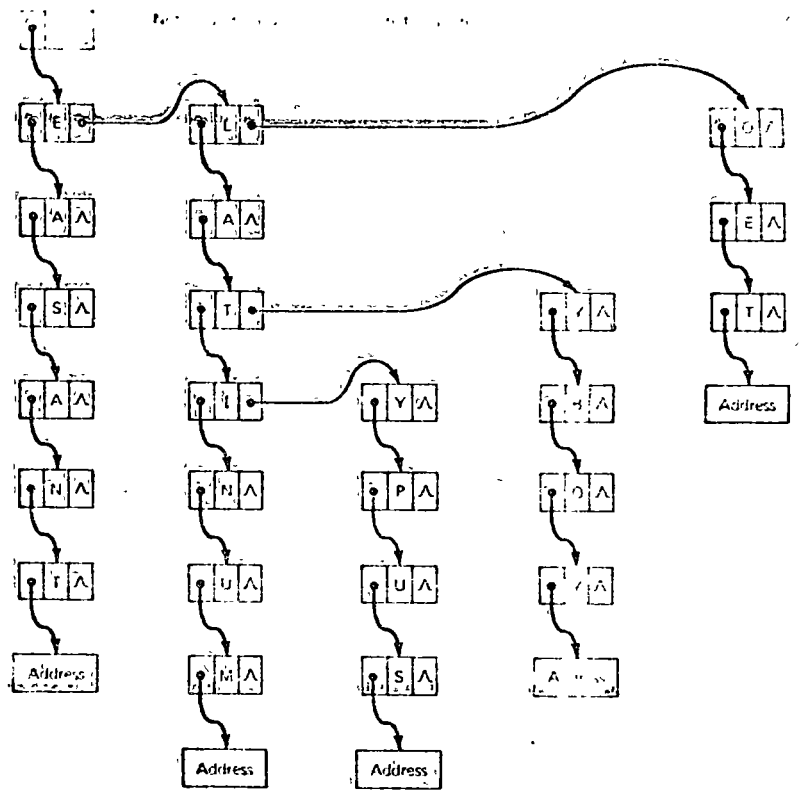



Figure 11.22 Data structure for symbol tree

## EXERCISES

1. For Fig. 11.1 list the record relating to F. R. Jackson.
2. A good key for Fig. 11.1 would be license number. If we use name of owner for key, a key may not be unique. Why?
3. Give the collating sequence for letters and numbers in the ANSI (or ASCII) code in Chap. 2.
4. Give the collating sequence for letters and numbers for the EBCDIC code in Chap. 2.
5. Discuss sorting a list of names in EBCDIC or ASCII using the numerical values as numbers (integers) in the sort routine.
6. To search the file in Fig. 11.1 for a record given the license number we would probably keep the file sorted with license number as a key. If many demands were made to search the file for "name of owner," it would be convenient to

# Chapter 6 Trees

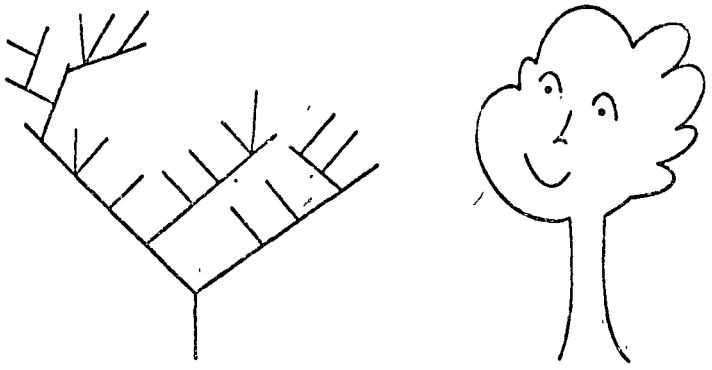
## 6·1 Tree examples

The use of flowcharts to represent algorithms has helped us to recognize their underlying *structure.* Furthermore, attention paid to the structure of an algorithm usually results in a better understanding of the computational process, and often results in our recognizing alternatives and potential improvements to the original design. Similar rewards result from attention paid to the structural relationships among the components of a set of data.

There is, in fact, a close connection between the steps we need to express an algorithm and the way we choose to think about or *represent* the data that are to be transformed by that algorithm. Experience in constructing algorithms fosters an increased appreciation of this interdependence. You will gain some of this experience by studying the next several chapters. Your ability to analyze the structure of a set of data and how alternate representations of it can affect algorithms using such data, undoubtedly will improve. We have already considered two structural forms for data, lists and arrays. Another type of structure is called a *tree* (Figure 6·1). Tree structures are important in representing certain types of data and, oddly enough, the essential steps of a number of algorithms exhibit a treelike structure.

First we will give two simple examples of a process whose strategy of execution (algorithm) can be pictured as a tree and two examples of data that can be pictured as a tree. Later, we will tackle three fascinating problems, the first one at the end of this chapter and the other two in Chapter 7. When we have finished this study, we may claim the title *tree expert.*

Let us agree now, before we get too far along, that trees in this chapter will be drawn upside down (Figure 6·2). We



This is a tree. This is not a tree.

FIGURE 6·1

do this only because it is convenient. You have to be willing to think of a tree growing toward the earth, its trunk "hanging" from the sky.

### Example 1. Treelike Algorithms

Our first example shows how we can represent the 16 conclusions to the well-known eight-coin problem as a "decision tree." The problem is this. You are given eight coins, a, b, c, d, e, f, g, and h, and are told that they are all of uniform weight except one, which is either heavier or lighter than the others. You are given an equal arm balance, but you may only use it three times for comparing coins or groups of coins. Your job is to determine the maverick coin and whether it is lighter or heavier than the rest.

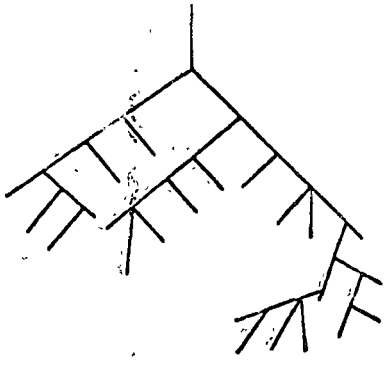


FIGURE 6·2
This is an upside-down tree.

Here is a strategy to use (see Figure 6·3) for all possible cases.

1. Compare the weights of two subsets of equal numbers of coins and consider the significance of the *three* possible outcomes. If the weights of the two subsets are equal, the coin in which we are interested cannot belong to either of the compared subsets.

2. Once we have isolated a pair containing the "odd" coin and we want to know whether one of them is heavy or light, we weigh one of the two candidates against any other that is known to be "standard."

There are 16 possible cases, each of which may occur, given the eight labeled coins. The algorithm shown in Figure 6·3 has a treelike structure. Conclusions are reached by follow-

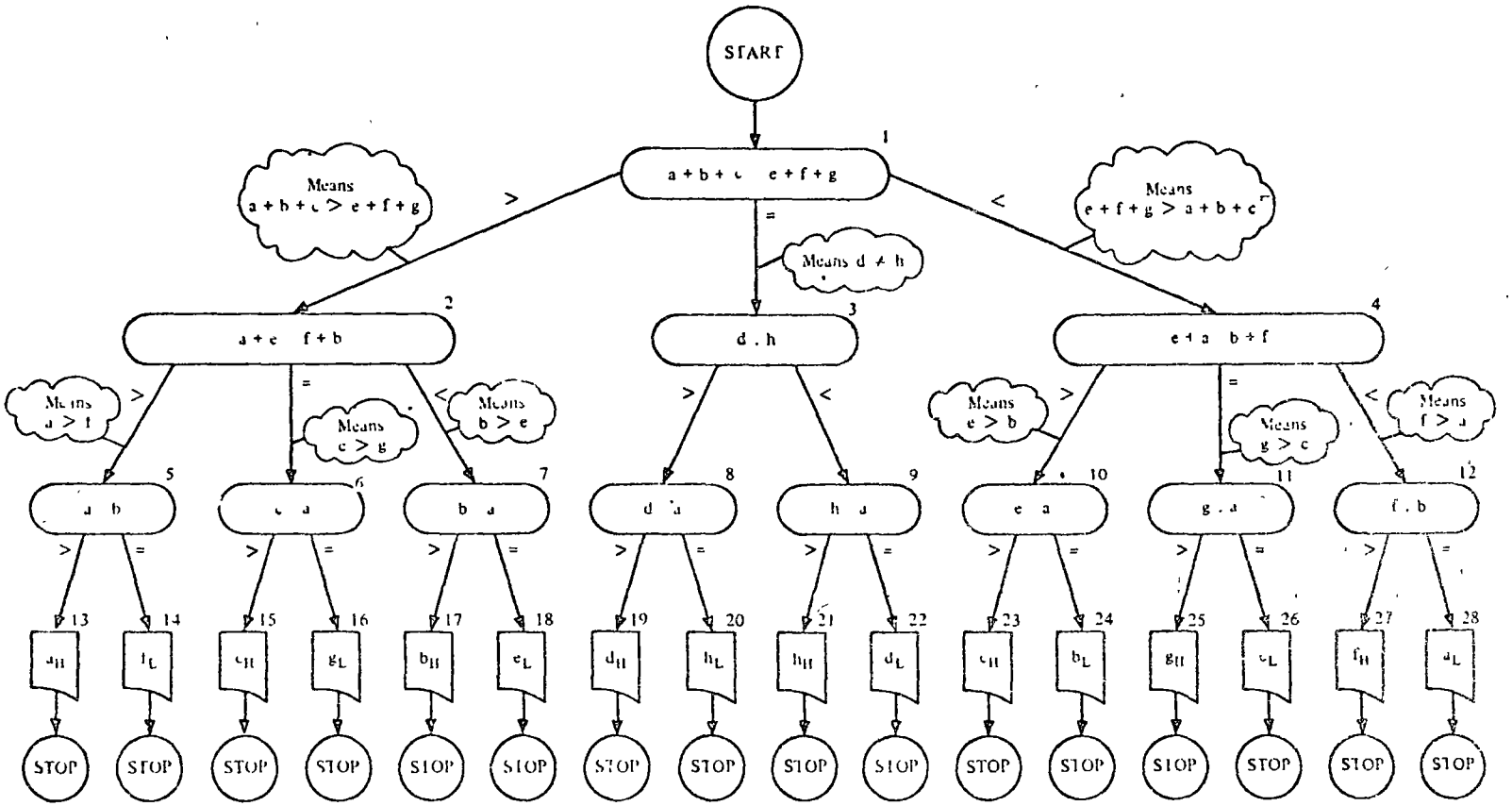
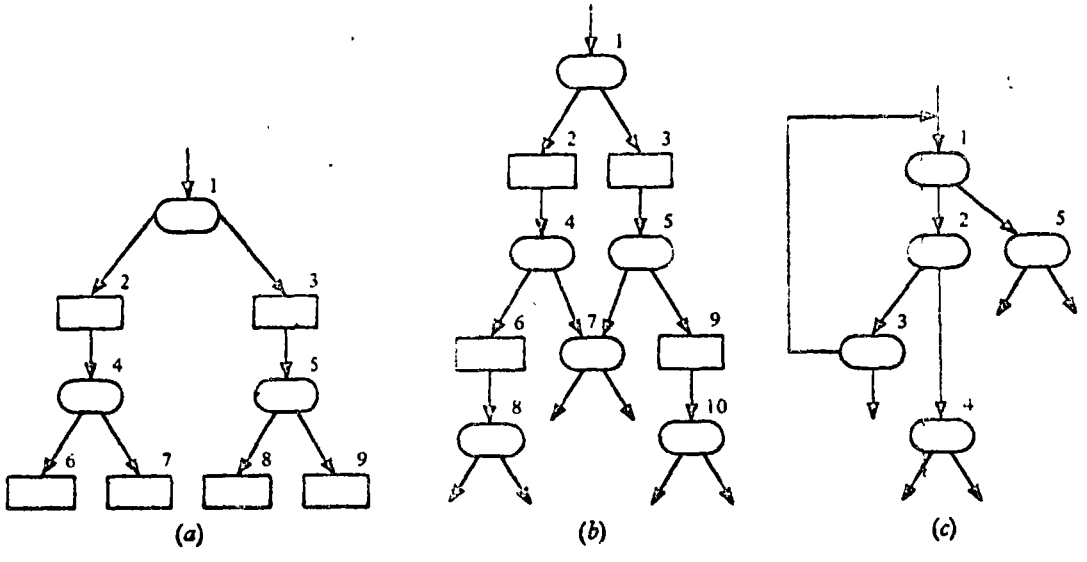



FIGURE 6·3
Tree diagram of a strategy to identify the odd coin. Conclusions are subscripted: subscript H means *heavy;* subscript L means *light*

ing a unique path (a sequence of three weighings) from the top or *root* of the tree diagram to one of the terminal boxes or *leaves* at the bottom.

EXERCISES 6·1, SET A

1. For decision box 2 in Figure 6·3, explain why:
   (a) If the relation a + e > f + b is true, one may conclude that a > f.
   (b) If the relation a + e = f + b is true, one may conclude that e > g.
   (c) If the relation a + e < f + b is true, one may conclude that b > e.
2. Explain why b may be regarded as a "standard" coin at decision box 5 but not at decision box 7.
3. Suppose you are given 12 seemingly identical balls and are told that one ball is *heavier* than the others, which are of the same weight. Draw the tree diagram algorithm to identify the heavy ball in three weighings.
4. Suppose you are given 12 seemingly identical balls and are told that one ball is *different* in weight (either heavier or lighter). Draw a tree diagram algorithm to identify the odd ball and to determine whether it is heavier or lighter in three weighings.
5. Are all decision sequences tree structures? Consider the three flowcharts below.




   (a) Which of these are tree structures?
   (b) Consider the following attempt to define a tree structure.

(1) A node having no segments extending from it is a *terminal* node.
(2) A node having one or more segments extending from it is a *nonterminal* node.
(3) A tree structure is a terminal node *or* a nonterminal node whose segments consist either of terminal nodes or tree structures.

What corrections or additions, if any, are needed in the above definition so that, when applied to flowcharts *a*, *b*, and *c*, you will reach the same conclusions that you came to in the answer to part a of this problem?

Example 2. Game Trees

A more interesting type of decision tree, frequently referred to as a *game tree,* shows the moves made by the players. Each time a player makes a move, he selects among the available choices of "legal" moves. Each line segment of the tree represents one choice by one player during the playing of one game. Figure 6·4 is a tree for the game of "Eight." This two-player game is so trivial you may not enjoy playing it very long. Its tree is simple enough, however, that we can study it easily, and it serves as a good illustration of similar but far more complicated games.

*The rules of "Eight"*

Each player takes a turn at picking a number from one to three, adding this number to a running sum that is initially set at zero. The first player has a free choice of numbers 1, 2, and 3. The choice in each play thereafter is restricted. A player may not choose the opponent's preceding selection. The player who brings the running sum to a total of exactly eight wins the game; a player exceeding eight loses. There is no draw possible.

When we study the game tree, we can observe that a complete game from start to finish is represented by one path (e.g., the colored line) from the beginning or *root* of the tree down to an end or terminal point. Player A always moves first. Thus, on the green line, A chooses 1 from among the three initial choices. Then B chooses 3, then A chooses 1, and so forth, until at the last move for A the running sum is 7, and his choices are 1 and 3. So he chooses 1 to make the sum 8 and wins. Triangular-shaped endpoints denote a win for A. Square-shaped endpoints denote a win for B.

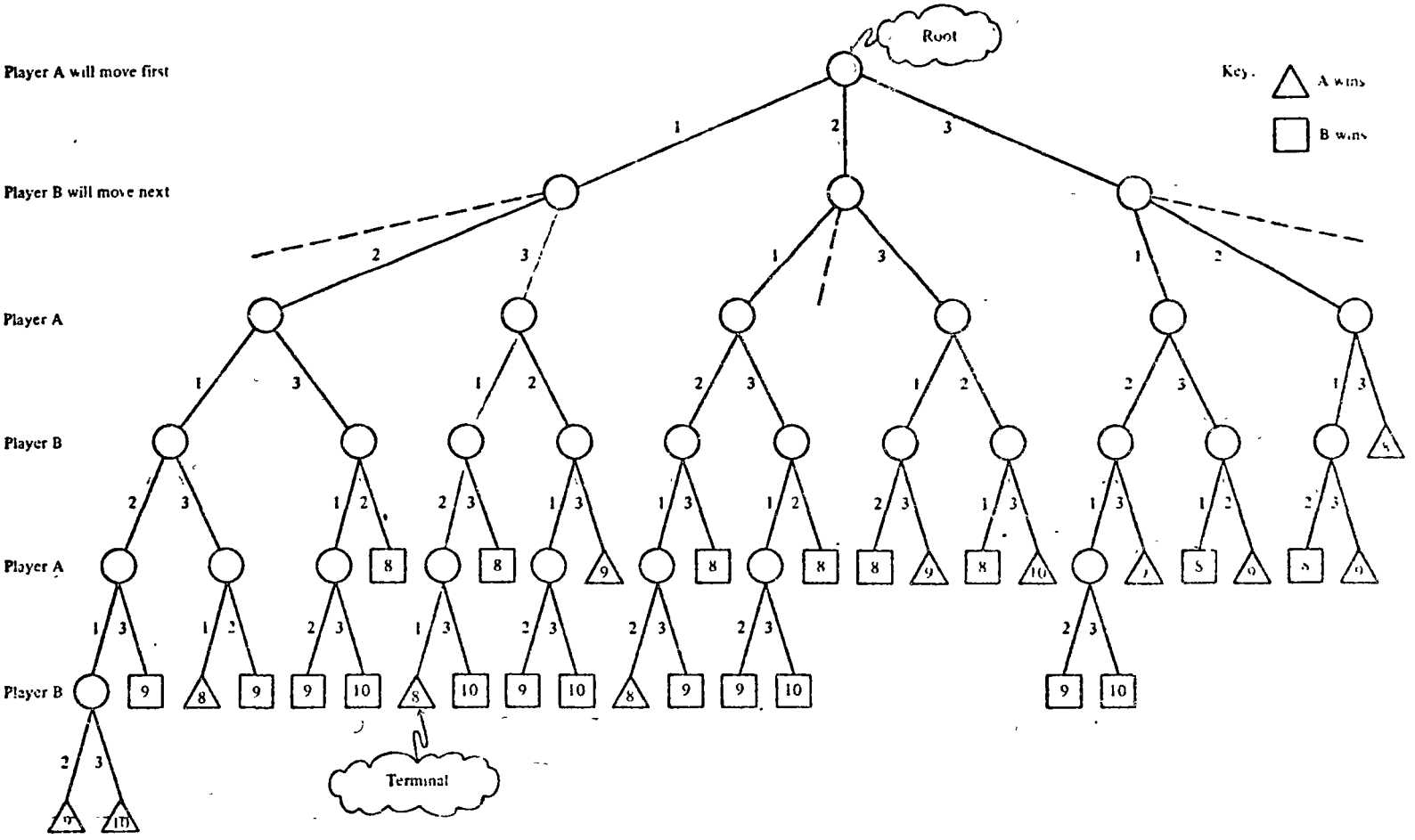
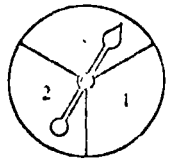



FIGURE 6·4
Tree for the game of Eight.

Each time a player makes a move, you can imagine he looks at three choices 1, 2, and 3. After the very first choice he rules out one of these, as the game rules demand. *Inadmissible* moves would not ordinarily appear in the tree because they tend to clutter the diagram. We have shown them as dashed lines for B's first move only as a reminder.

EXERCISES 6·1, SET B

**1.** (a) How many distinct games of "Eight" are there?

Imagine the game of Eight is played by children at the local kindergarten in the following way. Player A twirls the arrow of a three-part spinner, to select the initial move. 

Thereafter, each player flips a coin to decide among the two admissible choices. (heads the smaller, tails the larger.)

(b) What percentage of games played will follow the color line path in Figure 6·4?

(c) What are A's chances of winning? Express this result as the number of games won for every 100 games played.

(d) If each player chooses each move as shrewdly as possible, what do you think are A's chances of winning if A plays first? The answer is 0 times out of 100. See if you can develop a proof of this assertion. In Chapter 7 we will take another look at this problem.

In each of the next two exercises, you are given the rules of a simple, two-player game. Your job in each case is to show part or all of the game tree with at least four complete games displayed.

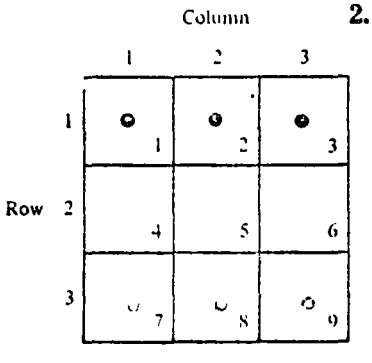



FIGURE 6·5
Board position at the beginning of the game of Hex.

**2.** The game of *Hex* (or Hexapawn) uses a 3 by 3 checkerboard. Each player begins the game with three pieces on his base line, as shown in Figure 6·5. Play alternates between green and gray. The rules of the game are as follows.

(a) Either green or gray, in his turn, can move forward one space to an unoccupied position.

(b) Or he can move diagonally one space to capture an opponent. A captured piece is removed from the board.

(c) The game is won by reaching the opponent's baseline.

(d) Or by leaving the opponent without a move.

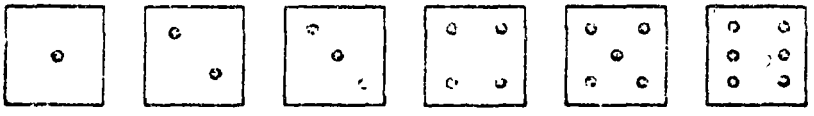
(e) Or by capturing all opponent pieces.

*Hint* Each segment of the tree should be labeled to indicate the move it represents. One way would be to show the *before* and *after* row,

column values of the piece that is moved. For example, on green's first play he has three choices: (3,1) → (2,1), or (3,2) → (2,2), or (3,3) → (2,3). Each of these moves can be further abbreviated to four-digit numbers without loss of information, that is, 3121, 3222, and 3323, respectively.

Alternatively, if we give the squares of the board the explicit names shown as small digits in the lower right corner of each square in Figure 6·5, then we can use a somewhat more compact representation of a move. Instead of four-digit abbreviations (e.g., 3121, 3222, and 3323), we can use two-digit abbreviations (e.g., 74, 85, and 96, respectively) with no risk of confusion.

3. *The game of "31."* Take a die and roll it. The side that turns up gives the running sum's initial value. Thereafter, each player moves by tilting the die over on its side (one of four possible sides, of course, and remember that opposite faces



always add to seven). The side that turns up after the tilt-over is then added to the running sum. The game proceeds tilt after tilt. A player whose tilt brings the total to exactly 31 wins the game; a player exceeding 31 loses. There are no draws.

## Data Trees

Flowcharts of algorithms often have the characteristic treelike structure, but it is also interesting that *data* can be arranged in a treelike structure. Here are two examples.

### Example 3. Monotone Subsequences

Suppose you are given a sequence (i.e., a list) of N numbers, all guaranteed to be different. What is the longest *monotone subsequence* in the given sequence?

By a *subsequence* we mean the list that remains after "crossing out" some numbers in the original list. If, for example, the given list is

5 0 9 6 1 12 3 7 2,

then one of the 511 ($2^9 - 1$) possible subsequences of this sequence is:

~~5~~ 0 9 6 ~~1~~ 12 3 ~~7~~ 2

that is, 9 6 12 3 2.

The reason for explaining this idea in terms of "crossing out" is to make it absolutely clear that the order of the *remaining* terms is not altered. By a *monotone* subsequence we mean one in which either the values are increasing from left to right or one in which they are decreasing. Thus the preceding subsequence,

9 6 12 3 2

is not monotone, but the following two are, the first being increasing and the second decreasing.

5 0 9 6 1 12 3 7 2
5 0 9 6 1 12 3 7 2

You can check that the increasing subsequence is the longest possible; that is, there is no increasing subsequence with more than four elements. The decreasing one is not the longest possible, since the subsequence

9 6 3 2

is longer.

It is possible to develop an algorithm for determining longest monotone subsequences of a given sequence. Our interest here is a bit different. Suppose you are asked to picture *all* the possible monotone decreasing subsequences of our example sequence,

5 0 9 6 1 12 3 7 2

A hopeless task? Not if we think in terms of trees! See the answer in Figure 6·6.

A most revealing discovery! We have taken a string of numbers, posed a particular problem concerning that string, and discovered that the problem's answer could lie in inspecting a related tree. Notice that every monotone decreasing subsequence in S can be represented as a *path* running from the root S to one of the terminal squares. From now on, we'll call these circles and squares *nodes*.

The longest of such subsequences is easy for the eye to pick out once the tree is drawn. It is the one whose *terminal node* is found at a level of the tree farthest from the root node. In this example, only one path reaches to level 4, so there is only one longest monotone decreasing subsequence.

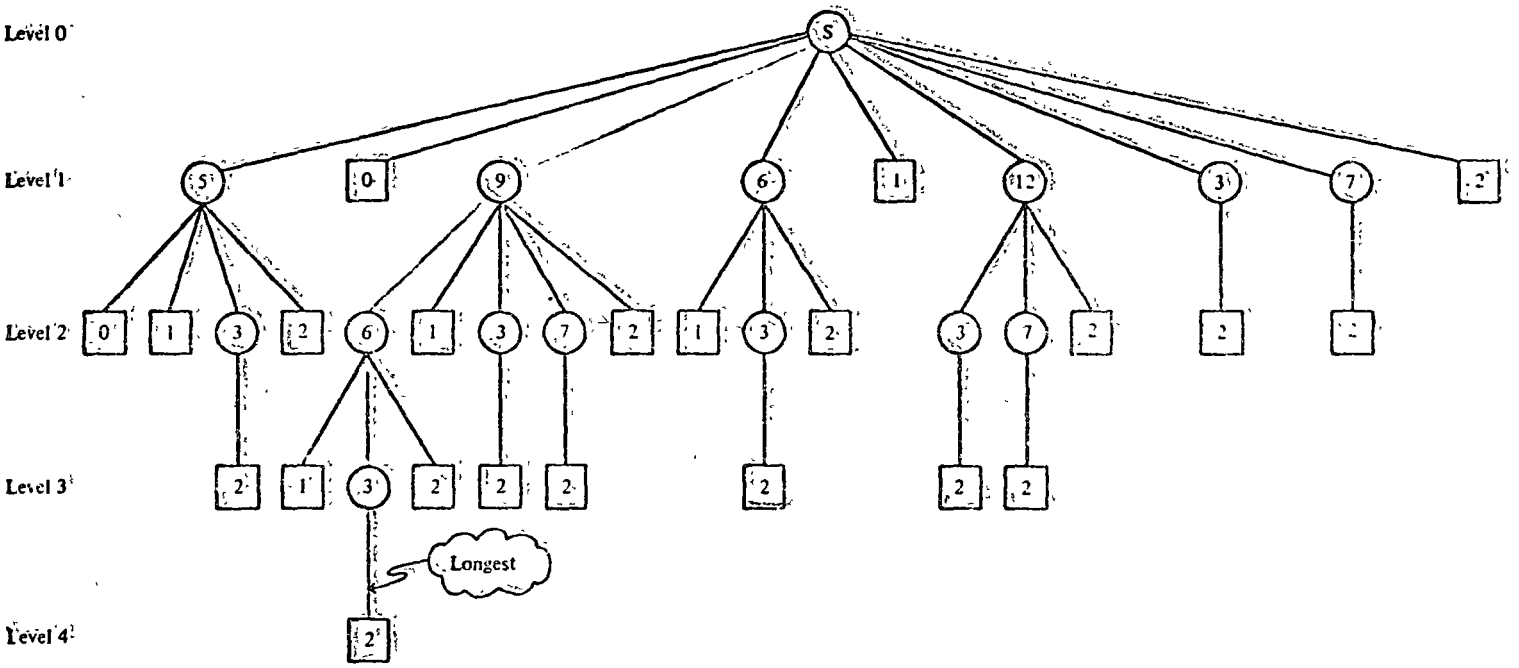



FIGURE 6·6
Tree of monotone *decreasing* subsequences for the list
S = 5, 0, 9, 6, 1, 12, 3, 7, 2.

If a computer is to be used for this approach, we must have an algorithm that, in effect, systematically scans the entire tree. This is the interesting part, which will be taken up starting with the next section.

EXERCISES 6·1, SET C

1. Construct a tree that displays all monotone *increasing* subsequences of the same sequence given in Figure 6·6.
2. Draw the tree that displays all monotone decreasing subsequences for the sequence S defined as

   $$S = \{3, 2, 1, 0\}$$

3. Imagine you are a student registering at a university and you have decided to enroll in a particular group of five courses. The five courses, together with the available sections and the times each will be taught, are listed in Figure 6·7. We presume these data are extracted from the official class schedule. Notice that the time periods are letter coded for convenience.

| *Course* | *Open Sections* | |
|---|---|---|
| ENG 132 | D (9–10 | MWF) |
| | E (10–11 | MWF) |
| | F (11–12 | MWF) |
| FRE 141 | F (11–12 | MWF) |
| | H (1–2 | MWF) |
| | Q (10–11:30 | TTH) |
| HIS 231 | F (11–12 | MWF) |
| | H (1–2 | MWF) |
| MTH 172 | D (9–10 | MWF) |
| | F (11–12 | MWF) |
| | Q (10–11:30 | TTH) |
| CSC 131 | F (11–12 | MWF) |
| | H (1–2 | MWF) |

Timetable

| Course | Letter Codes for Possible Sets | | | | |
|---|---|---|---|---|---|
| | No. 1 | No. 2 | No. 3 | No. 4 | No. 5 |
| ENG 132 | | | | | |
| FRE 141 | | | | | |
| HIS 231 | | | | | |
| MTH 172 | | | | | |
| CSC 131 | | | | | |

FIGURE 6·7
Data taken from the printed class schedule.

Does a possible set of nonconflicting sections for the five courses exist? That is, is it possible to select a set with no time conflicts? If so, how many distinct feasible sets can be selected? To be distinct, a set need differ in no more than one section from other possible sets. Complete a column of the timetable shown in Figure 6·7 for each feasible set.

*Hint* This problem and others like it can be solved systematically by constructing a tree of labeled nodes. The structure for the tree could be such that the set of nodes along any path emanating from the root represents a set of nonconflicting course sections. For example, labeled nodes at level 1 could represent the various available sections of ENG 132 (Figure 6·8). Nodes at level 2 could correspond to various sections of FRE 141, and so forth. Any path running from the root to level 5 such that every node has a different label would represent a feasible set of courses.

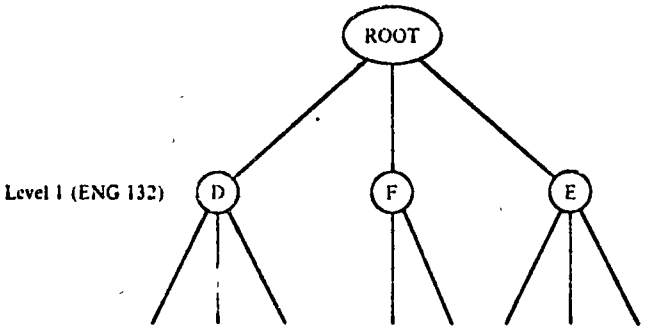



FIGURE 6·8

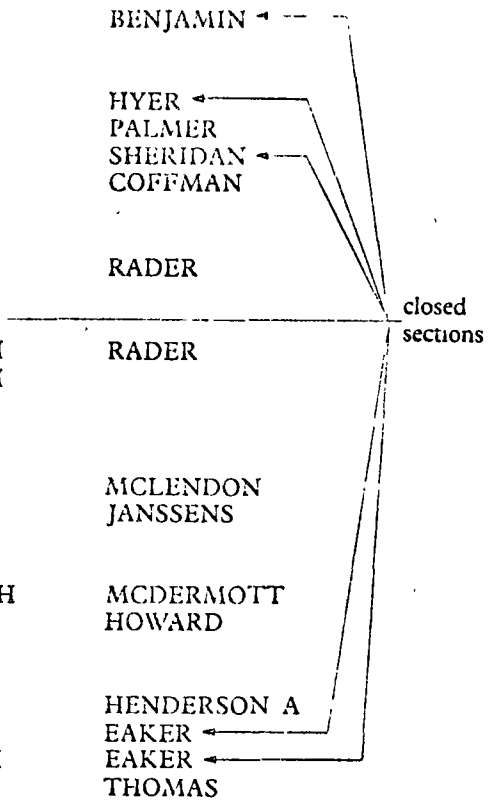
4. Imagine you are a student registering at a university and assume that you have decided to enroll in the following *six* courses:

| | |
|---|---|
| Communications 267 | (COM 267) |
| English 337 | (ENG 337) |
| French 231 | (FRE 231) |
| Geography 233 | (GGY 233) |
| Mathematics 272 | (MTH 272) |
| Music 120A | (MUS 120A) |

Below are data taken from the printed class schedule. Imagine that when you reach the registration desk, you find that certain sections of four of the desired courses are closed (as indicated). Prepare a tree diagram that shows whether there are one or more possible programs open to you at this time, permitting you to enroll in all six of the desired courses with no time conflicts. If one (or more) program(s) is (are) available, prepare a filled-out timetable whose format is similar to that given in Figure 6·7.

5. A student who was planning to work every afternoon (1–5 p.m.) for the Athletics Department was also hoping to enroll in all of the following six courses: COM 267, MUS 120A, GGY 233, MTH 272,

| | | | |
|---|---|---|---|
| COM 267 | TV-FILM SCENE AND LIGHT | | |
| R601 | 117-KUHT | 930-11AM TTH | COLLINS |
| MUS 120A | MUSICIANSHIP I | | |
| E601 | 129-E | 10-11AM WF | MILLER |
| H602 | 129-E | 1-2PM WF | MILLER |
| P603 | 129-E | 9-10AM TTH | HORVIT |
| Q604 | 101-E | 10-11AM TTH | BENJAMIN |
| GGY 233 | WORLD REALMS | | |
| F601 | 101-AH | 11-12 MWF | HYER |
| H602 | 101-AH | 1-2PM MWF | PALMER |
| S603 | 210-AH | 1-2:30PM TTH | SHERIDAN |
| U601 | 101-AH | 4-5:30PM TTH | COFFMAN |
| MTH 272 | CALCULUS III | | |
| D602 | 204-AH | 9-10AM MWF | RADER |
| E603 | 216-AH | 10-11AM MWF | |
| H604 | 116-T | 1-2PM MWF | |
| P607 | 7-AH | 8:30-10AM TTH | RADER |
| P608 | 111-Z | 8:30-10AM TTH | |
| S610 | 211A-SR | 1-2:30PM TTH | |
| FRE 231 | INTERMEDIATE FRENCH | | |
| E601 | 303-AH | 10-11AM MWF | MCLENDON |
| F602 | 106-Z | 11-12 MWF | JANSSENS |
| H603 | 105-Z | 1-2PM MWF | |
| I604 | 105-Z | 2-3PM MWF | |
| Q605 | 111-Z | 10-11:30AM TTH | MCDERMOTT |
| S606 | 112-Z | 1-2:30PM TTH | HOWARD |
| ENG 337 | SHAKESPEARE | | |
| D601 | 105-C | 9-10AM MWF | HENDERSON A |
| E602 | 110-C | 10-11AM MWF | EAKER |
| P604 | 110-C | 8:30-10AM TTH | EAKER |
| S605 | 113-C | 1-2:30PM TTH | THOMAS |




FRE 231, and ENG 337. He received special permission to register early (i.e., no *closed sections* to worry about). How many different feasible programs could he select, given the printed schedule used in Problem 4 (ignoring closed sections), and still take the afternoon job without a time conflict?

**Example 4. Tree Representation of Expressions**

Suppose we are given

$$((a \times w + b) \times w) \uparrow 2 / (d \times y)$$

It seems obvious that whoever first wrote this string of characters intended that it have a mathematical meaning. By now, you are quite expert at interpreting such strings. This interpretation, remember, involved the application of a relatively complicated set of rules (Tables 2·1 and 2·3). Figure 6·9

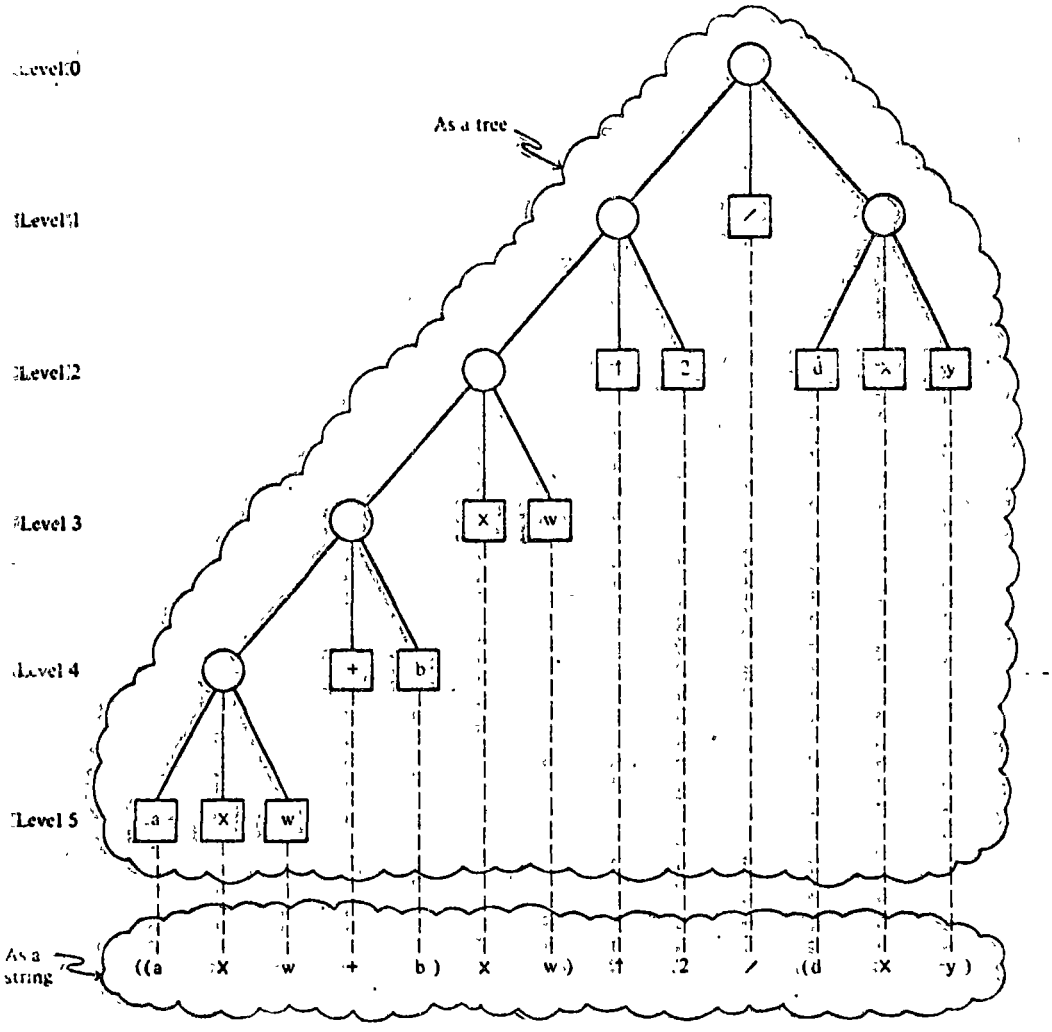



FIGURE 6•9
A tree representation of an arithmetic expression.

shows how we can represent the same string as a tree and give it the same interpretation. We will quickly discover that the rules for evaluating an expression represented as a tree are much simpler to state because *part of the interpretation is inherent in the structure of the tree.*

Before proceeding with this line of thought, it will be helpful to summarize and supplement the tree terminology developed thus far. This is done with the aid of Figure 6•10*a*.

We see that every tree has a *root node* from which extend *segments* (one or more) to other nodes, which in turn branch to others, eventually ending in *terminal* or *leaf nodes.* (Notice that a root node alone is not a tree.) Every segment leads to the root of a subtree, which may be a terminal node. Nodes

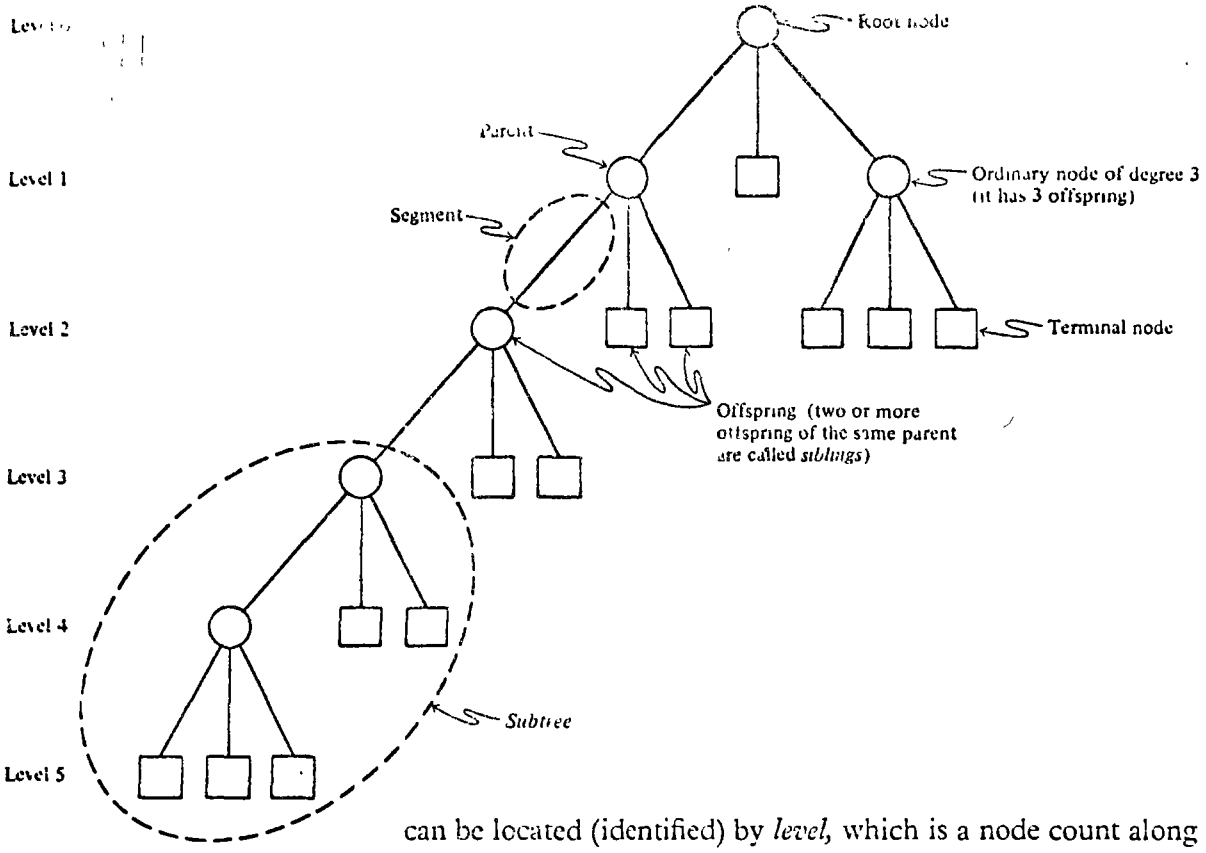



FIGURE 6·10*a*
Tree terminology.

can be located (identified) by *level,* which is a node count along a *path* from the root node to a terminal. The level we associate with the root node is purely arbitrary, but we will usually take it to be zero. The nodes along a path are often thought of as having an *ancestral* relationship one to another. By analogy with family trees, moving from a root toward a terminal, each node is the *parent* of its immediate successor nodes (*offspring*). Two or more nodes having a common parent are sometimes called *siblings.* Finally, we can say that the *degree* of a node is the number of its immediate offspring.

A node may also be identified in terms of the *path* that leads from the root to that node. How can we represent that path? An answer comes to mind when we realize that representing a tree in two dimensions imposes an ordering on the segments that emanate from each node. And we might as well recognize this fact of life by numbering the segments in some way, say from left to right or right to left. For simplicity and consistency we will generally number segments from left to right, as suggested in Figure 6·10*b*. These ordinal numbers amount, in effect, to *names* for the segments.

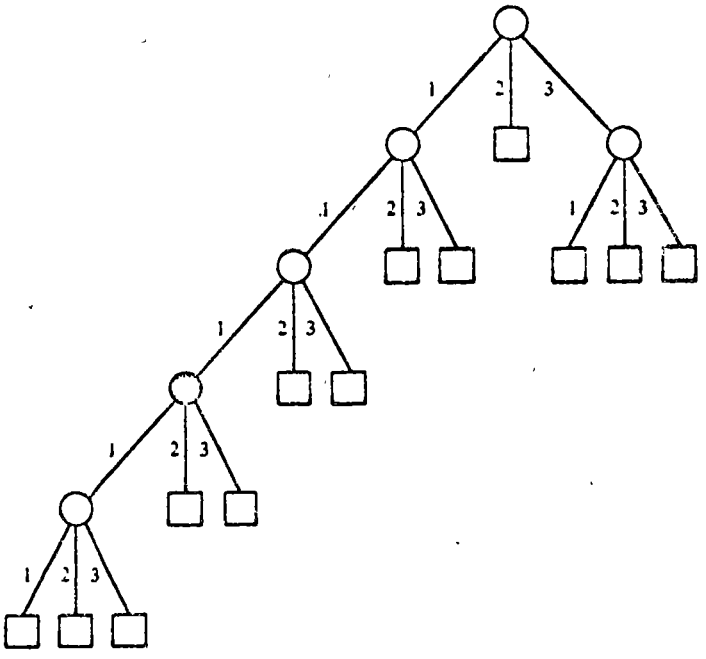


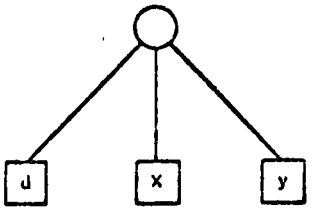
FIGURE 6·10*b*

Now a node can be designated uniquely by listing the *names of all the segments in the path leading to that node.* For example, the light green node in Figure 6·10*b* may be designated by the list (1, 1, 1, 1, 2), the dark green node by the list (3, 1), the gray node by the one-element list (2), and the black node by (1, 1, 1). (How would the root node be designated in this scheme?) Distinct nodes have distinct paths and hence distinct lists.

The *expression tree,* by its very structure, provides the key to evaluating the expression that is represented. For example, suppose values for the variables of our expression are:

| a | w | b | d | y |
|---|---|---|---|---|
| 3 | 2 | 2 | −1 | 7 |

A subtree of the form shown in Figure 6·11, together with the above table, can be understood to mean: Look up the values of a and w, compute $a \times w = 3 \times 2 = 6$, and replace the subtree by the terminal node 6.




Correspondingly, the subtree shown on the left can be interpreted as: Compute $d \times y = -1 \times 7 = -7$ and replace the subtree by the terminal node −7.

Figure 6·12 represents a sequence of meaningful substi-

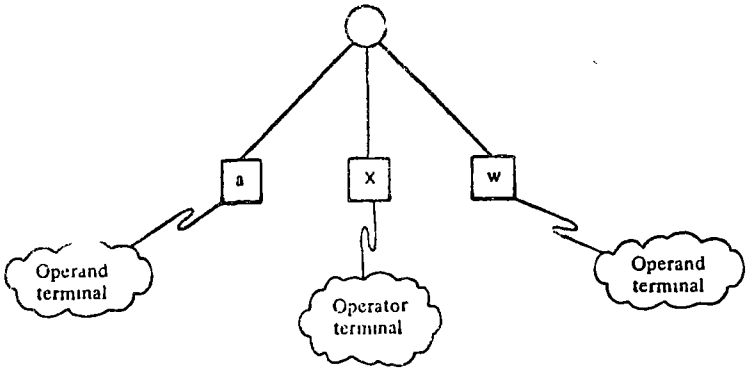



FIGURE 6·11

tutions that, when carried out, will ultimately lead to the replacement of the whole tree, root node and all, by a single terminal, which represents the value of the expression.

Proper evaluation is guaranteed if we follow one simple *replacement rule* that says:

> *Whenever you find a subtree consisting of a root node leading to three terminals (an operator and two operands), replace that subtree by a single terminal value.*

Thus the replacement sequence in Figure 6·13, although different from that of Figure 6·12, leads us irrevocably to the same value, $\boxed{\frac{-256}{7}}$. A computer performing either sequence would evaluate the same indicated quotient $\boxed{\frac{-256}{7}}$, notwithstanding the fact that computer operations on floating-point numbers are nonassociative, a fact explained in Chapter 11. Another point to note from the figures is that the treelike representation of a complicated arithmetic expression allows us to see all the meaningful subexpressions (all the subtrees) at a glance.

Once an expression is represented as a tree, evaluation depends only on repeatedly searching and finding subtrees that are subject to the replacement rule. At any given time, an expression tree, if not already fully evaluated, will exhibit at least one such subtree.

A question that has no doubt been uppermost in the minds of some readers is: How should tree structures be represented in storage? Linked lists, introduced in Chapter 4, suggest one

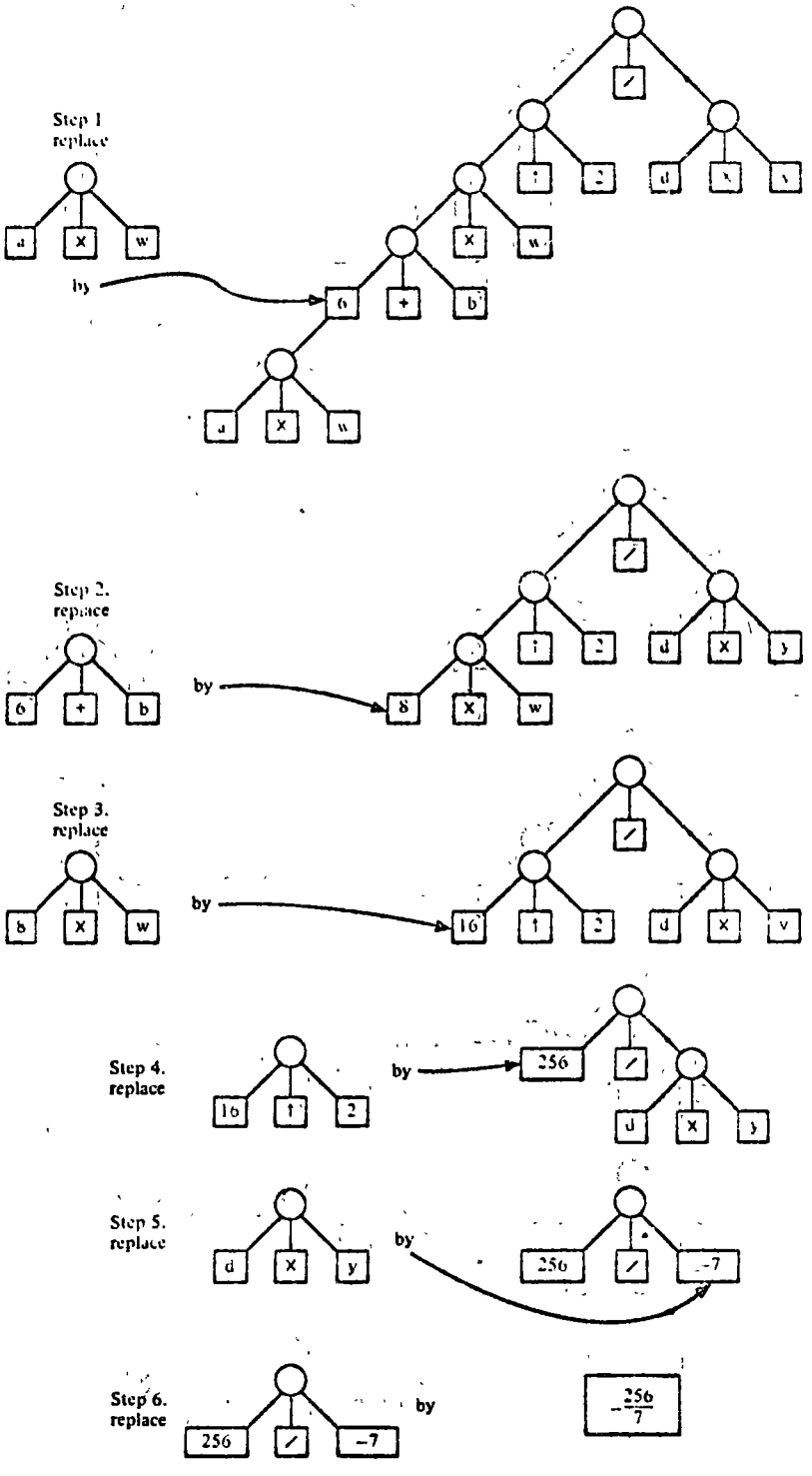



FIGURE 6·12
A stepwise evaluation of a tree expression.

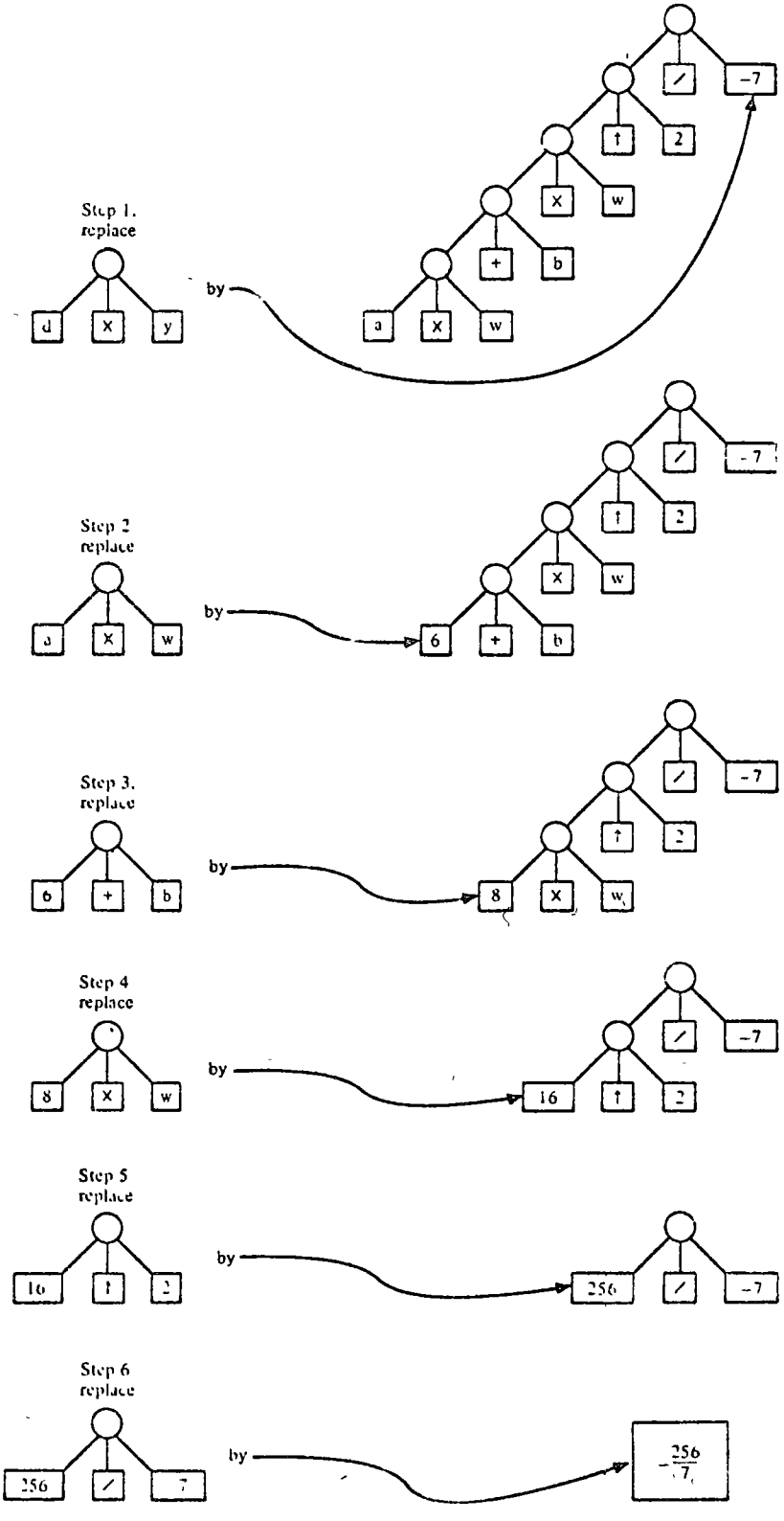



FIGURE 6·13
Alternative replacement sequence for evaluating the tree expression.

way. With this approach, one storage representation for the expression

((a × w + b) × w)↑2 / (d × y)

is shown in Figure 6·14. A four-column table is used. Each row represents a node with row 1 representing the root node.

Tree Table

| Node Number | Value (String) | Links | | |
|---|---|---|---|---|
| | | Left | Middle | Right |
| 1 | □ | 2 | 15 | 16 |
| 2 | □ | 3 | 13 | 14 |
| 3 | □ | 4 | 11 | 12 |
| 4 | □ | 5 | 9 | 10 |
| 5 | □ | 6 | 7 | 8 |
| 6 | a | — | — | — |
| 7 | × | — | — | — |
| 8 | w | — | — | — |
| 9 | + | — | — | — |
| 10 | b | — | — | — |
| 11 | × | — | — | — |
| 12 | w | — | — | — |
| 13 | ↑ | — | — | — |
| 14 | 2 | — | — | — |
| 15 | / | — | — | — |
| 16 | □ | 17 | 18 | 19 |
| 17 | d | — | — | — |
| 18 | × | — | — | — |
| 19 | y | — | — | — |

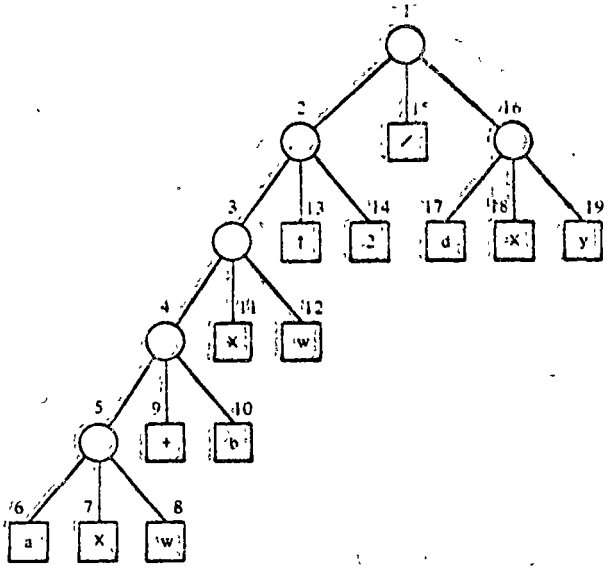
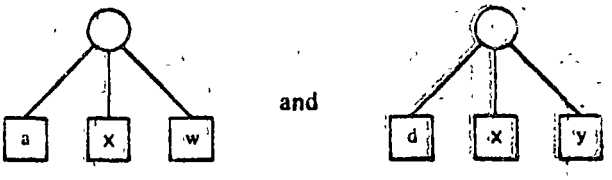



**FIGURE 6·14**
**Tabular representation of an expression tree.**

The first column holds the value of the node if it is a terminal or some special mark, for example, □, if the node is nonterminal. The remaining three columns hold node numbers that designate the left, middle, and right offspring. These positions may be left empty (undefined) for terminal nodes. Node numbers in the left, middle, and right columns serve as *links* to other nodes. Other tabular descriptions of tree structure using the linked list approach are discussed in Chapter 7.

### A Peek at Some Future Models of SIMPLOS

We noticed in the expression tree of Figure 6·12 that two separate subtrees



had only terminal nodes as offspring. At the start of evaluation we could invoke the replacement rule on either of these subtrees. It was immaterial which we picked first. This will always be the case for subtrees whose root nodes are siblings or the offspring of sibling nodes (i.e., cousins once or more *removed*). From the standpoint of expression evaluation, such subtrees are *mutually independent.*

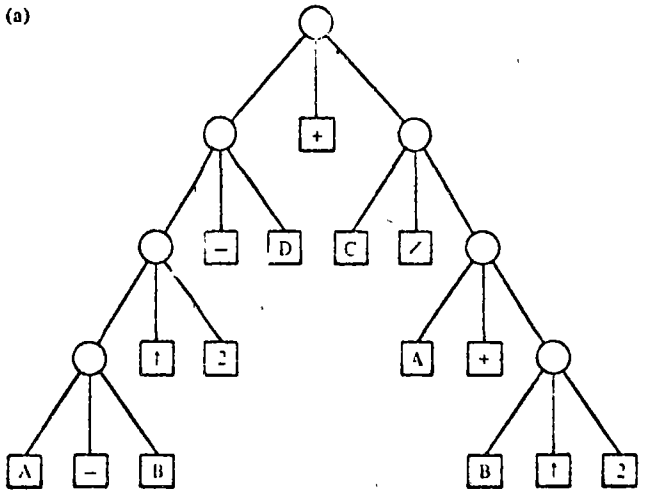
Under what circumstances can a computer work on the evaluation of two or more independent subtrees at the same time? With our present SIMPLOS model the answer is never, because at any one time there is only one team of personnel (Master Computer, Affixer, Reader, and Assigner) available to do work. On the other hand, advanced models of computer systems having several, perhaps many teams of personnel, are quite feasible.

Although it may boggle the mind to think about it, one may anticipate that future computers will evaluate mutually independent subtrees concurrently, that is, *in parallel,* whenever more than one team of personnel is available for the purpose. In cases where speed is essential the capability of concurrent computation offers the opportunity to solve problems that cannot be solved fast enough in any other way. Examples of such problems already abound in our technological society. More can be found on this topic in advanced texts.

**EXERCISES 6·1, SET D**

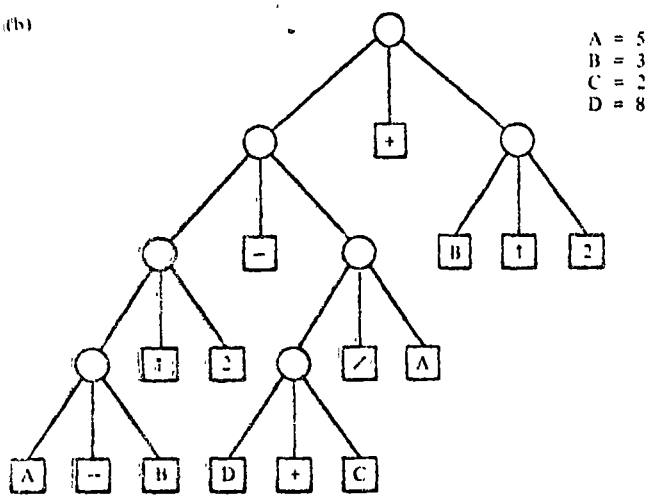
1. Evaluate the expression trees below, using the given values for the variables.

(a)

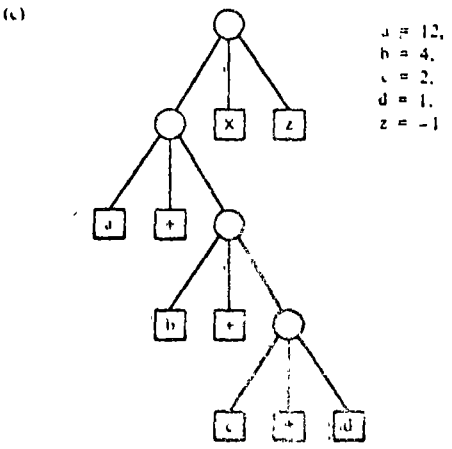



A = 1
B = −3
D = 10
C = −60

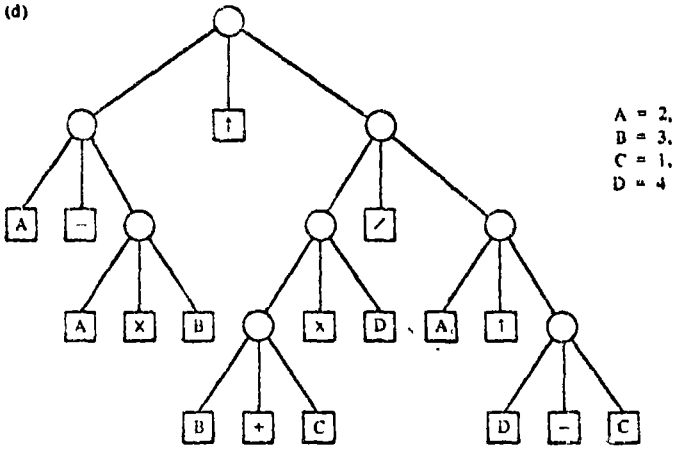
(b)




(c)




(d)



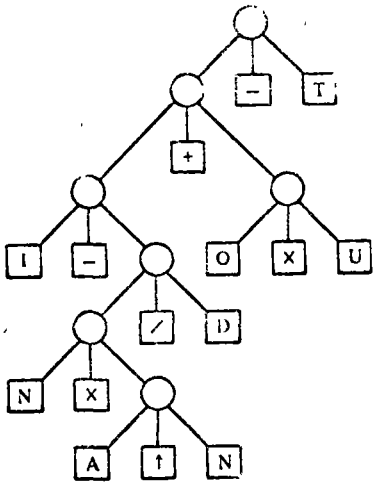

**2.** Draw a tree representation for the expression

$$B\uparrow 2 - 4 \times A \times C$$

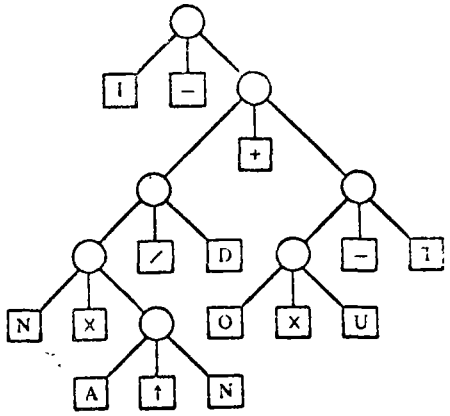
**3.** The following are two proposed tree representations for the expression

$$I - N \times A\uparrow N / D + O \times U - T$$

Which, if either, of these trees, evaluated by the replacement rule, yields a result computationally equivalent to the result we get by following the evaluation rules laid down in Tables 2·1 and 2·3? If the evaluation of either one of the trees is not compatible with these rules, describe the discrepancy.




(a)




(b)

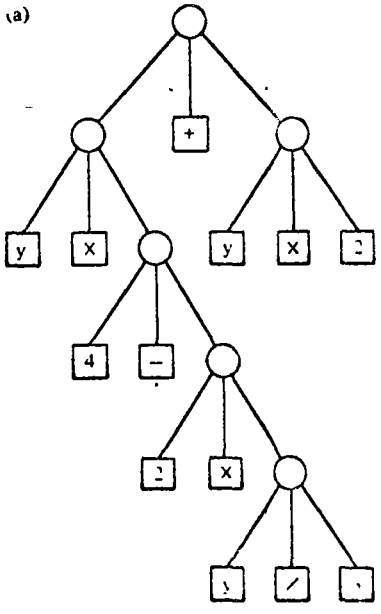
4. Draw a tree representation for the expression

$$(a - b) \times (c - d) / (e \times (f + g))$$

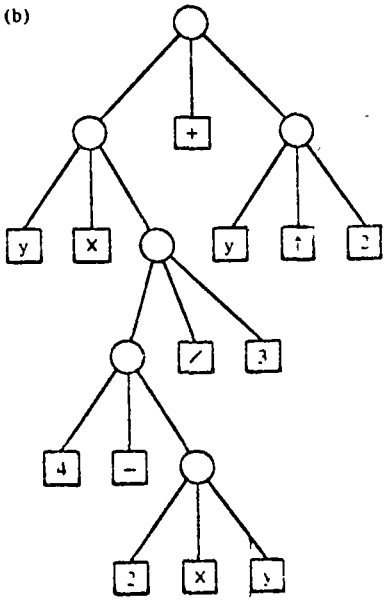
5. Find which of the three trees given below correctly represents the given expression and exhibit the expression represented by each of the other trees.

$$y\left(\frac{4 - 2y}{3}\right) + y^2$$

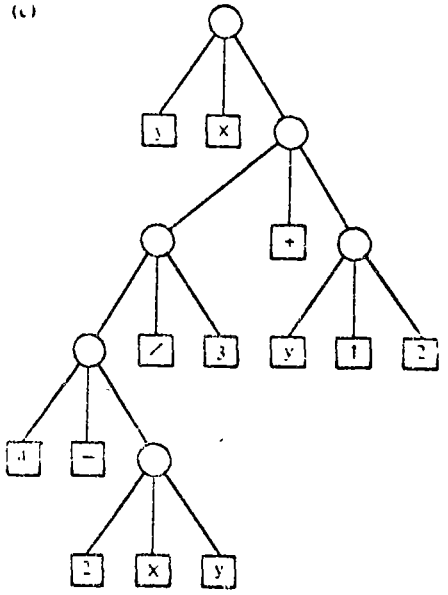
(a)




(b)



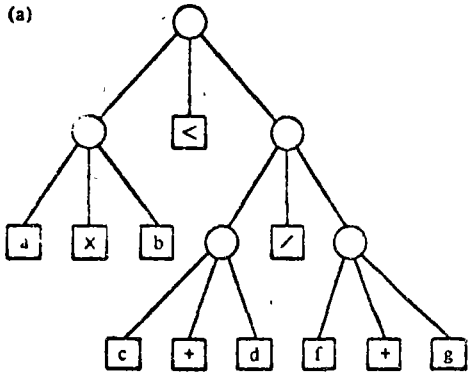
(c)



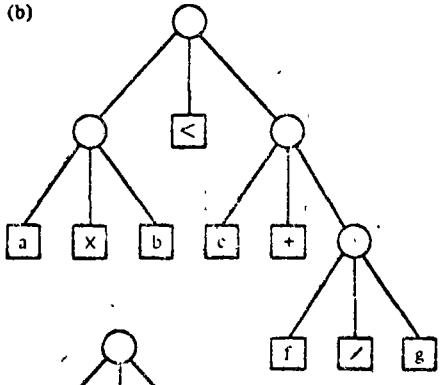
6. Find the tree among the four given below that represents the given expression correctly and exhibit the expressions represented by each of the other trees.

$$a \times b < c + d/(f + g)$$

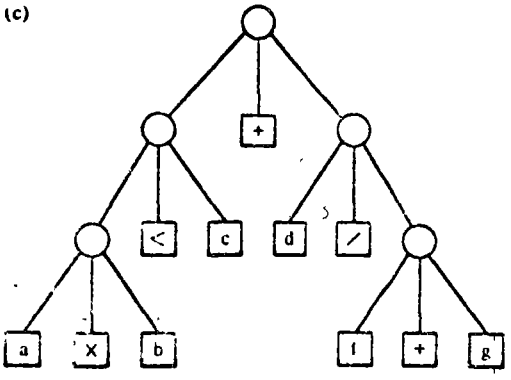
(a)



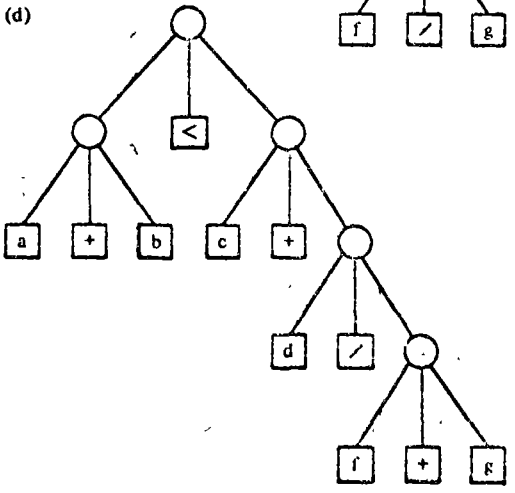
(b)



(c)

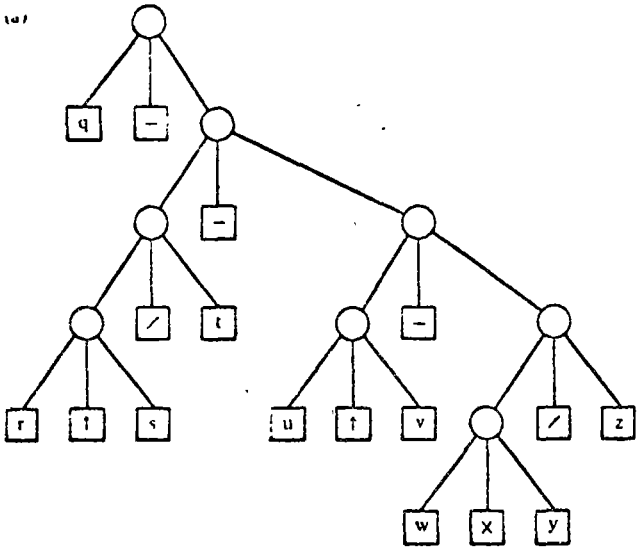


(d)

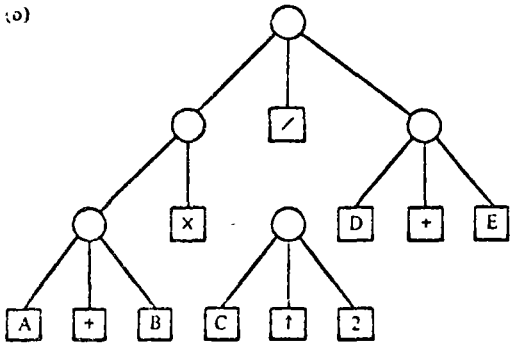


7. Convert the trees below into the corresponding arithmetic expressions.

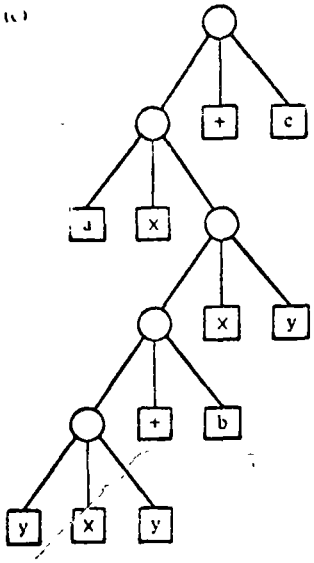
(a)



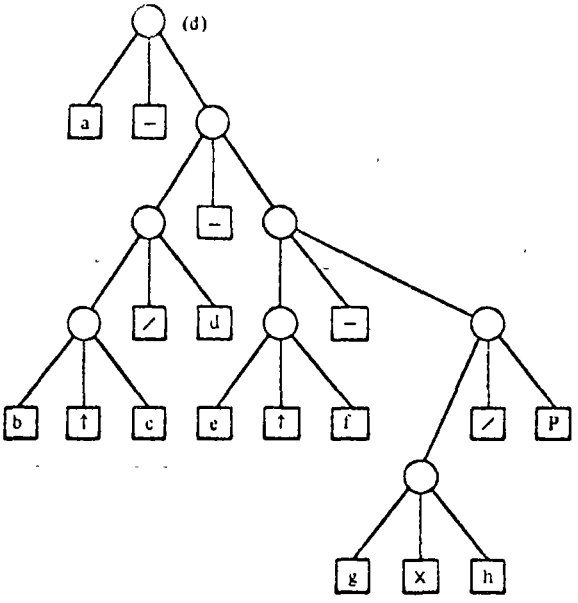

(b)



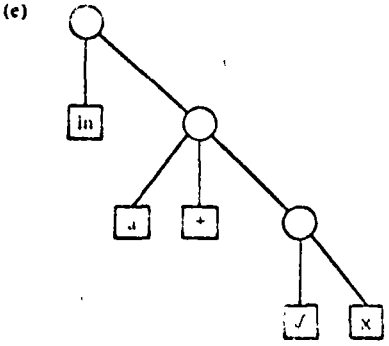

(c)

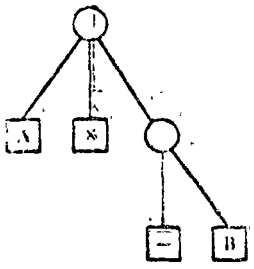



(d)




(e)






8. How would you draw the expression tree for the expression $A \times (-B)$? The problem here is to decide how one should represent a unary operation. One possibility is shown in the tree on the left.

Here the subtree expression $-B$ is treated as a root node having a unary operator [−] and one argument [B] as offspring nodes. One can deduce that the operator is unary by the fact that the (left-hand) operand node is missing. Functions such as $\sqrt{}$, cos, and so forth, can be thought of as unary operators. Using the above expression scheme, or another of your own choosing, develop expression trees for:

(a) $A + \sqrt{x}$
(b) $\cos(x^2 + y^2)$
(c) $\sqrt{\frac{1}{2} \times (1 + q / \sqrt{p^2 + q^2})}$

9. Which of the following statements is false?

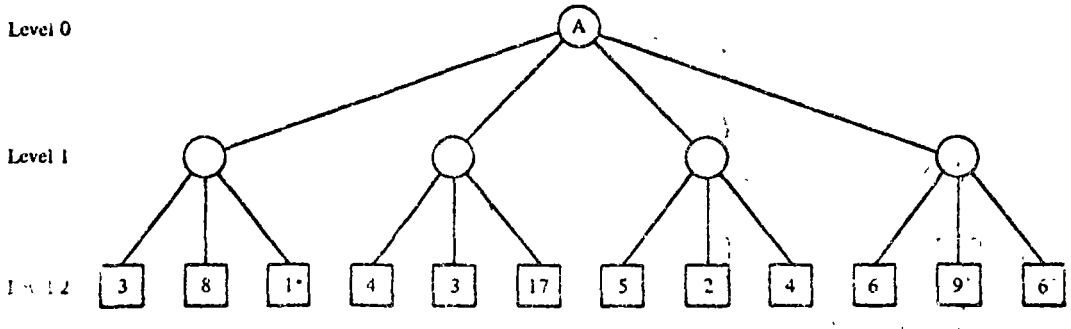
(a) A terminal node has one ancestor node and no descendant nodes.
(b) A root node has no ancestor nodes and may have no descendant.
(c) A nonterminal node has no descendant nodes.
(d) A nonterminal node may have only one ancestor node.
(e) A terminal node can be connected to an ancestor.

*Hint* If you have any question as to the meaning of "ancestor" and "descendant" just think of a family tree.

10. Any given two-dimensional matrix can be represented as a tree. For example, the matrix

$$A = \begin{bmatrix} 3 & 4 & 5 & 6 \\ 8 & 3 & 2 & 9 \\ 1 & 17 & 4 & 6 \end{bmatrix}$$

can be expressed as the tree:




(a) Given the representation above, the four nodes at Level 1 correspond to (choose one):

(1) The four elements of the main diagonal of the given matrix.
(2) The four elements of row 1 of the given matrix.

(3) The four columns of the given matrix.
(4) The sums of the elements in each of the columns of the given matrix.
(5) None of the above.

(b) Show at least one other way to represent the matrix A as a tree structure.

**11.** Using list notation, give the paths for the nodes labeled [D], [U], and [↑], in the second of the two trees referred to in Problem 3 of this exercise set.

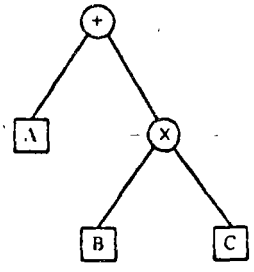
**12.** Develop a scheme to denote the saving in time that, in principle, is possible in a computer having multiple processing units that can execute concurrently in the same expression. Show how your scheme would work on the following expressions.

(a) $a^2 + b^2 + c^2$
(b) $(a - b) \times (c - d) / (e \times (f + g))$
(c) $(\sqrt{x} + \cos y) / z$

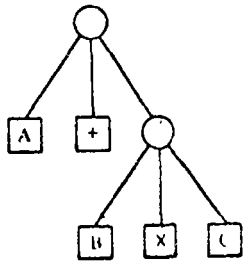
**13.** In the text we have always shown the operator symbol of an expression tree as a terminal node, so each nonterminal node of the tree has three offspring if the operator has two operands, and two offspring if the operator has one operand. Another way to draw the tree is to place each operator symbol *at* its parent (nonterminal) node. For example, the tree for the expression

$$A + B \times C$$

may be drawn as




instead of as



The new form, which we shall call a *binary* expression tree, has fewer nodes but the same amount of information.
Refer to Figure 6·14 and:

(a) Produce the binary expression tree equivalent to the expression tree given in the figure.
(b) Show how the tree table in that figure can be changed in structure and in content to represent the binary tree you developed in part a.
(c) Which tree table would require less storage in a computer representation?

## 6·2 Tree searches

We have now seen enough of trees to have observed their main structural characteristics; segments of a subtree always connect to new nodes that form a continuation of the same subtree; there is no looping back to nodes closer to the root; and there is no crossing or crisscrossing between subtrees.

There are many ways one can construct and store a tree structure. Depending on what use is to be made of the tree, some representations (we will call these *storage structures*) are better than others. Trees are searched for one reason or another, either to gain specific information, to reach a conclusion, or to modify the tree in a certain way. A tree search lies at the heart of a number of mathematical problems and a great number of games.

### Natural-order Tree Search

There is a systematic way to scan all the nodes of a tree that is used frequently in solving problems. We call it *natural-order* searching. Although a squirrel may have better ways of finding nuts in a tree, it will help us to understand natural-order search if we imagine a nutseeking squirrel willing to follow these rules.

1. Start at the trunk (root) and don't stop trying segments until you reach a leaf (terminal node) unless you find a nut and choose to stop at that point.

2. Upon reaching a terminal without finding a nut, back up to the node you just passed, that is, to the *parent* node of this terminal.

3. Now, choose the next untried segment, if any, and move forward along it toward another leaf node.

4. If there are no untried segments, crawl backward to the predecessor (parent) node and repeat the process of trying to reach another leaf node.

5. If you ever find yourself back at the root having already tried all segments from the root without finding a nut, you have finished searching the entire tree in natural order and can report a failure to find a nut.

Figure 6·15 shows a natural-order search of a tree. The numbers beside the nodes indicate the sequence in which they

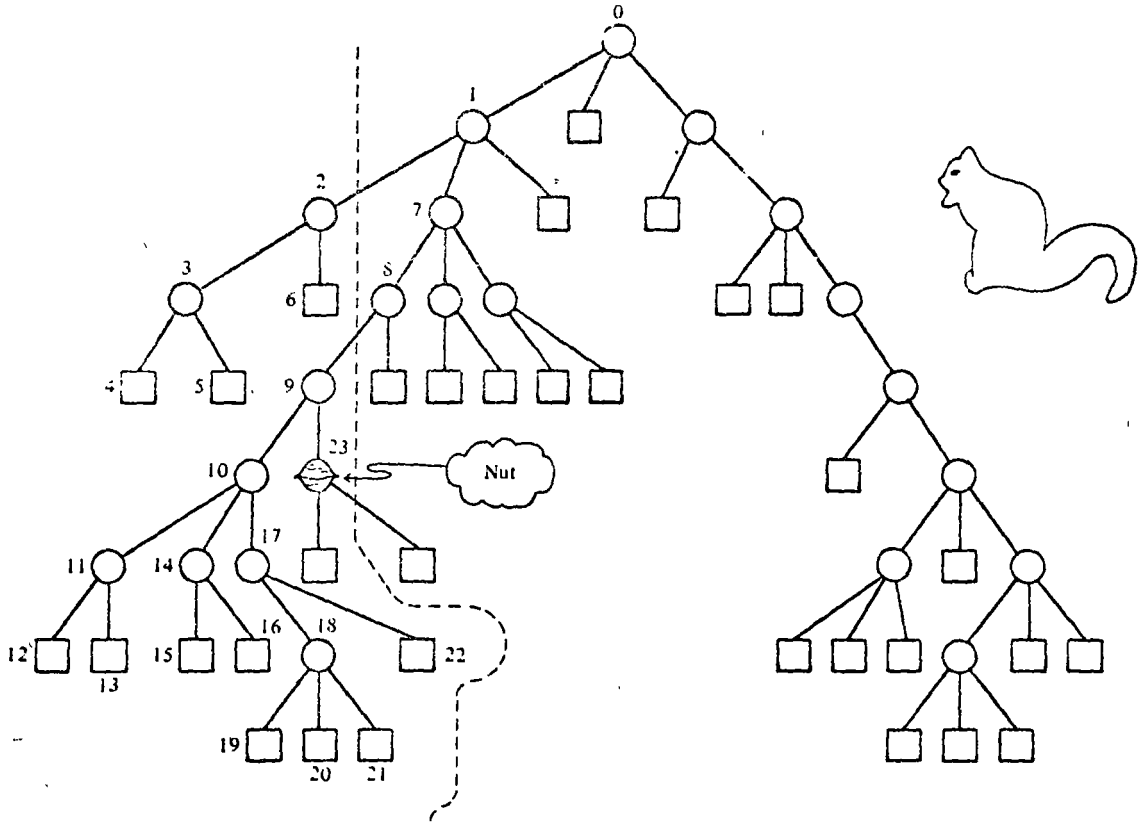



FIGURE 6·15
Systematic (*natural-order*) search for a nut.

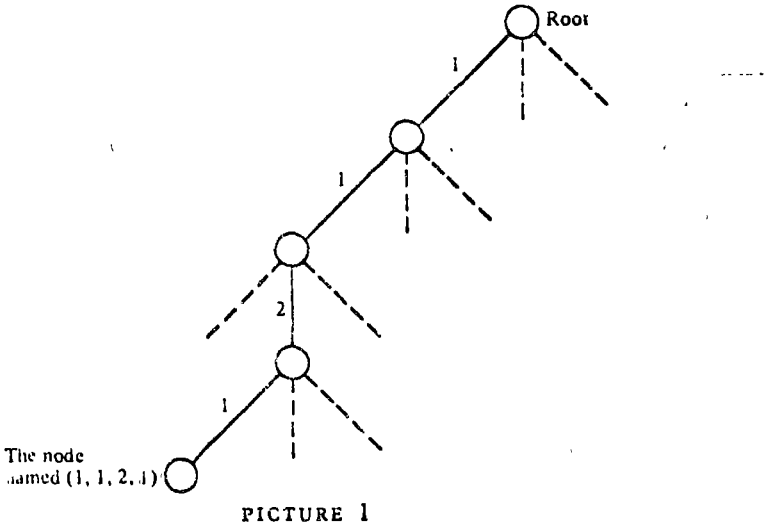
are first encountered (i.e., as the squirrel sees them in its forward progress). We picture one of these nodes as a nut. It is the 23rd node encountered. Notice the systematic, left-to-right selection of segments at each node.

## Algorithm for Natural-Order Tree Search

Now suppose we wish to construct a tree search algorithm that generally follows the stated set of rules. One of our problems is how to interpret rule 3, that is, how to choose among the

remaining untried segments. If we recall, however, that the segments emanating from each node are, or always can be, *ordered,* then a simple interpretation comes quickly to mind: choose the segment, if any, whose ordinal number name is one higher than that of the segment previously tried.

To make this choice implies that the algorithm can always identify the ancestor or parent node from which the "previously tried" segment emanated. That is, the algorithm can only identify additional segments in terms of the common parent. This *backup* capability is assured if the algorithm at all times has an up-to-date record of where it is in the tree search and can represent this data in the form of a path list. For example, suppose it has been discovered that the segments from the node whose path designation is (1, 1, 2, 1) need be examined no further (Picture 1).

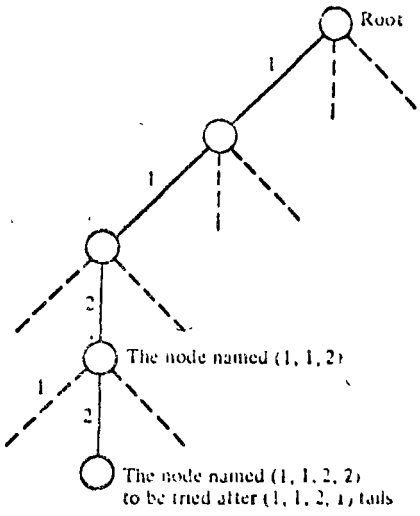



PICTURE 1

What is the path name for the parent of that node? The answer is (1, 1, 2).

How do we apply rule 3 to this parent (1, 1, 2)? (Rule 3: choose the next untried segment, if any, and move forward along it toward another leaf node.) The answer is, if there *is* a node whose path is (1, 1, 2, 1 + 1), try it (Picture 2).

In general, suppose we have tried the segment leading from node (1, 1, 2) to node (1, 1, 2, i) and the subtree whose root is (1, 1, 2, i) has failed to contain the nut we are looking for. To select the next untried segment, if any, of node (1, 1, 2), we have only to check whether there exists a valid node




PICTURE 2

whose path is (1, 1, 2, i + 1). If so, select the segment leading to this node and if not, back up, and check whether there is a valid node (1, 1, 2 + 1). If so, select the segment leading to this node, but if not, back up again and see whether (1, 1 + 1) is valid, and so on.

We now sense that by starting out with a path list that represents the root node (an empty list of segments), and by continuing to update that list as we move through the tree to reflect where we are in the search, then simple adjustments to the path allow us to determine each new direction of search.

Figure 6·16 shows a systematic procedure, that is, an algorithm for conducting natural-order tree search. The algorithm is represented in top-down style, with Figure 6·16*a* giving the topmost view. Any necessary data are input in box 1 and the tree search begins at box 2. In Tree_Search, whose details are given in Figure 6·16*b*, there are two key variables: *level* and *path*. The value of level tells us the number of elements in path. Values of these two variables determine the *current node* of the search. In a sense the current node is the one we are standing on while we try to find the next node to move to. These variables are initialized in box 1 of Figure 6·16*b*. Tree_Search sets some sort of switch to indicate success or failure. (Recall that a root node by itself does not constitute a tree. There must be at least one subtree. For this reason the first time box 2.2 is executed the Yes outlet will be taken.) Upon exit from Tree_Search, the main program, in effect, tests

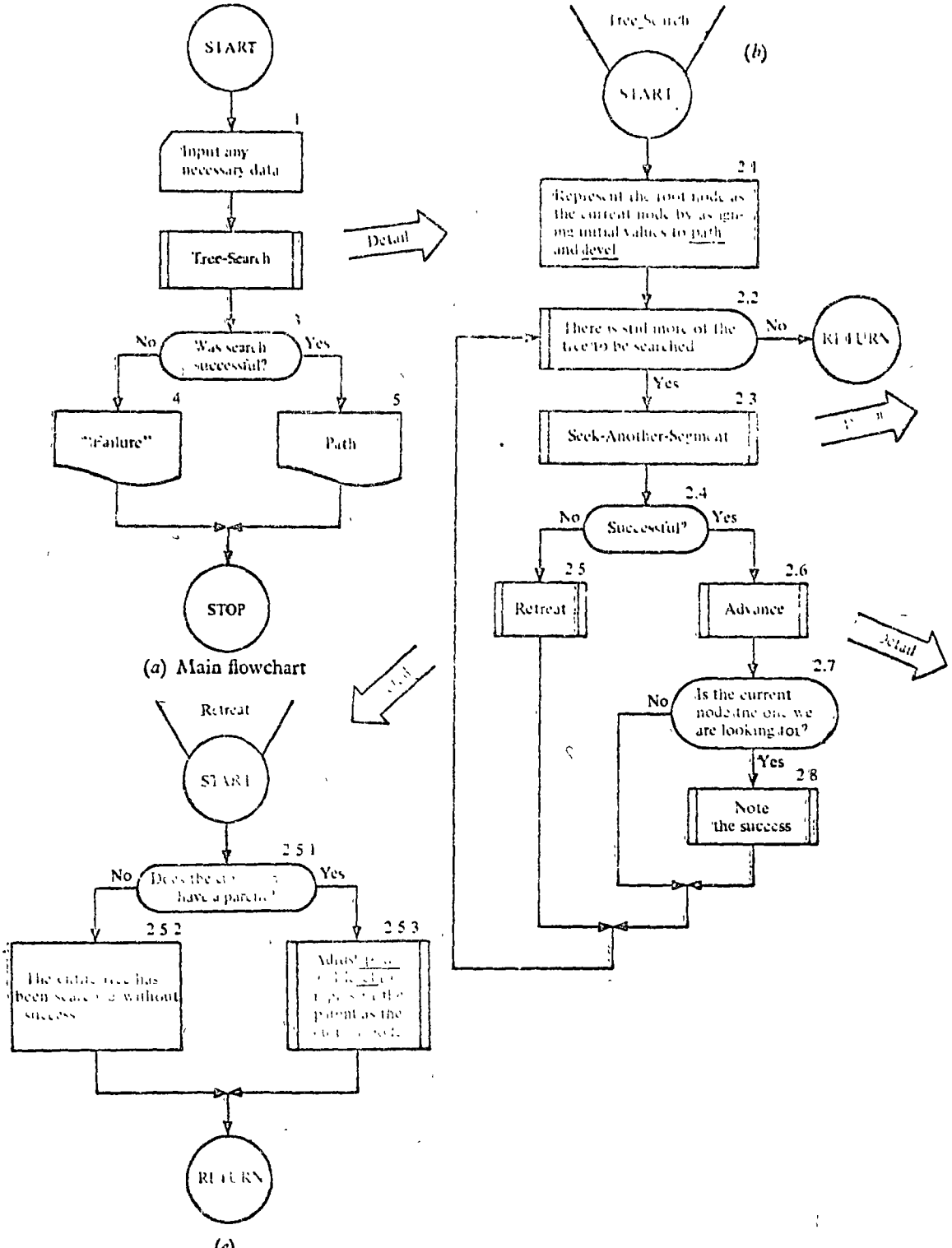
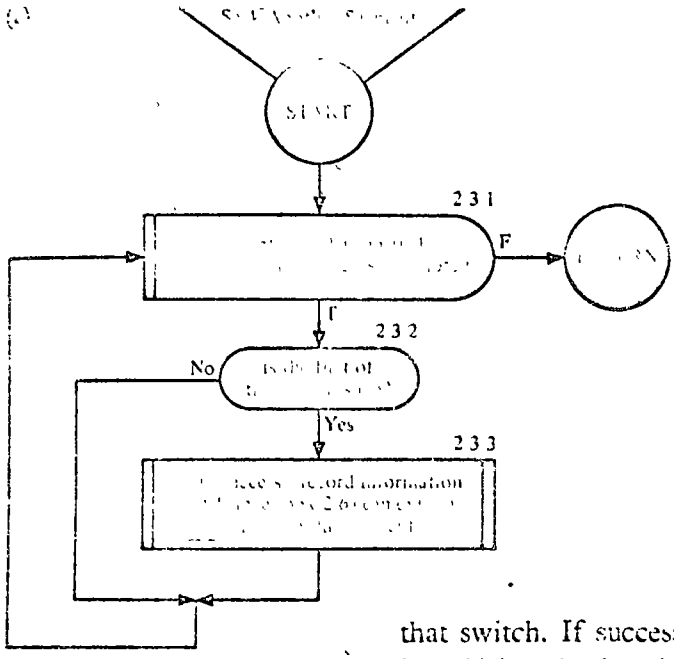
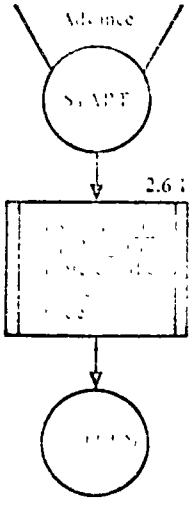



FIGURE 6·16
Top-down development of an algorithm for natural-order tree search.




(*d*)

that switch. If success is indicated, the path list is displayed, identifying the location in the tree where the nut was found; otherwise a failure message is displayed.

To simplify the details of Tree_Search, three of its boxes are given in more detail in Figure 6·16*c*, 6·16*d*, and 6·16*e*. Notice that rule 3 is implemented in box 2.3 as a call to a procedure, "Seek_Another_Segment," whose details (Figure 6·16*c*) include a test for admissibility of untried segments. Although not shown in the level of detail given in Figure 6·16*c*, we imagine that some sort of switch is set by Seek_Another_Segment, which can be tested upon return to Tree_Search so that the former's success or failure can be determined at box 2.4. If successful, there is a new node to which the search may *advance* (details in Figure 6·16*d*). If unsuccessful, it is necessary to *retreat* to the parent node, if any (Figure 6·16*e*). (Remember that Seek_Another_Segment reports failure only after *all* segments have been tested.)

The bookkeeping of the retreat operation (box 2.5.3) is a two-step process.

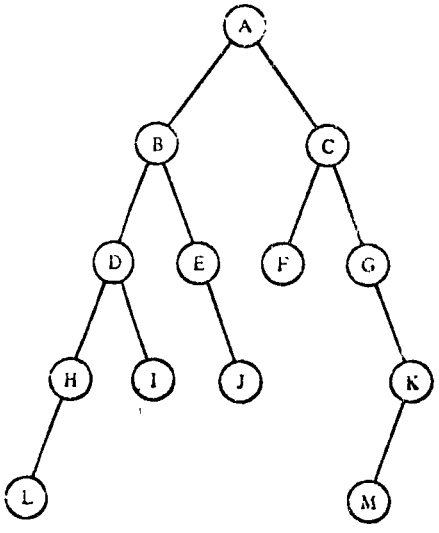
1. Detach the last element of *path*, which is a segment number, and save it to use the next time Seek_Another_Segment is called at box 2.3.

2. Decrement *level* by one to reflect the shortened length of *path*.

The segment number saved in step 1 is needed in boxes 2.3.1 and 2.3.3. When a new segment number is selected in box 2.3.3, it in turn is saved for use by Advance, the next time that procedure is called at box 2.6. Advance increments *level* by one and appends the new segment number on to the end of *path*. We leave to the reader the pleasure of reviewing these details and convincing himself of their correctness. As a parting remark, it is worth observing that the nature of the admissibility check hinted at in box 2.3.2 may be crucial to the success of the search. As many inadmissible structures as possible must be ruled out at each stage. For example, the squirrel should recognize each dead limb and not search it. Otherwise, the proportion of useless paths may grow rapidly, meaning that the efficiency of the search method can plummet toward zero. Next we examine several interesting problems that employ this type of search in their algorithmic solution.

EXERCISE 6·2

1. List the nodes of the tree below in the order in which they would be encountered in a natural-order search.




## 6·3 The four-color problem

Maps are colored to make it easy to see at a glance the extent of each country. It is necessary that neighboring countries (i.e. countries with a common boundary line) be assigned different colors. Does the mapmaker then need more than four colors to do his job? He doesn't care, but we *do*.

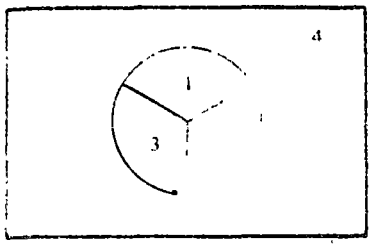



FIGURE 6·17

This problem was one of the most celebrated challenges in mathematics. It is of great intellectual interest and has intrigued many people from all paths of life. Actually, its solution has little or nothing whatsoever to do with making maps. A mapmaker is and always will be able to print maps using as many different colors as he needs.

A checkerboard is an example of a map that can be colored with only *two* colors. The four-country map shown in Figure 6·17 requires *four* colors. Because each pair of countries is adjacent, no two can have the same color.

It didn't take us long to find a map requiring four colors. Yet, in over 100 years of searching, no one has succeeded in finding a map requiring five! It is natural to conjecture that every map can be colored with four colors, and many mathematicians have racked their brains trying to prove this conjecture. The best they have been able to do so far is to show that every map can be colored using no more than five colors.*

We are about to see how computer methods can be applied to the four-color problem. We will not use the computer to show that the four-color conjecture is *true*. Indeed, it is entirely possible that no computer can ever prove this. However, true or false, we can use the computer to determine whether *a particular map* can be colored in only four colors. This is the task for which we want to construct an algorithm.

Before starting on this algorithm, a few remarks concerning the coloring of maps may be helpful.

A *minimal five-color map* is a map requiring five colors, so that every other map requiring five colors has at least as many countries. Of course, no minimal five-color map has ever been found. But mathematicians have shown that if such maps exist at all, then some of them satisfy these two conditions.

1. No point is a boundary point of more than three countries.
2. Each country is a neighbor of at least five others.

Moreover, it can be shown that *every* minimal map, if any exist, must satisfy the second condition.

It is therefore customary to consider as candidates for counterexamples to the four-color conjecture only maps fulfilling these conditions.

*A simple proof of the five-color theorem exists. It may be found, for example, in *What is Mathematics?*, Oxford University Press (1941), by Courant and Robbins.

### Four-Coloring as a Tree Search

We can model the problem of four-coloring a given map, say, the one pictured in Figure 6·18, as one of traveling along a path through a tree such as that shown in Figure 6·19. Each segment represents a decision to color a country, with colors 1, 2, 3, or 4. The ith segment in a path from the root corresponds to the coloring of the ith country of the map.

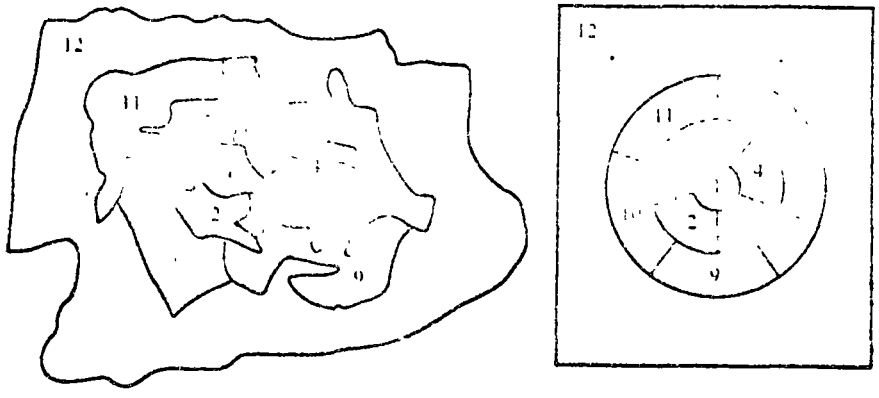


FIGURE 6·18

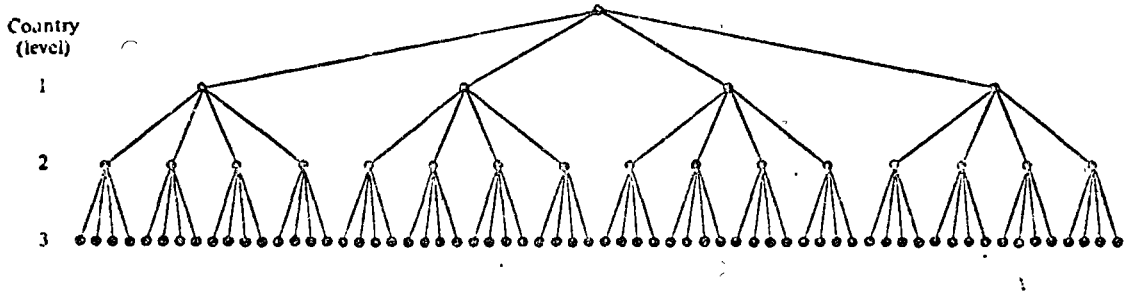



FIGURE 6·19

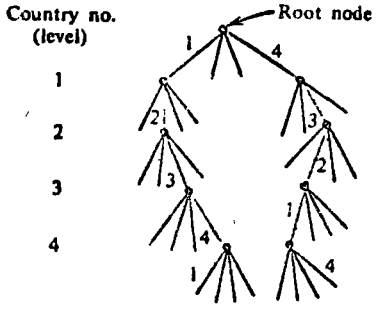



FIGURE 6·20

Observe that many paths through the tree turn out to represent identical colorings of the map except for renaming of the colors, and it is desirable to avoid searching through such duplicate patterns. (E.g., the two heavy-line paths in Figure 6·20 represent the same coloring patterns with different names used for the colors.) One way to avoid the unnecessary search is to fix at the outset in a quite arbitrary way the colors for neighboring countries 1, 2, and 3 and to begin the real search with the coloring of country 4.

In coloring all countries, from the fourth country on, as seen in Figure 6·21, we assume that all four choices are possible. Most of the time, however, as can be seen in Figure 6·22, only one, two, or three of these choices will be admissible. Sometimes even all four choices will be inadmissible, as ex-

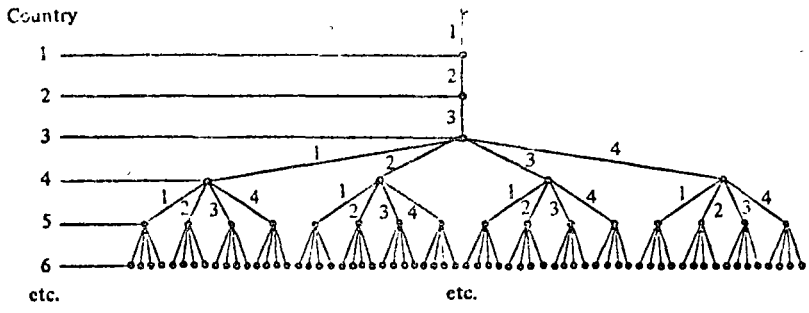



FIGURE 6·21
Showing coloring tree and one path representing the coloring of the first six countries (colored line).

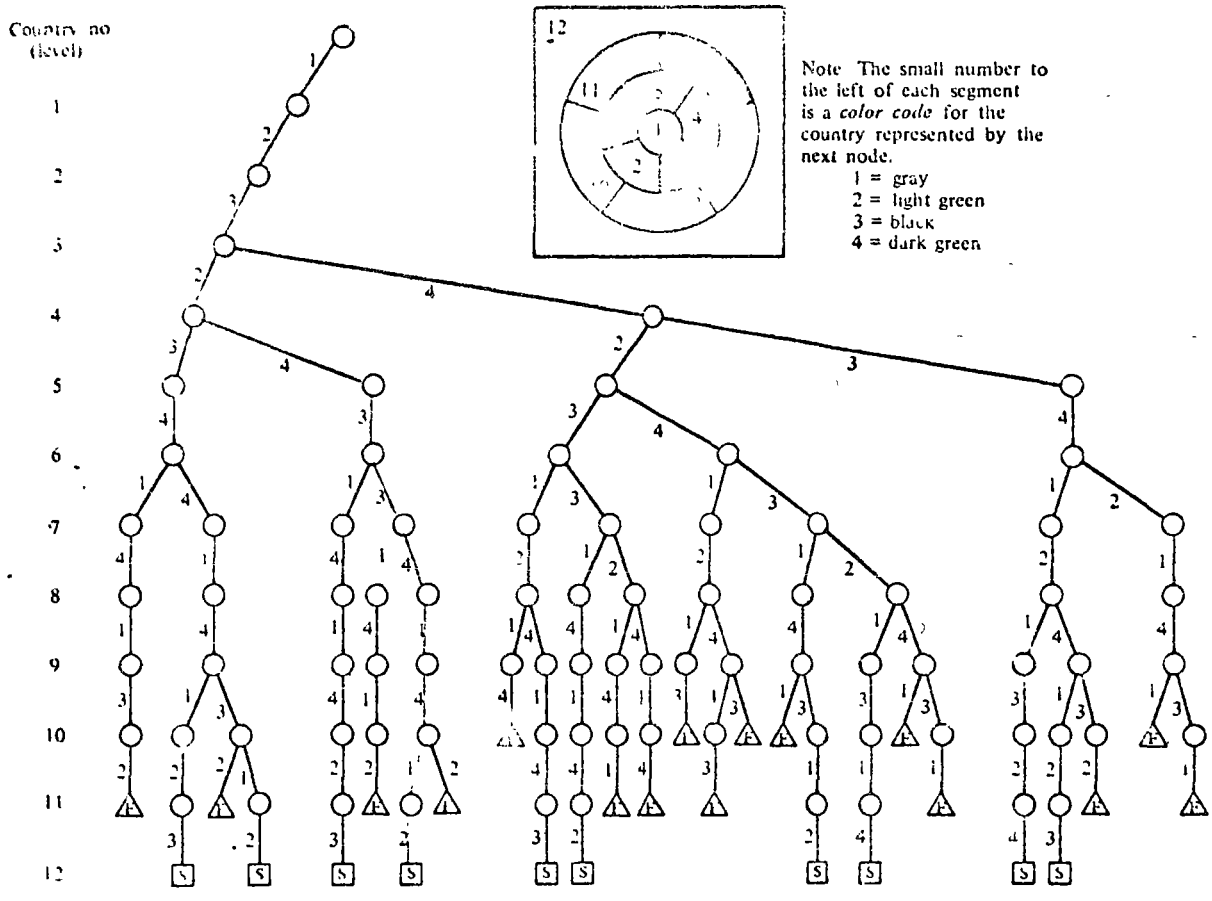



FIGURE 6·22
An entire coloring tree showing how to four-color the map of Figure 6·18.

emplified by terminal nodes marked F in Figure 6·22. Only 10 paths lead to S (success) terminals.

EXERCISES 6·3, SET A

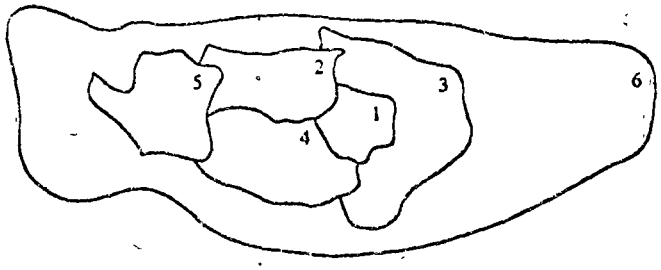
1. Compute the theoretical maximum number of possible terminal nodes for the coloring tree of the 12-country map in Figure 6·18.

   *Hint* Use Figure 6·21 as a guide.

2. Assume that it takes only 1 microsecond to check another path to a terminal, and that the search of half of these paths is required before the desired terminal is reached. How long would the computer chug away before it found what it was looking for in a 39-country map? Assume all segments to be admissible. Express your answer in units of *years*.

3. By renumbering the countries on the map of Figure 6·18, show that a coloring tree can have nodes with three and even four permissible segments emanating from them.

4. Using a form similar to that of Figure 6·22, draw a "coloring tree" for the map shown below.




## A Four-Coloring Algorithm

Let us see how to apply what we have just learned about tree search to an actual problem. It is one thing to discuss a tree in the abstract and another to start with a problem, define in some detail the tree search that is involved, and then develop a detailed flowchart algorithm. In this case, we will take as our problem statement: Develop a detailed flowchart algorithm for four-coloring any n-country map.

The first step toward this objective might be to devise a method to represent any n-country map. To do this we need a sample map for study as, for example, in Figure 6·23. The

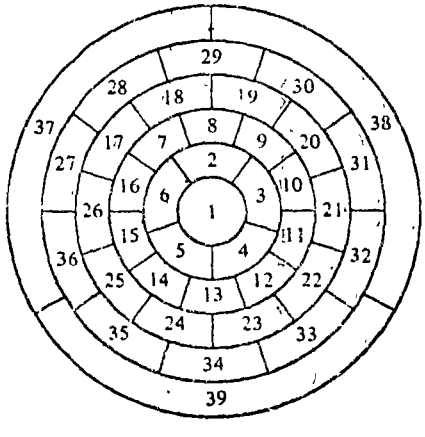



FIGURE 6·23
Example of map to be four-colored by a computer algorithm.

map consists of 39 countries, and the countries have been numbered or indexed in the order that the algorithm will attempt to "color" them.

The efficiency of the algorithm will be greatly improved if each country borders on as many lower-numbered countries as possible. We do not absolutely insist on this but, if you have done Problem 3 of Exercise 6·3, you will appreciate why we recommend this approach. We do, however, require that the first three countries all be neighbors of each other.

How do we represent the map in computer storage? One way is to construct a "connection table," listing after each country all of its neighbors in increasing order. This is shown for our example in Table 6·1.

Our algorithm should consult this table when deciding how to color a particular country. For example, if we were coloring country 15, we could see in row 15 that countries 5, 6, and 14 are neighbors already colored. Our choice of color for 15, then, depends solely on the currently chosen colors for 5, 6, and 14.

Knowing that country 15 also has neighbors numbered 16, 25, and 26 appears to be superfluous. This leads us to the idea of a shaved-down table, which we will call the "*reduced* connec-

TABLE 6·1
The Connection Table for the Map in Figure 6·23

| Country | Neighbors | | | | | | Country | Neighbors | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 2 | 3 | 4 | 5 | 6 | | 21. | 10 | 11 | 20 | 22 | 31 | 32 | |
| 2. | 1 | 3 | 6 | 7 | 8 | 9 | 22. | 11 | 12 | 21 | 23 | 32 | 33 | |
| 3. | 1 | 2 | 4 | 9 | 10 | 11 | 23. | 12 | 13 | 22 | 24 | 33 | 34 | |
| 4. | 1 | 3 | 5 | 11 | 12 | 13 | 24. | 13 | 14 | 23 | 25 | 34 | 35 | |
| 5. | 1 | 4 | 6 | 13 | 14 | 15 | 25. | 14 | 15 | 24 | 26 | 35 | 36 | |
| 6. | 1 | 2 | 5 | 7 | 15 | 16 | 26. | 15 | 16 | 17 | 25 | 27 | 36 | |
| 7. | 2 | 6 | 8 | 16 | 17 | 18 | 27. | 17 | 26 | 28 | 36 | 37 | | |
| 8. | 2 | 7 | 9 | 18 | 19 | | 28. | 17 | 18 | 27 | 29 | 37 | | |
| 9. | 2 | 3 | 8 | 10 | 19 | 20 | 29. | 18 | 19 | 28 | 30 | 37 | 38 | |
| 10. | 3 | 9 | 11 | 20 | 21 | | 30. | 19 | 20 | 29 | 31 | 38 | | |
| 11. | 3 | 4 | 10 | 12 | 21 | 22 | 31. | 20 | 21 | 30 | 32 | 38 | | |
| 12. | 4 | 11 | 13 | 22 | 23 | | 32. | 21 | 22 | 31 | 33 | 38 | 39 | |
| 13. | 4 | 5 | 12 | 14 | 23 | 24 | 33. | 22 | 23 | 32 | 34 | 39 | | |
| 14. | 5 | 13 | 15 | 24 | 25 | | 34. | 23 | 24 | 33 | 35 | 39 | | |
| 15. | 5 | 6 | 14 | 16 | 25 | 26 | 35. | 24 | 25 | 34 | 36 | 39 | | |
| 16. | 6 | 7 | 15 | 17 | 26 | | 36. | 25 | 26 | 27 | 35 | 37 | 39 | |
| 17. | 7 | 16 | 18 | 26 | 27 | 28 | 37. | 27 | 28 | 29 | 36 | 38 | 39 | |
| 18. | 7 | 8 | 17 | 19 | 28 | 29 | 38. | 29 | 30 | 31 | 32 | 37 | 39 | |
| 19. | 8 | 9 | 18 | 20 | 29 | 30 | 39. | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
| 20. | 9 | 10 | 19 | 21 | 30 | 31 | | | | | | | | |

TABLE 6·2
Reduced Connection Table for the Map in Figure 6·23

| Country i | Neighbors $CONN_{ij}$ | | | | Width $w_i$ | Country i | Neighbors $CONN_{ij}$ | | | | | | | Width $w_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | 0 | 21 | 10 | 11 | 20 | | | | | 3 |
| 2 | 1 | | | | 1 | 22 | 11 | 12 | 21 | | | | | 3 |
| 3 | 1 | 2 | | | 2 | 23 | 12 | 13 | 22 | | | | | 3 |
| 4 | 1 | 3 | | | 2 | 24 | 13 | 14 | 23 | | | | | 3 |
| 5 | 1 | 4 | | | 2 | 25 | 14 | 15 | 24 | | | | | 3 |
| 6 | 1 | 2 | 5 | | 3 | 26 | 15 | 16 | 17 | 25 | | | | 4 |
| 7 | 2 | 6 | | | 2 | 27 | 17 | 26 | | | | | | 2 |
| 8 | 2 | 7 | | | 2 | 28 | 17 | 18 | 27 | | | | | 3 |
| 9 | 2 | 3 | 8 | | 3 | 29 | 18 | 19 | 28 | | | | | 3 |
| 10 | 3 | 9 | | | 2 | 30 | 19 | 20 | 29 | | | | | 3 |
| 11 | 3 | 4 | 10 | | 3 | 31 | 20 | 21 | 30 | | | | | 3 |
| 12 | 4 | 11 | | | 2 | 32 | 21 | 22 | 31 | | | | | 3 |
| 13 | 4 | 5 | 12 | | 3 | 33 | 22 | 23 | 32 | | | | | 3 |
| 14 | 5 | 13 | | | 2 | 34 | 23 | 24 | 33 | | | | | 3 |
| 15 | 5 | 6 | 14 | | 3 | 35 | 24 | 25 | 34 | | | | | 3 |
| 16 | 6 | 7 | 15 | | 3 | 36 | 25 | 26 | 27 | 35 | | | | 4 |
| 17 | 7 | 16 | | | 2 | 37 | 27 | 28 | 29 | 36 | | | | 4 |
| 18 | 7 | 8 | 17 | | 3 | 38 | 29 | 30 | 31 | 32 | 37 | | | 5 |
| 19 | 8 | 9 | 18 | | 3 | 39 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 7 |
| 20 | 9 | 10 | 19 | | 3 | | | | | | | | | |

tion table." It is constructed by striking out of each row in the table all numbers greater than the number of the row itself. The reduced connection table for our example is seen in Table 6·2 and can be thought of in this case as a 39-row by 7-column array called CONN. The number of nonnull elements in each row is given by elements of an associated list w. Thus the algorithm can search the first $w_i$ elements in the ith row of CONN to determine which neighbors have already been colored.

If we are to apply our generalized tree search algorithm (Figure 6·16) to the map-coloring problem, we must also decide how to represent the *current node* (i.e., how to represent the variables *path* and *level*). The variable *path* is a list of elements, each of which designates a segment choice. Our decision to use color codes 1, 2, 3, and 4 for the four possible color choices leads us directly to the decision that a search from a node may be accomplished by selecting (trying) the segments in the same order, 1, 2, 3, and 4. The decision to make this correspondence between the color codes and the segment order imposes the required ordering on the segments from each node of our coloring tree. Moreover, the ith element

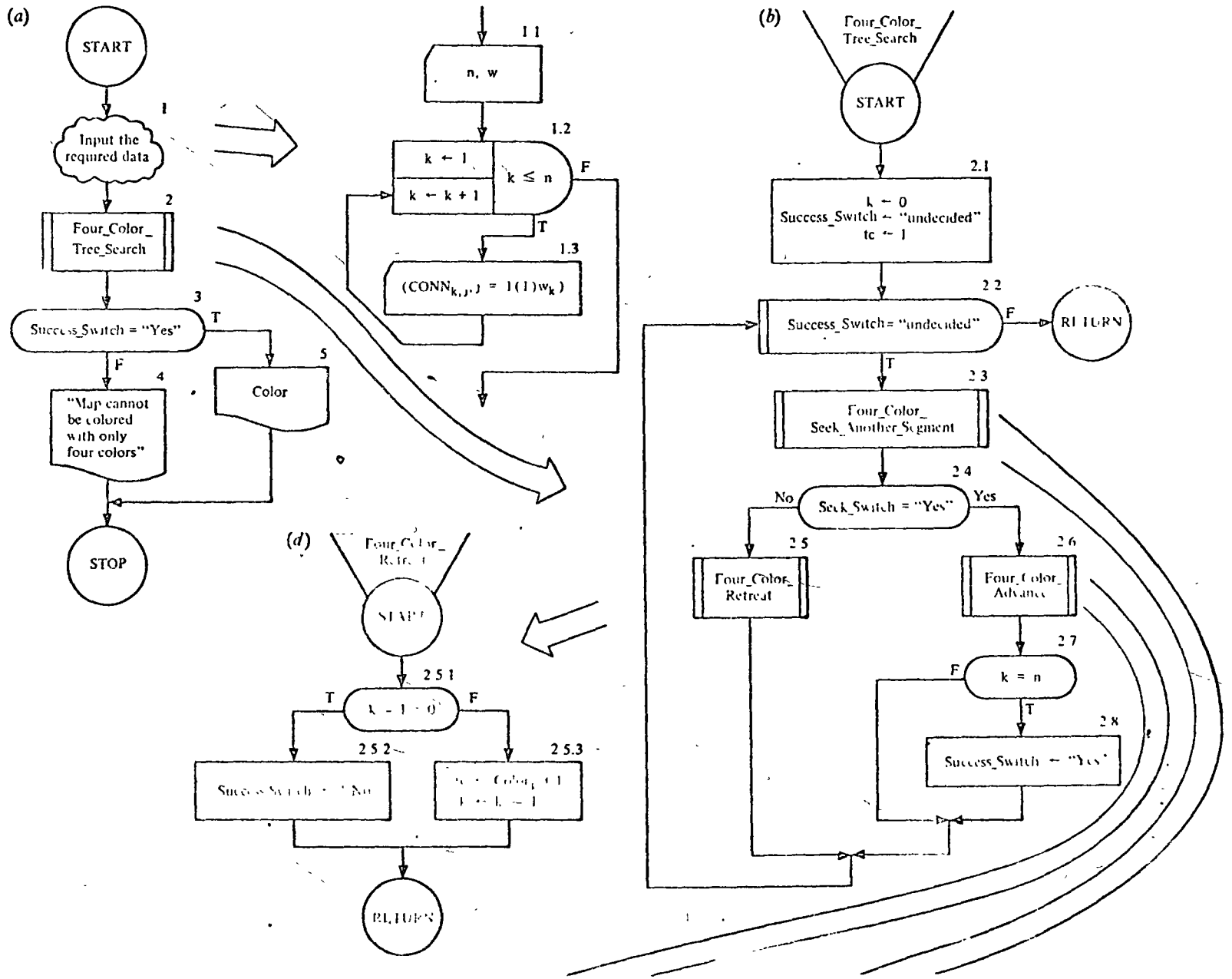
of *path* automatically identifies the color chosen for the ith country! This means that whenever we have been able to choose a valid color for the nth country, the current contents of the path is the desired list of colors for the n countries of the map. Nothing could be simpler. For this problem, let us call the path list COLOR, since it is more suggestive of our desired objective.

Figure 6·24 shows a flowchart algorithm incorporating the foregoing concepts and details and following the identical top-down structure given in the generalized search (Figure 6·16). It will be easy to verify the claimed similarity. If you have any difficulty in following Figure 6·24, remember that boxes with corresponding numbers in the generalized flowchart have similar meanings. Only Figure 6·24*f*, the detail of the admissibility test in box 2.3.2, is really new.

Discussion

Box 1 of Figure 6·24 is a counterpart to box 1 of Figure 6·16. In the detailed algorithm we must input the data explicitly to represent the map if we are going to deduce the actual structure of the tree. In Figure 6·24*b*, to keep track of what tree level has been reached, a level or path length counter k is needed. This counter is initially set to 0 in box 2.1 to reflect the start of the search at the root node. [The algorithm could be made more efficient by initializing the level counter to 3 and path to (1, 2, 3) to reflect coloring the first three countries with the first three colors, as suggested in Figure 6·21.]

Success_Switch is a three-valued switch variable that is initially set to "undecided" (at box 2.1), and then is set to either "Yes" if the tree search succeeds or to "No" if the search fails. To see why or where this switch is set to either "Yes" or to "No," you may have to descend to the next levels of detail. Thus, whenever we discover that the search is about to backtrack to level zero, the search has failed (boxes 2.5.1 and 2.5.2 of Retreat). If $k - 1 = 0$ in box 2.5.1, the current node is at level 1. In other words we have backtracked to the first country. The first country was colored with color 1 at the beginning of the search. We have not tested colors 2, 3, or 4 on country 1. Should we? Not really, because we know that any coloring we find will simply be a renaming of a previous coloring (if any) when country 1 had color 1. We won't find


(a)
START
1
Input the required data
2
Four_Color_ Tree_Search
3
Success_Switch = "Yes"
T
F
4
"Map cannot be colored with only four colors"
5
Color
STOP
11
n, w
1.2
k ← 1
k ← k + 1
k ≤ n
F
T
1.3
(CONN_k,j = 1(1)w_k)
(b)
Four_Color_ Tree_Search
START
2.1
k ← 0
Success_Switch ← "undecided"
tc ← 1
2.2
Success_Switch = "undecided"
F
RETURN
T
2.3
Four_Color_ Seek_Another_Segment
2.4
No
Seek_Switch = "Yes"
Yes
2.5
Four_Color_ Retreat
2.6
Four_Color_ Advance
2.7
F
k = n
T
2.8
Success_Switch ← "Yes"
(d)
Four_Color_ Retreat
START
2.5.1
T
k - 1 = 0
F
2.5.2
2.5.3
RETURN

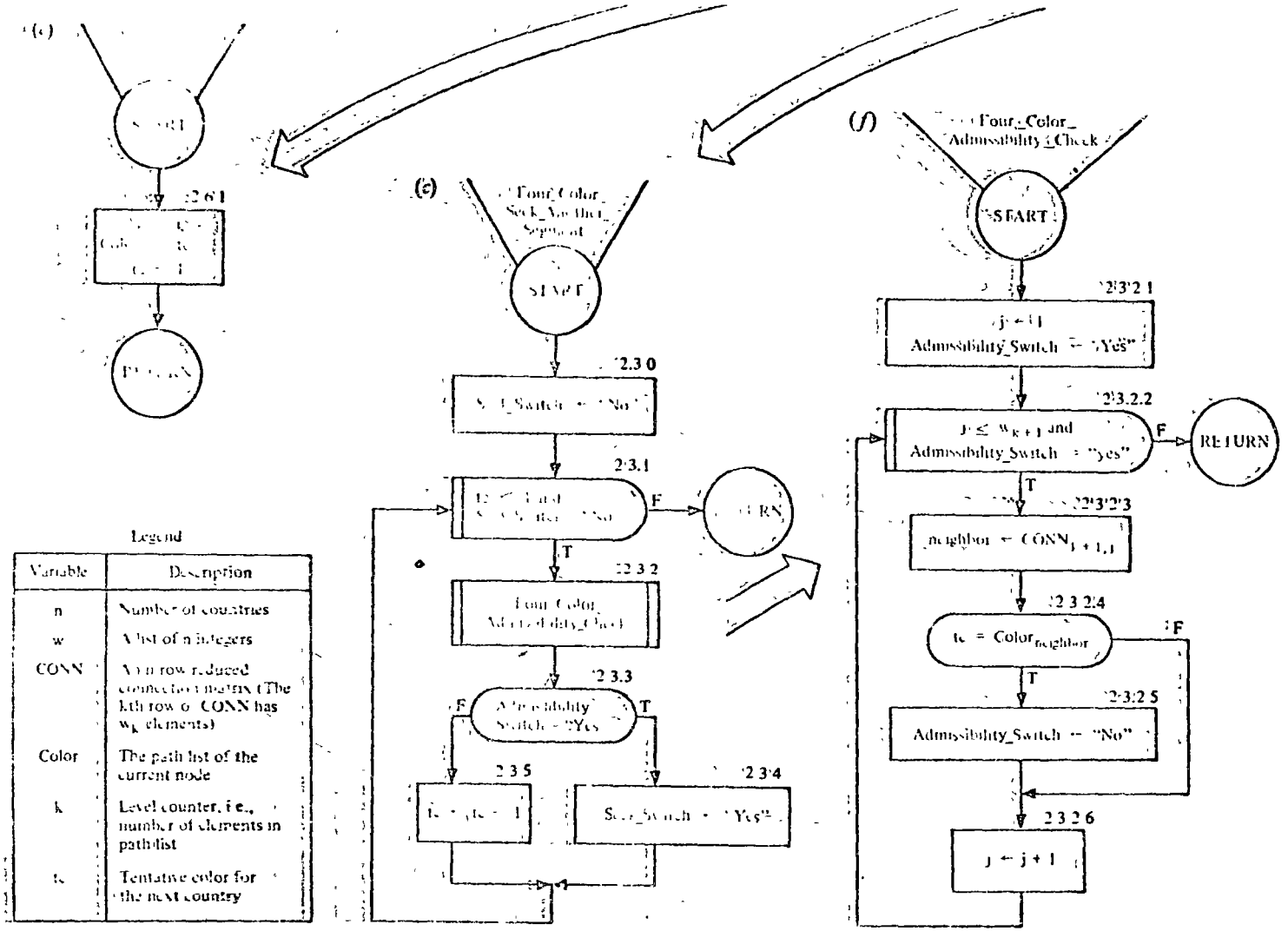
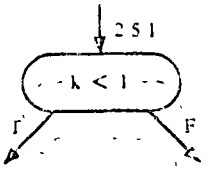



FIGURE 6.24
Four-coloring algorithm.
Top-down approach.

any new patterns. If we want to search the complete tree, however, including the four possible colors for the first country, we have only to change the test in box 2.5.1 to read




Whenever the level counter k is found to equal n (boxes 2.7 and 2.8 of Tree_Search), Success_Switch is set to "Yes." The variable Seek_Switch tested by Tree_Search in box 2.4 is set to either "Yes" or "No" by Seek_Another_Segment (Figure 6·24*c*). This procedure in turn reports success if and only if the subprocedure Admissibility_Check reports success. This latter procedure (Figure 6·24*f*) determines whether any of the previously colored neighbors (there are $w_{k+1}$ of them) have the same color as the *tentative* color, tc, that is being considered for the k + 1 st country. If so, Admissibility_Switch is set to "No" so that, after the RETURN to Seek_Another_Segment, another (the next) color may be tried. Notice that only in Admissibility_Check is there any reference to the map's representation. This suggests that detailed flowcharts for different natural-order tree search problems will differ mainly in the details of this particular part of the search algorithm.

The bookkeeping of Retreat and Advance in Figure 6·24 uses auxiliary variable, tc, tentative color. This variable is also used in Seek_Another_Segment during the search for an admissible segment and, in box 2.3.5, tc is incremented whenever an inadmissible segment is found. In Advance, k is incremented to represent the longer successful coloring path. The successful tentative color is stowed away in $Color_k$, and the auxiliary variable tc is reset to 1. (See box 2.6.1.) During *Retreat* the current color choice for country k must be remembered so that the search for another segment of country k's parent can resume at a value of tc that is one greater than the last one tried. The saving of this information is accomplished by the assignment step,

$$tc \leftarrow Color_k + 1$$

as seen in box 2.5.3. Then the path length k is shortened by 1.

In the problem set that follows, you are introduced to several well-known problems involving tree search. Here is your chance to apply our generalized natural-order tree search method.

EXERCISES 6·3, SET B

1. *The Eight Queens Problem.* A chessboard is an eight-by-eight array of positions. The Queen is the most powerful piece in the game of chess in that it can capture any other piece encountered on the same row, column, or diagonal. The problem is to so place eight Queens on the chessboard so that no Queen can capture another Queen.

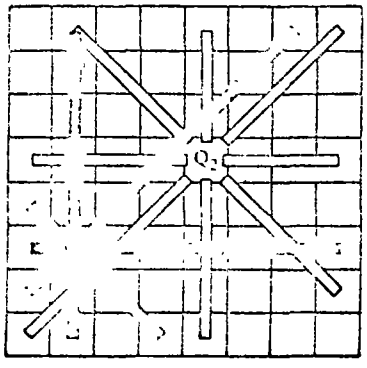


FIGURE 6·25
A chessboard with two Queens on different rows, columns, and diagonals.

If there is a solution to the Eight Queens Problem, it is evident that each Queen must be on a different row, column, and diagonal of the chessboard (Figure 6·25). This suggests the need for a systematic way of placing the eight Queens on the board, one at a time. It is certainly immaterial where the first Queen should go but, to be systematic, we can think of putting it somewhere in column 1 with the object of placing each successive Queen in a succeeding column.

In placing the first Queen in column one, there are eight choices, each of which eliminates some of the choices for placing a Queen in column two. These eight choices may be represented by a tree with eight segments emanating from the root node. As one moves down this tree of choices, there will be fewer and fewer admissible branches. A solution to the Eight Queens Problem is represented by a path through the tree reaching all the way to level eight.

The natural-order tree search is suitable for searching the tree, but it is necessary to be explicit about the test to determine which segments of the tree are admissible. Although it is tempting to represent the chessboard as an eight-by-eight array, it is easy to see that a single eight-element list, say $\{Q_i,\ i = 1(1)8\}$, will suffice, since in the Q list we can store the row number for each Queen.

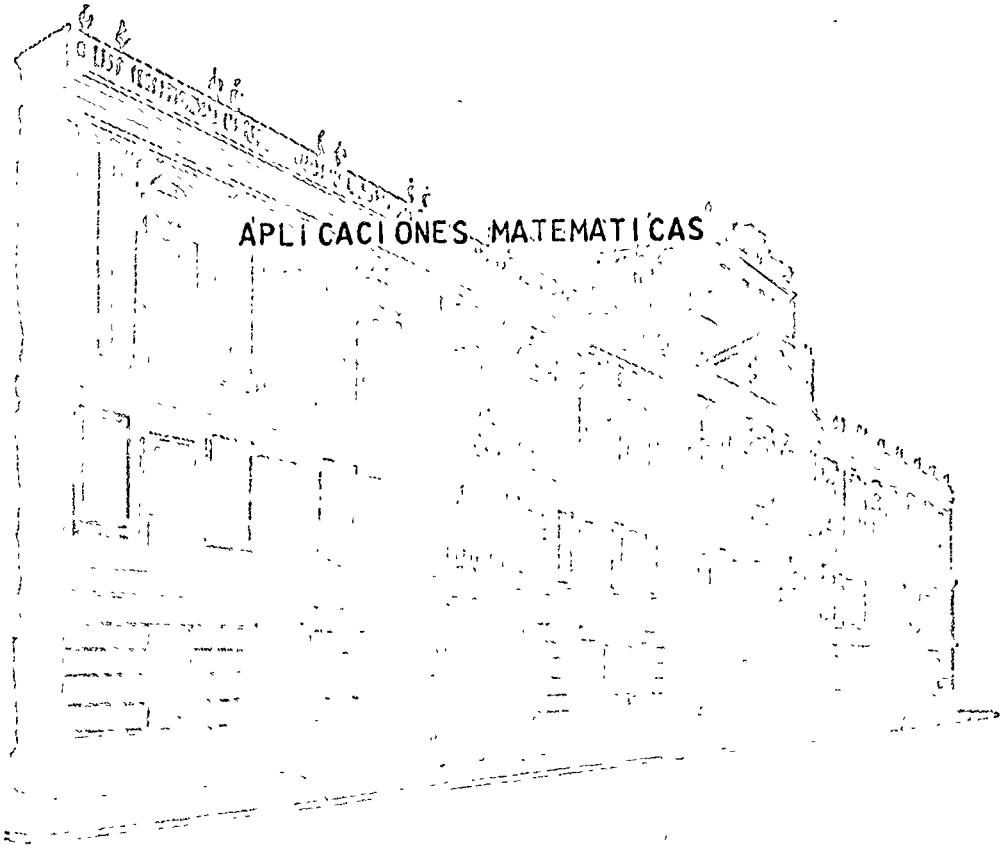
Suppose that k Queens have already been placed admissibly in the first k columns of the board. To determine whether the next Queen can be placed in position j, k + 1, at least two tests must be made.

(a) Is there already a Queen in row j? That is, has the value j already been assigned to an element of the Q list? If so, this position (j, k + 1) is inadmissible.
(b) Is there already a Queen on one of the two diagonals that pass through the new position? The first diagonal, which we will call a "major" diagonal, slants from upper left to lower right. The second one, a "minor" diagonal, slants from lower left to upper right.

If the answers to all these tests are negative, the new position is admissible. You should give thought to various ways of representing the needed data and performing the required tests. One way to record the positions of the Queens (least amount of storage) is with a single eight-element list whose ith element is the row number of the ith

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA




ING. ARMANDO TORRES FENTANES

Febrero, marzo de 1977

# 7. APROXIMACION POLINOMIAL Y COEFICIENTE DE CORRELACION

## 7.1 Introducción

En muchas ocasiones a partir de una serie de valores muestrales, donde existe una variable dependiente y una o varias -- variables independientes, es necesario ajustar dichos puntos -- por una curva tal que permita determinar el valor de la variable dependiente para cualquier valor de las variables independientes. La curva de ajuste por el método de los mínimos cuadrados puede ser un polinomio de grado "n", una función de tipo logarítmico, etc.; dicha curva se escoge de acuerdo a la distribución de los puntos muestrales y en forma tal que se minimice la suma de los cuadrados de los errores. En procesos estadísticos a tal tipo de ajuste se le denomina regresión simple o múltiple de la variable dependiente sobre las variables independientes. El grado de relación existente entre la variable dependiente y la independiente se denomina correlación y a la medida de tal relación se le llama coeficiente de correlación, el cual se suele denotar con el símbolo $\rho$ ó $r$ 1(2.3...n). Donde:

$$\rho = \sqrt{\frac{\text{Variación explicada}}{\text{Variación total}}} \qquad (7.1)$$

Si se considera a Y como la variable dependiente, las -- variaciones se definen en la siguiente forma:

$$\text{variación total} = \Sigma (Y - \bar{Y})^2 \; * \qquad (7.2)$$

$$\text{variación explicada} = \Sigma (Yest - \bar{Y})^2 \qquad (7.3)$$

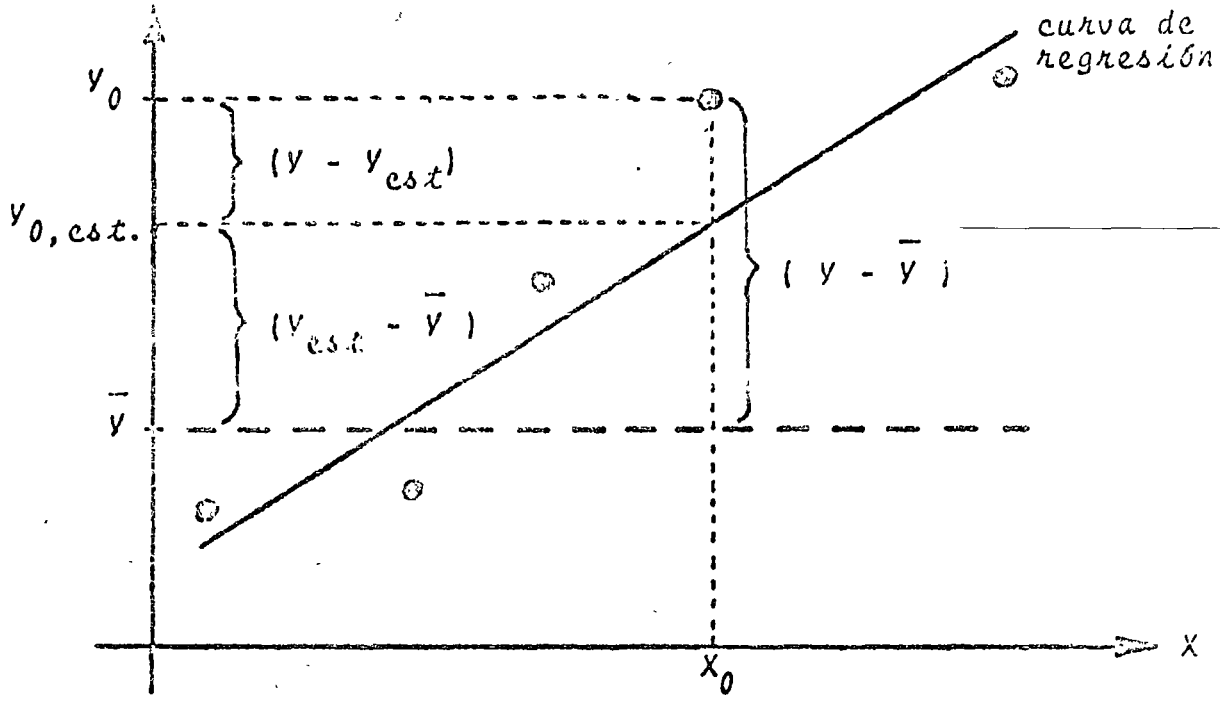
$$\text{variación no explicada} = \Sigma (Y - Yest)^2 \qquad (7.4)$$

$$\Sigma (Y - \bar{Y})^2 = \Sigma (Yest - \bar{Y})^2 + \Sigma (Y - Yest)^2 \qquad (7.5)$$

---

* $\bar{Y} = \dfrac{\sum_{i=1}^{N} Y_i}{N}$, para N puntos muestrales

Gráficamente, para el caso de regresión simple se tendrá:




## 7.2 Método de los Mínimos Cuadrados

### 7.2.1 Objeto

Efectuar la regresión lineal o exponencial, simple o múltiple, de la variable $X_1$ sobre la(s) variable(s) $X_2$, ..., $X_n$ a partir de una tabla de "m" puntos muestrales con la siguiente configuración:

| Punto | $X_1$ | $X_2$ | ... | $X_n$ |
|---|---|---|---|---|
| 1 | | | | |
| . | | | | |
| . | | | | |
| . | | | | |
| m | | | | |

Para efectuar el ajuste se emplea el método de los mínimos cuadrados. Además, se proporciona el coeficiente de correlación y las desviaciones estándar de los parámetros de la cur

va.

Las curvas de ajuste serán del tipo:

$$\left.\begin{aligned} X_1 &= A_1 + A_2X_2 + A_3X_3 + \ldots + A_nX_n \\ \text{ó} \\ X_1 &= e^{A_1}\ e^{A_2X_2}\ e^{A_3X_3} \ldots e^{A_nX_n} \end{aligned}\right\} \qquad (7.6)$$

Si se desea como curva de ajuste un polinomio de grado - "(n-1)" o sea:

$$X_1 = A_1 + A_2X_2 + A_3\,X_2^2 + \ldots + A_nX_2^{n-1} \qquad (7.7)$$

solo se requerirá efectuar el siguiente cambio de variable:

$$\left.\begin{aligned} X_2 &= X_2 \\ X_3 &= X_2 \\ X_4 &= X_2 \\ &\cdot \\ &\cdot \\ &\cdot \\ X_n &= X_2^{n-1} \end{aligned}\right\} \qquad (7.8)$$

en el momento de proporcionar los datos.

7.2.2 Método

Dadas un conjunto de "m" observaciones de la variable -- dependiente $X_1$ sobre una o varias variables independientes $X_2$, ..., $X_n$ se busca ajustar los datos mediante la siguiente curva:

$$f(\underline{X}) = \widehat{X_1} = A_1 + A_2X_2 + A_3X_3 + \ldots + A_nX_n \ ^* \qquad (7.9)$$

El valor de la variable dependiente correspondiente al - valor de las variables independientes en un punto muestral $\underline{X_i}$ es $X_{1,i}$, por lo que el error será:

$$e_i = f(\underline{X_i}) - X_{1,i} \qquad (7.10)$$

$$e_i = A_1 + A_2X_{2,i} + A_3X_{3,i} + \ldots + A_nX_{n,i} - X_{1,i} \qquad (7.11)$$

---

* $\widehat{X_1}$ = valor estimado de $X_1$

y la suma de los cuadrados de los errores considerando todos los puntos muestrales es:

$$\sum_{i=1}^{m} e_i^2 = \sum_{i=1}^{m} \left[ A_1 + A_2 X_{2,i} + A_3 X_{3,i} + \ldots + A_n X_{n,i} - X_{1,i} \right]^2 \qquad (7.12)$$

para obtener el mínimo de la suma de los cuadrados de los errores se deriva la expresión (7.12) con respecto a los parámetros $A_j$ y cada una de las derivadas se iguala a cero para toda $j$:

$$\frac{\partial}{\partial A_j} \sum_{i=1}^{m} e_i^2 = \frac{\partial}{\partial A_j} \sum_{i=1}^{m} \left[ A_1 + A_2 X_{2,i} + \ldots + A_n X_{n,i} - X_{1,i} \right]^2$$

$$= \sum_{i=1}^{m} 2 \left[ A_1 + A_2 X_{2,i} + \ldots + A_n X_{n,i} - X_{1,i} \right] X_{j,i}$$

$$= 0 \qquad (7.13)$$

lo cual se cumple solo si:

$$A_1 \sum_{i=1}^{m} X_{j,i} + A_2 \sum_{i=1}^{m} X_{2,i} X_{j,i} + \ldots + A_n \sum_{i=1}^{m} X_{n,i} X_{j,i} = \sum_{i=1}^{m} X_{1,i} X_{j,i} \qquad (7.14)$$

al evaluar (7.14) para toda "j" se tiene:

$$\begin{aligned}
m A_1 &+ A_2 \Sigma X_2 &+ A_3 \Sigma X_3 &+ \ldots + A_n \Sigma X_n &= \Sigma X_1 \\
A_1 \Sigma X_2 &+ A_2 \Sigma X_2^2 &+ A_3 \Sigma X_3 X_2 &+ \ldots + A_n \Sigma X_n X_2 &= \Sigma X_1 X_2 \\
&\vdots & & \vdots & \vdots \\
A_1 \Sigma X_n &+ A_2 \Sigma X_2 X_n &+ A_3 \Sigma X_3 X_n &+ \ldots + A_n \Sigma X_n^2 &= \Sigma X_1 X_n
\end{aligned} \qquad (7.15)$$

expresando en forma matricial:

$$\begin{bmatrix} m & \Sigma X_2 & \cdots & \Sigma X_n \\ \Sigma X_2 & \Sigma X_2^2 & \cdots & \Sigma X_2 X_n \\ \vdots & & & \vdots \\ \Sigma X_n & \Sigma X_2 X_n & \cdots & \Sigma X_n^2 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} = \begin{bmatrix} \Sigma X_1 \\ \Sigma X_1 X_2 \\ \vdots \\ \Sigma X_1 X_n \end{bmatrix} \qquad (7.16)$$

Al resolver el sistema de ecuaciones (7.16) se obtienen los parámetros $A_j$ de la curva de regresión.

Dentro del programa se plantea el sistema de ecuaciones, obteniendo la sumatoria de los puntos muestrales para cada variable, para el cuadrado de la variable y para los productos - cruzados. El sistema de ecuaciones se resuelve por el método - de Gauss-Jordan modificado.

.El coeficiente de correlación se obtiene de la siguiente forma:

$$r_{1(23\ldots n)} = \sqrt{\frac{\text{variación explicada}}{\text{variación total}}}$$

$$= \sqrt{\frac{\Sigma x^2_{1(23\ldots n)}}{\Sigma x_1^2}} \; * \qquad (7.17)$$

donde:

$$\underset{\text{var. total}}{\Sigma x_1^2} = \underset{\text{var. no explicada}}{\Sigma x^2_{1.23\ldots n}} + \underset{\text{var. explicada}}{\Sigma x^2_{1(23\ldots n)}} \qquad (7.18)$$

las componentes de la ecuación (7.18) están dadas por:

$$\sum_{i=1}^{m} x^2_{1,i} = \sum_{i=1}^{m} X_{1,i} - \frac{\left(\sum_{i=1}^{m} X_{1,i}\right)^2}{m} \qquad (7.19)$$

---

* $x = X - \bar{X}$

$$\Sigma x_1^2(23\ldots n) = A_2 \ \Sigma x_1 x_2 + A_3 \ \Sigma x_1 x_3 + \ldots + A_n \ \Sigma x_1 x_n \qquad (7.20)$$

$$\left.\begin{aligned} \Sigma x_1 x_2 &= \Sigma X_1 X_2 - \frac{\Sigma X_1 \ \Sigma X_2}{m} \\ \Sigma x_1 x_3 &= \Sigma X_1 X_3 - \frac{\Sigma X_1 \ \Sigma X_3}{m} \\ &\vdots \\ \Sigma x_1 x_n &= \Sigma X_1 X_n - \frac{\Sigma X_1 \ \Sigma X_n}{m} \end{aligned}\right\} \qquad (7.21)$$

A los términos de las ecuaciones (7.19) y (7.21) se les - denomina elementos de variación y covariación respectivamente, ya que $x = X - \bar{X}$ y la variancia y covariancia se definen como:

$$\sigma_X^2 = E\left[X - \bar{X}\right]^2 \qquad (7.22)$$

$$cov_{XY} = E\left[X - \bar{X}\right]\left[Y - \bar{Y}\right] \qquad (7.23)$$

A continuación se describe la obtención de la desviación estándar de los parámetros $A_j$.

Se define a los productos $\underline{X}'\underline{X}$ y $\underline{X}'\underline{Y}$ como:

$$\underline{X}'\underline{X} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ X_{2,1} & X_{2,2} & \ldots & X_{2,m} \\ \vdots & & & \vdots \\ X_{n,1} & X_{n,2} & \ldots & X_{n,m} \end{bmatrix} \begin{bmatrix} 1 & X_{2,1} & \ldots & X_{n,1} \\ 1 & & & \\ \vdots & & & \vdots \\ 1 & X_{2,m} & \ldots & X_{n,m} \end{bmatrix}$$

$$= \begin{bmatrix} n & \Sigma X_2 & \ldots & \Sigma X_n \\ \Sigma X_2 & \Sigma X_2^2 & \ldots & \Sigma X_2 \Sigma X_n \\ \vdots & & & \vdots \\ \Sigma X_n & \Sigma X_2 \Sigma X_n & \ldots & \Sigma X_n^2 \end{bmatrix} \qquad (7.24)$$

$$\underline{X}'\,\underline{Y} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ X_{2,1} & & \cdots & X_{2,m} \\ \vdots & & & \vdots \\ X_{n,1} & & \cdots & X_{n,m} \end{bmatrix} \begin{bmatrix} X_{1,1} \\ X_{1,2} \\ \vdots \\ X_{1,m} \end{bmatrix} = \begin{bmatrix} \Sigma X_1 \\ \Sigma X_1 X_2 \\ \vdots \\ \Sigma X_1 X_n \end{bmatrix} \qquad (7.25)$$

Sea $\underline{A}$ el vector de los parámetros de la ecuación de regresión, el sistema de ecuaciones (7.16) se puede denotar en base a lo anterior como:

$$(\underline{X}'\,\underline{X})\,\underline{A} = \underline{X}'\,\underline{Y} \qquad (7.26)$$

donde $\underline{Y}$ representa el vector de valores muestrales de la variable dependiente. Por lo tanto:

$$\underline{Y} = \underline{X}\,\underline{A} + \underline{e} \quad * \qquad (7.27)$$

$$\underline{A} = (\underline{X}'\,\underline{X})^{-1}\,\underline{X}'\,\underline{Y} \qquad (7.28)$$

Se asume que la ecuación de regresión es de la forma:

$$\underline{Y} = \underline{X}\,\underline{\alpha} + \underline{\varepsilon} \qquad (7.29)$$

A partir de las expresiones anteriores se puede demostrar (ver ref. 1) que la variancia de los parámetros $A_i$ está dada por:

$$E\ (\underline{A} - \underline{\alpha})(\underline{A} - \underline{\alpha})' = \sigma^2\,\underline{I}\,(\underline{X}'\,\underline{X})^{-1} \qquad (7.30)$$

El valor de $\sigma^2$ se obtiene mediante la expresión:

$$\sigma^2 = S^2_{1.23\ \ldots\ n} = \frac{\Sigma x^2_{1.23\ \ldots\ n}}{m-n} \qquad (7.31)$$

donde "m" representa la cantidad de puntos muestrales y "n" la cantidad de variables independientes.

Para efectos de regresión exponencial del tipo:

$$X_1 = e^{A_1} e^{A_2 X_1} \ldots e^{A_{n+1} X_n} \qquad (7.32)$$

---

* $\underline{e}$ : es el vector de diferencias entre los valores estimados y los valores reales.

el programa aplica el operador "Ln" en ambos términos de la ecuación (7.32) con lo que dicha expresión se linealiza y esta expresión linealizada es la que se utiliza para la obtención de los parámetros $\Lambda_j$.

7.2.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE SISTOR(N,M,C,A,B), plantea el sistema de ecuaciones requerido para la obtención de los parámetros de la curva de regresión.

SUBROUTINE GAUTOR(A,B,M,EPS,DET), obtiene la solución del sistema de ecuaciones mediante el método de Gauss-Jordan. Consultar el capítulo 3.

SUBROUTINE MULTMA(A,B,N,M,L,X), efectua productos matriciales. Consultar el capítulo 2.

SUBROUTINE MATINV(A,N,EPS,DET), obtiene la inversa de una matriz por el método de Gauss-Jordan. Consultar el capítulo 2.

SUBROUTINE GRAFI(A,N,M), obtiene la gráfica de los valores muestrales y de los valores estimados. Consultar el capítulo 1.

b) Descripción de las variables:

Para la subrutina SISTOR:

| | |
|---|---|
| N | cantidad de puntos muestrales |
| M | cantidad de variables, incluyendo la dependiente |
| C(I,J) | valor de la variable $X_j$ para el punto muestral "i" |
| A(I,J) | matriz de coeficientes del sistema de ecuaciones |
| B(I) | vector de términos independientes del sistema de ecuaciones |
| SUM | variable que guarda la sumatoria de los puntos muestrales |

Para el programa principal:

| | |
|---|---|
| N | cantidad de puntos muestrales |
| M | cantidad de variables incluyendo la de- |

| | |
|---|---|
| | pendiente |
| NTIPO | variable que indica el tipo de regresión a efectuar |
| C(I,J) | valor de la variable $X_j$ para el punto muestral "i" |
| A(I,J) | matriz de coeficientes del sistema de ecuaciones |
| B(I) | vector de términos independientes del sistema de ecuaciones |
| EPS | criterio para determinar si el determinante de $\underline{A}$ es nulo |
| DET | variable que indica si el determinante de la matriz $\underline{A}$ es o no nulo |
| PROD(I) | sumatorias de los productos cruzados $X_i X_j$ |
| CM(I,J) | matriz modificada de la matriz $\underline{C}$ donde CM(I,1)=1 |
| XTR(I,J) | matriz transpuesta de la matriz CM |
| XI(I,J) | matriz inversa del producto matricial (XTR)(CM) |
| A1 | coeficiente de correlación |
| VAR | variancia no explicada |
| BVAR(I) | desviación estándar de los parámetros de la curva de regresión |
| ATEMP(I) | variable de reemplazo |

c) Dimensiones:

El programa está estructurado para trabajar como máximo con cinco variables independientes. La proposición DIMENSION deberá modificarse en el caso de que la cantidad de puntos muestrales sea mayor de 30.

d) Formatos para los datos de entrada:

| SEC. TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (2I5,A4) | N,M,NTIPO, para la variable NTIPO se deberá perforar LINE en el caso de regresión lineal y EXPO para regresión exponencial. |

| | | |
|---|---|---|
| 2 | (8F10.0) | C(I,J), los elementos de la matriz se dan columna por columna. Emplear tantas tarjetas como sean necesarias. |
| . | | |
| . | | |
| . | | |

---

otros paquetes de datos (opcional)

---

| | |
|---|---|
| n | TARJETA EN BLANCO, al finalizar toda la información |

e) Diagrama de bloques:

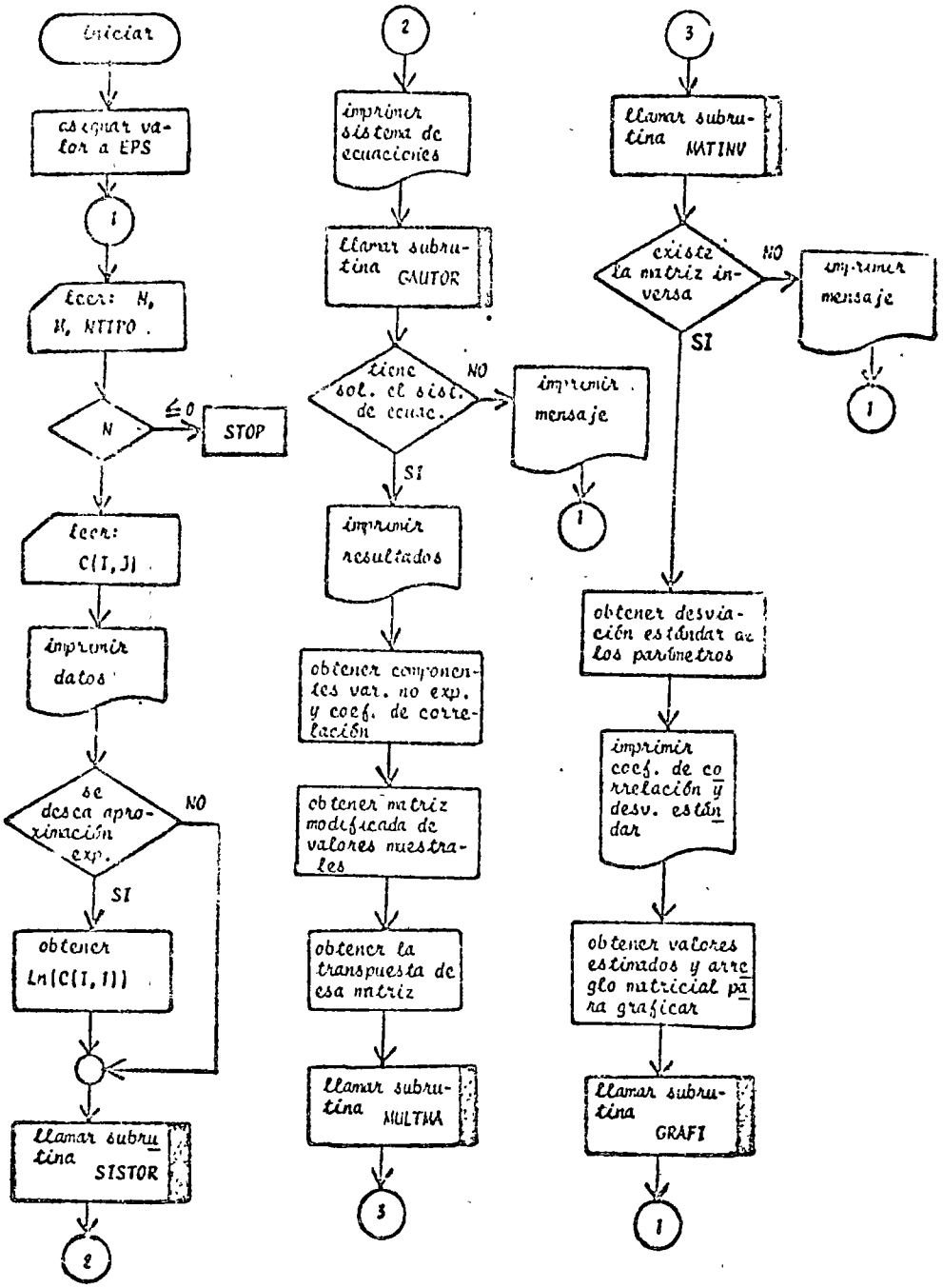



Fig. 7.1 Diagrama de bloques para el programa principal

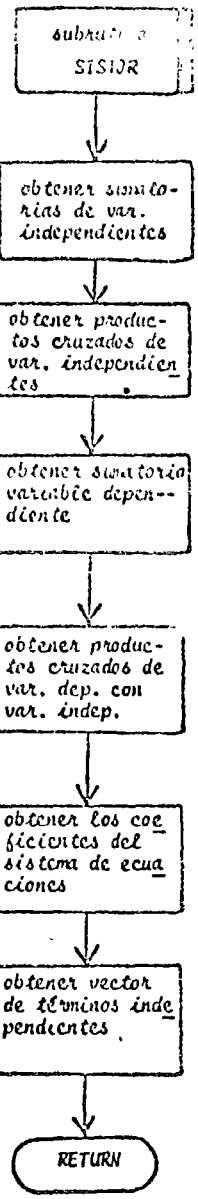



*Fig. 7.2 Diagrama de bloques para la subrutina SISTOR.*

6) Listado:



```
C
C     PROGRAMA PARA EFECTUAR REGRESION LINEAL O EXPONENCIAL MULTIPLE
C     EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C     N=CANTIDAD DE PUNTOS MUESTRALES
C     M=CANTIDAD DE VARIABLES(INCLUYENDO LA V.DEPENDIENTE)
C     NTIPO=VARIABLE QUE INDICA EL TIPO DE REGRESION A EFECTUAR(EXPO O
C           LINE)
C     C(I,J)=VALOR DE LA VARIABLE X(J) PARA EL PUNTO MUESTRAL I
C     C(I,1)=VARIABLE DEPENDIENTE
C     A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES, QUE SIRVEN PA-
C       RA ENCONTRAR LOS PARAMETROS DE LA FUNCION DE REGRESION
C     B=PARAMETROS DE LA FUNCION DE REGRESION
C     EPS=VALOR CON EL QUE SE CONSIDERA NULO EL DETERMINANTE DE LA MA-
C       TRIZ A
C     DET=VALOR ABSOLUTO DEL DETERMINANTE DE LA MATRIZ A
C     PROD=SUMATORIAS DE LOS PRODUCTOS CRUZADOS CON LA V. DEPENDIENTE
C     CM=MATRIZ MODIFICADA DE LA MATRIZ C DONDE CM(I,1)=1
C     XTR=TRANSPUESTA DE LA MATRIZ CM
C     XI=MATRIZ INVERSA DEL PRODUCTO MATRICIAL XTR*CM
C     BVAR=DESVIACION ESTANDAR DE LOS PARAMETROS B
C     VAR=VARIACION NO EXPLICADA
C     AI=COEFICIENTE DE CORRELACION
C
C     LA FUNCION DE REGRESION ES DEL SIGUIENTE TIPO
C     X(1)=B(1) + B(2)X(2) + B(3)X(3) + ... + B(M)X(M)
C
C     DIMENSIONES PARA OBTENER LOS PARAMETROS DE LA FUNCION DE REGRESION

      DIMENSION C(30,10),X(10),B(10),A(10,10),ATEMP(30)
C
C     DIMENSIONES PARA OBTENER LA DESVIACION ESTANDAR DE B(I)
      DIMENSION PROD(10),CM(30,10),XTR(10,30),XI(10,10),BVAR(10)
C
      DATA X/4HX(1),4HX(2),4HX(3),4HX(4),4HX(5),4HX(6),4HX(7),4HX(8),4HX
     1(9),5H(10)/
C
C     LECTURA DE DATOS
C
      EPS=0.000001
    1 READ(5,50) N,M,NTIPO
      IF(N) 2,2,3
    2 CALL EXIT
    3 DO 4 J=1,M
    4 READ(5,51) (C(I,J),I=1,N)
      LIN=4HLINE
C
C     IMPRESION DE
C
      WRITE(6,52) N,M
      WRITE(6,53) (X(I),I=1,M)
      DO 5 I=1,N
    5 WRITE(6,54) (C(I,J),J=1,M)
C
C     LLAMADO DE SUBRUTINA  PARA FORMAR EL SISTEMA DE ECUACIONES
C
      IF(NTIPO.EQ.LIN) GO TO 26
      DO 25 I=1,N
      ATEMP(I)=C(I,1)
   25 C(I,1)=ALOG(C(I,1))
   26 CALL SISTOR(N,M,C,A,B)
C
C     IMPRESION DEL SISTEMA DE ECUACIONES
C
      WRITE(6,551)
      DO 6 I=1,M
    6 WRITE(6,550) (A(I,J),J=1,M),B(I)
C
C     LLAMADO DE SUBRUTINA PARA RESOLVER EL SISTEMA DE ECUACIONES
C
      CALL GAUTOR(A,B,M,EPS,DET)
      IF(DET.LE.EPS) GO TO 8
C
C     IMPRESION DE RESULTADOS
C
      WRITE(6,55)
      DO 7 I=1,M
    7 WRITE(6,56) I,B(I)
      WRITE(6,552) NTIPO
C
C     OBTENCION DE SUMX(1)=AI
C
      AI=0.0
      DO 9 I=1,N
    9 AI=AI + C(I,1)
```

```
C
C     OBTENCION DE SUM(X(I)*X(I))=A2, SUMA(I)=A3
C
      DO 11 I=1,M
      A2=0.0
      A3=0.0
      DO 10 J=1,N
      A2=A2 + C(J,I)*C(J,I)
   10 A3=A3 + C(J,I)
      PROD(I)=A2 - (A1*A3)/FLOAT(N)
   11 CONTINUE
      WRITE(6,54) (PROD(I),I=1,M)
C
C     OBTENCION DE VAR. NO EXPLICADA Y DEL COEF. DE CORRELACION
C
      VAR=0.0
      DO 12 I=2,M
   12 VAR=VAR + B(I)*PROD(I)
      A1=SQRT(VAR/PROD(1))
      VAR=PROD(1) - VAR
      VAR=VAR/FLOAT(N*M)
      WRITE(6,54) VAR
C
C     OBTENCION DE LA MATRIZ MODIFICADA DE VALORES MUESTRALES
C
      DO 15 I=1,N
      DO 14 J=1,M
      IF(J.EQ.1) GO TO 13
      CM(I,J)=C(I,J)
      GO TO 14
   13 CM(I,J)=1.0
   14 CONTINUE
   15 CONTINUE
C
C     OBTENCION DE CM TRANSPUESTA
C
      DO 16 I=1,N
      DO 16 J=1,M
   16 XTR(J,I)=CM(I,J)
C
C     LLAMADO DE SUBRUTINA PARA EFECTUAR PRODUCTO MATRICIAL
C
      CALL MULTMA(XTR,CM,M,N,M,XI)
C
C     LLAMADO DE SUBRUTINA PARA OBTENER LA MATRIZ INVERSA
C
      CALL MATINV(XI,M,EPS,DET)
      IF(DET.LE.EPS) GO TO 19
      DO 201 I=1,M
  201 WRITE(6,54) (XI(I,J),J=1,M)
C
C     OBTENCION DE LA DESVIACION ESTANDAR DE LOS PARAMETROS
C
      DO 17 I=1,M
      BVAR(I)=VAR*XI(I,I)
   17 BVAR(I)=SQRT(BVAR(I))
C
C     IMPRESION DE COEFICIENTES Y DESVIACIONES ESTANDAR
C
      WRITE(6,58)
      DO 18 I=1,M
   18 WRITE(6,59) B(I),BVAR(I)
      WRITE(6,60) A1
C
C     REACOMODO DE DATOS PARA GRAFICAR VALORES REALES Y VALORES ESTIMA-
C     DOS CON LA FUNCION DE REGRESION
C
      IF(NTIPO.NE.LIN) GO TO 27
      DO 21 I=1,N
      CM(I,1)=FLOAT(I)
      CM(I,2)=C(I,1)
      SUM=0.0
      DO 20 J=2,M
   20 SUM=SUM + C(I,J)*B(J)
      CM(I,3)=B(1) + SUM
   21 CONTINUE
      GO TO 36
   27 DO 29 I=1,N
      CM(I,1)=FLOAT(I)
      CM(I,2)=ATEMP(I)
      SUM=EXP(B(1))
      DO 28 J=2,M
   28 SUM=SUM*EXP(C(I,J)*B(J))
      CM(I,3)=SUM
   29 CONTINUE
```

```
C
C     LLAMADO DE SUBRUTINA PARA GRAFICAR
C
   30 CALL GRAFI(CM,N,3)
      GO TO 1
    8 WRITE(6,57)
      GO TO 1
   19 WRITE(6,61)
      GO TO 1
C
C     FORMATOS DE LECTURA E IMPRESION
C
   50 FORMAT(2I5,A4)
   51 FORMAT(8F10.0)
   52 FORMAT(1H1,5(/),5X,35HLA CANTIDAD DE PUNTOS MUESTRALES ES ,I5,3(/)
     1,5X,28HLA CANTIDAD DE VARIABLES ES ,I5)
   53 FORMAT(///,5(/),10X,26HLOS VALORES MUESTRALES SON,///,5X,A5,9(7X,A
     15),///)
   54 FORMAT(/,2X,10(1PE11.4,1X))
   55 FORMAT(///,5(/),10X,17HLOS COEFICIENTES DE LA FUNCION DE REGRESION
     1 SON,///,46X,1H1,17(,1HR(I),/)
   56 FORMAT(/,45X,I2,10(,1PE15.8)
   57 FORMAT(5(/),5X,42HEL SISTEMA DE ECUACIONES NO TIENE SOLUCION)
   58 FORMAT(1H1,3(/),10X,49HLAS DESVIACIONES ESTANDAR DE LOS COEFICIENT
     1ES SON,///,46X,4HR(I),21X,4HD.S.,///)
   59 FORMAT(/,40X,2(1PE15.8,10X))
   60 FORMAT(5(/),5X,34HEL COEFICIENTE DE CORRELACION ES  ,1PE15.8)
   61 FORMAT(5(/),5X,62HNO EXISTE LA INVERSA DE LA MATRIZ EMPLEADA PARA
     1CALCULAR LAS DESVIACIONES ESTANDAR)
  550 FORMAT(/,2X,1PE10.3,10(1X,1PE10.3))
  551 FORMAT(1H1,5(/),5X,27HEL SISTEMA DE ECUACIONES ES,/)
  552 FORMAT(//,5X,21HTIPO DE REGRESION    ,A4)
      END
```

*Fig. 7.3 Listado del programa principal*

```
      SUBROUTINE SISTOR(N,M,C,A,B)
C
C     SUBRUTINA PARA PLANTEAR EL SISTEMA DE ECUACIONES QUE PERMITE OBTE-
C     NER LOS PARAMETROS DE UNA FUNCION DE REGRESION LINEAL MULTIPLE
C     EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C     N=CANTIDAD DE PUNTOS MUESTRALES
C     M=CANTIDAD DE VARIABLES(INCLUYENDO LA DEPENDIENTE)
C     C(I,J)=VALOR DE LA VARIABLE X(J) PARA EL PUNTO MUESTRAL I
C     A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C     B=VECTOR DE TERMINOS INDEPENDIENTES DEL SISTEMA DE ECUACIONES
C
C     EL PLANTEAMIENTO APROVECHA LAS CARACTERISTICAS DE SIMETRIA DE LA
C     MATRIZ A
C
      DIMENSION C(30,10),A(10,10),B(10)
C
C     OBTENCION DEL SISTEMA DE ECUACIONES
C
      A(1,1)=N
      DO 6 I=1,M
      IF(I.NE.1) GO TO 4
      SUM=0.0
      DO 1 J=1,N
    1 SUM=SUM + C(J,I)
      B(I)=SUM
      IP1=I+1
      DO 3 J=IP1,M
      SUM=0.0
      DO 2 K=1,N
    2 SUM=SUM + C(K,J)
      A(I,J)=SUM
      A(J,I)=SUM
    3 CONTINUE
      GO TO 9
    4 SUM=0.0
      DO 5 J=1,N
    5 SUM=SUM + C(J,1)*C(J,I)
      B(I)=SUM
      DO 7 J=I,M
      SUM=0.0
      DO 6 K=1,N
    6 SUM=SUM + C(K,I)*C(K,J)
      A(I,J)=SUM
      A(J,I)=SUM
    7 CONTINUE
    8 CONTINUE
      RETURN
      END
```

Fig 7.4 Listado de la subrutina SISTOR

7.2.4 Ejemplo

Los valores observados para la demanda de energía eléctrica en el sector residencial desde 1962 hasta 1973 son:

| AÑO | DEMANDA RESIDENCIAL (MWH) |
|---|---|
| 1962 | 1418.757 |
| 1963 | 1578.572 |
| 1964 | 1816.236 |
| 1965 | 1970.987 |
| 1966 | 2256.216 |
| 1967 | 2548.05 |
| 1968 | 2803.79 |
| 1969 | 3152.095 |
| 1970 | 3582.568 |
| 1971 | 3979.667 |
| 1972 | 4431.655 |
| 1973 | 4930.197 |

Si se sabe que la demanda de energía eléctrica se encuentra estrechamente relacionada con el tiempo, el PNB, la población y el producto bruto del sector eléctrico del año anterior. Obtenga una curva de tipo lineal y otra de tipo exponencial que se ajuste lo mejor posible a los valores muestrales de la demanda residencial de energía eléctrica.

* SOLUCION

TABLA 7.1 Datos del problema del ejemplo 7.2.4

N = 12

M = 4

NTIPO = LINE 6 EXPO

| BANDA RESID. | TIEMPO | PNB | POBLACION | PB CFE_1 |
|---|---|---|---|---|
| 1418.757 | 62. | 165310. | 37439. | 1609. |
| 1578.572 | 63. | 178516. | 38727. | 1753. |
| 1816.236 | 64. | 199390. | 40059. | 2170. |
| 1970.987 | 65. | 212320. | 41437. | 2529. |
| 2256.216 | 66. | 227037. | 42863. | 2769. |
| 2548.05 | 67. | 241272. | 44338. | 3157. |
| 2803.79 | 68. | 260901. | 45863. | 3533. |
| 3152.095 | 69. | 277400. | 47441. | 4228. |
| 3582.568 | 70. | 296600. | 49031. | 4812. |
| 3979.667 | 71. | 306800. | 50778. | 5357. |
| 4431.655 | 72. | 329100. | 52539. | 5784. |
| 4930.197 | 73. | 354100. | 54560. | 6297. |

TABLA 7.2 Resultados del problema de ejemplo 7.2.4

EL SISTEMA DE ECUACIONES ES

| | | | | | |
|---|---|---|---|---|---|
| 1.200E+01 | 6.100E+02 | 3.049E+06 | 5.451E+05 | 4.400E+04 | 3.447E+04 |
| 6.100E+02 | 5.462E+04 | 2.987E+08 | 3.701E+07 | 3.033E+06 | 2.372E+06 |
| 3.049E+06 | 2.982E+08 | 8.146E+11 | 1.422E+11 | 1.224E+10 | 9.518E+09 |
| 5.451E+05 | 3.701E+07 | 1.422E+11 | 2.510E+10 | 2.097E+09 | 1.630E+09 |
| 4.400E+04 | 3.033E+06 | 1.224E+10 | 2.097E+09 | 1.897E+08 | 1.467E+08 |

LOS COEFICIENTES DE LA FUNCION DE REGRESION SON

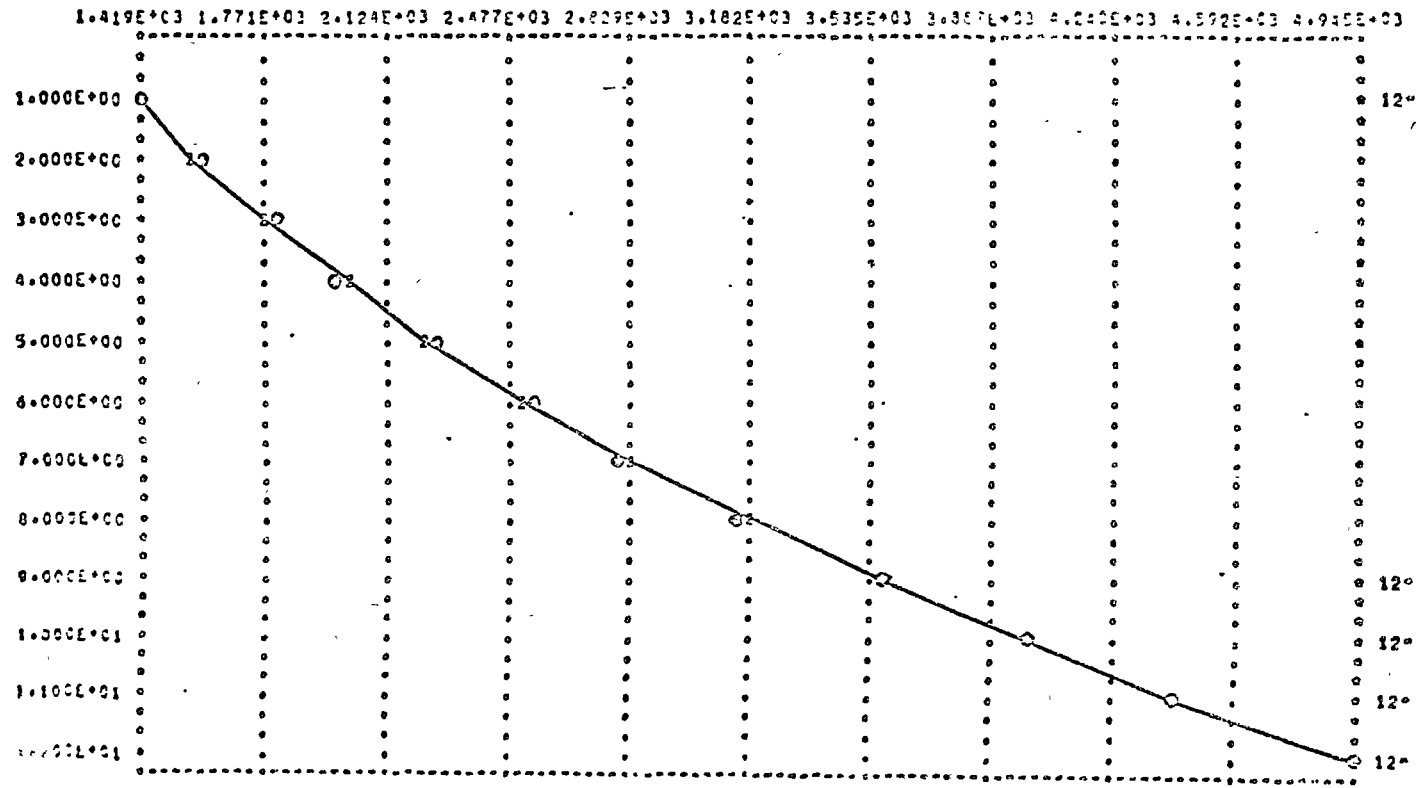
| I | B(I) |
|---|---|
| 1 | 1.25536780E+04 |
| 2 | -4.60028709E+02 |
| 3 | -7.20920646E-04 |
| 4 | 4.61541007E-01 |
| 5 | 1.76002042E-01 |

TIPO DE REGRESION LINE

LAS DESVIACIONES ESTANDAR DE LOS COEFICIENTES SON

| B(I) | D.S. |
|---|---|
| 1.25536780E+04 | 2.34876970E+03 |
| -4.60028709E+02 | 5.72169127E+01 |
| -7.20920646E-04 | 3.95054007E-03 |
| 4.61541007E-01 | 6.76429802E-02 |
| 1.76002042E-01 | 8.48344206E-02 |

EL COEFICIENTE DE CORRELACION ES 9.99738818E-01


1.419E+03 1.771E+01 2.124E+03 2.477E+03 2.829E+03 3.182E+03 3.535E+03 3.887E+03 4.240E+03 4.592E+03 4.945E+03
1.000E+00
2.000E+00
3.000E+00
4.000E+00
5.000E+00
6.000E+00
12°
12°
12°
12°
12°

EL SISTEMA DE ECUACIONES ES

| | | | | | |
|---|---|---|---|---|---|
| 1.200E+01 | 8.100E+02 | 3.049E+06 | 5.451E+05 | 4.400E+04 | 9.464E+01 |
| 8.100E+02 | 5.482E+04 | 2.062E+08 | 3.701E+07 | 3.033E+06 | 6.405E+03 |
| 3.049E+06 | 2.062E+08 | 8.146E+11 | 1.422E+11 | 1.224E+10 | 2.432E+07 |
| 5.451E+05 | 3.701E+07 | 1.422E+11 | 2.510E+10 | 2.097E+09 | 4.324E+06 |
| 4.400E+04 | 3.033E+06 | 1.224E+10 | 2.097E+09 | 1.897E+08 | 3.542E+05 |

LOS COEFICIENTES DE LA FUNCION DE REGRESION SON

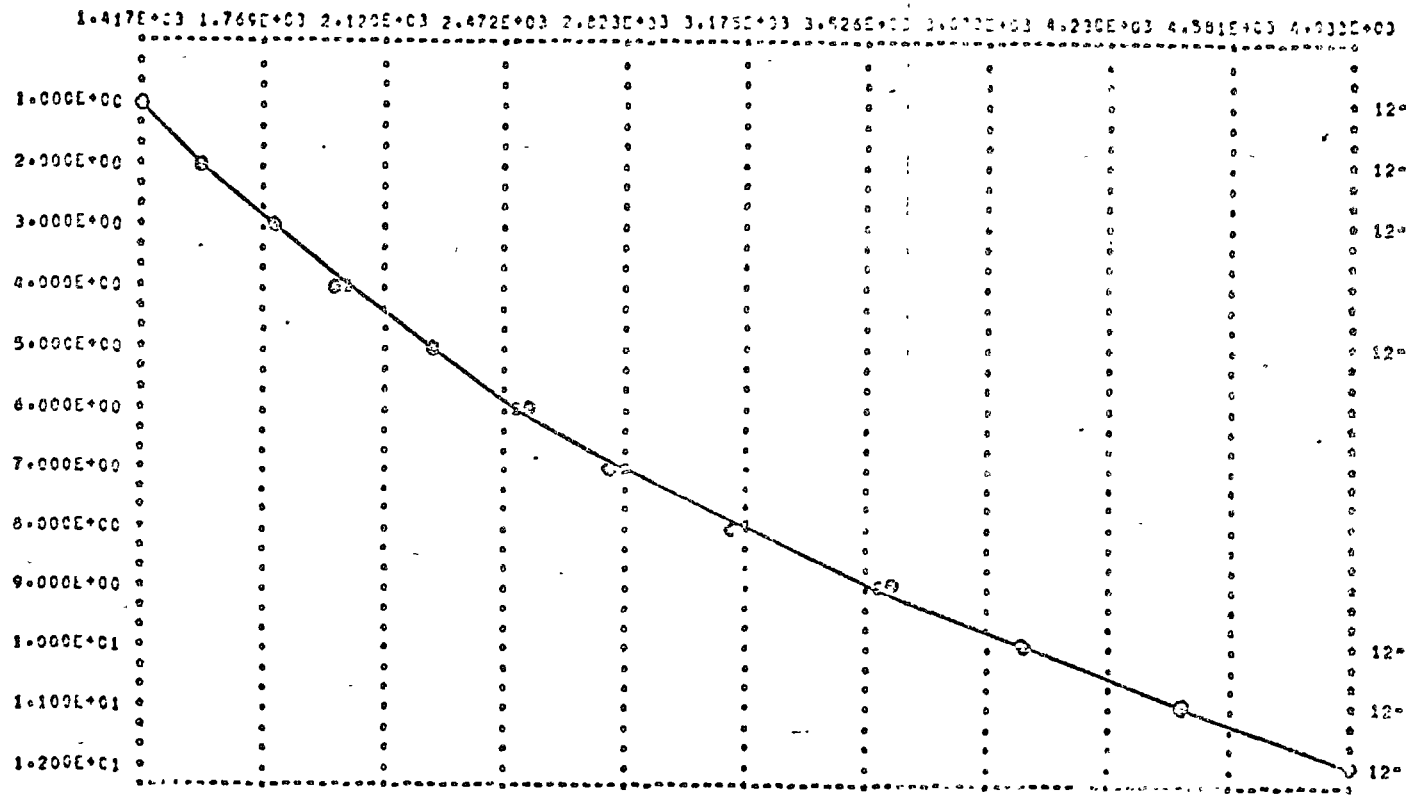
| I | B(I) |
|---|---|
| 1 | -4.11430552E-01 |
| 2 | 1.38567095E-01 |
| 3 | 5.55134606E-07 |
| 4 | -2.80140415E-05 |
| 5 | 2.08932652E-05 |

TIPO DE REGRESION EXPO

LAS DESVIACIONES ESTANDAR DE LOS COEFICIENTES SON

| B(I) | D.S. |
|---|---|
| -4.11430552E-01 | 7.91075704E-01 |
| 1.38567095E-01 | 1.92910641E-02 |
| 5.55134606E-07 | 1.33460183E-06 |
| -2.80140415E-05 | 2.26054052E-05 |
| 2.08932652E-05 | 2.66014915E-05 |

EL COEFICIENTE DE CORRELACION ES 9.99765155E-01


1.417E+03 1.766E+03 2.120E+03 2.472E+03 2.823E+03 3.175E+03 3.526E+03 3.878E+03 4.230E+03 4.581E+03 4.933E+03
1.000E+00
2.000E+00
3.000E+00
4.000E+00
5.000E+00
6.000E+00
7.000E+00
8.000E+00
9.000E+00
1.000E+01
1.100E+01
1.200E+01
12°
12°
12°
12°
12°
12°
12°

## 7.3 Bibliografía


1. CROXTON F., COWDEN D., BOLCH B., "Practical Business Statistics". Englewood Cliffs N.J.: Prentice-Hall Inc., 1969.
   pp.192-263.
2. GEREZ G. Victor, GRIJALVA L. Luis,"El Enfoque de Sistemas". México: Editorial Limusa S.A., 1976.
   pp.150-164.
3. KUO S. Shan, "Computer Applications of Numerical Methods". Reading Mass.: Addison-Wesley Co., 1972.
   pp.252-261.
4. SPIEGEL Murray, "Estadística".México: Libros Mc Graw-Hill de México, S.A., 1961.
   pp.217-282.

## VI) SOLUCION ECUACIONES DIFERENCIALES ORDINARIAS.

Las ecuaciones diferenciales ordinarias son aquellas en las que la variable dependiente es función de una sola variable independiente :

* $Y^{(n)} = (X, Y, Y', \ldots, Y^{(n-1)})$

### a) Método de Euler

Se tratará el caso de ecuaciones diferenciales ordinarias de primer orden :

$$dy = y' \, dx$$

Substituyendo por los incrementos en la expresión anterior se tiene :

$$\Delta y = y' \ \Delta X \qquad (VI.0)$$

Tomando un punto inicial para arrancar y conservando un incremento constante $\Delta x$ se obtiene la siguiente fórmula iterativa :

$$\begin{aligned} Y_1 &= Y_o + Y' \Big|_{(X_o, Y_o)}^{\Delta x} \\ Y_2 &= Y_1 + Y' \Big|_{(X_1, Y_1)}^{\Delta x} \\ &\vdots \\ Y_{n+1} &= Y_n + Y' \Big|_{(X_n, Y_n)}^{\Delta x} \end{aligned} \qquad (VI.1)$$

---

* $Y^{(n)} \equiv \dfrac{d^n Y}{dX^n}$

La ecuación VI.1 nos da la fórmula recursiva de Euler. Para aplicar el método se requiere que $\Delta x$ sea pequeño y además contar con un punto de inicio $(X_o, Y_o)$. El error producido es del orden de $\Delta x^2$.

b) Euler modificado

El procedimiento básico es el mismo solo que para cada $Y_{i+1}$ se hace una serie de iteraciones con los valores obtenidos sucesivamente de $Y_{i+1}$ a - fin de obtener el valor más exacto de $Y_{i+1}$.

Al tener :

$$Y_{i+1} = Y_i + Y'\Big|_{(X_i, Y_i)} \Delta X \qquad (VI.2)$$

se efectúan las siguientes iteraciones :

$$Y'_{i+1} = F(X_{i+1}, Y_{i+1})$$

$$\hat{Y}_{i+1} = Y_i + \frac{(Y'|_i + Y'|_{i+1})}{2} \Delta X$$

$$\hat{Y}'_{i+1} = F(X_{i+1}, \hat{Y}_{i+1})$$

$$\hat{\hat{Y}}_{i+1} = Y_i + \frac{(Y'|_i + \hat{Y}'|_{i+1})}{2} \Delta X$$

$$\vdots$$

y así sucesivamente hasta que :

$$\left| \hat{\hat{Y}}_{i+1} - \hat{Y}_{i+1} \right| < \varepsilon \qquad (VI.3)$$

al cumplirse, se procede a obtener $Y_{i+2}$ y así sucesivamente.

Al igual que en el método anterior es necesario emplear incrementos ( $\Delta x$) pequeños. El error producido es del orden $\Delta x^3$.

c) Método de Runge - Kutta

Este método utiliza las fórmulas de integración antes vista para llegar a la obtención de su propia fórmula recursiva. Dicho proceso es bastante laborioso por lo que no se tratará.

La solución para una ecuación diferencial de primer orden $Y' = f(x,y)$ está dada por :

$$Y_{n+1} = Y_n + \Delta Y_n$$

donde :

$$\Delta Y_n = \frac{\Delta X}{6} ( K_0 + 2K_1 + 2K_2 + K_3)$$

$$K_0 = f ( X_n, Y_n)$$

$$K_1 = f ( X_n + \frac{\Delta X}{2} , Y_n + \frac{K_0 \Delta X}{2} )$$

$$K_2 = f ( X_n + \frac{\Delta X}{2} , Y_n + \frac{K_1 \Delta X}{2} )$$

$$K_3 = f ( X_n + \Delta X, Y_n + K_2)$$

La fórmula anterior es la de Runge-Kutta de 4o. orden, hay otras - fórmulas con mayor cantidad de términos que se obtienen empleando diferencias de mayor orden al deducir la fórmula.

Los parámetros $K_i$ representan la pendiente de la función en los-- puntos en que se está evaluando. El método da un error del orden de $\Delta x^5$ y es uno de los más precisos.

Ejemplo

Obtener la solución de la ecuación diferencial $Y' = 1 - X + 4$ y para- 5 puntos consecutivos empleando los métodos de Euler, Euler mejorado y Rúnge - Kutta usando un incremento $\Delta x = 0.1$. Comparar dichos valores con la solución real si $X_0 = 0$, $Y_0 = 1$.

Sol.

La solución exacta está dada por :

$$Y' - 4Y = 1 - X$$

$$Y_h = Ce^{4X}$$

$$Y_p = A + BX$$

$$\therefore -4A - 4BX = 1 - X$$

$$A = -\frac{1}{4} \quad ; \quad B = \frac{1}{4}$$

$$Y(X) = Ce^{4X} - \frac{1}{4} + \frac{1}{4}X$$

$$Y(0) = 1 = C - \frac{1}{4}$$

$$C = \frac{5}{4}$$

$$Y(x) = \frac{5}{4} e^{4x} - \frac{1}{4} + \frac{1}{4} x$$

Las fórmulas de solución para los métodos son :

$$Y_n = Y_{n-1} + Y']_{n-1} \Delta X \qquad \text{(Euler)}$$

$$\left.\begin{aligned} Y_n &= Y_{n-1} + Y']_{n-1} \Delta X \\ \hat{Y}_n &= Y_{n-1} + \frac{(Y']_{n-1} + Y']_n}{2} \Delta X \end{aligned}\right\} \text{Euler mejorado}$$

$$Y_n = Y_{n-1} + \frac{\Delta X}{6} [K_1 + 2K_2 + 2K_3 + K_4]$$

$$K_1 = f(X_{n-1}, Y_{n-1})$$

$$K_2 = f\left(X_{n-1} + \frac{\Delta X}{2}, Y_{n-1} + \frac{K_1}{2} \Delta X\right) \qquad \text{Runge Kutta}$$

$$K_3 = f\left(X_{n-1} + \frac{\Delta X}{2}, Y_{n-1} + \frac{K_2}{2} \Delta X\right)$$

$$K_4 = f(X_{n-1} + \Delta X, Y_{n-1} + K_3 \Delta X)$$

las soluciones se muestran en la siguiente tabla:

| K | $X_k$ | Euler | E. Maj. | R.Kutta | Real |
|---|---|---|---|---|---|
| 0 | 0 | 1. | 1. | 1. | 1. |
| 1 | 0.1 | 1.5 | 1.595 | 1.608 | 1.609 |
| 2 | 0.2 | 2.19 | 2.463 | 2.505 | 2.505 |
| 3 | 0.3 | 3.146 | 3.737 | 3.829 | 3.830 |
| 4 | 0.4 | 4.774 | 5.609 | 5.792 | 5.794 |
| 5 | 0.5 | 6.324 | 8.369 | 8.709 | 8.712 |

d) Diferencias finitas

Este método se emplea cuando se tienen problemas con valores en la frontera.

El procedimiento consiste en lo siguiente : dividir el intervalo de integración en "n" espacios iguales, emplear las fórmulas de derivación de diferencias finitas en la ecuación diferencial (todas las diferencias deben ser del mismo orden), substituir las condiciones de frontera y por último resolver el sistema de ecuaciones planteado. Se tiene que aplicar el operador diferencial a todos los pivotes del intervalo.

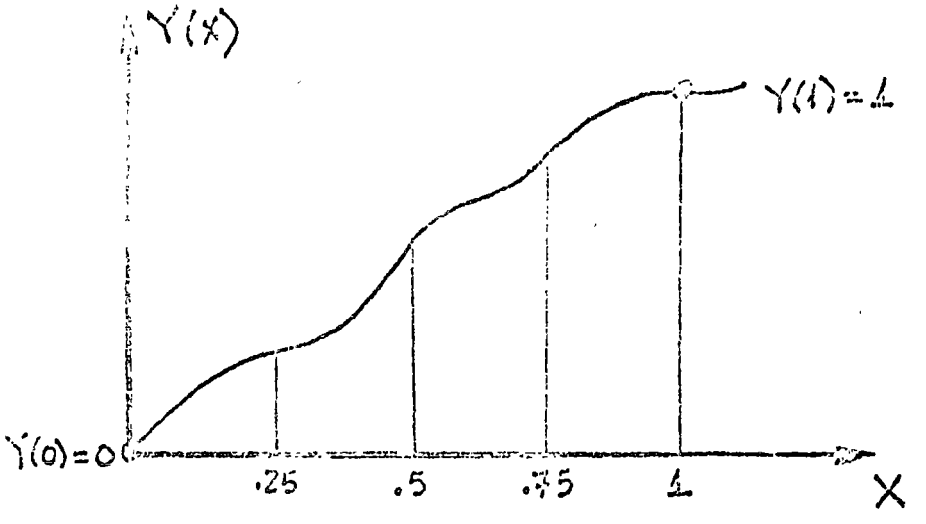
Ejemplo

Resolver la ecuación diferencial $\frac{d^2 y}{d X^2} - y = 0$, en el intervalo (0,1) si $y(0) = 0$, $y(1) = 1$

Sol.

Se divide el intervalo en "n" partes iguales, sean 4 en este caso :

$$\Delta X = \frac{1 - 0}{4} = 0.25$$




Empleando diferencias de 2o. orden :

$$Y_i'' = \frac{1}{(\Delta X)^2} \left[ Y_{i-1} - 2\,Y_i + Y_{i+1} \right]$$

Sustituyendo en la ecuación diferencial :

$$\frac{1}{\Delta X^2} \left[ Y_{i-1} - 2\,Y_i + Y_{i+1} \right] - Y_i = 0$$

$$Y_{i-1} - 2.0625\ Y_i + Y_{i+1} = 0 \qquad \text{(VI.4)}$$

las condiciones de frontera son :

$$Y_0 = 0$$

$$Y_4 = 1$$

aplicando VI.4 en los pivotes :

$$X_1 = 0.25$$

$$Y_o - 2.0625\,Y_1 + Y_2 = 0$$

$$-2.0625\,Y_1 + Y_2 = 0 \qquad \text{(VI.5)}$$

$$X_2 = 0.5$$

$$Y_1 - 2.0625\,Y_2 + Y_3 = 0 \qquad \text{(VI.6)}$$

$$X_3 = 0.75$$

$$Y_2 - 2.0625\,Y_3 + Y_4 = 0$$

$$Y_2 - 2.0625\,Y_3 = -1 \qquad \text{(VI.7}$$

el sistema de ecuaciones es :

$$-2.062\,Y_1 + Y_2 = 0$$

$$Y_1 - 2.062\,Y_2 + Y_3 = 0$$

$$Y_2 - 2.062\,Y_3 = -1$$

de donde :

$$Y_1 = 0.216$$

$$Y_2 = 0.445$$

$$Y_3 = 0.701$$

NOTA: Cuando se trata de ecuaciones diferenciales de mayor orden y se cuenta como condiciones $y'' = 0$, etc., hay que substituir en dichas ecuaciones las fórmulas de diferencias y despejar de ahí las condiciones de frontera desconocidas.

## 10.4 Solución de Sistemas de Ecuaciones Diferenciales Lineales No Homogéneas de Primer Orden

### 10.4.1 Objeto

Obtener la solución de sistemas de ecuaciones diferenciales no homogéneas, lineales, de primer orden mediante el método de Variación de Parámetros.

La representación en forma matricial para este tipo de sistemas de ecuaciones diferenciales es:

$$\dot{\underline{X}}(t) = \underline{A}\,\underline{X}(t) + \underline{B}\,\underline{U}(t) \qquad (10.16)$$
$$\underline{X}(t_0) = \underline{X}_0$$

donde $\underline{U}(t)$ representa el vector de entradas externas si se habla de sistemas físicos.

Debido a que cuando se modelan sistemas dinámicos lineales las salidas no siempre corresponden a las variables empleadas en las ecuaciones diferenciales, en este programa se considera la representación completa mediante variables de estado de un sistema lineal, la cual es:

$$\left.\begin{aligned} \dot{\underline{X}}(t) &= \underline{A}\,\underline{X}(t) + \underline{B}\underline{U}(t) \\ \underline{Y}(t) &= \underline{C}\underline{X}(t) + \underline{D}\underline{U}(t) \\ \underline{X}(t_0) &= \underline{X}_0 \end{aligned}\right\} \qquad (10.17)$$

donde $Y(t)$ representa el vector de salidas del sistema.

### 10.4.2 Método

El método de variación de parámetros establece que la solución del sistema de ecuaciones diferenciales lineales (10.17) tiene por solución:

$$\underline{X}(t) = e^{\underline{A}(t-t_0)}\underline{X}_0 + \int_{t_0}^{t} e^{\underline{A}(t-\sigma)}\underline{B}\,\underline{U}(\sigma)\,d\sigma \qquad (10.18)$$

donde la matriz $e^{\underline{A}(t - t_0)}$ es la matriz de transición definida en la sección 10.3.2.

Por lo tanto la solución total será la suma de la respuesta debida a las condiciones iniciales más la respuesta debida a las excitaciones externas. Para la primera parte de la solución se discutió su obtención en la sección 10.3.2.

Dado que el primer término de la solución se evalúa mediante una evolución de estados a incrementos iguales de tiempo, la segunda parte de la solución:

$$\int_{t_0}^{t} e^{\underline{A}(t - \sigma)} \underline{B}\, \underline{U}(\sigma)\, d\sigma \qquad (10.19)$$

también se evaluará a incrementos iguales de tiempo.

Para poder evaluar la expresión (10.19) mediante la computadora se requiere discretizar el vector de entradas $\underline{U}(t)$, aproximando cada entrada $u(t)$ mediante pulsos o rectas como se muestra a continuación:

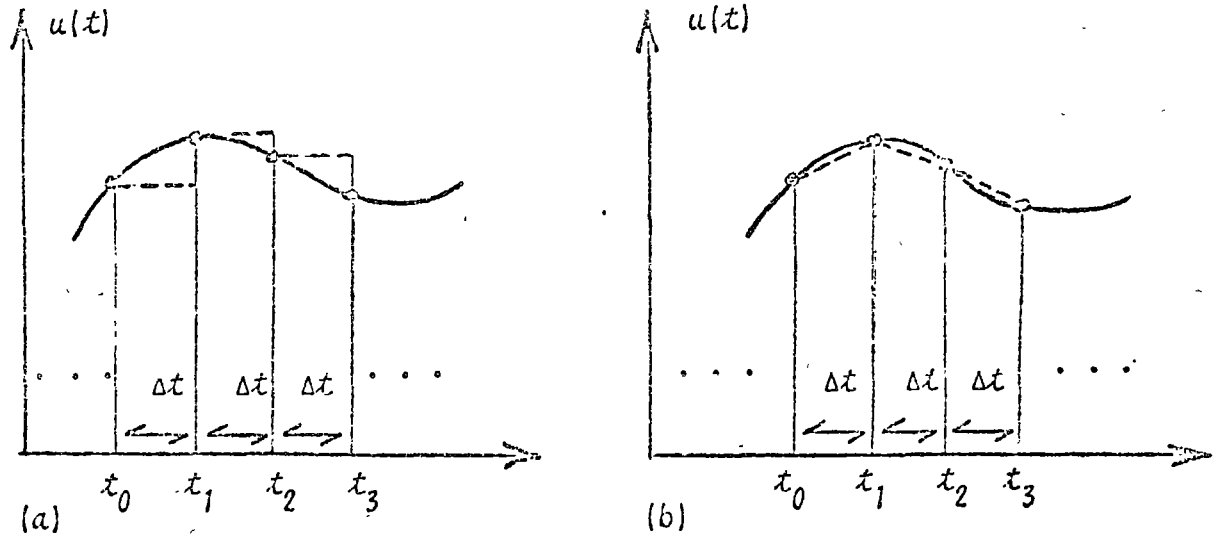



Fig. 10.10 Aproximación de una función mediante:
a) pulsos b) rectas

En el programa se aproxima la función $u(t)$ mediante rectas, las ecuaciones necesarias para la evaluación de (10.19) se desarrollan a continuación.

Sea la función $u(t)$ mostrada en la figura 10.11 :

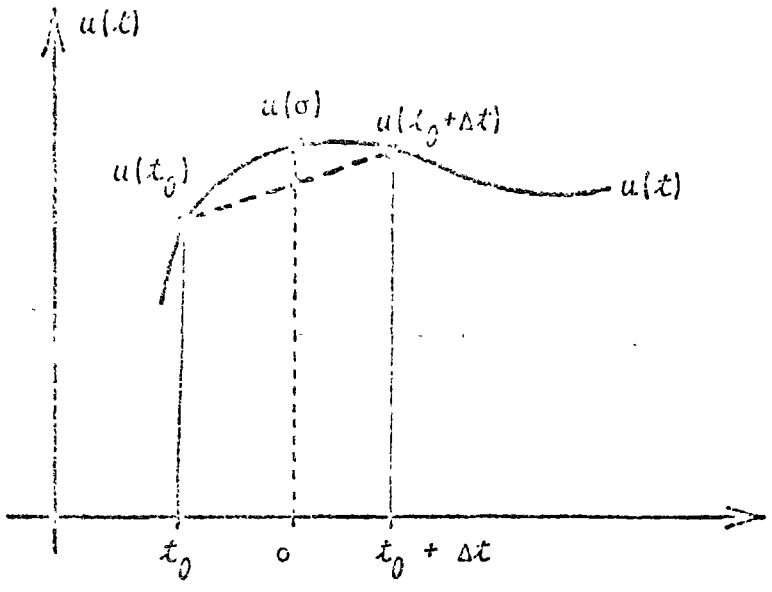



Fig. 10.11 Función $u(t)$ y su aproximación mediante una recta en el intervalo $t_0$ a $t_0 + t$

Se desea evaluar la expresión (10.19) pero:

$$\int_{t_0}^{t} e^{\underline{A}(t-\sigma)}\,\underline{B}\,\underline{U}(\sigma)\,d\sigma = e^{\underline{A}t}\int_{t_0}^{t} e^{-\underline{A}\sigma}\,\underline{B}\,\underline{U}(\sigma)\,d\sigma \qquad (10.20)$$

evaluando de $t_0$ a $t_0 + \Delta t$:

$$\int_{t_0}^{t_0+\Delta t} e^{\underline{A}(t-\sigma)}\,\underline{B}\,\underline{U}(\sigma)\,d\sigma = e^{\underline{A}\,\Delta t}\int_{0}^{\Delta t} e^{-\underline{A}\sigma}\,\underline{B}\,\underline{U}(\sigma)\,d\sigma \qquad (10.21)$$

De la figura 10.11 se observa que:

$$u(\sigma) \cong \frac{u(t_0+\Delta t) - u(t_0)}{\Delta t}\,\sigma + u(t_0) \qquad (10.22)$$

Sustituyendo (10.22) en (10.21):

$$\Delta t\int_{0}^{\Delta t} e^{-\underline{A}\sigma}\,\underline{B}\,\underline{U}(\sigma)\,\sigma = \left\{e^{\underline{A}\,\Delta t}\int_{0}^{\Delta t} e^{-\underline{A}\sigma}\sigma\,d\sigma\right\}\,\underline{B}\,\frac{\underline{U}(t_0+\Delta t) - \underline{U}(t_0)}{\Delta t} + \left\{e^{\underline{A}\,\Delta t}\int_{0}^{\Delta t} e^{-\underline{A}\sigma}\,d\sigma\right\}\,\underline{B}\,\underline{U}(t_0) \qquad (10.23)$$

empleando la siguiente relación:

$$e^{-\underline{A}\sigma} = \underline{I} - \frac{A\sigma}{1!} + \frac{A^2\sigma^2}{2!} - \frac{A^3\sigma^3}{3!} + \ldots \qquad (10.24)$$

en los términos entre corchetes se llega a:

$$e^{\underline{A}\,\Delta t}\int_0^{\Delta t} e^{-\underline{A}\sigma}\,\sigma d\sigma = \frac{\underline{I}(\Delta t)^2}{2!} + \ldots + \frac{A^n(\Delta t)^{n+2}}{(n+2)!} + \ldots \qquad (10.25)$$

$$e^{\underline{A}\,\Delta t}\int_0^{\Delta t} e^{-\underline{A}\sigma}\,d\sigma = \underline{I}(\Delta t) + \ldots + \frac{A^n(\Delta t)^{(n+1)}}{(n+1)!} + \ldots (10.26)$$

Para la evaluación de las series (10.25) y (10.26) la cantidad de términos a emplear dependerá de la exactitud deseada. Se fija un criterio de convergencia $\mathcal{E}$ tal que si $\underline{Z}$ representa a la matriz de la serie (10.25) y $\underline{W}$ a la matriz de la serie (10.26), se cumpla que:

$$\left| z_{ij}^{(n+1)} - z_{ij}^{(n)} \right| < \mathcal{E} \text{ , para toda } ij$$
$$\left| w_{ij}^{(n+1)} - w_{ij}^{(n)} \right| < \mathcal{E} \text{ , para toda } ij \qquad * \qquad (10.27)$$

Como las series (10.25) y (10.26) solo dependen del espaciamiento, se tendrán que evaluar una sola vez.

En el programa para obtener la relación (10.24) hay que evaluar el vector de entradas $\underline{U}(t)$ en cada uno de los puntos en que se subdivide el intervalo de integración.

En términos generales el proceso a seguir es:

1. evaluar la matriz de transición $e^{\underline{A}(\Delta t)}$
2. evaluar las series de las ecuaciones (10.25) y (10.26)
3. obtener la respuesta debida a las condiciones iniciales para $t_i$

---

* $z_{ij}^{(n)}$ representa el elemento $z_{ij}$ de la matriz $\underline{Z}$ compuesta por la sumatoria de "n" términos.

(4) evaluar $\underline{U}(t)$ en $t_i$ y $t_{i+1}$

(5) obtener la respuesta debida a las excitaciones externas mediante la relación (10.23)

(6) hacer $i=i+1$ y regresar al paso (3) hasta barrer todo el intervalo de integración.

10.4.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE EXPMA(DELT,M,A,EXPO), obtiene la matriz de transición. Consultar sección 10.3.5.

SUBROUTINE INTPE(DELT,M,A,SUMA), obtiene la matriz de la serie (10.26); esta expresión se emplea para evaluar la respuesta debida a las excitaciones externas

SUBROUTINE INTRE(DELT,M,A,RECTA), evalua la expresión dada por la serie de la ecuación (10.25); esta expresión se utiliza para calcular la respuesta debida a las excitaciones externas.

SUBROUTINE MULTMA(A,B,N,M,L,X), obtiene el producto matricial AB. Consultar el capítulo 2.

SUBROUTINE GRAFI(A,N,M), grafica las soluciones de las variables dependientes y de las respuestas del sistema. Consultar el capítulo 1.

SUBROUTINE EXCITA(T,F), evalua el vector de entradas $\underline{U}(t)$ en el instante $t_i$.

b) Descripción de las variables:

Para la subrutina INTPE:

| | |
|---|---|
| DELT | espaciamiento entre los valores de la variable independiente |
| M | cantidad de ecuaciones diferenciales |
| A(I,J) | matriz de coeficientes constantes del sistema de ecuaciones diferenciales |
| EPS | criterio de convergencia |
| SUMA(I,J) | matriz resultante de evaluar la serie |
| CN | contador de iteraciones |
| DIV | factorial divisor |
| TNEW | incremento de la variable independiente elevado a la potencia "n" |
| CMA(I,J) | matriz identidad |

B(I,J) matriz $\underline{A}$ elevada a la potencia "n"
X(I,J) matriz resultante del producto AB

Para la subrutina INTRE:

DELT espaciamiento entre los valores de la variable independiente
M cantidad de ecuaciones diferenciales
A(I,J) matriz de coeficientes del sistema de ecuaciones diferenciales
RECTA(I,J) matriz resultante de evaluar la serie
CMA(I,J) matriz identidad
B(I,J) matriz $\underline{A}$ elevada a la potencia "n"
X(I,J) matriz resultante del producto AB
EPS criterio de convergencia
TNEW incremento de la variable independiente elevado a la potencia "n"
CN contador
DIV factorial divisor

Para la subrutina EXCITA:

T valor del instante de tiempo en el cual se desea evaluar la expresión $\underline{U}(t)$
F(1,1) valor del primer renglón de la expresión $\underline{U}(t)$ en el instante $t_i$
F(2,1) valor del segundo renglón de la expresión $\underline{U}(t)$ en el instante $t_i$
F(3,1) valor del tercer renglón de la expresión $\underline{U}(t)$ en el instante $t_i$
F(4,1) valor del cuarto renglón de la expresión $\underline{U}(t)$ en el instante $t_i$
F(5,1) valor del quinto renglón de la expresión $\underline{U}(t)$ en el instante $t_i$

Para el programa principal:

M cantidad de ecuaciones diferenciales
N cantidad de subintervalos en que se divide el intervalo de integración
NS cantidad de salidas del sistema
NU cantidad de entradas del sistema
PERIO intervalo de integración

| | |
|---|---|
| DELT | magnitud de los subintervalos de integración |
| A(I,J) | matriz A del sistema de ecuaciones |
| B(I,J) | matriz B del sistema de ecuaciones |
| C(I,J) | matriz C del sistema de ecuaciones |
| D(I,J) | matriz D del sistema de ecuaciones |
| X(1,1) | condición inicial de la variable independiente |
| X(I,J) | condiciones iniciales para cada una de las variables dependientes, J > 1 |
| X(I,J) | solución del sistema de ecuaciones |
| YY(I,J) | valor de las salidas del sistema |
| SUMA(I,J) | integral del término constante de la ecuación de la recta empleada para aproximar la entrada |
| RECTA(I,J) | integral del término variable de la ecuación de la recta empleada para aproximar la entrada |
| SHOM(I,J) | solución del sistema debida a las excitaciones externas |
| Y(I,J) | valor de la entrada en el instante $t_{i-1}$ |
| PEND(I,J) | pendiente de la recta empleada para aproximar la entrada |

[illegible] de reemplazo

| SEC. TARJETAS | FORMATO | INFORMACION |
|---|---|---|
| 1 | (4I5,F10.0) | M, N, NS, NU, PERIO |
| 2 | (8F10.0) | A(I,J), los elementos de la matriz se dan renglón por renglón. Emplean tan- |

| | | |
|---|---|---|
| . | | tas tarjetas como sean ne |
| . | | cesarias |
| . | | |
| 3 | (8F10.0) | B(I,J), igual que para la |
| . | | matriz A |
| . | | |
| . | | |
| 4 | (8F10.0) | C(I,J), igual que para la |
| . | | matriz A |
| . | | |
| . | | |
| 5 | (8F10.0) | D(I,J), igual que para la |
| . | | matriz A |
| . | | |
| . | | |
| 6 | (8F10.0) | X(I,J), el primer valor de |
| | | be corresponder a la condi |
| | | ción inicial de la varia-- |
| | | ble independiente. |

---

otros paquetes de datos (opcional)

---

| | |
|---|---|
| n | TARJETA EN BLANCO, al fina |
| | lizar toda la información. |

:) Diagrama de bloques:

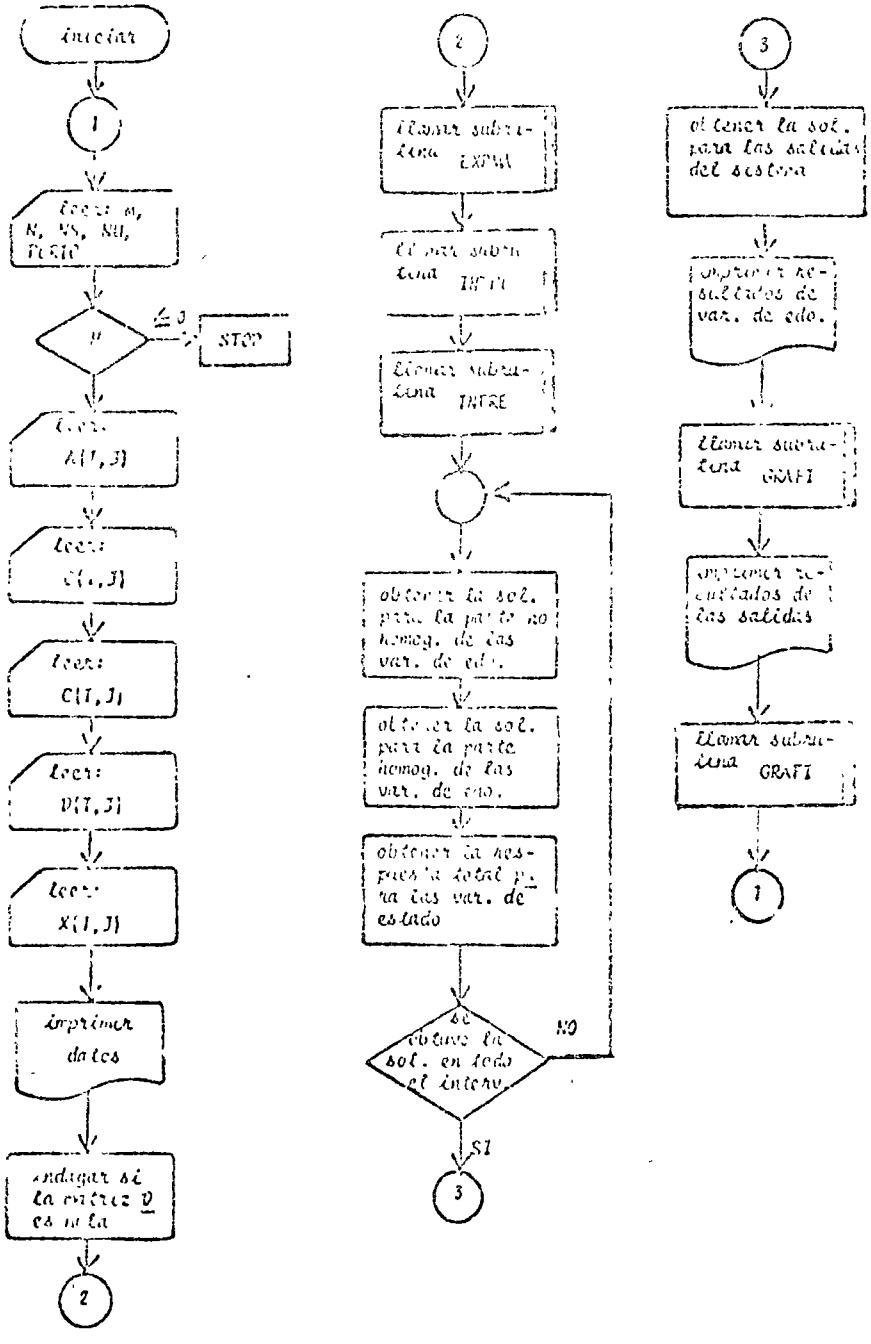



Fig. 10.12 Diagrama de bloques de programa principal

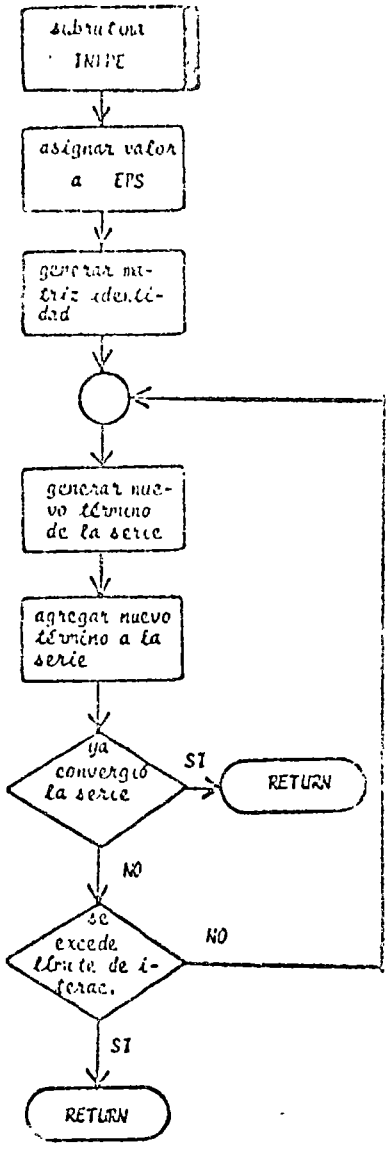



Fig. 10.13 Diagrama de bloques de la subrutina INTPE

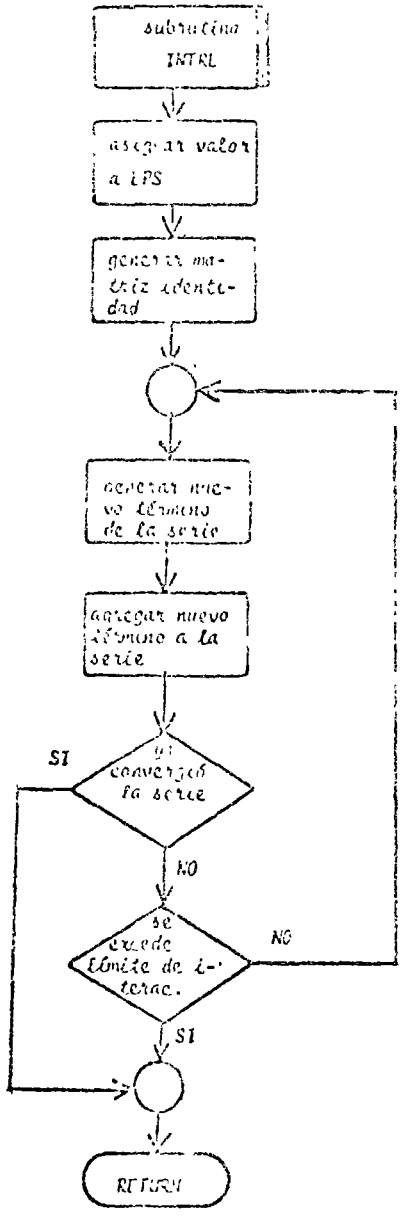



Fig. 10.14 Diagrama de bloques de la subrutina INTRE

6) Listado:

```
C      PROGRAMA PARA SIMULAR SISTEMAS DINAMICOS LINEALES EN UNA COMPUTA-
C     -DORA DIGITAL MEDIANTE EL METODO DE VARIACION DE PARAMETROS
C      SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     -N=CANTIDAD DE ECUACIONES DIFERENCIALES
C     -M=CANTIDAD DE PARTES EN QUE SE SUBDIVIDE EL INTERVALO DE INTEGRA-
C        CION
C      NS=CANTIDAD DE SALIDAS
C     -NU=CANTIDAD DE ENTRADAS
C      PERIO=MAGNITUD DEL INTERVALO DE INTEGRACION
C      DELT=MAGNITUD DE LOS SUBINTERVALOS
C     -A=MATRIZ A DEL SISTEMA DE ECUACIONES
C      B=MATRIZ B DEL SISTEMA DE ECUACIONES
C      C=MATRIZ C DEL SISTEMA DE ECUACIONES
C     -D=MATRIZ D DEL SISTEMA DE ECUACIONES.
C      X(1,1)=CONDICION INICIAL DE LA VARIABLE INDEPENDIENTE
C      X(1,J)=CONDICIONES INICIALES
C     -X=SOLUCION PARA LAS VARIABLES DE ESTADO
C      YY=VALOR DE LAS SALIDAS
C      SUM=RESPUESTA DEBIDA A LAS EXCITACIONES EXTERNAS
C     -EXPO=MATRIZ DE TRANSICION
C      SUMA=INTEGRAL DE LA PARTE CONSTANTE DE LA ECUACION DE LA RECTA
C      PECTA=INTEGRAL DEL TERMINO VARIABLE DE LA ECUACION DE LA RECTA
C     -Y=VALOR DE LA ENTRADA EN EL INSTANTE T(I-1)
C      F=VALOR DE LA ENTRADA EN EL INSTANTE T(I)
C      PEND=PENDIENTE DE LA RECTA
C     -MP1=CANTIDAD DE COLUMNAS DEL ARREGLO MATRICIAL X
C     -RE=MATRIZ DE REEMPLAZO

      DIMENSION A(6,6),B(6,6),C(6,6),D(6,6),X(101,6),EXPO(6,6),PECTA(6,6
     1),SUMA(6,6),SUM(6,6),Y(6,6),PEND(6,6),PE(6,6),F(6,6),YY(101,6)
      IR=5
      IW=6
C     LECTURA DE DATOS
    1 READ(IR,100) M,N,NS,NU,PERIO
      IF(M) 2,2,3
    2 CALL EXIT
    3 MP1=M+1
      DO 4 I=1,N
    4 READ(IR,150) (A(I,J),J=1,N)
      DO 5 I=1,N
    5 READ(IR,150) (B(I,J),J=1,NU)
      IF(NS.EQ.0) GO TO 71
      DO 6 I=1,NS
    6 READ(IR,150) (C(I,J),J=1,N)
      DO 7 I=1,NS
    7 READ(IR,150) (D(I,J),J=1,NU)
   71 READ(IR,150) (X(1,J),J=1,MP1)
C     IMPRESION DE DATOS
      WRITE(IW,200)
      DO 8 I=1,N
    8 WRITE(IW,250) (A(I,J),J=1,N)
      WRITE(IW,300)
      DO 9 I=1,N
    9 WRITE(IW,250) (B(I,J),J=1,NU)
      IF(NS.EQ.0) GO TO 72
      WRITE(IW,350)
      DO 10 I=1,NS
   10 WRITE(IW,250) (C(I,J),J=1,N)
      WRITE(IW,400)
      DO 11 I=1,NS
   11 WRITE(IW,250) (D(I,J),J=1,NU)
   72 WRITE(IW,450)
      WRITE(IW,250) (X(1,J),J=1,MP1)
      IF(NS.EQ.0) GO TO 73
C     IMPRESION SI LA MATRIZ D ES NULA
      DO 12 I=1,NS
      DO 12 J=1,NU
      IF(D(I,J).EQ.0.0) GO TO 12
      ID=1
      GO TO 13
   12 CONTINUE
   73 ID=0
C     OBTENCION LA MATRIZ DE TRANSICION
   13 DELT=PERIO/FLOAT(M)
      CALL EXPMA(DELT,N,A,EXPO)
C     OBTENCION SUMA Y PECTA
      CALL INTSU(DELT,N,A,SUMA)
      CALL INTPE(DELT,N,A,PECTA)
```

```
      OBTENER LA SOLUCION TOTAL DE LAS VARIABLES DE ESTADO
      NP1=N+1
      DO 18 K=2,NP1
      X(K,1)=X(K-1,1) + DELT
C     OBTENER LA SOLUCION DEBIDA A LAS ENTRADAS
      T=X(K-1,1)
      CALL EXCITA(T,F)
      CALL MULTMA(A,F,N,NU,1,Y)
      T=X(K,1)
      CALL EXCITA(T,F)
      CALL MULTMA(B,F,N,NU,1,RE)
      DO 14 I=1,N
   14 PE(I,1)=(RE(I,1)-Y(I,1))/DELT
      CALL MULTMA(SOMA,Y,N,N,1,SHO)
      CALL MULTMA(RECTA,DELTA,N,N,1,RE)
      DO 15 I=1,N
   15 SOMA(I,1)=SHO(I,1) + PE(I,1)
C     OBTENER SOLUCION DEBIDA A LAS CONDICIONES INICIALES
      DO 17 J=1,N
      X(1,J+1)=0.0
      DO 16 I=2,NP1
   16 X(K,J+1)=X(K,J+1) + EXPO(J,I-1)*X(K-1,I)
C     OBTENER LA SOLUCION TOTAL
   17 X(K,J+1)=X(K,J+1) + SOMA(J,1)
   18 CONTINUE
      IF(NS.EQ.0) GO TO 30
C     OBTENER EL VALOR DE LAS SALIDAS PARA TODOS LOS PUNTOS MUESTRALES
      IF(IC.EQ.0) GO TO 29
      DO 21 K=1,NP1
      DO 25 I=1,NS
   25 YY(K,I)=0.
      Y=X(K,1)
      CALL EXCITA(T,F)
      CALL MULTMA(D,F,NS,NU,1,A)
      DO 31 I=1,NS
   31 YY(K,I)=YY(K,I) + A(I,1)

      DO 1° I=1,N
   1° PE(I,1)=X(K,I+1)
      CALL MULTMA(C,RE,NS,N,1,A)
      DO 20 I=1,NS
   20 YY(K,I)=YY(K,I) + A(I,1)
   21 CONTINUE
      GO TO 30
   26 DO 29 K=1,NP1
      DO 27 I=1,N
   27 PE(I,1)=X(K,I+1)
      CALL MULTMA(C,RE,NS,N,1,A)
      DO 28 I=1,NS
   28 YY(K,1)=YY(K,I) + A(I,1)
   29 CONTINUE
C     IMPRIMIR RESULTADOS CORRESPONDIENTES A LAS VARIABLES DE ESTADO
   30 WRITE(IW,500)
      DO 22 I=1,NP1
   22 WRITE(IW,250) (X(I,J),J=1,NP1)
      CALL GRAFI(X,NP1,NP1)
      IF(NS.EQ.0) GO TO 1
C     IMPRIMIR RESULTADOS CORRESPONDIENTES A LAS SALIDAS DEL SISTEMA
      WRITE(IW,550)
      DO 23 I=1,NP1
   23 WRITE(IW,250) X(I,1),(YY(I,J),J=1,NS)
      DO 24 I=1,NP1
      DO 24 J=1,NS
   24 X(I,J+1)=YY(I,J)
      NS=NS + 1
      CALL GRAFI(X,NP1,NS)
      GO TO 1
C     FORMATOS DE LECTURA E IMPRESION
  100 FORMAT(4I5,F10.0)
  150 FORMAT(8F10.0)
  200 FORMAT(1H1,5(/),15X,'MATRIZ A',//)
  250 FORMAT(/,15X,8(1PE12.5,3X))
  300 FORMAT(5(/),15X,'MATRIZ B',/)
  350 FORMAT(5(/),15X,'MATRIZ C',/)
  400 FORMAT(5(/),15X,'MATRIZ D',/)
  450 FORMAT(5(/),15X,'LAS CONDICIONES INICIALES SON',/,18X,'TIEMPO',1X
     1,'X(1)',11X,'X(2)',11X,'X(3)',11X,'X(4)',11X,'X(5)',/)
  500 FORMAT(5(/),15X,'LA SOLUCION PARA LAS VARIABLES DE ESTADO ES',/,8
     1X,'TIEMPO',10X,'X(1)',11X,'X(2)',11X,'X(3)',11X,'X(4)',11X,'X(5)',
     2//)
  550 FORMAT(5(/),15X,'EL VALOR DE LAS SALIDAS ES',/,18X,'TIEMPO',10X,'Y
     1(1)',11X,'Y(2)',11X,'Y(3)',11X,'Y(4)',11X,'Y(5)',/)
      END
```

*Fig. 10.15 Listado del programa principal*

```
      SUBROUTINE INTPE(DELT,N,A,SUMA)
C     SUBRUTINA PARA EVALUAR LA INTEGRAL DEL TERMINO CONSTANTE DE LA
C     SERIE
C     SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C     DELT=ESPACIAMIENTO ENTRE LOS VALORES DE LA VARIABLE INDEPENDIENTE
C     N=CANTIDAD DE ECUACIONES DIFERENCIALES
C     A=MATRIZ A DEL SISTEMA DE ECUACIONES
C     SUMA=MATRIZ RESULTANTE DE EVALUAR LA INTEGRAL
C     EPS=CRITERIO DE CONVERGENCIA
C     CMA=MATRIZ IDENTIDAD
C     CN=CONTADOR DE ITERACIONES
C     DIV=FACTORIAL DIVISOR
C     TMEN=INCREMENTO DE TIEMPO ELEVADO A LA POTENCIA N
C     B=MATRIZ A ELEVADA A LA POTENCIA N
C     X=MATRIZ RESULTANTE DEL PRODUCTO AB
      DIMENSION A(6,6),CMA(6,6),B(6,6),X(6,6),SUMA(6,6)
      EPS=0.00001
C     CALCULAR MATRIZ IDENTIDAD
      DO 3 I=1,N
      DO 2 J=1,N
      IF(I.EQ.J) GO TO 1
      CMA(I,J)=0.0
      GO TO 2
    1 CMA(I,J)=1.0
    2 CONTINUE
    3 CONTINUE
C     OBTENER LOS DOS PRIMEROS TERMINOS DE LA SERIE
      DO 4 I=1,N
      DO 4 J=1,N
      SUMA(I,J)=CMA(I,J)*DELT + (A(I,J)*DELT*DELT)/2.0
    4 B(I,J)=A(I,J)
      TMEN=DELT*DELT
      CN=3.0
      DIV=2.0
C     OBTENER LOS TERMINOS RESTANTES DE LA SERIE .
    5 DIV=DIV*CN
      TMEN=TMEN*DELT
      CALL MULTMA(A,B,N,N,N,X)
      DO 6 I=1,N
      DO 6 J=1,N
      B(I,J)=X(I,J)
    6 CMA(I,J)=(X(I,J)*TMEN)/DIV
      IF(CN-6.0) 9,9,13
   13 AMAX=CMA(1,1)
      DO 8 I=1,N
      DO 7 J=1,N
      IF(ABS(CMA(I,J))-ABS(AMAX)) 7,7,14
   14 AMAX=CMA(I,J)
    7 CONTINUE
    8 CONTINUE
C     VERIFICAR LA CONVERGENCIA DE LA SERIE
      IF(ABS(AMAX)-EPS) 11,11,9
    9 DO 10 I=1,N
      DO 10 J=1,N
   10 SUMA(I,J)=SUMA(I,J) + CMA(I,J)
      IF(CN-20.0) 15,15,11
   15 CN=CN + 1.0
      GO TO 5
   11 RETURN
      END
```

*Fig. 10.16 Listado de la subrutina INTPE*

```
      SUBROUTINE INTRE(DELT,N,A,RECTA)
C     SUBRUTINA PARA EVALUAR LA INTEGRAL DE LA PARTE VARIABLE DE LA EX
C     PONENCIAL DE LA SERIE
C     DELT  INCREMENTO ENTRE LOS VALORES DE LA VARIABLE INDEPENDIENTE
C     N  GRADO DE POLINOMIOS DIFERENCIALES
C     A  MATRIZ DEL SISTEMA DE ECUACIONES
C     RECTA  MATRIZ RESULTANTE DE EVALUAR LA SERIE
C     CHA  MATRIZ IDENTIDAD
C     B  MATRIZ ELEVADA A LA POTENCIA N
C     X  MATRIZ RESULTANTE DEL PRODUCTO AB
C     EPS  CRITERIO DE CONVERGENCIA
C     T  INCREMENTO DE LA VARIABLE INDEPENDIENTE ELEVADO A LA POTENCIA
C        N
C     C  CONTADOR DE ITERACIONES
      DIMENSION A(6,6),RECTA(6,6),CHA(6,6),B(6,6),X(6,6)
      EPS=0.00001
C     CALCULO MATRIZ IDENTIDAD
      DO 3 I=1,N
      DO 2 J=1,N
      IF(I.EQ.J) GO TO 1
      CHA(I,J)=0.0
      GO TO 2
    1 CHA(I,J)=1.0
    2 CONTINUE
    3 CONTINUE
C     CALCULO LOS DOS PRIMEROS TERMINOS DE LA SERIE
      DO 4 I=1,N
      DO 4 J=1,N
      RECTA(I,J)=(CHA(I,J)*DELT*DELT)/2.0 + (A(I,J)*(DELT**3))/6.0
    4 B(I,J)=A(I,J)
      T=DELT*DELT*DELT
      C=4.0
      DIV=6.0
C     CALCULO EL RESTO DE LOS TERMINOS DE LA SERIE
    5 DIV=DIV*C
      T=T*DELT
      CALL MULTMA(A,B,N,N,N,X)
      DO 6 I=1,N
      DO 6 J=1,N
      B(I,J)=X(I,J)
    6 CHA(I,J)=(X(I,J)*T)/DIV
      IF(CHA.LE.0.0) GO TO 9
      AMAX=CHA(1,1)
      DO 8 I=1,N
      DO 7 J=1,N
      IF(ABS(CHA(I,J)).LE.ABS(AMAX)) GO TO 7
      AMAX=CHA(I,J)
    7 CONTINUE
    8 CONTINUE
C     VERIFICAR CONVERGENCIA DE LA SERIE
      IF(ABS(AMAX).LE.EPS) GO TO 11
    9 DO 10 I=1,N
      DO 10 J=1,N
   10 RECTA(I,J)=RECTA(I,J) + CHA(I,J)
      IF(C.GT.20.) GO TO 11
      C=C + 1.0
      GO TO 5
   11 RETURN
      END
```

*Fig. 10.17 Listado de la subrutina INTRE*

```
      SUBROUTINE EXCITA(T,F)
      DIMENSION F(6,6)
      F(1,1)=5.*EXP(-4.*T)
      F(2,1)=0.5*SIN(3.*T)
      F(3,1)=0.0
      F(4,1)=0.0
      F(5,1)=0.0
      RETURN
      END
```

*Fig. 10.18 Listado de la subrutina EXCIT*

10.4.4 Ejemplo

Para el siguiente circuito eléctrico:

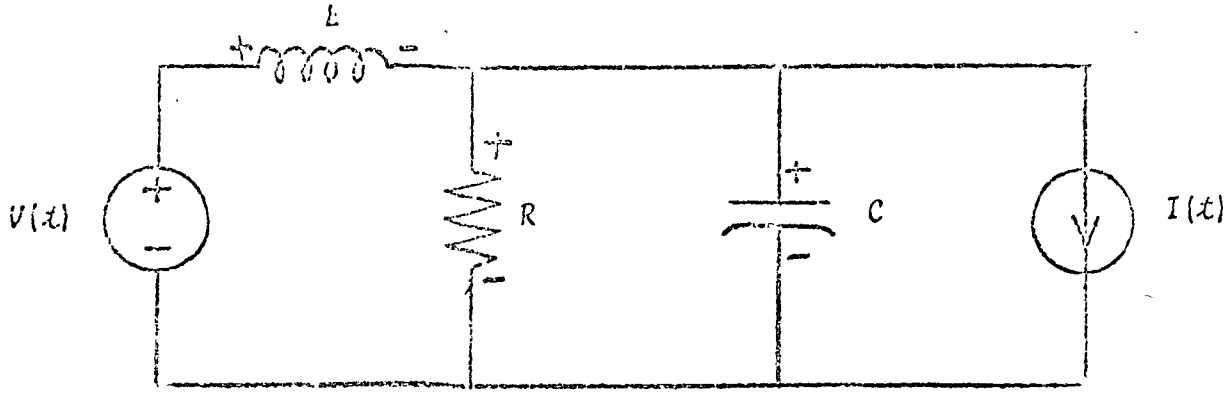



Fig. 10.19 Circuito eléctrico del problema del ejemplo 10.4.4

si se consideran como salidas $I_c$ y $V_R$, su representación mediante variables de estado es:

$$\begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_c}{dt} \end{bmatrix} = \begin{bmatrix} 0 & -1/L \\ 1/C & -1/RC \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} 1/L & 0 \\ 0 & -1/C \end{bmatrix} \begin{bmatrix} V(t) \\ I(t) \end{bmatrix}$$

$$\begin{bmatrix} I_c \\ V_R \end{bmatrix} = \begin{bmatrix} 1 & -1/R \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V(t) \\ I(t) \end{bmatrix}$$

Determine los valores de $I_L$, $V_c$, $I_c$ y $V_R$ para $t \geq 0$, cuando:

$V(t) = 5e^{-4t}u_{-1}(t)$ (V)

$I(t) = 0.5\,sen(3t)u_{-1}(t)$ (A)

$R = 100$ ohms

$C = 0.1$ F

$L = 1.0$ H

$t_0 = 0.$

$V_c(t_0) = 2$ V

$I_L(t_0) = 0.3$ A

$t_f = 10$ s

* SOLUCION

TABLA IV.5 Datos para el problema del ejemplo 10.4.4

N= 2

N= 100

NS= 2

NU= 2

PERIO= 10

$$A = \begin{bmatrix} 0 & -1 \\ 10 & -0.1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ 0 & -10 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & -0.01 \\ 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}$$

X(1,J) =( 0 , 0.3 , 2 )

---

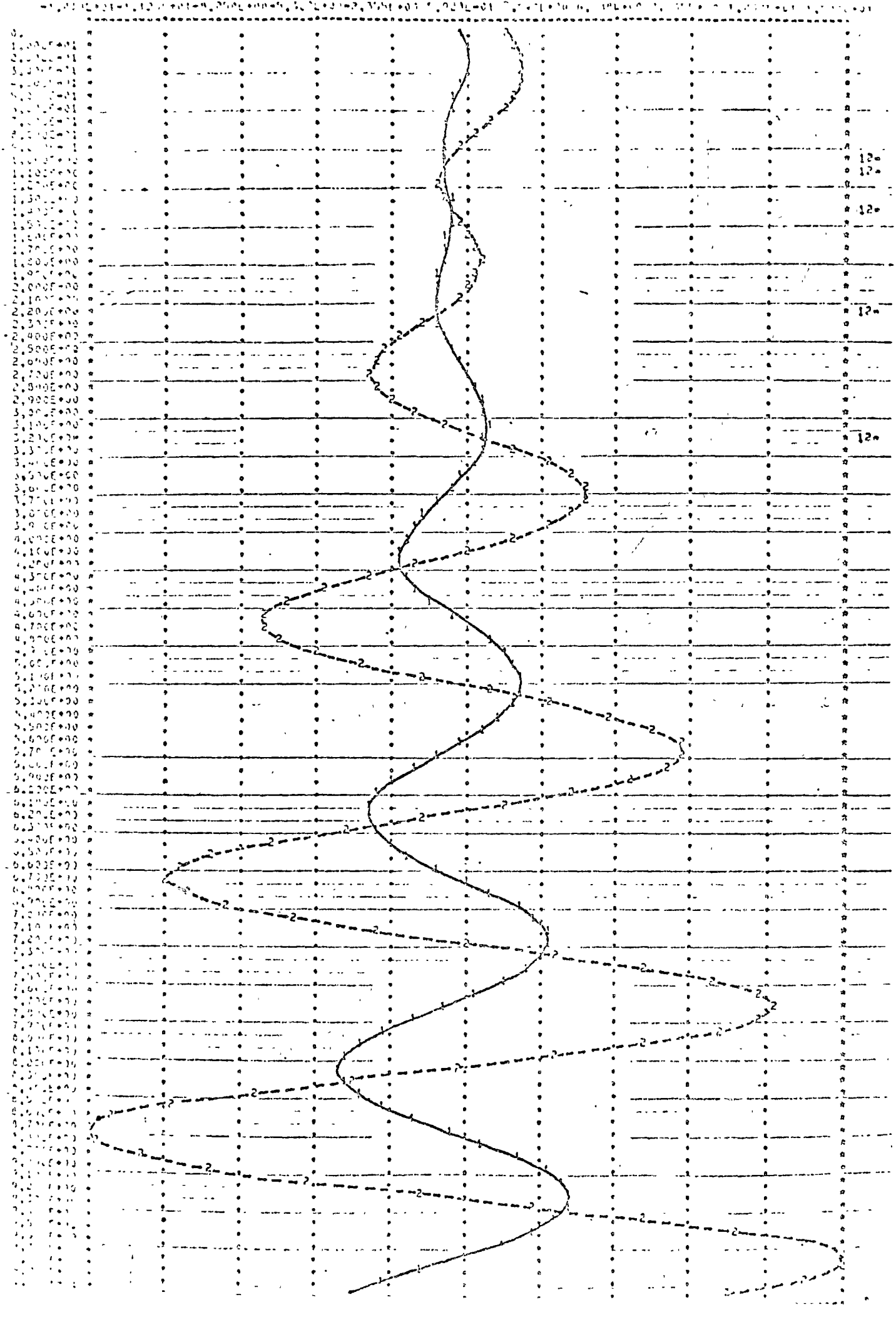
F(1,1) = 5.*EXP(-4.*T)

```
F(2,1)= 0.5*SIN(3.0*T,
F(3,1)= 0.
F(4,1)= 0.
F(5,1)= 0.
```

TABLA 10.6 Resultados del problema del ejemplo 10.4.4

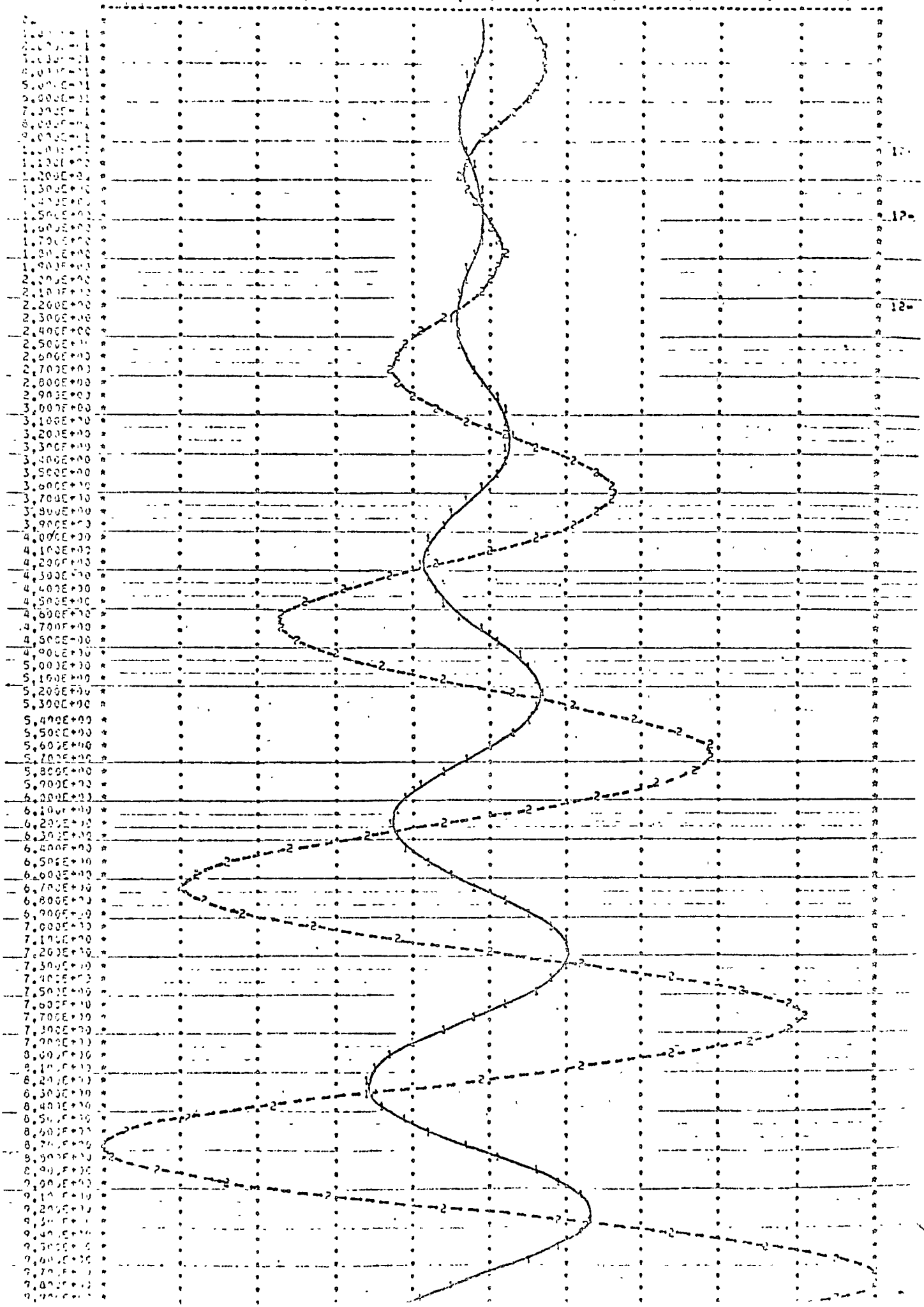
LA SOLUCION PARA LAS VARIABLES DE ESTADO ES

| TIEMPO | X(1) | X(2) |
|---|---|---|
| 0. | .30000E+00 | .20070E+01 |
| .10000E+00 | .50191E+00 | .23226E+01 |
| .20000E+00 | .53425E+00 | .26119E+01 |
| .30000E+00 | .49525E+00 | .27905E+01 |
| .40000E+00 | .33563E+00 | .28740E+01 |
| .50000E+00 | .13186E+00 | .28367E+01 |
| .60000E+00 | -.25677E-01 | .19240E+01 |
| .70000E+00 | -.15241E+00 | .13566E+01 |
| .80000E+00 | -.23291E+00 | .78453E+00 |
| .90000E+00 | -.29400E+00 | .23115E+00 |
| .10000E+01 | -.29499E+00 | -.17340E+00 |
| .11000E+01 | -.21637E+00 | -.59983E+00 |
| .12000E+01 | -.10778E+00 | -.84242E+00 |
| .13000E+01 | -.12577E+00 | -.90147E+00 |
| .14000E+01 | -.10031E+00 | -.23559E-01 |
| .15000E+01 | -.11965E+00 | .32723E+00 |
| .16000E+01 | -.10905E+00 | .67420E+00 |
| .17000E+01 | -.25037E+00 | .93223E+00 |
| .18000E+01 | -.35072E+00 | .10544E+01 |
| .19000E+01 | -.45351E+00 | .97213E+00 |
| .20000E+01 | -.53738E+00 | .67331E+00 |
| .21000E+01 | -.58102E+00 | .17120E+00 |
| .22000E+01 | -.56612E+00 | -.48249E+00 |
| .23000E+01 | -.46037E+00 | -.12319E+01 |
| .24000E+01 | -.31998E+00 | -.19651E+01 |
| .25000E+01 | -.10139E+01 | -.25359E+01 |
| .26000E+01 | .19019E+00 | -.29957E+01 |
| .27000E+01 | .99838E+00 | -.31137E+01 |
| .28000E+01 | .13221E+01 | -.13828E+02 |
| .29000E+01 | .20339E+01 | -.12194E+02 |
| .30000E+01 | .37213E+01 | -.93734E+01 |
| .31000E+01 | .44771E+01 | -.59142E+01 |
| .32000E+01 | .48219E+01 | -.12543E+01 |
| .33000E+01 | .47215E+01 | .32984E+01 |
| .34000E+01 | .41721E+01 | .76154E+01 |
| .35000E+01 | .32209E+01 | .11295E+02 |
| .36000E+01 | .19038E+01 | .13946E+02 |
| .37000E+01 | .73498E+00 | .15335E+02 |
| .38000E+01 | -.46496E+00 | .15297E+02 |
| .39000E+01 | -.17533E+01 | .13773E+02 |
| .40000E+01 | -.27419E+01 | .10994E+02 |



EL VALOR DE LAS SALIDAS ES

| TIEMPO | Y(1) | Y(2) |
|---|---|---|
| .00000E+00 | .00000E+00 | .00000E+00 |
| .10000E+00 | .31993E+00 | .23273E+01 |
| .20000E+00 | .52431E+00 | .26117E+01 |
| .30000E+00 | .31185E-01 | .27485E+01 |
| .40000E+00 | -.18773E-01 | .27405E+01 |
| .50000E+00 | -.34402E+00 | .27369E+01 |
| .60000E+00 | -.53134E+00 | .19243E+01 |
| .70000E+00 | -.52354E+00 | .13960E+01 |
| .80000E+00 | -.47809E+00 | .76453E+00 |
| .90000E+00 | -.49980E+00 | .25115E+00 |
| .10000E+01 | -.32781E+00 | -.17706E+00 |
| .11000E+01 | -.13397E+00 | -.40443E+00 |
| .12000E+01 | .57922E-01 | -.44242E+00 |
| .13000E+01 | .22113E+00 | -.30147E+00 |
| .14000E+01 | .32979E+00 | -.23558E+00 |
| .15000E+01 | .36555E+00 | .32723E+00 |
| .16000E+01 | .32229E+00 | .67490E+00 |
| .17000E+01 | .20350E+00 | .93923E+00 |
| .18000E+01 | .25120E-01 | .10534E+01 |
| .19000E+01 | -.18709E+00 | .97213E+00 |
| .20000E+01 | -.30944E+00 | .67331E+00 |
| .21000E+01 | -.50114E+00 | .17120E+00 |
| .22000E+01 | -.71700E+00 | -.43804E+00 |
| .23000E+01 | -.75727E+00 | -.12316E+01 |
| .24000E+01 | -.69716E+00 | -.19651E+01 |
| .25000E+01 | -.53119E+00 | -.25359E+01 |
| .26000E+01 | -.27913E+00 | -.29357E+01 |
| .27000E+01 | .44547E-01 | -.31437E+01 |
| .28000E+01 | .40303E+00 | -.28877E+01 |
| .29000E+01 | .75550E+00 | -.23033E+01 |
| .30000E+01 | .10539E+01 | -.13892E+01 |
| .31000E+01 | .12729E+01 | -.21239E+00 |
| .32000E+01 | .95243E+01 | -.12545E+01 |
| .33000E+01 | .45052E+01 | .32994E+01 |
| .34000E+01 | .40568E+01 | .76164E+01 |
| .35000E+01 | .32109E+01 | .11285E+02 |
| .36000E+01 | .20822E+01 | .13948E+02 |
| .37000E+01 | .80767E+00 | .15330E+02 |
| .38000E+01 | -.77116E+00 | .15207E+02 |
| .39000E+01 | -.21788E+01 | .13798E+02 |
| .40000E+01 | -.34007E+01 | .10998E+02 |

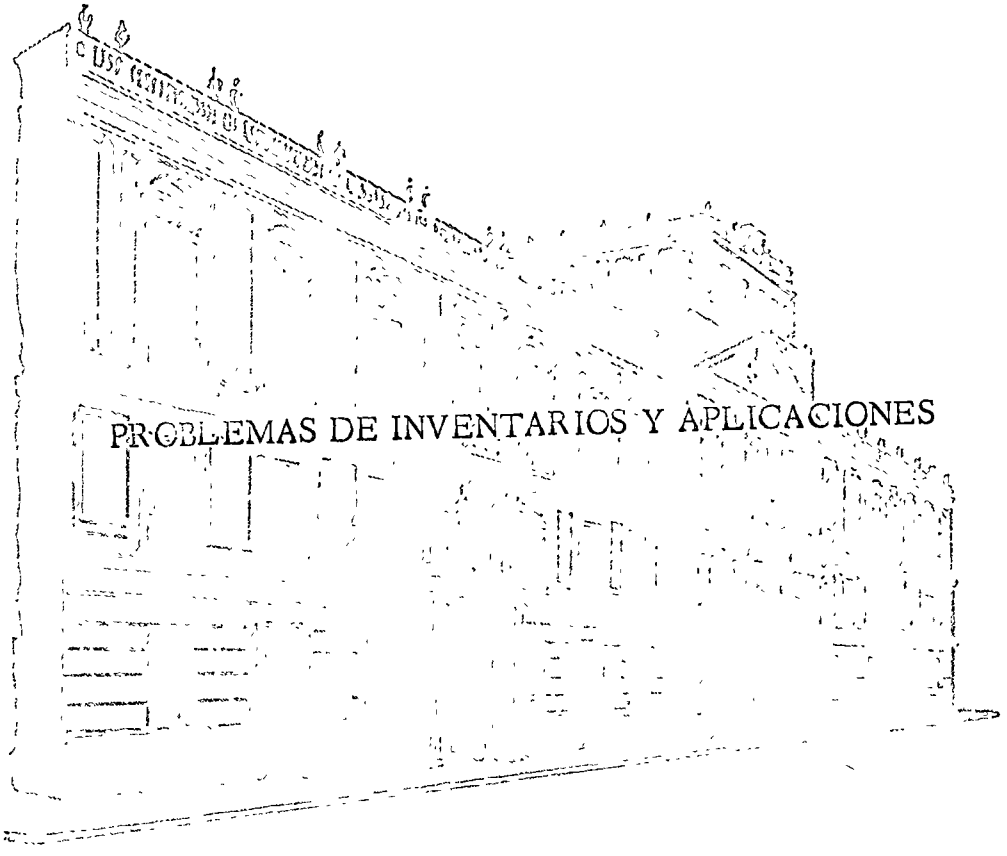


10.5 Bibliografía


1. CANALES Roberto y BARRERA Renato, "Apuntes de Ingeniería de Control 1". México: Fac. de Ingeniería, UNAM, 1973.
   pp.1-36(cap. 3),1-37(cap. 5).
2. CARNAHAN B., LUTHER H., WILKES J.,"Applied Numerical Methods". New York: John Wiley & Sons Inc., 1969.
   pp.361-380.
3. DORF C. Richard, "Modern Control Systems". Reading Mass.: Addison-Wesley Co., 1970.
   pp.250-301, 374-379.
4. GEREZ G. Víctor y MURRAY-LASSO M.A.,"Teoría de Sistemas y Circuitos". México: Representaciones y Servicios de Ingeniería S.A., 1972.
   pp.330-385, 459-463.
5. KUO S. Shan, "Computer Applications of Numerical -- Methods". Reading Mass.: Addison-Wesley Co., 1972.
   pp.128-166.
6. OGATA Katsuhiko , "Modern Control Theory". Englewood Cliffs N.J.: Prentice-Hall Inc., 1970.
   pp.663-704.
7. SMITH G., JAMES M., WOLFORD J., "Applied Numerical Methods for Digital Computation with FORTRAN". Scranton Penn.: International Textbook Co., 1967.
   pp.350-356.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA



# PROBLEMAS DE INVENTARIOS Y APLICACIONES

DR. MARCIAL PORTILLA ROBERTSON

Febrero-Marzo 1977

Palacio de Minería Calle de Tacuba 5, primer piso. México 1, D. F. Tels.: 521-40-23 521-73-35 5123-123

# INSYS

## an inventory system model

*computes inventory levels and factory output for a factory-wholesaler-retailer inventory system. Inventory replenishment and lead time policies may be changed in order to test the effect of these policies on the performance of the overall system.*

The purpose of an inventory is to provide a separation in time or location between the production of goods and the consumption of goods. In our specialized economy a man is no longer his own butcher, baker, and candlestick-maker. Rather, we have production centers (factories) which are specialized, centrally located, and have high production rates. There is a great gain in production efficiency from this specialization, but it also requires a large increase in inventories to separate the centralized factory from the ultimate consumer. No longer do we follow the example of the little red hen who planted, reaped, milled, baked, and ate (without the help of the pig, cow, rabbit, or duck) her own loaf of bread.

The most common inventory system in our economy is the factory-wholesaler-retailer system. The wholesaler provides a time decoupling service between the factory and the retailer, in that he holds the factory output until ordered by the retailer. The wholesaler also provides a location decoupling, in that he generally ships goods over a wide geographic area. Similarly, the retailer provides a decoupling service between the wholesaler and the consumer, in that he maintains an inventory of goods on display for sale to customers.

The purpose of this exercise is to provide an illustration of the dynamic nature of the factory-wholesaler-retailer inventory system. A computer model is used to calculate week by week how the retail inventory, the wholesale inventory, and the factory output rate change in response to retail sales. The model user may make changes in retail and wholesale inventory policy in an attempt to control the overall inventory system.

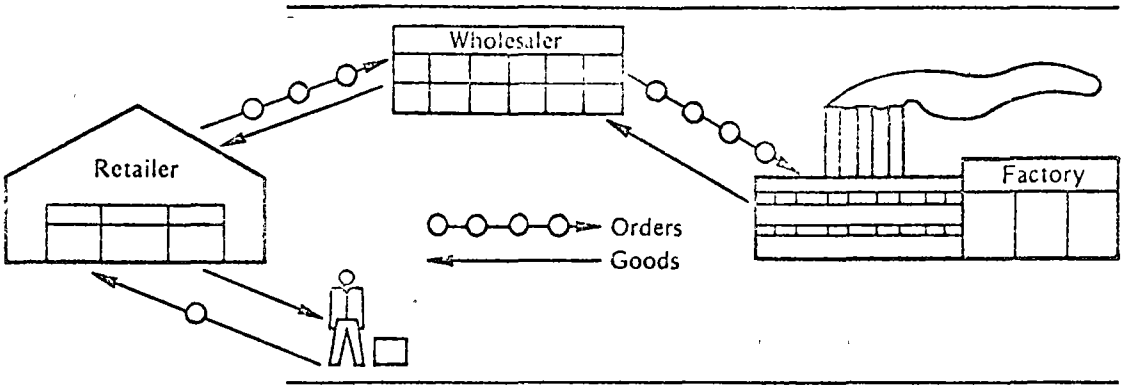



Figure 2-1 The factory-wholesaler-retailer system.

Section 2.1 explains the normal inventory systems and the rules for maintaining inventory levels. The following sections present three illustrated computer problems for maintaining and controlling the inventory system.

## 2.1 FACTORY-WHOLESALER-RETAILER MODEL

The normal system for the production and distribution of goods in our economy is through the factory-wholesaler-retailer system. A visual conceptualization of this system is shown in Figure 2-1. The function of the retailer in this system is to

—take orders from customers
—deliver goods to customers from on-the-shelf inventory
—reorder goods from the wholesaler
—receive shipments from the wholesaler

The function of the wholesaler is similar to the retailer except that the wholesaler's customer is the retailer and there is a time lag between the ordering and the delivery of goods. The wholesaler must

—receive orders from the retailer
—ship goods from the warehouse inventory
—reorder goods from the factory
—receive shipments from the factory.

Finally, the factory must produce the goods which are ultimately sold to the customers. The factory may or may not hold inventories. In the current model the factory does not maintain any inventory so that its only functions are to

—produce goods at some rate
—change production rate as requested by wholesaler

The model just described is a simple abstraction of that which is found in the industrial system. Durable goods, such as applicances, more or less follow the system described. There are variations in that some large retailers order directly from the factory, or the factory may maintain a

showroom and make direct retail sales. In other cases, the factory maintains large inventories and performs the function of the wholesaler. In all of these modifications, however, there is a dynamic interplay of sales with the inventories maintained and the factory rate as illustrated in this model.

**Retailer Model Formulas**

The parameters and formulas for the actual computer model of the retailer are presented in this section. These formulas are a mathematical statement of the verbal model description above. We also present some sample computations using the retail formulas.

*Retail sales* are controlled by the customer. They are part of the input to the program by the reader. Retail sales in the past have been about 100 units per week.

*Retail receipts* are the units received from the wholesaler each Monday morning that were ordered Friday one week (10 days) prior.

*Retail inventory level* is the number of units on hand Friday afternoon at the close of business. The inventory level varies through the week as shown in Figure 2-2. The formula for determining the inventory level is

Inventory level = prior inventory level + retail receipts − retail sales

*Retail orders* are placed with the wholesaler each Friday afternoon after determining the inventory level. The order policy is to order the retail sales for the week plus or minus enough units to return the base stock level to 100 units. Thus

Retail order = retail sales + (100 − inventory level)

The effects of these formulas on inventory level and the retail order can be seen in the following sample computation.

In a normal week

Retail sales = 100
Retail receipts = 100
Retail inventory level = 100 + 100 − 100
= 100
Retail order = 100 + (100 − 100)
= 100

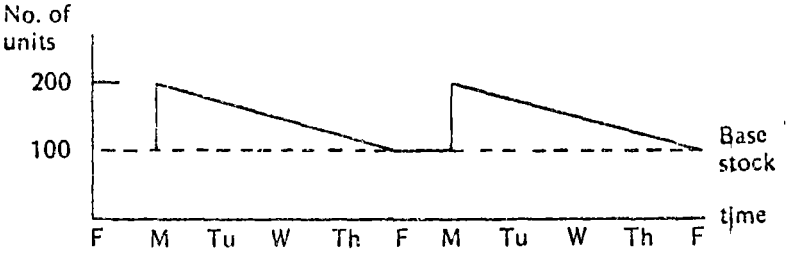



**Figure 2-2** Retail inventory level.

In a week in which sales increase

Retail sales = 110
Retail receipts = 100
Retail inventory level = 100 + 100 − 110
= 90
Retail order = 110 + (100 − 90)
= 120

In a week in which sales decrease

Retail sales = 90
Retail receipts = 100
Retail inventory level = 100 + 100 − 90
= 110
Retail order = 90 + (100 − 110)
= 80

**Wholesaler Model Formulas**

The wholesaler's policies for maintaining inventory and reordering from the factory are similar to the retailer's policies. The formulas for the wholesaler and sample computations are now presented.

*Wholesale shipments* are dispatched each Wednesday from orders submitted by the retailer on the prior Friday. These orders arrive at the retailer's on the following Monday.

Wholesale shipments = retail order (prior week)

*Wholesale receipts* is the factory production of the previous week which is received each Monday morning.

Wholesale receipts = factory production (prior week)

*Wholesale inventory level* is the number of units on hand Friday afternoon at the close of business. The inventory level actually varies through the week as shown in Figure 2-3.

The formula for determining the Friday afternoon inventory level is as follows:

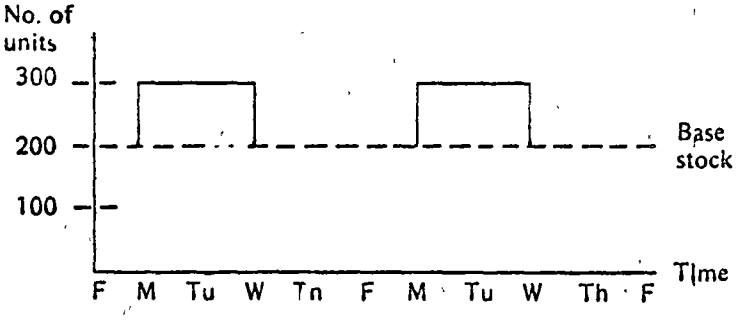



**Figure 2-3 Wholesale inventory.**

Wholesale inventory level = prior inventory level + wholesale receipts
− wholesale shipments

*Wholesale orders* are placed with the factory each Friday afternoon after taking inventory. The factory, however, requires a week to change the production rate, so two weeks pass before the wholesaler receives the actual order. The policy is to order the current week's shipments plus enough units to return the base stock to a normal level of 200 units. The formula is

Wholesale order = wholesale shipments
+ (200 − inventory level)

The effects of the wholesaler's policies can be seen in the following sample computation.

In a normal week:

Wholesale shipments = 100
Wholesale receipts = 100
Wholesale inventory level = 200 + 100 - 100
= 200
Wholesale order = 100 + (200 - 200)
= 100

In a week in which shipments increase

Wholesale shipments = 110
Wholesale receipts = 100
Wholesale inventory level = 200 + 100 − 110
= 190
Wholesale order = 110 + (200 − 190)
= 120

In a week in which shipments decrease

Wholesale shipments = 90
Wholesale receipts = 100
Wholesale inventory level = 200 + 100 − 90
= 210
Wholesale order = 90 + (200 − 210)
= 80

It should be noted that in the present simplified model, the wholesaler only services one retailer. This is an obvious oversimplification from the real world and allows the analysis to isolate the effect of a single retailer in the entire system.

**Factory Production Rate**

In this model, the factory maintains no inventory. The factory produces at the rate specified by the wholesale order. There is, however, a one-week delay when changing the production rate and a one-week delay for shipping. The net effect is that the wholesaler receives the actual order two weeks after it is placed with the factory. Thus, for example, one might have the situation shown in Figure 2-4.

| Week number | Wholesale order | Factory rate | Wholesale receipts |
|---|---|---|---|
| 1 | 100 | 100 | 100 |
| 2 | 120 | 100 | 100 |
| 3 | 80 | 100 | 100 |
| 4 | 100 | 120 | 100 |
| 5 | 100 | 80 | 120 |
| 6 | 100 | 100 | 80 |

Figure 2-4 Changing factory production rate.

## 2.2 NORMAL INVENTORY POLICY

This section presents the results of following a normal inventory policy. By "normal" we mean that the retailer and wholesaler follow the rules described in the preceding sections. The most significant rule, which will be analyzed in detail later, is the reorder rule. The normal reorder rule which is followed in the current problem is

*Order the current week's sales plus or minus enough to bring the base stock back to its normal level.*

Following this reorder rule and the other inventory policies outlined in Section 2.1, one can compute over a number of weeks the inventory level and orders in response to retail sales. For example, if retail sales are 100 in weeks 1 and 2, then increase to 110 in weeks 4, 5, 6, and 7, results will occur as shown in Figure 2-5.

These results are arrived at by following the computation formulas given in the preceding section. For example, the formula for the retail order each week is as follows:

Retail order = weekly sales + (100 − inventory level)

The retail order in week 5 is 120 units, derived from the above formula as follows:

Retail order = 110 + (100 − 90)
= 120

| Week | Retail | | | | | Wholesale | | | Factory |
|---|---|---|---|---|---|---|---|---|---|
| No. | Sales | Rec | Inv | Order | Ship | Rec | Inv | Order | Rate |
| 1 | 100 | 100 | 100 | 100 | 100 | 100 | 200 | 100 | 100 |
| 2 | 100 | 100 | 100 | 100 | 100 | 100 | 200 | 100 | 100 |
| 3 | 110 | 100 | 90 | 120 | 100 | 100 | 200 | 100 | 100 |
| 4 | 110 | 100 | 80 | 130 | 120 | 100 | 180 | 140 | 100 |
| 5 | 110 | 120 | 90 | 120 | 130 | 100 | 150 | 180 | 100 |
| 6 | 110 | 130 | 110 | 100 | 120 | 100 | 130 | 190 | 140 |
| 7 | 110 | 120 | 120 | 90 | 100 | 140 | 170 | 130 | 180 |

Figure 2-5 "Normal" inventory policy.

It is quite possible to compute the results by hand for as many weeks of operation as desired. Fortunately, the computer program will do the tedious calculations. We next consider briefly the data cards required to run the computer program and then we will analyze twenty-five weeks of operation by using the computer model.

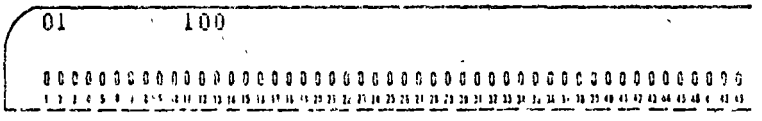
**Computer Input**

This section presents the data cards required as input to the computer program. The program listing is found in Section 2.6.

*User name card* is the first card in the data deck. It is used to identify who is making the analysis and any other identification desired. This card is *free field*, in that any information may be keypunched in columns 1 to 40


*Control card* is the second card in the data deck. This card contains the number of weeks to be analyzed, which is punched in columns 1 and 2.


*Weekly sales cards* contain the week number and the retail sales for that week. There is one card per week and the total number of cards must be exactly the number specified by the control card. The week number is keypunched in columns 1 and 2. The retail sales for the week are in columns 11, 12, and 13, *right justified* (that is, the last digit is always in column 13).




The complete data deck setup for problem one is shown in Figure 2-6. Each typewritten line in the figure corresponds to one keypunched data card.

**Computer Output**

The computer printout resulting from the normal reorder policy is depicted in Figure 2-7. The first line in the computer output is a reproduction of the information keypunched on the first data card. The last line of the computer output is information from the second data card, the control card. The notation 25 WEEKS RUN is a reminder that the control card specified that 25 weeks of data were to be run. If there is too little weekly sales data or if the weekly sales data is out of order the computer will print the message SOMETHING WRONG WITH YOUR DATA and stop processing the program.

```
EXERCISE ONE BY ROY HARRIS
25
01          100
02          100
03          110
04          110
05          110
06          110
07          110
08          110
09          110
10          110
11          110
12          110
13          110
14          110
15          110
16          110
17          110
18          110
19          110
20          110
21          110
22          110
23          110
24          110
25          110
```

Figure 2-6 Computer input—normal policy.

```
PROGRAM INSYS FOR EXERCISE ONE BY ROY HARRIS

WEEK -------RETAIL---------  *****WHOLESALE******** FACTORY
NO.  SALES REC   INV  ORDER   SHIP  REC  INV  ORDER   RATE
 1   100   100   100   100    100   100  200   100    100
 2   100   100   100   100    100   100  200   100    100
 3   110   100    90   120    100   100  200   100    100
 4   110   100    80   130    120   100  180   140    100
 5   110   120    90   120    130   100  150   180    100
 6   110   130   110   100    120   100  130   190    140
 7   110   120   120    90    100   140  170   130    180
 8   110   100   110   100     90   180  260    30    190
 9   110    90    90   120    100   190  350     0    130
10   110   100    80   130    120   130  360     0     30
11   110   120    90   120    130    30  260    70      0
12   110   130   110   100    120     0  140   180      0
13   110   120   120    90    100     0   40   260     70
14   110   100   110   100     90    70   20   270    180
15   110    90    90   120    100   180  100   200    260
16   110   100    80   130    120   260  240    80    270
17   110   120    90   120    130   270  380     0    200
18   110   130   110   100    120   200  460     0     80
19   110   120   120    90    100    80  440     0      0
20   110   100   110   100     90     0  350     0      0
21   110    90    90   120    100     0  250    50      0
22   110   100    80   130    120     0  130   190      0
23   110   120    90   120    130     0    0   330     50
24   110   130   110   100    120    50    0   320    190
25   110   120   120    90    100   190   90   210    330

25 WEEKS RUN  0.0  0  0
```

Figure 2-7 Computer output—normal policy.

**Analysis of the Normal Inventory Policy**

It is quite evident that the so called normal inventory policy is not a very smart policy. A simple increase in retail sales to a new level 10 percent higher than before has set off uncontrollable fluctuations in the wholesale inventory and in the factory rate. Even though the factory services only one wholesaler and one retailer these uncontrollable swings cause the factory to completely shut down by week 11. Negative inventories, orders, or factory rates are not allowed.

By week 25 the situation is still not in control. The retailer has not stabilized his inventory level back to 100 units, the wholesaler has not stabilized, and the factory is going from boom to bust. This cyclic behavior in the system is the result of the lead times in the system and the "blind" ordering policies of the retailer and the wholesaler. The next two sections consider some methods for bringing this situation under control.

## 2.3 CONTROLLING THE REPLENISHMENT RATE

This section considers the problem of controlling fluctuation in the inventory system through a change in the reorder policies of the retailer and the wholesaler. The basic concept applied is that of *dampening* the amplitude of change. This concept is implemented by changing the reorder policy to decrease the amount of replenishment of the base stock. The new policy, the computer output, and an analysis of the results are presented here.

**The Replenishment Concept**

According to the old policy, the reorder formula for the retailer is

Retail order = retail sales + (100 − inventory level)

This policy says in effect that the retailer wants to replenish the stock he has actually sold during the week. In addition, if sales are above or below the base stock level of 100 units he wants to maintain, he will order enough to bring the base stock to 100.

This policy appears reasonable enough but it is based on the rather nearsighted assumptions that

—future sales will be the same as this week's
—stock replenishment is instantaneous

The first assumption is obviously risky for almost any retail operation. The second assumption is obviously not fulfilled in the present system. The retailer orders on Friday, the goods are shipped on the next Wednesday and received the following Monday. Each Friday, the retailer orders enough "to bring the base stock back to normal" even though the goods he ordered the prior Friday to bring the base stock back to normal still have not arrived. When the order does arrive the retailer overreacts by ordering too little the next time. The net result, as seen in Section 2.2, is that the retailer is never able to stabilize his order or inventory level. Business cycles may be caused by just this kind of behavior.

One way to dampen the swings is to change the replenishment policy to specify that only a percentage of the base stock difference is to be ordered. Thus we change the formula to

Retail order = retail sales + (100 − inventory level) (*A*%)

If we set $A$ at 50 percent and thus try to make up only one-half the difference, then we can compare the retail order when sales are up to 110 units.

| *Old Policy* | *New Policy* |
|---|---|
| Retail order = 110 + (100 − 90) | Retail order = 110 + [(100 − 90) X 0.5] |
| = 110 + 10 | = 110 + 5 |
| = 120 | = 115 |

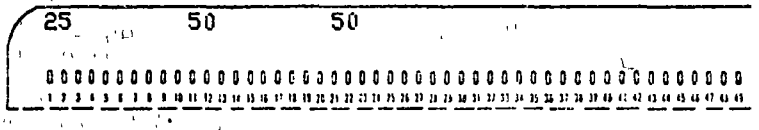
When sales are down to 90 units the result is

| *Old Policy* | *New Policy* |
|---|---|
| Retail order = 90 + (100 − 110) | Retail order = 90 + [(100 − 110) X 0.5] |
| = 90 − 10 | = 90 − 5 |
| = 80 | = 85 |

The overall effect of the new policy is that the retailer only partly reacts to increases or decreases in the base stock and allows some time for inventories to return to normal. The wholesaler may follow a similar policy in ordering from the factory by including $B$ percent in the wholesale order formula.

**Computer Input —Replenishment Rate**

*User name card* remains unchanged. The new reorder policy is implemented by specifying on the *control card* the percentage value for $A$ (retailer) in columns 11 and 12, and $B$ (wholesaler) in columns 21 and 22.




If the field is left blank, the value for $A$ or $B$ is set to 100 percent. Otherwise, $A$ or $B$ may be set from 01 to 99 by the user.

*Weekly sales cards* are keypunched as in Section 2.2.

A complete data deck listing for the new policy is shown in Figure 2-8. The retailer and the wholesaler only try to make up 50 percent of the difference in base stock under the new policy.

**Computer Output —Replenishment Rate**

The computer printout for the new 50 percent reordering level policy which is generated from these data cards is shown in Figure 2-9. Note that the last line of the printout includes the input values for $A$ and $B$ specified in the control card.

**Analysis of the Replenishment Rate Control**

The overall result of the new reordering policy is a dramatic improvement in the performance of the inventory system.

Retail reorders match the new sales level within eleven weeks.
Wholesale reorders match the new sales level within twelve weeks.
Factory rate is not yet stable, but appears to be dampening out.

Most significantly, the system is no longer out of control, i.e., caught up in uncontrollable fluctuation. The fluctuations have been dampened out and the system stabilizes itself to the new sales level.

```
 EXERCISE TWO BY ROY HARRIS
25          50          50
01          100
02          100
03          110
04          110
05          110
06          110
07          110
08          110
09          110
10          110
11          110
12          110
13          110
14          11
15          11
16          11
17          11
18          11
19          11
20          110
21          11
22          11
23          11
24          11
25          11
```

Figure 2-8 Computer input—replenishment rate.

```
PROGRAM INSYS FOR  EXERCISE TWO BY ROY HARRIS

WEEK ---------RETAIL----------  *****WHOLESALE******** FACTORY
NO.  SALES REC  INV  ORDER  SHIP REC  INV  ORDER  RATE
 1   100   100  100  100    100  100  200  100    100
 2   100   100  100  100    100  100  200  100    100
 3   110   100  90   115    100  100  200  100    100
 4   110   100  80   120    115  100  185  123    100
 5   110   115  85   118    120  100  165  138    100
 6   110   120  95   113    118  100  148  144    123
 7   110   118  103  109    113  123  156  134    138
 8   110   113  105  108    109  138  166  116    144
 9   110   109  104  108    108  144  223  96     134
10   110   108  101  109    108  134  248  84     116
11   110   108  99   110    109  116  254  82     96
12   110   109  99   111    110  96   240  90     84
13   110   110  99   110    111  84   214  104    82
14   110   111  100  110    110  82   185  118    90
15   110   110  100  110    110  90   165  127    104
16   110   110  100  110    110  104  159  130    118
17   110   110  100  110    110  118  167  126    127
18   110   110  100  110    110  127  185  118    130
19   110   110  100  110    110  130  205  107    126
20   110   110  100  110    110  126  221  99     118
21   110   110  100  110    110  118  229  96     107
22   110   110  100  110    110  107  226  97     99
23   110   110  100  110    110  99   216  102    96
24   110   110  100  110    110  96   201  109    97
25   110   110  100  110    110  97   188  116    102

25 WEEKS RUN 50 50  0  0
```

Figure 2-9 Computer output—replenishment rate.

However, all is still not perfect. There is still a long time lag before the factory "catches on" to the new rate. Moreover, a simple 10 percent increase in sales still causes a 20 percent change in the wholesale shipments and a 44 percent change in the factory rate. Section 2.4 considers additional control measures for bringing the inventory system under even tighter control.

## 2.4 CONTROL OF LEAD TIME

This section considers the problem of controlling fluctuations in the inventory system through a decrease in the lead time between the order and the receipt of goods. The basic concept is to change the lead time required to respond to changes in the system. This concept is implemented by testing the effect of faster delivery from the wholesaler and faster changeover to a new production rate by the factory.

Lead Time Concept

Under the "normal" system setup the two basic lead times in the system were (1) between the order and receipt of goods by the retailer, and (2) between the order and receipt of goods from the factory. These were as follows:

*Retailer normal lead time*

| Order on | Delivered on |
|---|---|
| Friday week 1 | Monday week 3 |

*Wholesaler normal lead time*

| Order on | Change rate | Deliver goods |
|---|---|---|
| Friday week 1 | Week 3 | Monday week 4 |

The effects of these lead times are clearly seen in Section 2.2 when the retailer reorders every Friday to make up goods that have previously been ordered. In effect, he makes a double reorder for the same goods. In addition, the factory takes seven weeks to begin to respond to a change in retail sales.

In the current example we will consider the effects of decreasing this lead time. The new policy is to work the wholesaler on Saturday in order to deliver Friday-afternoon orders the very next Monday. Thus

*Retailer decreased lead time*

| Order on | Delivered on |
|---|---|
| Friday week 1 | Monday week 2 |

Similarly the lead time for the wholesaler may be changed if the factory can shift to a new production rate without a week lag and if the factory ships over the weekend.

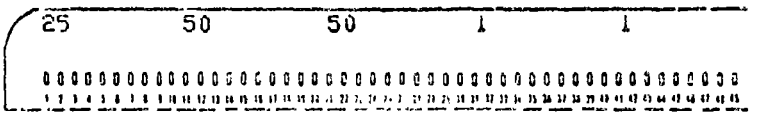
*Wholesaler decreased lead time*

| Order on | Change rate | Delivered in |
|---|---|---|
| Friday week 1 | week 2 | week 3 |

**Computer Input —Lead Times**

The *user name card* is the same as that in Section 2.2. The change in the lead times is implemented in the computer model by two fields in the *control card.* A speedup of one week in the wholesaler deliveries is accomplished by placing a 1 in column 31 of the control card. A speedup of one week in the changeover of the factory is accomplished by placing a 1 in column 41 of the control card.

The control card now reads




The *weekly sales cards* retain the same format.

The complete data deck setup to test the effects of the decreased lead time is shown in Figure 2-10.

**Computer Output**

The computer printout with lead time decreased is shown in Figure 2-11

**Analysis of Lead Time Control**

The result of the new lead time policy is a further improvement in the overall performance of the inventory system.

Retail orders match the new sales rate within five weeks.

```
 EXERCISE THREE BY ROY HARRIS
25          50          50          1           1
01          100
02          100
03          110
04          110
05          110
06          110
07          110
08          110
09          110
10          110
11          110
12          110
13          110
14          110
15          110
16          110
17          110
18          110
```

```
PROGRAM INSYS FOR  EXERCISE THREE BY ROY HARRIS

WEEK --------RETAIL---------  ********WHOLESALE********  FACTORY
NO.  SALES REC   INV  ORDER   SHIP  REC  INV  ORDER   RATE
 1   100   100   100  100     100   100  200  100     100
 2   100   100   100  100     100   100  200  100     100
 3   110   100    90  115     115   100  185  123     100
 4   110   115    95  113     113   100  173  126     123
 5   110   113    98  111     111   123  184  119     126
 6   110   111    99  111     111   126  199  111     119
 7   110   111    99  110     110   119  208  106     111
 8   110   110   100  110     110   111  209  106     106
 9   110   110   100  110     110   106  205  107     106
10   110   110   100  110     110   106  201  110     107
11   110   110   100  110     110   107  198  111     110
12   110   110   100  110     110   110  198  111     111
13   110   110   100  110     110   111  199  111     111
14   110   110   100  110     110   111  200  110     111
15   110   110   100  110     110   111  200  110     110
16   110   110   100  110     110   110  201  110     110
17   110   110   100  110     110   110  200  110     110
18   110   110   100  110     110   110  200  110     110
19   110   110   100  110     110   110  200  110     110
20   110   110   100  110     110   110  200  110     110
21   110   110   100  110     110   110  200  110     110
22   110   110   100  110     110   110  200  110     110
23   110   110   100  110     110   110  200  110     110
24   110   110   100  110     110   110  200  110     110
25   110   110   100  110     110   110  200  110     110

25 WEEKS RUN 50 50  1  1
```

Figure 2-11 Computer output—lead times.

Wholesaler orders match the new sales rate within eight weeks.
Factory rate is set to the new sales level in nine weeks.

In addition to cutting response lags down, there is less fluctuation in the inventory levels.

A simple increase in sales of 10 percent causes a 15 percent change in wholesale shipments, down from the prior 20 percent change. Also, the factory rate changes 20 percent, down from the prior 44 percent. Thus, in general, it may be said that the inventory system is now in better control.

There are, however, many complications one could add to the model before it would approximate the real world. For example, customers are never so kind as to provide such nice uniform retail sales. Hence, the user might want to try his hand at controlling the inventory system if retail sales were to randomly fluctuate between, say, 80 units and 120 units in any given week.

## REFERENCES

Ansoff, H. I., and D. Slevin, "An Appreciation of Industrial Dynamics," *Management Science*, vol. 14, no. 7 (March 1968), pp. 398-415.

Buffa, E. S., *Modern Production Management*, New York: Wiley, 1969.

———, *Models for Production and Operations Management*, New York: Wiley, 1963.

Fetter, R. B., et al, *Decision Models for Inventory Management*, Homewood, Ill.; Irwin, 1961.

Forrester, J. W., *Industrial Dynamics*, Cambridge, Mass.: M.I.T., 1961.

———, "Industrial Dynamics—After the First Decade," *Management Science*, vol. 14, no. 7 (March 1968), pp. 398-415.

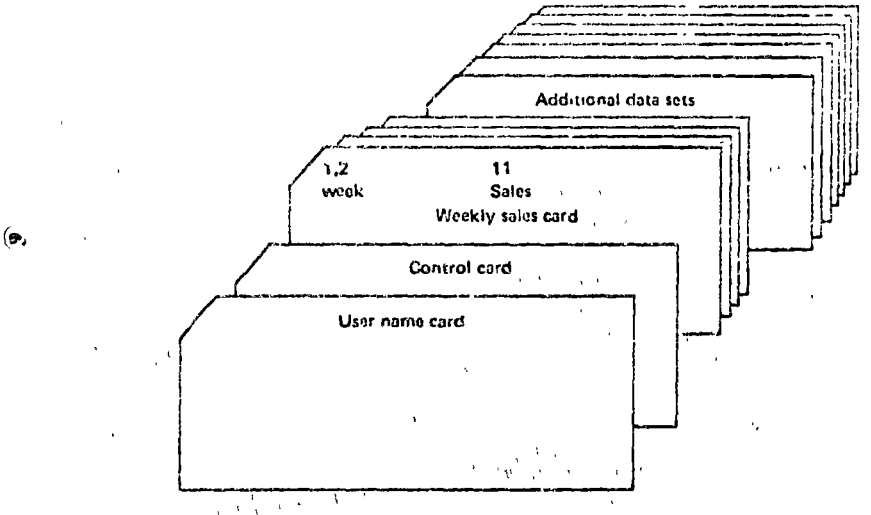
Gavett, J. W., *Production and Operations Management*, New York: Harcourt, Brace & World, 1968.

Meier, R. C., W. T. Newell, and H. L. Pazer, *Simulation in Business and Economics*, Englewood Cliffs, N.J.: Prentice-Hall, 1969.

Olson, R. A., *Manufacturing Management: A Quantitative Approach*, Scranton, Pa.: International Textbook Co., 1968.

Plossl, G. W., et al, *Production and Inventory Control*, Englewood Cliffs, N.J.: Prentice-Hall, 1967.

## 2.5 INSYS DATA DECK STRUCTURE




| Control Card | *Card columns* | *Format* | *Item* |
|---|---|---|---|
| | 1, 2 | I 2 | No. of weeks |
| | 11, 12 | I 2 | Percentage value for retailer |
| | 21, 22 | I 2 | Percentage value for wholesaler |
| | 31 | I 1 | Wholesaler lead time |
| | 41 | I 1 | Factory lead time |

## 2.6 INSYS PROGRAM LISTING

```
C     PROGRAM INSYS                                                    IBM   1
C     INVENTORY SYSTEM CONTROL MODEL                                     A   2
C     COPYRIGHT 1969 BY ROY D HARRIS                                     A   3
C     THIS VERSION FOR IBM 360                                         IBM   4
      DIMENSION ALPHA(10)                                                A   5
C                                                                        A   6
C     READ AND PRINT STUDENTS NAME CARD                                  A   7
      MI = 5                                                           IBM   8
      MO = 6                                                           IBM   9
 1    READ (MI,26) ALPHA                                                 A  10
      WRITE (MO,27) ALPHA                                                A  11
C                                                                        A  12
C     READ CONTROL CARD AND INITIALIZE                                   A  13
      READ (MI,28) N, IR, IW, LW, LF                                     A  14
      IF (IR) 2,2,3                                                      A  15
 2    A = 1.0                                                            A  16
      GO TO 4                                                            A  17
 3    A = IR/100.                                                        A  18
 4    IF (IW) 5,5,6                                                      A  19
 5    B = 1.0                                                            A  20
      GO TO 7                                                            A  21
 6    B = IW/100.                                                        A  22
 7    RI = 100.                                                          A  23
      RO = 100.                                                          A  24
      WS = 100.                                                          A  25
      WI = 200.                                                          A  26
      WO2 = 100.                                                         A  27
      WO1 = 100.                                                         A  28
      FR = 100.                                                          A  29
C                                                                        A  30
C     PRINT HEADINGS FOR WEEKLY OUTPUT                                   A  31
      WRITE (MO,29)                                                      A  32
      WRITE (MO,30)                                                      A  33
C                                                                        A  34
C     START OF DO LOOP THROUGH WEEKLY COMPUTATIONS                       A  35
      DO 24 I = 1, N                                                     A  36
C                                                                        A  37
C     READ AND CHECK DATA CARD CONTAINING WEEKLY SALES                   A  38
      READ (MI,31) KWEEK, SALES                                          A  39
      IF (I-KWEEK) 8,9,8                                                 A  40
 8    WRITE (MO,32)                                                      A  41
      GO TO 25                                                           A  42
C                                                                        A  43
C     COMPUTE RETAIL INVENTORY LEVEL AND ORDER                           A  44
 9    RREC = WS                                                          A  45
      RINV = RI+RREC-SALES                                               A  46
      IF (RINV) 10,10,11                                                 A  47
 10   RINV = 0.0                                                         A  48
 11   RORD = SALES*((100.0-RINV)*A)                                      A  49
      IF (RORD) 12,12,13                                                 A  50
 12   RORD = 0.0                                                         A  51
C                                                                        A  52
C     SET WHOLESALE DELIVERY RATE FROM CONTROL CARD                      A  53
 13   IF (LW-1) 15,14,15                                                 A  54
 14   WSHIP = RORD                                                       A  55
      GO TO 16                                                           A  56
 15   WSHIP = RO                                                         A  57
C                                                                        A  58
C     COMPUTE WHOLESALE INVENTORY LEVEL AND ORDER                        A  59
 16   WREC = FR                                                          A  60
      WINV = WI+WREC-WSHIP                                               A  61
      IF (WINV) 17,17,18                                                 A  62
```

```
 17    WINV = 0.0                                                        A  63
 18    WORD = WSHIP+((200.-WINV)*B)                                      A  64
       IF (WORD) 19,19,20                                                A  65
 19    WORD = 0.0                                                        A  66
C                                                                        A  67
C      SET FACTORY RATE FROM CONTROL CARD                                A  68
 20    IF (LF-1) 22,21,22                                                A  69
 21    FRATE = W01                                                       A  70
       GO TO 23                                                          A  71
 22    FRATE = W02                                                       A  72
C                                                                        A  73
C      PRINT OUT CURRENT WEEK RESULTS                                    A  74
 23    WRITE (MO,33) I, SALES, RREC, RINV, RORD, WSHIP, WREC, WINV, WORD,A  75
      1 FRATE                                                            A  76
C      UPDATE ORDERING AND FACTORY RATE FOR NEXT WEEK                    A  77
       R1 = RINV                                                         A  78
       RO = RORD                                                         A  79
       WS = WSHIP                                                        A  80
       W1 = WINV                                                         A  81
       W02 = W01                                                         A  82
       W01 = WORD                                                        A  83
       FR = FRATE                                                        A  84
 24    CONTINUE                                                          A  85
C                                                                        A  86
C      PRINT CONTROL CARD VALUES                                         A  87
       WRITE (MO,34) N, IR, IW, LW, LF                                   A  88
 25    CONTINUE                                                          A  89
       GO TO 1                                                           A  90
C                                                                        A  91
C                                                                        A  92
C                                                                        A  93
 26    FORMAT     (10A4)                                                 A  94
 27    FORMAT     (1H1,18HPROGRAM INSYS FOR  ,10A4)                      A  95
 28    FORMAT    (I2,9X,I2,8X,I2,8X,I1,9X,I1)                            A  96
 29    FORMAT     (3HOWEEK -------RETAIL---------                        A  97
      1   3HO*****WHOLESALE********,FACTORY)                             A  98
 30    FORMAT     (1H ,36HNO.  SALES REC   INV  ORDER   SHIP            A  99
      1   23HREC   INV  ORDER  RATE)                                     A 100
 31    FORMAT     (I2,8X,F3.)                                           A 101
 32    FORMAT     (3H SOMETHING WRONG WITH YOUR DATA)                   A 102
 33    FORMAT    (1H ,I2,4F6.0,F7.0,3F6.0,F7.0)                         A 103
 34    FORMAT     (1HO,I2,10H WEEKS RUN,4I3)                            A 104
       END                                                              A 105
```

## a model for inventory ordering policy

*computes the most economical inventory order quantity under a variety of conditions, including price discounts, shortage cost, and storage limitations.*

A primary purpose of inventories is to decouple production from consumption. Inventories are goods which may be used as a hedge against uncertainty in demand or as a buffer for production fluctuations.

The *replenishment* of inventories is the topic of this exercise. We describe an elementary, but fundamental, inventory replenishment model the Economic Order Quantity (EOQ) model. In Section 3.1, the development of the basic EOQ model is given. In Section 3.2, the basic EOQ model is extended to include a real world phenomenon: quantity price discounts. In Section 3.3, the basic model is modified to incorporate shortage costs. Consideration of storage limitations and their effects upon the order quantity decision are the subject of Section 3.4.

The rational basis for deciding how much, if any, inventory to hold, and to order, is an economic basis There are costs associated with holding inventory in stock, e.g., insurance, taxes, interest on capital, and so on. Conversely, there are costs related to not holding inventory, e.g., lost sales, frequent purchase orders, production delays, etc. There is also the cost of purchasing the replenishment for inventories, e.g., paperwork and material handling.

The intent of this exercise is to allow the user an opportunity to get a feel for the *effects* of changing parametric values in the basic economic order quantity formulas. Hence, the reader is encouraged to conduct *sensitivity analysis* on each parameter in the EOQ formula.

### 3.1 ECONOMIC ORDER QUANTITY

This section introduces the basic Economic Order Quantity (EOQ) model. It also describes in detail the data cards required for the accompanying computer program, and the computer output.

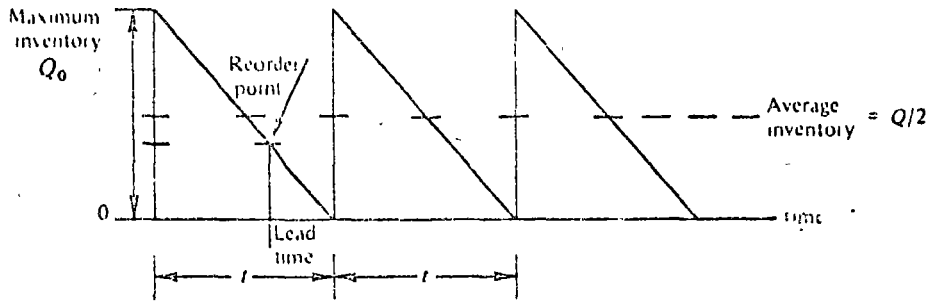



Figure 3-1 Inventory level and usage pattern for EOQ model.

**The Economic Order Quantity Inventory Model**

A basic assumption of the EOQ model is that the consumption of the inventory is constant over time and that it is possible to replenish the inventory on very short notice. The quantity in inventory at any point in time, for this circumstance, is shown in Figure 3-1. The basic EOQ model also assumes no quantity price discounts and no backorders.

In this "sawtooth" usage pattern the inventory is consumed over time until it is depleted. It is then instantly replenished (straight vertical line), and the usage continued.

**Inventory Cost**

The rational basis for determining inventory levels is to balance the cost of holding inventory against the cost of not holding inventory. In the consumption situation described above, the types of cost for holding the inventory are fairly obvious. Storage must be provided for the inventory and the inventory must be financed.

The cost of not holding inventory may not be so obvious. Since it was assumed that the inventory was easily and instantaneously obtainable, then there is no cost attached to being caught short. However, like a housewife who goes to the grocery store three times each day to purchase a single meal, there is a cost attached to procurement of the inventory. Not holding inventory may lead to very high procurement cost.

The issue in the economic order quantity model is to determine how much inventory to order each time. The cost of procurement per unit goes down if more is ordered each time, but the cost of holding inventory goes up. Figure 3-2 shows a graphic representation of these costs as they vary with the size of the order. The total incremental cost of inventory is shown as the top curve on the graph in the figure.

$$\text{Total incremental cost} = \text{holding cost} + \text{ordering cost}$$

The best inventory policy is to order the amount of inventory each time which yields the minimum total cost. This "correct quantity" to order is called the economic order quantity (EOQ).

The following definitions and variables will be used in deriving a mathematical expression for the EOQ.

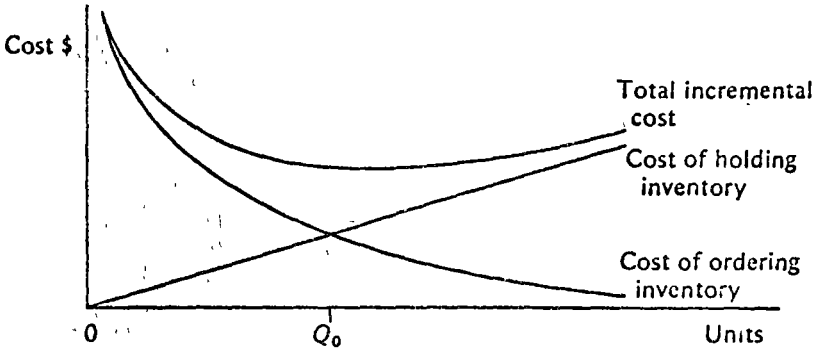



**Figure 3-2** Costs of holding and ordering inventory.

Holding cost = (average inventory ) X (unit inventory holding cost per year)

$= (Q/2) \times (P \times FH)$

where $Q$ = quantity ordered

$P$ = price per unit

$FH$ = annual unit holding cost as percentage of the unit price

Order cost = (number of orders per year) X (cost per order)

$$= \left(\frac{R}{Q}\right) \times (Cp)$$

where $R$ = annual requirements in units, level demand

$Cp$ = procurement cost (includes costs of paperwork, handling, etc.)

Cost of inventory = (unit price) X (annual requirements)

$= (P) \times (R)$

Total cost = holding cost + order cost + cost of inventory

$$\text{Total cost} = \frac{Q \times P \times FH}{2} + \frac{R \times Cp}{Q} + P \times R$$

$$\text{Total incremental cost} = \frac{Q \times P \times FH}{2} + \frac{R \times Cp}{Q}$$

Solving for the economic order quantity $Q_0$ by algebra: the minimum point on the total incremental cost curve is where the inventory holding cost and the procurement cost curves intersect. Where they intersect they must be equal. Therefore, at $Q_0$:

$$\frac{Q}{2}(P \times FH) = \left(\frac{R}{Q}\right) Cp$$

Clearing denominators:

$$Q(Q)(P \times FH) = 2(R)Cp$$

$$Q^2 = \frac{2RCp}{P \times FH}$$

$$Q = \sqrt{\frac{2RCp}{P \times FH}}$$

Solving for $Q$ by calculus the minimum point on the total incremental cost curve is where the first derivative equals zero. Taking the first derivative with respect to $Q$ and setting it equal to zero:

$$0 = (P \times FH)/2 - (R/Q^2)Cp$$

$$\frac{(P \times FH)}{2} = \frac{RCp}{Q^2}$$

$$Q^2 = \frac{2RCp}{P \times FH}$$

$$Q = \sqrt{\frac{2RCp}{P \times FH}}$$

**Sample Problem One**

It is a straightforward matter to find the economic order quantity (EOQ) and the total inventory cost ($TC$), when the values for $R$, $Cp$, $P$, and $FH$ are known. For example, if

$R$ = 1600 units (total annual usage)
$Cp$ = $5.00 (cost of one procurement)
$P$ = $1.00 (unit price of product)
$FH$ = 0.10 (unit holding cost per year as percentage of price), then

$$Q = \sqrt{\frac{2 \times 1600 \times 5.00}{1.00 \times 0.1}}$$

$Q$ = 400 units

$$\text{Total cost } (TC) = \frac{400 \times 1.00 \times 0.1}{2} + \frac{1600 \times 5.00}{400} + 1.00 \times 1600$$

$$TC = 20.00 + 20.00 + 1600.00$$

$TC$ = $1640.00

This computation is not too tedious to do, if there is only one. However, when there are many alternatives to test and when more complicated formulas are required, then a computer program is a great computational aid. The next section introduces the data input and computer output for the simple example shown above. In following sections more complicated problems illustrating the use of the computer program will be presented.

**Computer Input --Problem One**

Before describing the data cards, several comments will be made pertaining to the program itself. The user should keep these comments in mind when using the program.

The program is applicable only to a fixed-order-quantity inventory system, and all quantities in the program are expressed in annual amounts or rates. In the case of $R$ (annual inventory requirement), a level usage rate is assumed throughout the year. In order to convert the program for monthly or seasonal calculations one would have to adjust the imputs to the same time scale.

Although the figure available for inventory holding costs is often stated as an annual cost per unit, this program *requires* that holding costs be expressed as a percentage of the unit value of inventory.

To run the economic order quantity computer model, only two cards

are required: (1) the user name card, and (2) the "data" card. When multiple problems are batched together, a new name card is required for each problem.

The *user name card* may contain any identifying information (such as the user's name) which is desired. This identifying information is keypunched in the first forty columns.

```
MAGGARD EOQ PROBLEM ONE
0000000000000000000000000000000000000000000000000000000000000000000000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
```

The *data card* contains the numerical data for the economic order quantity computation. Columns 1-5 contain the annual usage requirement, and the ordering cost is punched in columns 6-10. The holding cost, expressed as a percentage of the unit price, is keypunched in columns 11-15, and the unit price is punched in columns 16-20.

```
1600.5.00 0.10 1.00
0000000000000000000000000000000000000000000000000000000000000000000000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
```

The user name card and the data card are the only two cards required. A listing of the two cards which produced the output shown in the next section is shown in Figure 3-3.

```
MAGGARD EOQ PROBLEM ONE
1600.5.00 0.10 1.00
```

Figure 3-3 Computer input—Problem One.

**Computer Output —Problem One**

The computer output (Figure 3-4) includes the identification information from the user name card and the information specified on the data card. Below this, the program prints out the quantities calculated in the program. These are (1) the optimum order quantity, (2) the total inventory cost, (3) the number of orders to be placed annually, and (4) the unit price at the order quantity determined.

## 3.2 PRICE DISCOUNTS

When discussing the economic order quantity model in Section 3.1, it was noted that the basic model assumes no quantity price discounts. However, the basic EOQ model may be extended to include price discounts

```
PROGRAM EOQ FOR MAGGARD EOQ PROBLEM ONE
INPUT DATA IS **********
     R     CP     FH     P1     CS     B1     P2     B2     P3      W
  1600   5.00    .10   1.00      0      0      0      0      0      0

ANALYSIS RESULTS ARE ****
 OPTIMUM ORDER QUANTITY IS                                  400.00
 AT A PRICE PER ITEM OF                                       1.00
 YIELDING A TOTAL INVENTORY COST OF                        1640.00
 WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS                 4.00
```

Figure 3-4 Computer output—Problem One.

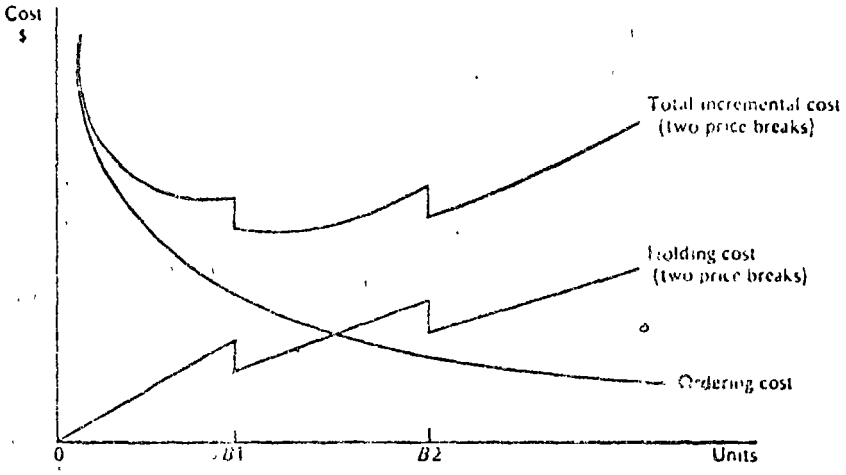



Figure 3-5 The effects of price discounts on the economic order quantity.

as input variables. Inasmuch as price discounts do happen in reality, the extension of the EOQ model to include price discounts will be the subject of this section.

**The EOQ Model —With Price Discounts**

Referring to Section 3.1, the user should note that in the derivation of the EOQ model the price per unit ($P$) affects the holding cost $(Q/2)$ X $(PXFH)$, but not the ordering costs. Nevertheless, if price discounts are introduced as variables, they will influence the total incremental costs ($TIC$). The effects of price discounts are graphically illustrated in Figure 3-5.

The addition of the quantity discounts to the economic order quantity model makes it somewhat more difficult to obtain a solution. It is not possible to find directly the lowest point on the Total Incremental Cost ($TIC$) curve shown in Figure 3-4. The general approach used is to investigate the $TIC$ curve at each price break. In addition, the curve must be analyzed at different points near the price break giving the lowest $TIC$ to see if an even better solution can be found. Problem Two illustrates this general search solution when price discounts are to be considered.

**Sample Problem Two**

The supplier has recently revised his pricing policies and now offers the following price discounts. If one orders in lot sizes of at least $Q_{B1}$ ($Q_{B1}$ = 300), the price will be $0.90/unit ($P_2$). If one orders quantity at least $Q_{B2}$ ($Q_{B2}$ = 2000), the price will be $0.80/unit ($P_3$).

**Solution for EOQ with Two Price Breaks**

First calculate $Q_3$ using $P_3$. If it is greater than $Q_B$, then order $Q_3$. If it is less than $Q_B$, then (using $P_3$) it is infeasible.

Next calculate $Q_2$ using $P_2$. If $Q_2 > Q_{B2}$, then order $Q_{B2}$.

If $Q_2$ is less than $Q_{B2}$ but greater than $Q_{B1}$, i.e., $Q_{B1} < Q_2 < Q_{B2}$, then compare $TC_2$ with $TC_{B2}$.

If $TC_2 > TC_{B2}$, then order $Q_{B2}$.

If $TC_2 < TC_{B2}$, then order $Q_2$.
If $Q_2$ is less than $Q_{B1}$, calculate $Q_1$.
If $Q_1 > Q_{B1}$, then compare $TC_{B1}$ with $TC_{B2}$.
If $TC_{B1} > TC_{B2}$, then order $Q_{B2}$.
If $TC_{B1} < TC_{B2}$, then order $Q_{B1}$.
If $Q_1$ is less than $Q_{B1}$, then compare $TC_1$ with $TC_{B1}$ with $TC_{B2}$. Order the quantity corresponding to the minimum total cost.

$$Q_3 = \sqrt{\frac{2(1600)(5.00)}{0.80(0.10)}} = 447.2 \text{ (using } P_3\text{)}$$

Since $Q_3 < Q_{B2}$, calculate $Q_2$ using $P_2$.

$$Q_2 = \sqrt{\frac{2(1600)(5.00)}{0.90(0.10)}} = 421.6 \text{ (using } P_2\text{)}$$

Since $Q_2$ is less than $Q_{B2}$ (2000), but greater than $Q_{B1}$ (300), we must compare $TC_2$ with $TC_{B2}$.

$$TC_2 = \frac{(0.90)(0.10)(421.6)}{2} + \frac{1600(5.00)}{421.6} + 1600(0.90)$$
$$= 18.99 + 18.99 + 1440$$
$$= \$1477.98$$

$$TC_{B2} = \frac{0.80(0.10)(2000)}{2} + \frac{1600(5.00)}{2000} + 1600(0.80)$$
$$= 80.00 + 4.00 + 1280.00$$
$$= \$1364.00$$

$TC_2$ is greater than $TC_{B2}$, therefore, order in quantities of $B2 (Q_{B2} =$ 2000 units @ \$0.80/unit).

**Computer Input —Problem Two**

The *user name card* is the first card in the data deck. The *data card* for this example problem contains some additional information. Columns 1-20 are the same as described in Section 3.1. The minimum quantity that can be ordered to take advantage of the first price discount is punched in columns 26-30 and the unit price at the first price discount is punched in columns 31-35. Columns 36-40 and 41-45 contain the corresponding information for the second price discount. The data card is as shown:

```
1600.5.00 0.10 1.00 0    300. 0.90 2000.0.8
```

The complete computer input for this example problem is exhibited in Figure 3-6.

```
MAGGARD EOQ PROBLEM TWO, PRICE DISCOUNTS
1600.5.00 0.10 1.00 0    300. 0.90 2000.0.8
```

Figure 3-6. Computer input—Problem Two.

```
PROGRAM EOQ FOR MAGGARD EOQ PROBLEM TWO, PRICE DISCOUNTS
INPUT DATA IS **********
     R     CP     FH     P1     CS     B1     P2     B2     P3     W
  1600   5.00    .10   1.00      0    300    .90   2000    .80     0

ANALYSIS RESULTS ARE ****
 OPTIMUM ORDER QUANTITY IS                                  2000.00
 AT A PRICE PER ITEM OF                                       .80
 YIELDING A TOTAL INVENTORY COST OF                         1364.00
 WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS                  .80
```

Figure 3-7 Computer output—Problem Two.

**Computer Output —Problem Two** The computer printout resulting from the above data is shown in Figure 3-7.

## 3.3 SHORTAGE COSTS

Just as it is true that in the real world quantity price discounts exist, it is also true that *backorders* are a reality. By allowing backorders we are saying that if an order cannot be filled at this time due to stock shortages, then as soon as inventory is available previously unfilled orders, i.e., backorders, will be the first orders to be filled. However, in an inventory system allowing for backorders (see Figure 3-8) a *shortage cost* is usually input relating to the backorder quantities. Generally, this shortage cost consists of costs due to (1) possible lost sales due to stockouts, (2) decreased customer satisfaction, (3) additional costs associated with rush shipments, and so on.

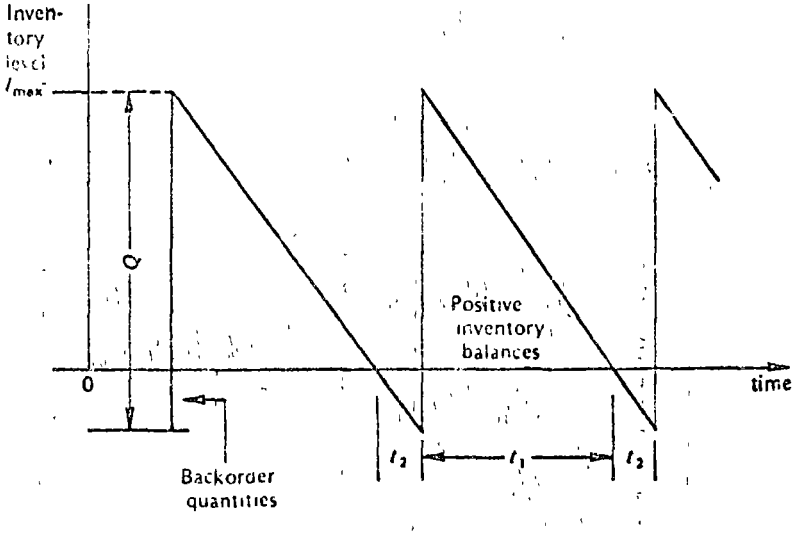



Figure 3-8 An inventory system with $(Q - I_{max})$ backorders allowed.

The basic EOQ formula may be modified to incorporate shortage costs as follows:

1. $Cp$ = procurement cost per order (unchanged)
2. $\frac{(P \times FH)(I_{max})}{2} t_1$ = the holding cost of the *positive inventory balance* during time $t_1$ ($t$ is on an annual basis, i.e., $t$ = a fraction of a year). Since $t_1 = I_{max}/R$, this becomes:

   $$\frac{(P \times FH) I^2_{max}}{2R}$$

3. $Cs \frac{(Q - I_{max})}{2} t_2$ = the shortage cost of the backorders during time $t_2$. Since $t_2 = \frac{Q - I_{max}}{R}$, this becomes:

   $$Cs \frac{(Q - I_{max})^2}{2R}$$

   where $I_{max}$ = maximum level of inventory and
   $Cs$ = shortage cost.

Hence, the total incremental cost for one cycle, $t_1 + t_2$, of an inventory system which allows backorders is

$$Cp + (P \times FH) \frac{(I^2_{max})}{2R} + Cs \frac{(Q - I_{max})^2}{2R}$$

The annual total incremental cost is now obtained by multiplying the above equation through by the number of orders placed per year, $R/Q$:

$$TIC = \frac{R \times Cp}{Q} + (P \times FH) \frac{(I^2_{max})}{2Q} + Cs \frac{(Q - I_{max})^2}{2Q}$$

To determine optimal values for $Q$ and $I_{max}$, take the partial derivatives of the above equations with respect to $Q$ and $I_{max}$, equate to zero and obtain

$$Q = \sqrt{\frac{2RCp}{P \times FH}} \times \sqrt{\frac{(P \times FH) + Cs}{Cs}}$$

$$I_{max} = \sqrt{\frac{2RCp}{P \times FH}} \times \sqrt{\frac{Cs}{(P \times FH) + Cs}}$$

$$TIC = \sqrt{2(P \times FH)RCp} \times \sqrt{\frac{Cs}{(P \times FH) + Cs}}$$

However, if either $Q$ or $I_{max}$ is constrained, their respective values are obtained as follows:

1. When $Q$ is constrained, for example, fixed at price discount quantities, then $I_{max}$ is calculated as

$$I_{max} = \frac{CsQ}{CH + CS}$$

2. When $I_{max}$ is constrained, for example, limited by storage constraints (see next section), then $Q$ is calculated as

$$Q = \sqrt{\frac{2CpR + (P \times FH)I_{max}^2 + CsI_{max}^2}{Cs}}$$

**Sample Problem Three**

This problem is basically the same as Sample Problem One except that shortage costs ($Cs$) are included. In this problem $Cs$ = \$0.30.

$$Q = \sqrt{\frac{2(1600)\,5.00}{1.00\,(0.10)}} \times \sqrt{\frac{(100 \times 0.10) + 0.30}{0.30}} = 400(1.154)$$

$$= 461.88 \text{ units}$$

$$TIC = \sqrt{2(5.00)\,(1.00 \times .10)1600} \times \sqrt{\frac{0.30}{(1.00 \times 0.10) + 0.30}}$$

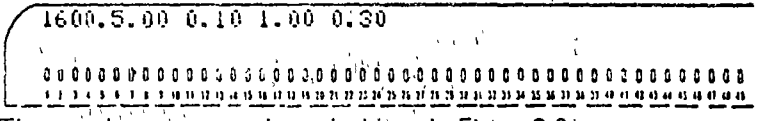
$$= 40(.866) = 34.64$$

$$TC = (1.00)\,(1600) + 34.64 = \$1634.64$$

The reader should note that the effect of including shortage costs is to increase the size of $Q$. This is because the annual inventory holding costs are smaller due to the smaller average inventory. In addition, the total incremental costs ($TIC$) are less than in the classical model because both holding costs and preparation costs are lower. (This may be verified by comparing the results of this analysis with those from Section 3.1.)

**Computer Input —Problem Three**

The data card for this example problem follows the same general form outlined previously. One additional data input is necessary. The shortage cost of \$0.30 is punched in columns 21-25.




The complete computer input is shown in Figure 3-9.

```
MAGGARD EOQ PROBLEM THREE, SHORTAGE COST
1600.5.00 0.10 1.00 0.30
```

Figure 3-9 Computer input--Problem Three.

The computer printout, using this input, is shown in Figure 3-10.

```
PROGRAM EOQ FOR MAGGARD EOQ PROBLEM THREE, SHORTAGE COST
INPUT DATA IS **********

     R     CP     FH     R1     CS     B1     P2     B2     P3      W
  1600   5.00    .10   1.00    .30      0      0      0      0      0

ANALYSIS RESULTS ARE ****
 OPTIMUM ORDER QUANTITY IS                              461.88
 WITH OPTIMUM INVENTORY OF                              346.41
 AT A PRICE PER ITEM OF                                   1.00
 YIELDING A TOTAL INVENTORY COST OF                    1634.64
 WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS             3.46
```

Figure 3-10 Computer output—Problem Three.

## 3.4 STORAGE LIMITATIONS

In this section, an additional constraint of maximum storage limitations, either in terms of available warehouse space or available capital, will be placed upon the basic EOQ model. This is intended to be an illustrative example of an additional type of constraint which may be (and in the real world is) imposed upon the basic EOQ model. By incorporating this added constraint the reader should get some additional insight into the problems that face management when determining what quantities to purchase from suppliers.

**Sample Problem Four**

This problem is basically the same as Sample Problem One except that the additional constraint of maximum warehouse space available ($W$) has been imposed.

In this problem, $W = 100$ units. From Sample Problem One, $Q = 400$ units, but since $W < Q$, the order quantity must be $Q = W = 100$ units.

In this problem, the cost of the limited storage constraint is $TC_{prob\,1} - TC_W$.

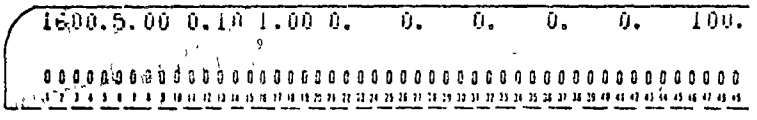
$$TC_W = \frac{100\,(1.00)\,(0.10)}{2} + \frac{1600(5.00)}{100} + 1600(1.00)$$

$$= 5 + 80 + 1600 = \$1685.00$$

$$\text{Cost}_W = TC_1 - TC_W = \$1640.00 - \$1685.00 = \$45.00$$

**Computer Input —Problem Four**

The data card of the above example problem follows the same form as outlined previously, with one addition. The maximum warehouse space available, expressed in units, is keypunched in columns 46-50. The data card looks like this:




The computer input for this example problem is shown in Figure 3-11.

```
HAGGARD EOQ PROBLEM FOUR, STORAGE LIMITS
1600.5.00 0.10 1.00 0.   0.   0.   0.   0.   100.
```

Figure 3-11 Computer input—Problem Four.

**Computer Output —Problem Four**

The computer output will indicate whether or not the warehouse constraint has had an effect on the economic order quantity. If an economic order quantity has been determined which exceeds $W$, the output will indicate that this has happened.

Furthermore, on the output will be a statement to the effect that, if the warehouse restrictions are operative, the order quantity determined may not be optimal. Suggestions are made for a method to determine the optimal quantity if this restraint is present. The computer output is shown in Figure 3-12.

```
PROGRAM EOQ FOR MAGGARD EOQ PROBLEM FOUR, STORAGE LIMITS
INPUT DATA IS **********
     R     CP     FH     P1     CS     B1     P2     B2     P3      W
  1600   5.00    .10   1.00      0      0      0      0      0    100

ANALYSIS RESULTS ARE ****
 BEFORE THE WAREHOUSE STORAGE LIMITATION IS APPLIED
 OPTIMUM ORDER QUANTITY IS                                  400.00
 AT A PRICE PER ITEM OF                                       1.00
 YIELDING A TOTAL INVENTORY COST OF                        1840.00
 WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS                 4.00

 THE ORDER QUANTITY IS LIMITED BY THE WAREHOUSE SPACE
 RESTRICTION AND IS NOT AT AN OPTIMUM. LOOSEN THE
 RESTRICTION AND RUN AGAIN OBSERVING THE EFFECT.

ANALYSIS RESULTS ARE ****
 AFTER THE WAREHOUSE STORAGE LIMITATION IS APPLIED
 OPTIMUM ORDER QUANTITY IS                                  100.00
 AT A PRICE PER ITEM OF                                       1.00
 YIELDING A TOTAL INVENTORY COST OF                        1685.00
 WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS                16.00
 THIS ORDER QUANTITY IS AT THE MAXIMUM WAREHOUSE CAPACITY
```

Figure 3-12 Computer output—Problem Four.

To conclude our discussion of economic order quantity models and, in particular this computer model, we would point out:

1. that the model is, in its present form, limited to only two price breaks.
2. the inclusion of shortage costs certainly complicates the storage limitation problem. In this model, when backorders and storage limitations are included in the same problem, the assumption is made that the backorders are instantaneously filled and that the storage limitation $W$ is a constraint upon $I_{max}$ and not upon $Q$. The user must remember that if $I_{max}$ is constrained by $W$ then neither $Q$ nor $I_{max}$ will be optimal.
3. this model will solve problems including one or all of the constraints previously described in a single problem. To appreciate this fact the reader may wish to solve the following problem manually and then by the use of the herein described EOQ model.

*Data*

| $R$ | $Cp$ | $FH$ | $P_1$ | $Cs$ | $B1$ | $P_2$ | $B2$ | $P_3$ | $W$ |
|---|---|---|---|---|---|---|---|---|---|
| 1600 | 5.00 | 0.10 | 1.00 | 0.30 | 300 | 0.90 | 500 | 0.80 | 350 |

## REFERENCES

Bowman, E. H., and R. B. Fetter, *Analysis for Production and Operations Management*, 3d ed., Homewood, Ill., Irwin, 1967.

Buffa, E. S., *Modern Production Management*, 3d ed., New York: Wiley, 1969.

———, *Operations Management: Problems and Models*, 2d ed., New York: Wiley, 1968.

———, *Production-Inventory Systems: Planning and Control*, Homewood, Ill.: Irwin, 1968.

Eilon, S., *Elements of Production Planning and Control*, New York: Macmillan, 1962.

Fabrycky, W. J., and P. E. Torgersen, *Operations Economy Industrial Applications of Operations Research*, Englewood Cliffs, N.J.: Prentice-Hall, 1966.

Garrett, L. J., and M. Silver, *Production Management Analysis*, New York: Harcourt, Brace & World, 1966.

Hadley, G., and T. M. Whitin, *Analysis of Inventory Systems*, Englewood Cliffs, N.J.: Prentice-Hall, 1963.

Hopeman, R. J., *Systems Analysis and Operations Management*, Columbus, Ohio: Merrill, 1969.

Magee, J. F., and D. M. Boodman, *Production Planning and Inventory Control*, 2d ed., New York: McGraw-Hill, 1967.

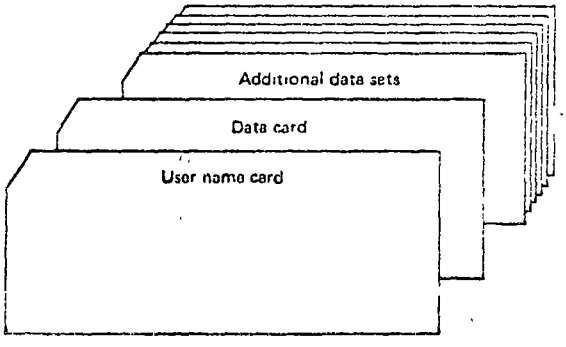
Naddor, E., *Inventory Systems*, New York: Wiley, 1966.

Olsen, R. A., *Manufacturing Management: A Quantitative Approach*, Scranton, Pa.: International Textbook Company, 1968.

Plossl, G. W., and O. W. Wright, *Production and Inventory Control*, Englewood Cliffs, N.J.: Prentice-Hall, 1967.

Riggs, J. L., *Production Systems: Planning, Analysis and Control*, New York, Wiley, 1970.

## 3.5 EOQ DATA DECK STRUCTURE




| Data Card | *Card column* | *Format* | *Item* |
|---|---|---|---|
| | 1-5 | F5.0 | annual usage requirement |
| | 6-10 | F5.0 | ordering cost |
| | 11-15 | F5.0 | holding cost |
| | 16-20 | F5.0 | unit price |
| | 21-25 | F5.0 | shortage cost |
| | 26-30 | F5.0 | minimum order quantity—first price discount |
| | 31-35 | F5.0 | unit price—first price discount |
| | 36-40 | F5.0 | minimum order quantity—second price discount |
| | 41-45 | F5.0 | unit price—second price discount |
| | 46-50 | F5.0 | maximum warehouse space available |

## 3.6 EOQ PROGRAM LISTING

```
C      PROGRAM EOQ                                                    IBM   1
C*   ECONOMIC ORDER QUANTITY MODEL                                      A   2
C*   COPYRIGHT ROY D HARRIS OCTOBER 1970                                A   3
C*   THIS VERSION FOR THE IBM 360                                     IBM   4
C*                                                                      A   5
C*   R = ANNUAL USAGE REQUIREMENT, LEVEL DEMAND                         A   6
C*   CP = COST OF ONE PURCHASE ORDER                                    A   7
C*   FH = HOLDING COST AS A PERCENTAGE OF UNIT PRICE                    A   8
C*   P(1) = PRICE OF EACH UNIT BEFORE DISCOUNT P(1)=P1                  A   9
C*   P(2) = PRICE OF EACH UNIT AT FIRST DISCOUNT BREAK POINT            A  10
C*   P(3) = PRICE OF EACH UNIT AFTER FIRST DISCOUNT P(3) = P2           A  11
C*   P(4) = PRICE OF EACH UNIT AT SECOND DISCOUNT BREAK POINT           1  12
C*   P(5) = PRICE OF EACH UNIT AFTER SECOND DISCOUNT P(5) = P3          A  13
C*   P(6) = PRICE OF EACH UNIT AT WAREHOUSE CAPACITY RESTRAINT          A  14
C*   EOQ = ECONOMIC ORDER QUANTITY                                      A  15
C*   B(1) = FIRST DISCOUNT BREAK POINT B(1) = B1                        A  16
C*   B(2) = SECOND DISCOUNT BREAK POINT B(2) = B2                       A  17
C*   CS = SHORTAGE COST                                                 A  18
C*   W = MAXIMUM WAREHOUSE SPACE AVAILABLE                              A  19
C*   TCST = TOTAL COST AT EOQ                                           A  20
C*   ON = NUMBER OF ORDERS PER YEAR AT EOQ                              A  21
C*   Q(1) = EOQ AT P(1)                                                 A  22
C*   Q(3) = EOQ AT P(3)                                                 A  23
C*   Q(5) = EOQ AT P(5)                                                 A  24
C*   Q(2) = B(1)                                                        A  25
C*   Q(4) = B(2)                                                        A  26
C*   Q(6)= W FOR CS=0  Q(6)=OPTIMUM ORDER AT ENV(6) = W FOR CS NOT 0    A  27
C*   TC(I) = TOTAL COCST                                                A  28
C*   TC(I) = TOTAL COST AT Q(I) AND ENV(I) FOR I = 1 TO 6               A  29
C*   ENV(I) = MOST ECONOMICAL INVENTORY LEVEL AT EOQ =Q(I)  I=1 TO 5    A  30
C*   ENV(6) = W                                                         A  31
C*   ENVT = OPTIMAL INVENTORY WHEN CS NOT 0                             A  32
C*                                                                      A  33
C**********************************************************************A  34
      DIMENSION P(6), Q(6), ENV(6), TC(6), B(3), ALPHA(10)              A  35
      MI = 5                                                           IBM  36
      MO = 6                                                           IBM  37
C     READ AND PRINT NAME CARD - - - - - - - - - - - - - - - - - - - - A  38
 1    READ (MI,71) ALPHA                                                A  39
      WRITE (MO,72) ALPHA                                               A  40
C     READ AND PRINT DATA CARD - - - - - - - - - - - - - - - - - - - - A  41
      READ (MI,73) R, CP, FH, P(1), CS, B(1), P(3), B(2), P(5), W       A  42
      WRITE (MO,74)                                                     A  43
      WRITE (MO,75)                                                     A  44
      WRITE (MO,76) R, CP, FH, P(1), CS, B(1), P(3), B(2), P(5), W      A  45
C     CHECK OUT THE DATA - - - - - - - - - - - - - - - - - - - - - - - A  46
      IF (R) 19,19,2                                                    A  47
 2    IF (CP) 19,19,3                                                   A  48
 3    IF (FH) 19,19,4                                                   A  49
 4    IF (P(1)) 19,19,5                                                 A  50
 5    IF (B(1)) 19,14,6                                                 A  51
 6    IF (P(3)) 19,19,7                                                 A  52
 7    IF (B(2)) 19,16,8                                                 A  53
 8    IF (B(2)-B(1)) 19,9,9                                             A  54
C     TWO PRICE BREAKS - - - - - - - - - - - - - - - - - - - - - - - - A  55
 9    NSEG = 3                                                          A  56
 10   IF (P(5)) 19,19,11                                                A  57
 11   IF (W) 19,12,13                                                   A  58
 12   W = 1.E25                                                         A  59
 13   B(3) = W                                                          A  60
      IF (CS) 19,20,20                                                  A  61
C     NO PRICE BREAKS - - - - - - - - - - - - - - - - - - - - - - - - A  62
```

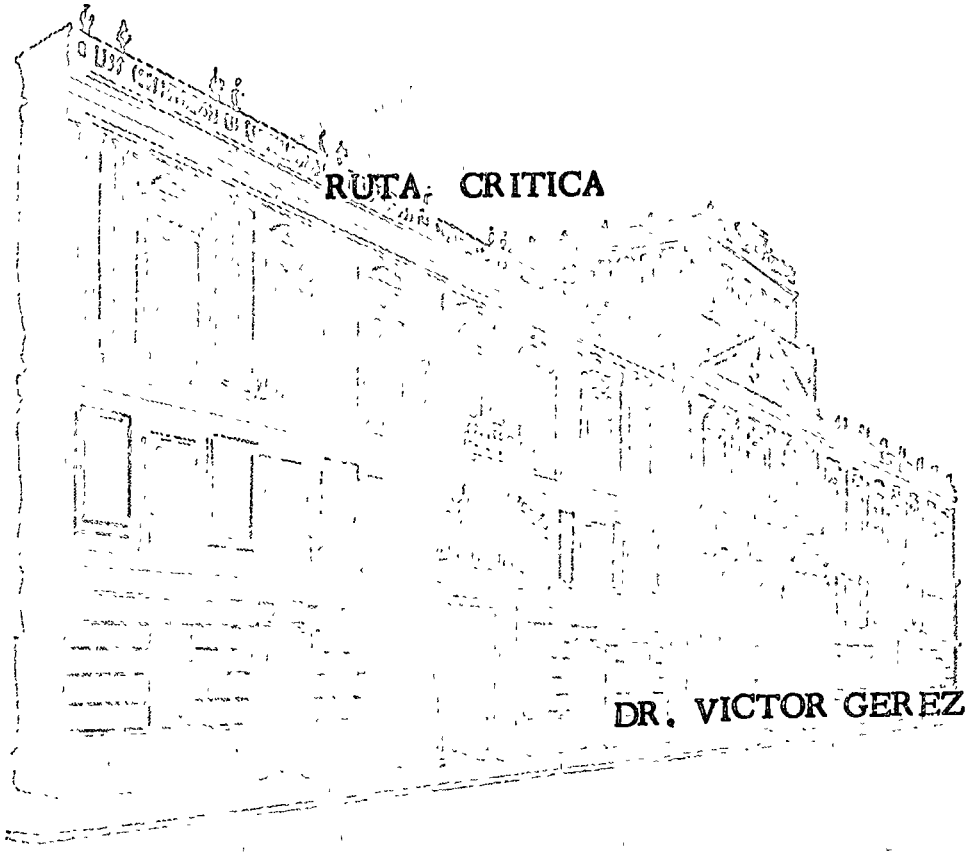
```
 14   NSEG = 1                                                          A  63
      B(1) = 1.E25                                                      A  64
      IF (P(3)) 19,15,19                                                A  65
 15   P(3) = P(1)                                                       A  66
      IF (B(2)) 19,17,19                                                A  67
C   ONE PRICE BREAK  - - - - - - - - - - - - - - - - - - - - - - - -    A  68
 16   NSEG = 2                                                          A  69
 17   B(2) = 1.E25                                                      A  70
      IF (P(5)) 19,18,19                                                A  71
 18   P(5) = P(3)                                                       A  72
      GO TO 10                                                          A  73
C   ERROR IN DATA  - - - - - - - - - - - - - - - - - - - - - - - - -    A  74
 19   WRITE (MO,77)                                                     A  75
      GO TO 1                                                           A  76
 20   P(4) = P(5)                                                       A  77
      P(2) = P(3)                                                       A  78
      IF (P(5)-P(3)) 22,22,21                                           A  79
 21   P(4) = P(3)                                                       A  80
 22   IF (P(3)-P(1)) 24,24,23                                           A  81
 23   P(2) = P(1)                                                       A  82
 24   P(6) = P(2)                                                       A  83
      IF (W-B(1)) 28,30,25                                              A  84
 25   IF (W-B(2)) 27,29,26                                              A  85
 26   P(6) = P(5)                                                       A  86
      GO TO 30                                                          A  87
 27   P(5) = P(3)                                                       A  88
      GO TO 30                                                          A  89
 28   P(6) = P(1)                                                       A  90
      GO TO 30                                                          A  91
 29   P(6) = P(4)                                                       A  92
 30   Q(2) = B(1)                                                       A  93
      Q(4) = B(2)                                                       A  94
      Q(6) = B(3)                                                       A  95
      IF (CS) 31,31,61                                                  A  96
C   SHORTAGES NOT ALLOWED  - - - - - - - - - - - - - - - - - - - - -    A  97
 31   DO 32 I = 1, 3                                                    A  98
      J = 2*I-1                                                         A  99
C   CALCULATE  Q(I)  I=1,3,5  - - - - - - - - - - - - - - - - - - -    A 100
 32   Q(J) = SQRT(2.*CP*R/(FH*P(J)))                                    A 101
      DO 35 I = 1, 6                                                    A 102
      IF (Q(I)-1.E25) 33,34,33                                          A 103
C   CALCULATE TC(I),  I=1 TO 6  - - - - - - - - - - - - - - - - - -    A 104
 33   TC(I) = (CP*R/Q(I)+P(I)*R+P(I)*F4*Q(I)/2.)                        A 105
      GO TO 35                                                          A 106
 34   TC(I) = 1.E25                                                     A 107
 35   CONTINUE                                                          A 108
      DO 36 I = 1, 6                                                    A 109
 36   ENV(I) = Q(I)                                                     A 110
C   TEST FOR FEASIBILITY WITH RESPECT TO THE PRICE BREAKS  - - - - - -  A 111
 37   IF (Q(1)-Q(2)) 39,39,38                                           A 112
 38   TC(1) = 1.E25                                                     A 113
 39   IF (Q(2)-Q(3)) 40,40,41                                           A 114
 40   IF (Q(3)-Q(4)) 42,42,41                                           A 115
 41   TC(3) = 1.E25                                                     A 116
 42   IF (Q(4)-Q(5)) 44,44,43                                           A 117
 43   TC(5) = 1.E25                                                     A 118
C   FIND EOQ WITH OUT STORAGE LIMITATIONS  - - - - - - - - - - - - -    A 119
 44   KFLG = 1                                                          A 120
      TCST = TC(1)                                                      A 121
      ECOQ = Q(1)                                                       A 122
      ENVT = ENV(1)                                                     A 123
      DO 46 I = 1, 5                                                    A 124
```

```
      IF (TCST-TC(I)) 46,46,45                                          A 125
 45   TCST = TC(I)                                                      A 126
      KFLB = I                                                          A 127
      ECOQ = Q(I)                                                       A 128
      ENVT = ENV(I)                                                     A 129
 46   CONTINUE                                                          A 130
      ON = R/ECOQ                                                       A 131
C     PRINT RESULTS BEFORE STORAGE RESTRICTIONS - - - - - - - - - - -   A 132
      WRITE (MO,70)                                                     A 133
      IF (W-1.E25) 47,48,47                                             A 134
 47   WRITE (MO,79)                                                     A 135
 48   WRITE (MO,80) ECOQ                                                A 136
      IF (CS) 49,5 ,49                                                  A 137
 49   WRITE (MO,90) ENVT                                                A 138
 50   WRITE (MO,81) P(KFLB)                                             A 139
      WRITE (MO,82) TCST                                                A 140
      WRITE (MO,83) ON                                                  A 141
      IF (W-1.E25) 51,1,51                                              A 142
C     TEST FOR FEASIBILITY WITH RESPECT TO THE STORAGE REQUIREMENT - - - A 143
 51   DO 53 I = 1, 5                                                    A 144
      IF (ENV(I)-W) 53,53,52                                            A 145
 52   TC(I) = 1.E25                                                     A 146
 53   CONTINUE                                                          A 147
      IF (TC(KFLB)-1.E25) 54,55,54                                      A 148
 54   WRITE (MO,84)                                                     A 149
      GO TO 1                                                           A 150
C     FIND EOQ WITH STORAGE LIMITATIONS - - - - - - - - - - - - - - -   A 151
 55   KFLB = 1                                                          A 152
      TCST = TC(1)                                                      A 153
      ECOQ= Q(1)                                                        A 154
      ENVT = ENV(1)                                                     A 155
      DO 57 I = 1,6                                                     A 156
      IF (TCST-TC(I)) 57,57,56                                          A 157
 56   TCST = TC(I)                                                      A 158
      ECOQ= Q(I)                                                        A 159
      KFLB = I                                                          A 160
      ENVT = ENV(I)                                                     A 161
 57   CONTINUE                                                          A 162
      ON = R/ECOQ                                                       A 163
C     PRINT RESULTS AFTER STORAGE LIMITATIONS - - - - - - - - - - - -   A 164
      WRITE (MO,85)                                                     A 165
      WRITE (MO,86)                                                     A 166
      WRITE (MO,87)                                                     A 167
      WRITE (MO,78)                                                     A 168
      WRITE (MO,88)                                                     A 169
      WRITE (MO,80) ECOQ                                                A 170
      IF (CS) 58,59,58                                                  A 171
 58   WRITE (MO,90) ENVT                                                A 172
 59   CONTINUE                                                          A 173
      WRITE (MO,81) P(KFLB)                                             A 174
      WRITE (MO,82) TCST                                                A 175
      WRITE (MO,83) ON                                                  A 176
      IF (KFLB-6) 1,60,1                                                A 177
 60   WRITE (MO,89)                                                     A 178
      GO TO 1                                                           A 179
C     SHORTAGES ALLOWED - - - - - - - - - - - - - - - - - - - - - - -   A 180
 61   ENV(6) = Q(6)                                                     A 181
      DO 62 I = 1, 5, 2                                                 A 182
      ENV(I) = SQRT(2.*CP*R*CS/(FH*P(I)*(FH*P(I)+CS)))                  A 183
 62   Q(I) = SQRT((2.*CP*R*(FH*P(I)+CS))/(FH*P(I)*CS))                  A 184
      DO 65 I = 1, 2                                                    A 185
      J = 2*I                                                           A 186
```

```
      IF (Q(J)-1.E25) 64,63,64                                          A 187
63    ENV(J) = 1.E25                                                    A 188
      GO TO 65                                                          A 189
64    ENV(J) = (Q(J)*CS)/(P(J)*FH+CS)                                   A 190
65    CONTINUE                                                          A 191
      IF (ENV(6)-1.E25) 66,67,66                                        A 192
66    Q(6) = SQRT((2.*CP*R*ENV(6)**2*(CS+P(6)*FH))/CS)                  A 193
67    DO 70 I = 1, 6                                                    A 194
      IF (Q(I)-1.E25) 68,69,68                                          A 195
68    TC(I) = (CP*R/Q(I)+FH*P(I)*ENV(I)**2/(2.*Q(I))+R*P(I)+(CS*(Q(I)-EN  A 196
     1V(I))**2)/(2.*Q(I)))                                              A 197
      GO TO 70                                                          A 198
69    TC(I) = 1.E25                                                     A 199
70    CONTINUE                                                          A 200
      GO TO 37                                                          A 201
C                                                                       A 202
C                       TABLE OF FORMATS                                A 203
C                                                                       A 204
71    FORMAT (20A4)                                                     A 205
72    FORMAT (17H1PROGRAM EOQ FOR  ,10A4)                               A 206
73    FORMAT (11F5.0)                                                   A 207
74    FORMAT (26H INPUT DATA IS **********  )                           A 208
75    FORMAT (60H      R    CP    FH    P1    CS    B1    P2    B2    P3   A 209
     1    *   )                                                         A 210
76    FORMAT (1X,F6.0,4F6.2,F6.0,F6.2,F6.0,F5.2,F6.0)                   A 211
77    FORMAT (44H  ERROR IN INPUT DATA, CHECK AND RUN AGAIN  )           A 212
78    FORMAT (26H0ANALYSIS RESULTS ARE ****)                            A 213
79    FORMAT (54H  BEFORE THE WAREHOUSE STORAGE LIMITATION IS APPLIED   A 214
     1)                                                                 A 215
80    FORMAT ( 27H  OPTIMUM ORDER QUANTITY IS,20X,F10.2)                A 216
81    FORMAT (25H  AT A PRICE PER ITEM OF ,22X,F10.2)                   A 217
82    FORMAT (37H  YIELDING A TOTAL INVENTORY COST OF ,10X,F10.2)       A 218
83    FORMAT (47H  WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS ,F10.2) A 219
84    FORMAT (51H  THE WAREHOUSE LIMITATION HAD NO EFFECT ON THE EOQ )  A 220
85    FORMAT (54H0 THE ORDER QUANTITY IS LIMITED BY THE WAREHOUSE SPACE) A 221
86    FORMAT (54H  RESTRICTION AND IS NOT AT AN OPTIMUM, LOOSEN THE     A 222
     1)                                                                 A 223
87    FORMAT (54H  RESTRICTION AND RUN AGAIN OBSERVING THE EFFECT.      A 224
     1)                                                                 A 225
88    FORMAT (60H  AFTER THE WAREHOUSE STORAGE LIMITATION IS APPLIED    A 226
     1                  )                                              A 227
89    FORMAT (60H  THIS ORDER QUANTITY IS AT THE MAXIMUM WAREHOUSE CAPAC A 228
     1ITY     )                                                         A 229
90    FORMAT  (28H  WITH OPTIMUM INVENTORY OF ,19X,F10.2)               A 230
      END                                                               A 231
```

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA



## RUTA CRITICA

DR. VICTOR GEREZ GREISER

Febrero-Marzo 1977

C O N T E N I D O

## OBJETIVO GENERAL

Los directivos de un proyecto adquirirán un enfoque - muy útil y preciso en cuanto corresponde a los métodos CPM y PERT, (CRITICAL PATH METHOD y PROGRAM EVALUATION AND REVIEW TECHNIQUE), para planificar y controlar proyectos complejos - de gran importancia, permitiéndoles comparar y evaluar de una manera rápida y eficaz los distintos programas de trabajo. Además de proporcionar los efectos de cada variación o retraso en los planes adoptados, y con ello identificar las operaciones que requieran cambios.

Nos permitimos acercar a todos los que directa o indirectamente tienen contacto con proyectos vitales de alguna -- Empresa, con el afán de aumentar su gran potencial en cuanto a la planificación, programación y control de los mismos.

Trataremos de dar respuesta a preguntas que siempre - surgen entre los directivos de un proyecto.

¿ Cómo identificar las actividades que se deben terminar de acuerdo a lo planeado ?.

¿ Si el proyecto compuesto se va a terminar de acuerdo también al programa ?.

¿ Cómo revisar los avances del proyecto conforme pasa el tiempo ?.

Rogamos que si nuestro trabajo hace nacer en ustedes - una inquietud de crítica siempre constructiva, uniremos nuestros esfuerzos para dar contestación a las interrogantes que presentaren.

## OBJETIVOS ESPECIFICOS

En suma el objetivo general antes expresado es la conjunción de los siguientes objetivos específicos:

1. - Informar sobre los antecedentes del CPM y PERT.

2. - Resaltar que la diferencia entre CPM y PERT no es subtancial solo de forma, formato y nombre.

3. - Valorar las ventajas que nos proporcionan estos métodos CPM y PERT.

4. - Ver que el diagrama de GANTT presenta serias dificultades para los fines de control en cualquier tipo de proyecto.

5. - Saber elaborar una tabla de secuencias necesarias para la coordinación de actividades dentro del proyecto.

6. - Diagramar la red lógica de actividades, apoyándose en tablas de secuencias.

7. - Interpretar la programación de un proyecto con ayuda de CPM y PERT.

8. - Calcular la solución de una red lógica de actividades por los métodos CPM y PERT.

9. - Llevar al éxito todas las etapas de un proyecto al utilizar métodos CPM y PERT.

10. - Mejorar la planeación, programación y control con ayuda de estas técnicas CPM y PERT.

## 1.- RESUMEN GENERAL DE LOS METODOS EXISTENTES PARA PLANEACION PROGRAMACION Y CONTROL DE PROYECTOS

### 1.1 Diagrama de Barras:

No es nada nuevo el saber utilizar el diagrama de barras, o diagrama de Gantt, para poder elaborar programas de trabajo y ejecutar un proyecto, ya que se forma como sigue:

a. - Se determinan cuales son las actividades importantes de un proyecto.

b. - Se asigna una estimación de tiempo para cada actividad.

c. - Se representa cada actividad por una recta horizontal acotada en tiempo.

d. - Se hace una lista de actividades por cada renglón y con un cierto orden de ejecución se colocan las barras según el tiempo efectivo.

e. - Se convierten los tiempos efectivos a una escala de fechas de calendario, y se hace coincidir el inicio del proyecto con esta escala de fechas calendario.

d. - Se ajustan las posiciones de las barras representativas de las actividades, tomando en cuenta las fechas no laborables ( días de descanso, festivos, vacaciones).

Un diagrama resultante, será el demostrado en la figura siguiente:

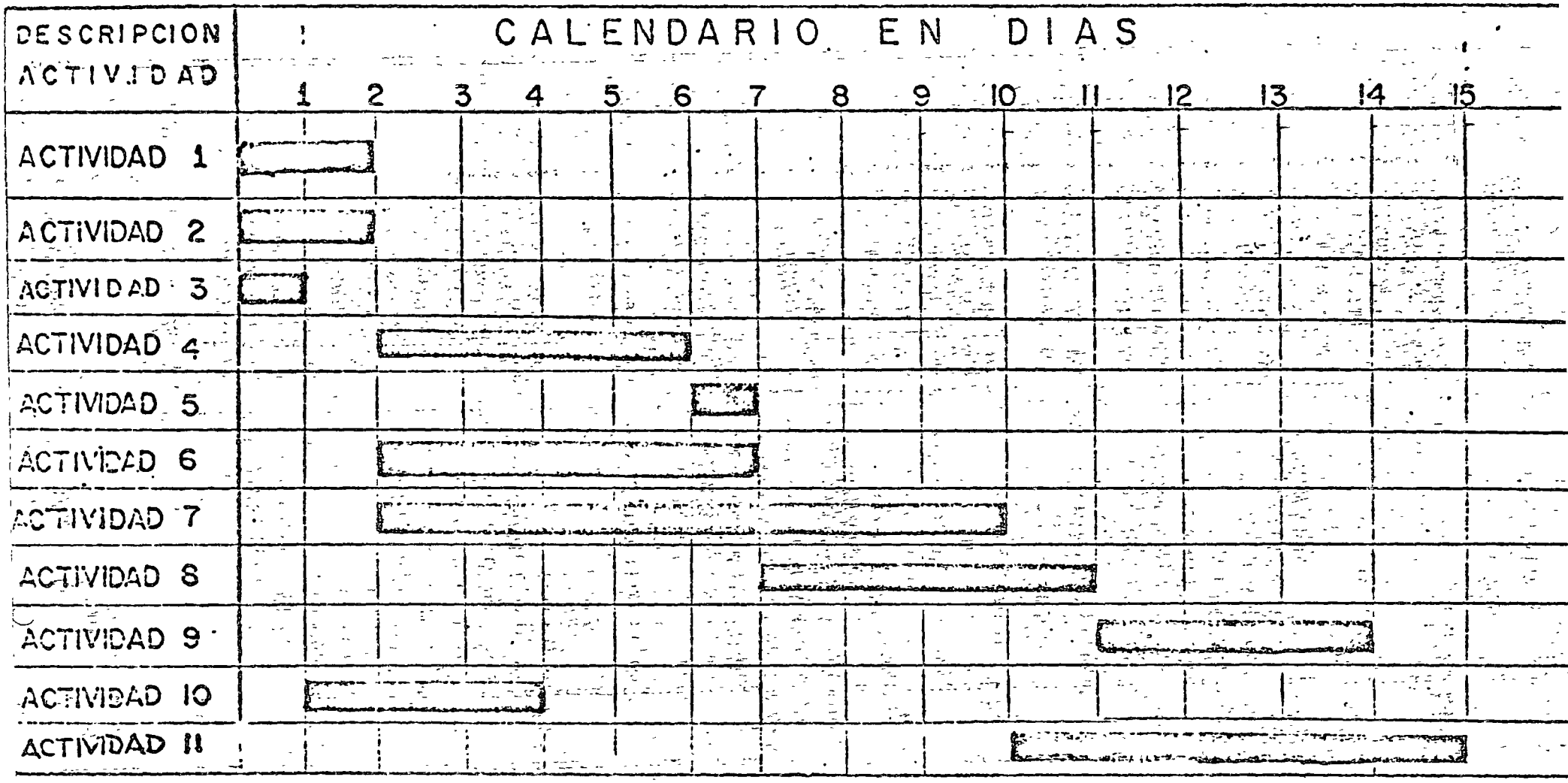



DIAGRAMA DE BARRAS

## 1.2 DEFICIENCIAS AL UTILIZAR UN DIAGRAMA DE GANTT, EN PLANEACION PROGRAMACION Y CONTROL DE PROYECTOS.

a.- Existen problemas para representar la coordinación lógica de actividades, acrecentándose aún más cuando el proyecto es complejo.
Al final de cuentas no es posible evaluar el progreso sin intervención contínua del personal principal.

b.- No se logra por este método diferenciar la planeación y programación, y no es posible ver claramente que actividades necesitan ser iniciadas al término de alguna actividad en cuestión.

c.- No se detectan facilmente cuales son las actividades que en realidad controlan la duración del proyecto, ya sea que aparentemente todas tienen la misma importancia, y por consiguiente si se llega a retrasar alguna principal, el proyecto sufre una descompensación con respecto a la duración de lo anteriormente programado sin poder predecir este tipo de efectos, tomándose a poco tiempo de iniciado un proyecto, medidas de aceleración del mismo, para compensar estas deficiencias.

d.- No es posible asegurar ninguna fecha de terminación de ciertas actividades ya que pueden ocurrir retrasos inevitables de condiciones de trabajo, clima, etc., provocando muy serios problemas.

e.- Es importante también hacer notar que el diagrama de Gantt, tampoco es recomendable para distribución de recursos (material, personal equipo, capital, etc.,) y programación de un proyecto.

## 1.3 VENTAJAS DEL DIAGRAMA DE GANTT, CUANDO MUESTRA LOS RESULTADOS DE CPM

Un diagrama de Gantt que represente a un proyecto con el auxilio de los métodos de programación, muestra objetivamente las duraciones, o sea las fechas de iniciación y de terminación posibles, y las holguras para cada actividad de que consta el proyecto, así como para determinar la distribución en el tiempo, de los recursos necesarios para el proyecto.

## 1.4 ANTECEDENTES DE LOS METODOS CPM y PERT ( METODOS DE PLANEACION, PROGRAMACION y CONTROL).

A últimas fechas se idearon dos métodos para la planeación, programación y control:

a.- Método de la ruta crítica (CPM), duración determinística.

b.- Método PERT, duración probabilística.

Para efectos de nuestro trabajo, hablaremos de CPM y PERT indistintamente.

El método de la ruta crítica fue desarrollado en los Estados Unidos, a principios de 1957, por el Sr. Morgan R. Walder, en ese entonces, miembro del Departamento de Ingeniería de la Compañía E. I. Dupont de Nemours & Co., y por el Sr. James E. Kelley, Jr., entonces investigador de la Compañía Remington Rand.

A partir de ello, el método CPM, lo utilizó la Compañía Dupont desde 1957, dedicándose a construir y modernizar plantas químicas con excelentes resultados en la etapas de planeación, programación y control. (ver referencias bibliográficas ).

En México, el CPM se ha utilizado en diversos organismos:

En 1961, en la Dirección General de Construcción de Edificios, y en la Secretaría de Obras Públicas, En 1962 en la Comisión Federal de Electrícidad y después en el Combinado Industrial Sahagún, y en otras grandes compañías constructoras del País.

El método PERT, fue desarrollado en los Estados Unidos en el año de 1958, por un grupo de investigadores de la Boos, Alien y Hamilton de Chicago, a solicitud de la "Special Projects Offices" de la Marina de los Estados Unidos.

Este método permitió acortar la duración del proyecto Polaris, en dos años.

## 1.5 BASES DE LOS METODOS DE PLANEACION PROGRAMACION Y CONTROL

Veremos a continuación en forma breve, los fundamentos de los métodos CPM y PERT, y los análisis que pueden efectuarse en ellos.

Sus Bases son:

a. - Permitir la diferenciación entre planeación y programación.

b. - Reconocer en la planeación:

   1) Actividades componentes del proyecto

   2) Coordinación de las actividades en orden lógico.

c. - Presentar un proyecto en diagrama de flechas.

d. - Asignar a las duraciones de cada actividad, en el Método PERT, tres tiempos: Más probable, Optimista, Pesimista, mediante los cuales se ajusta una distribución conveniente de probabilidad para la duración de la actividad.

e. - Dar información para hacer un análisis, en relación a cuanto se aminora el costo de una actividad si reducimos su duración.

f. - Proporcionar datos para analizar los recursos requeridos, para cada duración posible de cada actividad.

g. - Apoyarse en métodos como la programación lineal.

h. - Para el método PERT, se auxilia en métodos estadísticos.

## 1.6 ANALISIS BASICOS DE LOS METODOS CPM y PERT

Al tener la presentación de un proyecto, por medio de un diagrama de flechas, se procede a la programación o al análisis de tiempos.

En el diagrama la longitud de cada flecha es:

a. - En el Método CPM, la duración de la actividad.

b. - En el Método PERT, la duración probable de la actividad correspondiente.

Con base a estas longitudes, se consigue la duración de la ruta mas larga dándonos la mínima duración del proyecto, proporcionándonos así, una ruta crítica y las actividades que son excluyentes de las anteriores se consideran tener holguras; las cuales son importantes para programación de recursos, siempre y cuando no se consuman duraciones mayores de las permitidas y retrasen el proyecto.

En el método PERT, además es posible determinar las probabilidades de que se pueda terminar un determinado grupo de actividades, del proyecto en conjunto a un determinado tiempo.

## 1.7 GRAFICA Y ANALISIS QUE SE PUEDEN HACER HABIENDO UTILIZADO LOS METODOS CPM y PERT

Habiendo utilizado en un proyecto, los métodos de CPM y PERT, es posible elaborar diagramas de Gantt, que nos represen

te todas y cada una de las actividades con holgura respectiva decreciente, hasta llegar a holgura cero, siendo ésta la actividad crítica de un proyecto. Las holguras son parámetros importantes en cuanto a elaborar con ellos gráficas tipo recursos requeridos vs. tiempo, evaluando el exceso o falta de ciertos recursos para poderlos distribuir óptimamente a través de todo el proyecto.

Si la ruta crítica de un proyecto, da fechas mayores a la deseada, se puede recurrir a métodos de programación lineal, optimizando para la actividad en cuestión un costo mínimo a menor duración, lo mismo logrando reunir estos parámetros para las actividades de un proyecto, podemos decidir alcanzar el objetivo a un mínimo costo y a un mínimo tiempo.

## 1.8 APLICACION DEL CPM y PERT AL CONTROL DE EJECUCION DE UN PROYECTO

Los métodos CPM y PERT, permiten determinar las actividades críticas y las que tienen holguras pequeñas. Si el proyecto en cuestión sufre retrasos, en alguna actividad crítica, estas técnicas nos proporcionan información sobre el nuevo estado del proyecto.

Queda al Consejo Directivo decidir el comprimir la red, o dada la imposibilidad de hacerlo, llevar a cabo un estricto control de la nueva ruta crítica y de las actividades con pequeña holgura.

También la información permite la asignación óptima de recursos, conforme a los progresos alcanzados por el proyecto.

## 1.9 VENTAJAS DE LOS METODOS CPM y PERT

a. - Desglosar un proyecto en todas sus actividades componentes, el poder clasificar en orden de importancia y organizar la planeación, programación y control de ejecución del mismo, bajo esas mismas reglas.

b. - Coordinar de una manera eficiente a todos los organismos involucrados en el proyecto en las etapas de planeación, programación y control de ejecución del proyecto.

c. - Utilizar la experiencia de un grupo directivo de distintos organismos responsables y elaborar en conjunto un proyecto maestro que enfoque todas las actividades del mismo.

d. - Determinar cuales son las actividades del proyecto que controlan la duración (actividades criticas), y las holguras o márgenes de tiempo disponibles para retrasar la terminación de las otras actividades, sin retrasar la terminación del proyecto.

e. - Determinar de antemano y con toda precisión los recursos (materiales, personal, equipo capital, etc.), necesarios en cualquier tiempo durante la ejecución del proyecto.

f. - Comparar planes y programas alternativos para el mismo proyecto, o para una de sus partes, y ajustarse a las condiciones propias de la empresa en cuestión.

g. - Analizar los efectos de cualquier situación imprevista y de tomar las medidas correctivas eficientes.

h. - Permitir que el personal directivo de un proyecto sólo tenga que intervenir cuando ocurra alguna situación imprevista.

i. - Permitir el delegar responsabilidad de los diferentes organismos encargados de un proyecto o algunas de sus partes.

j. - Poder sustituir personal directivo en cualquier momento, sin trastornar la ejecución del proyecto o de una parte del mismo.

k. - Encauzar la experiencia adquirida en la ejecución de proyectos productivos similares, por lo tanto, la elaboración de planes standard.

1. - Comparar ordenadamente los datos estimados con los valores de ejecución y determinar el efecto de las desviaciones.

## 1.10 APLICACION DE COMPUTADORAS ELECTRONICAS EN LOS METODOS CPM y PERT.

Calcular los métodos CPM y PERT se puede hacer a mano, sin embargo, por la magnitud de los proyecto se hace imprescindible la ayuda del computador.

Tenemos a nuestro alcance computadoras de firmas como Burroughs, IBM, CDC, BULL, UNIVAC, que proporcionan todos los cálculos para la base de programación, y expeditar programas de costos.

## 2 TABLAS DE SECUENCIA

Al iniciar la planeación de un proyecto deberemos tomar en consideración sus diversas fases para lograr una base sólida en la aplicación de los métodos de programación CPM y PERT.

Sin embargo antes de iniciar la primera fase que será la de enumerar todas las actividades por orden de importancia, debemos de cumplir con tres reglas básicas:

a.- Debemos de tener colaboradores experimentados y con amplios conocimientos en la parte específica del proyecto, que les corresponde.

b.- Se requiere además información sobre los recursos disponibles como los humanos, económicos, equipo, espacio para la realización del proyecto.

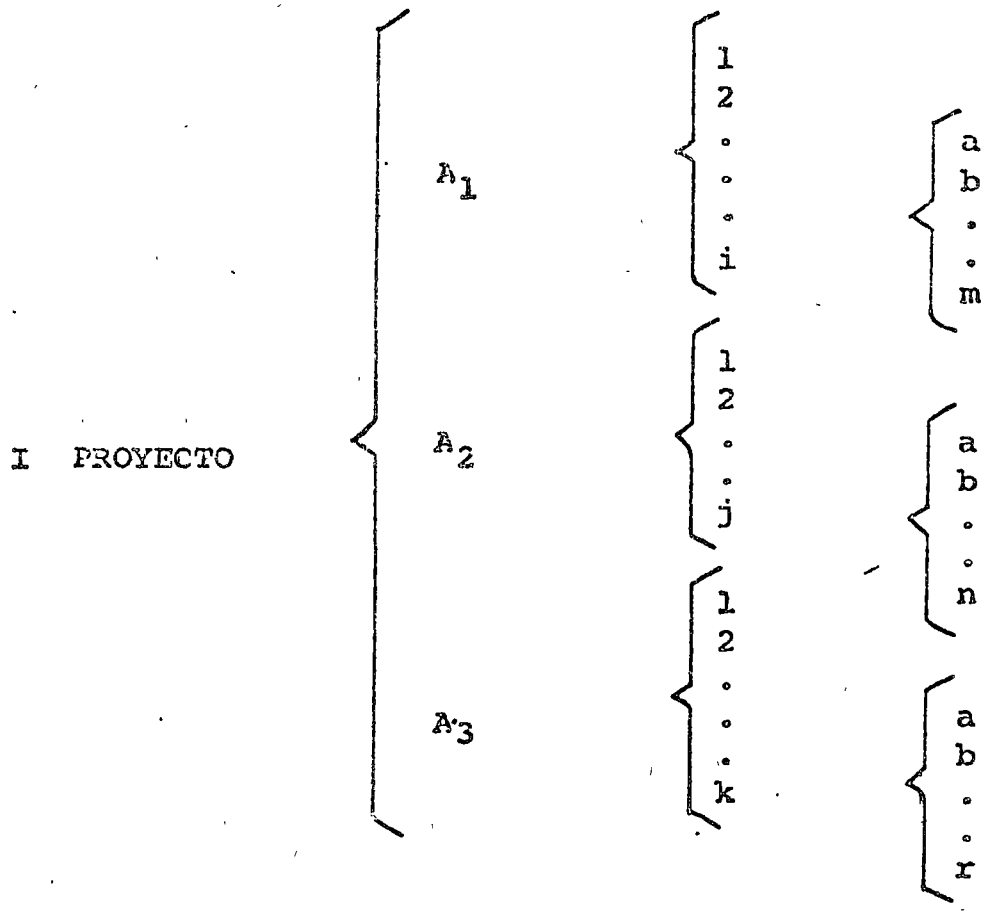
c.- Hay que tomar en consideración fechas claves para el cumplimiento de determinadas actividades y aunado al medio ambiente que influye en forma importante, en el desarrollo del proyecto.

### 2.1 PRIMERA FASE EN LA PLANEACION DE UN PROYECTO.

La primera fase es la elaboración de una lista de actividades componentes de un proyecto en actividades de primer orden o principales, y subdividir cada una en actividades de segundo orden y continuar asi sucesivamente.

Esta división mencionada anteriormente se puede representar en la siguiente ilustración:

| Número de Orden de las Actividades. | 1° | 2° | 3° |
|---|---|---|---|




## 2.2 SEGUNDA FASE EN LA PLANEACION DE UN PROYECTO

En esta fase se especifican el orden de secuencia de ejecución de actividades del proyecto, para lo cual se toman en cuenta los requisitos del proyecto, ya sean condiciones necesarias de una persona o empresa.

Para cumplir esta fase de planeación es recomendable preparar una tabla de secuencias.

La tabla de secuencias es una matriz cuadrada donde se describen todas las actividades en los renglones y columnas, de manera que a cada actividad renglón le corresponde una actividad columna.

Se siguen dos reglas para formar una tabla de secuencias:

a. - Se analiza cada actividad correspondiente al renglón en turno y se determinan cuales actividades se pueden hacer inmediatamente después, colocando una cruz en el casillero correspondiente.

b. - Se analizan las actividades columna y se determinan cuales actividades se pueden hacer inmediatamente antes de dichas actividades, colocando una cruz en el casillero correspondiente.

La aplicación de las reglas anteriores se pueden hacer en cualquier orden, una vez determinada la tabla de secuencias debe ser revisada una y otra vez para mejorar la planeación del proyecto.

Esta tabla de secuencia es esencial para la ejecución de un proyecto mas no forma parte del método CPM y PERT. Solo es una investigación de objetivos, métodos y elementos disponibles.

Toda esta etapa nos aclara si nuestro proyecto satisface nuestros objetivos y si es costeable su realización. Cuando dispongamos de un conocimiento de redes aunado a la tabla de secuencias, entonces podemos elaborar el diagrama de red de un proyecto en particular.

## 3 CREACION DE LA RED DE UN PROYECTO

El primer paso para utilizar los métodos de ruta crítica es la identificación de todas las actividades contenidas en el proyecto, y la representación de estas actividades por medio de un diagrama de flechas.

Este paso es usualmente llamado "fase de planeación". Aquí nos limitaremos a indicar las reglas básicas para hacer un primer dibujo de una red.

Existen ciertas reglas y convencionalismos que deberán seguirse en preparar redes, éstas nos mostrarán un alto grado de conocimiento sobre el proyecto y todos los juicios lógicos que hay que conservar. Por supuesto las reglas lejos de ser rígidas son muy flexibles dependiendo del usuario, sobre el - conocimiento de los conceptos y su experiencia en métodos de CPM y PERT.

Hay varias formas de dibujar una red. Aquí enfatiza-- remos el convencionalismo existente en el ramo industrial como la construcción de usar en métodos de ruta crítica, el sistema de actividad en una flecha.

## 3.1 TERMINOS BASICOS

Varios de los más comunes términos en trabajos de redes se definen a continuación:

Definición:

Una actividad es una porción de un proyecto que esta conforme a los siguientes indicadores: Esta no puede comenzar a menos que sus predecesoras en orden lógico sean terminadas.

Las actividades siempre tienen un principio y un fin y pueden estar asociados a las mismas, tiempos, recursos del proyecto. Las actividades se representan graficamente por - flechas acompañada de la descripción y el tiempo estimado de la misma.

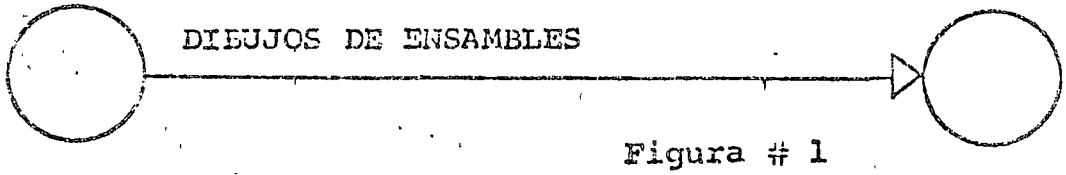



Figura # 1

Definición:

Una flecha que sólo indica una dependencia de una actividad con otra, es una actividad ficticia. Una ficticia tie-

no duración de cero y se representa tan comunmente como una flecha de línea interrumpida (Figura # 2), o una flecha sólida asociada con cero duración (figura # 3).

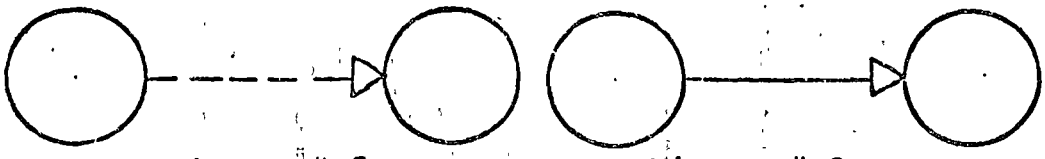


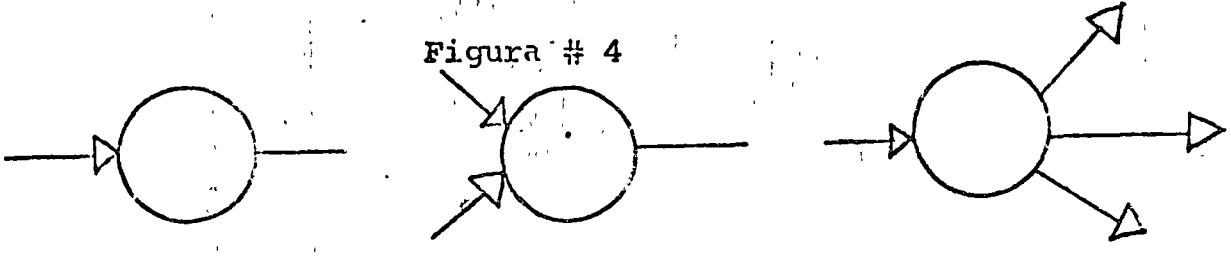
Figura # 2 Figura # 3

Definición:

Los puntos iniciales y los finales de las actividades son llamados EVENTOS.

Teóricamente los eventos son puntos instántaneos en el tiempo. Hay sinónimos como Nodos y Conectores. Si un evento representa la llegada final de más de una actividad, éste es llamado evento Receptor. Si un evento representa el punto de partida de varias actividades se llama evento Radiante. Un evento se presenta a menudo como una figura geométrica como está a continuación en la figura # 4.

Figura # 4



EVENTO EVENTO RECEPTOR EVENTO RADIANTE

Definición:

Una red es una representación gráfica de la planeación de un proyecto, mostrando las interrelaciones de varias actividades. Las redes pueden ser llamadas "Diagramas de Flechas". Figura # 5 cuando los resultados de los tiempos estimados y computados han sido agregados a la red, se puede usar como para programar un proyecto.

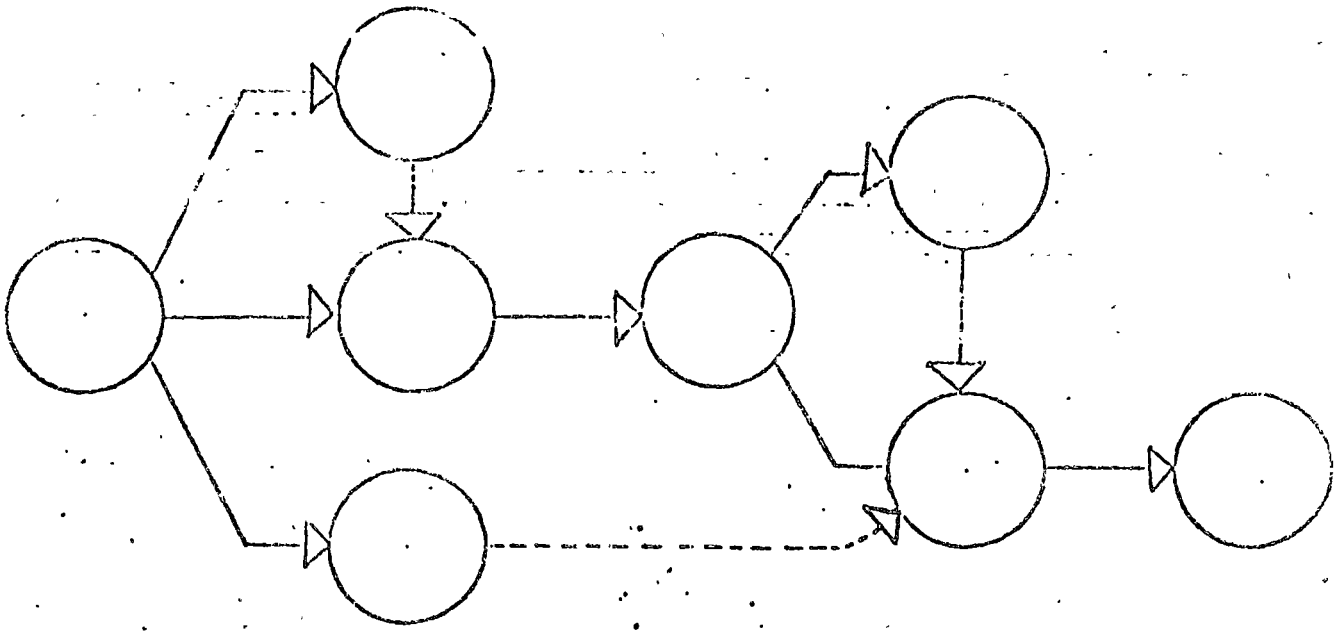


Figura # 5

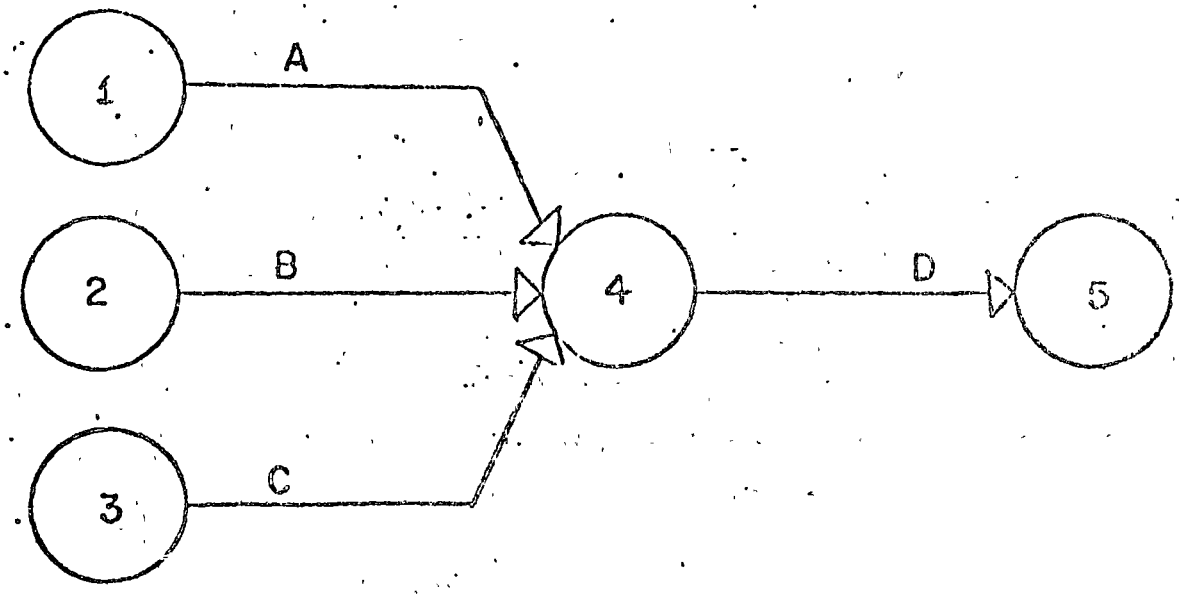



Figura # 6

## 4 REGLAS DE LA RED

Las pocas reglas para redes pueden ser clasificadas como todas comunes a todas las actividades de un diagrama de flechas, y estas reglas son impuestas para el uso de computadoras que manejen los métodos CPM y PERT.

### 4.1 REGLAS BASICAS DE UNA RED LOGICA

Regla 1.- Antes de iniciar una actividad, todas las actividades precedentes deben de terminarse.

Regla 2.- Las flechas sólo implican una precedencia lógica. La longitud de la flecha no tiene ningún significado. ( A menos que se utilice la escala tiempo para la red).

### 4.2 REGLAS IMPUESTAS POR LOS COMPUTADORES O METODOS DE CALCULO

Regla 3.- Dos eventos pueden directamente conectar una sola actividad.

Regla 4.- Los números de los eventos no deben estar duplicados.

Reglas.- Las Redes deben tener solo un evento inicial que no tenga predecesor), solo un evento final ( que no tenga sucesor).
Existen paquetes de computadora que no tienen esta restricción.

### 4.3 INTERPRETACION DE LAS REGLAS

La regla 1 y 2 pueden ser interpretadas si conocemos la porción de una red de la figura # 6.

De acuerdo a la regla 1, este diagrama establece que antes de que la actividad D pueda iniciarse, las actividades A, B, y C deben ser completadas.

Note que esta no implica que las actividades A, B y C sean completamente simultáneas.

Note que además que el evento 4 es un evento receptor por que ahí terminan las actividades A, B y C y comienza la actividad D.

## 4.4 ERRORES COMUNES

Los errores más comunes que se presentan son al no respetar la Regla 1.

Nos ilustraremos con la misma figura 6, supongamos que la actividad B, depende de haber terminado la actividad B y C de solo una parte de la actividad A y completar la segunda parte de A totalmente independiente. El diagrama que ilustra correctamente esta situación es:

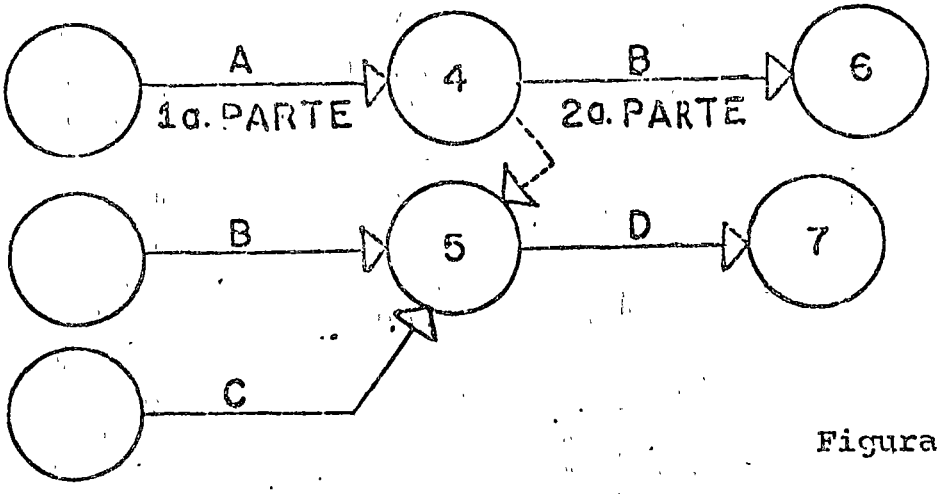



Figura # 7

Donde la actividad se divide en dos actividades A 1a. parte y A 2a. parte además de introducir una actividad ficticia, como veremos en la figura # 7, la actividad ficticia se ha usado para corregir el problema.

Otra condición se puede presentar en una red, se ilustra en la figura # 8.

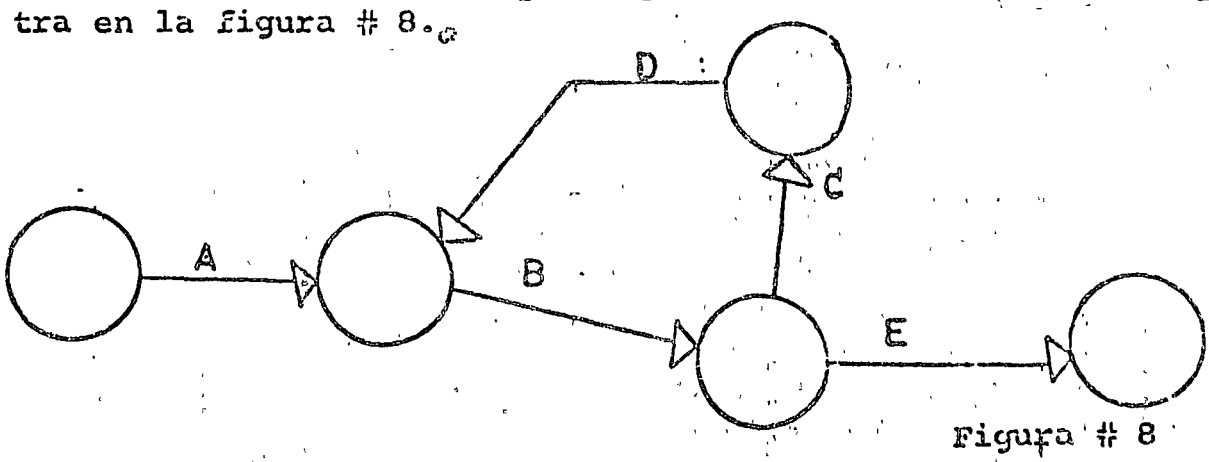



Figura # 8

Las actividades B, C, D, forman un Loop, el cual es - la indicación de un error en la lógica de la red.

En la figura 8 la actividad B no puede iniciarse hasta no completar la actividad D asimismo la D no se inicia al no terminarse la C y la C no principia porque no se ha terminado la B y caemos en un circuito cerrado que impide la lógica de la red.

## 4.5 REGLAS PARA INTRODUCIR EL PROBLEMA A UNA COMPUTADORA

Las reglas para redes la 3, 4 y 5 son los procedimientos para codificar redes para el análisis por computadora. La regla 3 se viola cuando ocurre lo demostrado en la figura # 9.

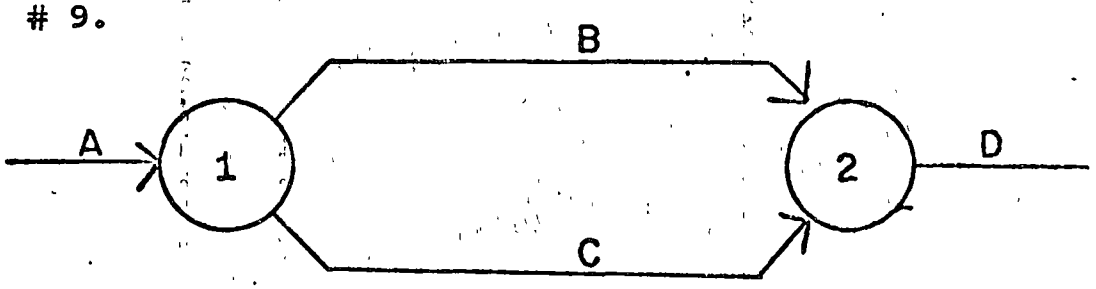



Figura # 9

Las actividades B y C puede llamarse actividades repetidas; como la forma de distinguir las actividades son - sus nodos

Quedará así:

| Código Computadora | Descripción Actividad |
|---|---|
| 1 - 2 | Actividad B |
| 1 - 2 | Actividad C |

Enconces tenemos que recurrir a las actividades ficticias para no caer en error de la regla 3, como se demuestra en la figura # 10.

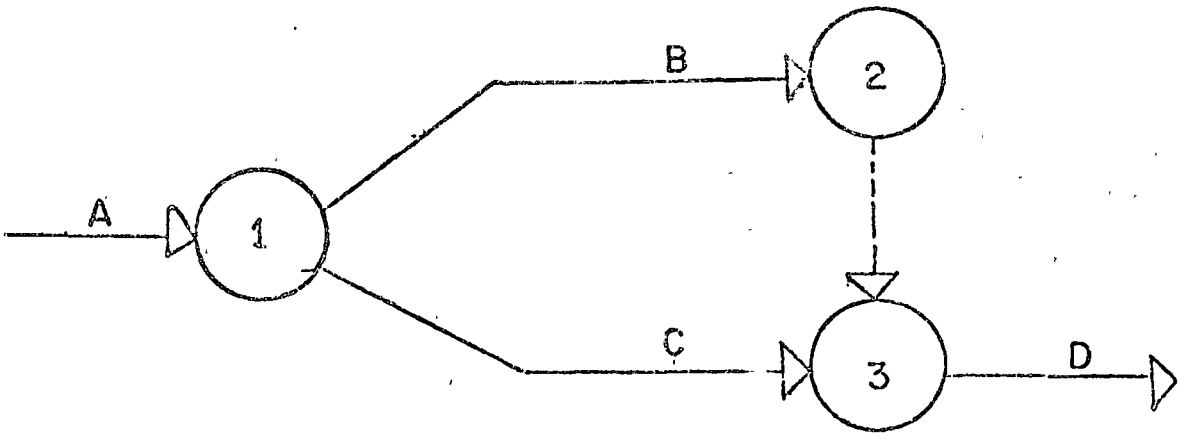



Figura # 10

Ahora si se pueden distinguir las dos actividades al ser bautizadas de la siguiente manera:

| Código Computadora | Descripción Actividad |
|---|---|
| 1 - 2 | Actividad B |
| 1 - 3 | Actividad C |
| 2 - 3 | Actividad Ficticia |

## 4.6 EL USO DE LAS ACTIVIDADES FICTICIAS

Algunas veces se emplean actividades ficticias en - forma redundante como en la figura # 11.

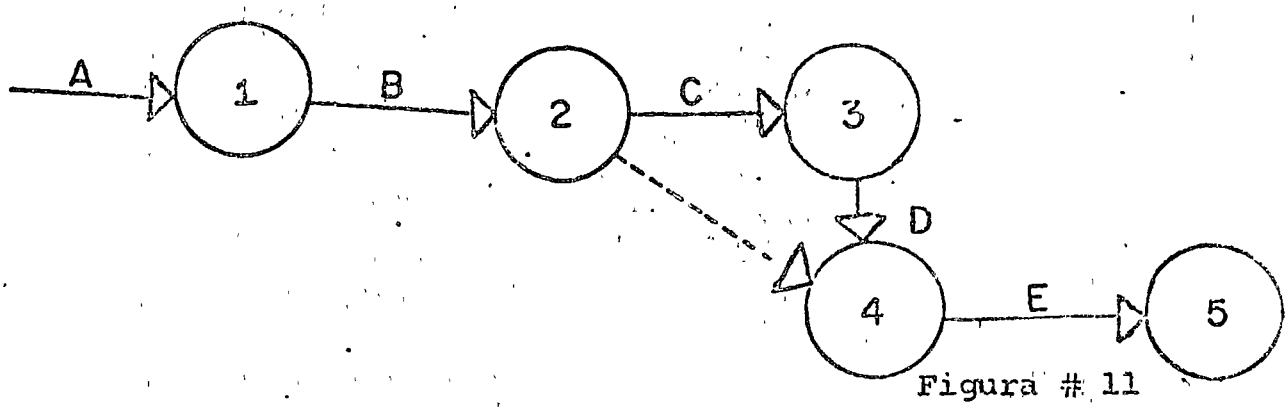



Figura # 11

Donde evidentemente para iniciar la actividad E es solo si se ha concluido la actividad D, C y B por lo tanto remarcar que la actividad B es necesaria para iniciar la actividad E, viene siendo redundante.

Otro caso donde las actividades ficticias no son necesarias es la figura 12.

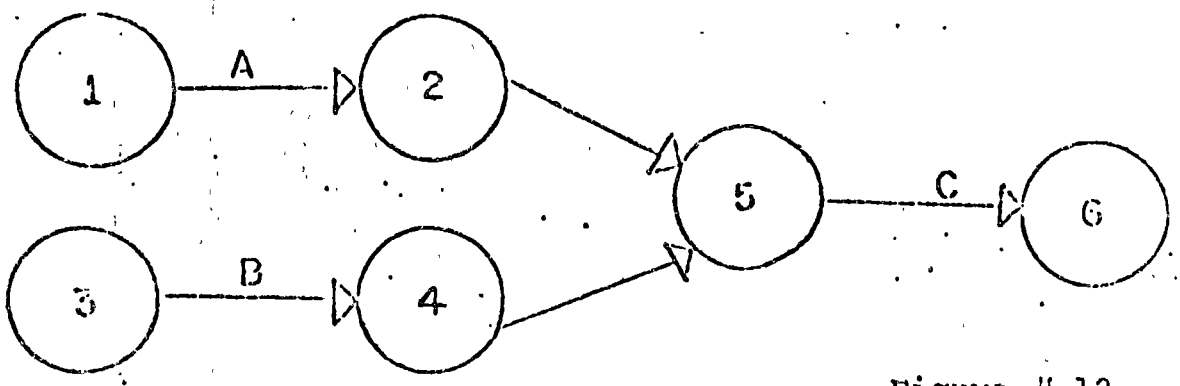



Figura # 12

En donde las actividades 2-5 y 4-5 que al ser ficticias no tienen duración si podemos prescindir de ellas y modificamos la figura 12 a la figura 13 siguiente:

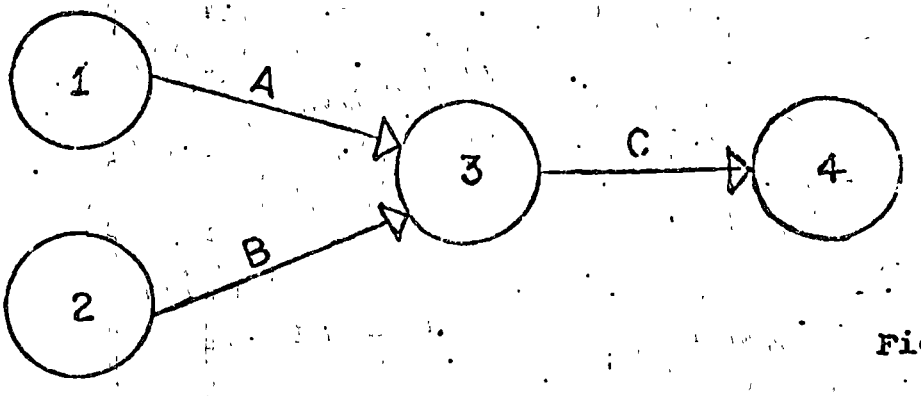



Figura # 13

Otro uso importante de las actividades ficticias esta ilustrado en este ejemplo. Supongamos en una parte de un proyecto de la figura 14 este se realiza

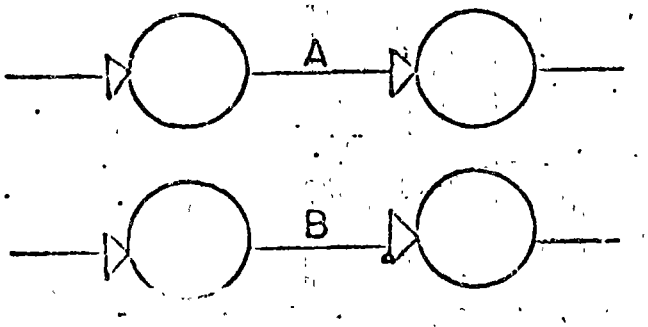



Figura # 14

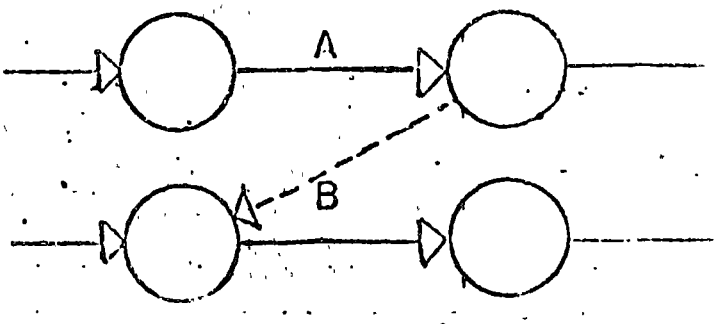



Figura # 15

que la actividad B depende de la actividad A por tener en común el mismo recurso, ya sea una persona en especial o una máquina específica. En este caso B debe dibujarse como dependiente de A por el uso de una actividad ficticia figura 15.

## 5 CALCULOS BASICOS PARA LA PROGRAMACION

Con base a los fundamentos para diseñar la red de un proyecto queda por calcular los tiempos aproximados que se asignarán a cada una de las actividades; y así comenzar a programarlas y calcular la ruta crítica de la red: en el caso de utilizar el método CPM los tiempos son determinísticos o sea una sola duración por cada actividad; en el método PERT los tiempos son probabilísticos en donde se asocian tres para cada una de las actividades, salvo esta diferencia entre los métodos CPM y PERT todos los cálculos que describiremos aquí son idénticos para ambos.

La programación de un proyecto comprende dos pasos básicos, el primero que hace un cálculo hacia adelante y el segundo es el cálculo hacia atrás en la red.

Basándose en un tiempo de ocurrencia del nodo inicial de red, el cálculo hacia adelante de la fecha de inicio más temprano y tardío para cada actividad, e indirectamente el inicio más temprano y más tardio para cada uno de los eventos.

Los tiempos reales, se conocen sólo después de haber concluído varias actividades y poder comparar contra los tiempos esperados, pueden diferir por la desviaciones entre los tiempos reales y estimados de los tiempos planeados, para cada actividad.

Por la especificación de los tiempos de ocurrencia terminales para los eventos de una red el cálculo hacia atrás nos calculará la terminación más temprana y tardía para cada actividad e indirectamente la terminación permitida de tiempo para cada evento. Después de un cálculo hacia adelante y hacia atrás se ha realizado, queda por hacer el cálculo de las holguras para cada actividad y determinar los arcos críticos de una red. Cuando una actividad tiene holgura, hay más tiempo disponible para hacer lo que esta actividad desea.

Casi siempre se adopta la unidad de trabajo de un día. Es muy conveniente planear los tiempos de las actividades en días de trabajo, en las redes comenzando con un

tiempo inicial de cero en el evento inicial del proyecto. La conversión de estos cálculos a fechas calendario requiere de hacer un calendario que contemple sólo los días de trabajo numerados. En suma se comprende que el proyecto tiene solamente un evento inicial y terminal y que la terminación permitida para el proyecto es igual a la iniciación más tardía obtenida de los cálculos hacia atrás. Estas consideraciones, se asocian normalmente con el método CPM, pero no usualmente para el método PERT. Finalmente, se toma en consideración que un diagrama de flechas para representar un proyecto es el que se usa para iniciar con los cálculos mencionados.

## 5.1 NOMENCLATURA

La siguiente nomenclatura se deberá usar en las fórmulas que describen los cálculos para la programación:

$N$ = Conjunto de todos los eventos de un proyecto.

$N_i$ = Nodo que representa el evento I $i = 1 \ldots m$. Llamado también Nodo Inicial.

$N_j$ = Nodo que representa el evento j $j = 1 \ldots n$ Llamado también Nodo Final.

$\Lambda$ = Conjunto de todas las actividades de un proyecto

$A_{ij}$ = Es la actividad que se inicia en el Nodo Inicial i y termina en el Nodo Final j.

$t_{ij}$ = Duración de la actividad del $N_i$ al $N_j$.

$\bar{11}k$ = K-esima Cadena que conduce del evento inicial $N_i$ al evento final $N_j$.

$k$ = Número posible de cadena que conectan el evento inicial $N_i$ al evento final $N_j$.

$t(\bar{11}_k)$ = Duración total de la cadena $\bar{11}_k$.

$$t\ (\bar{11}_k) = \sum_{A_{ij} e\ \bar{11}_k} t_{ij}$$

$E_i$ = Tiempo de ocurrencia más temprana para el evento inicial i.

$L_i$ = Tiempo de ocurrencia más tardía del evento inicial i

$ES_{ij}$ = Tiempo de iniciación más temprana para la actividad (i-j).

$LF_{ij}$ = Tiempo de terminación más tardía para la actividad(i-j).

$EF_{ij}$ = Tiempo de terminación más temprana de una actividad i-j.

$LS_{ij}$ = Tiempo de iniciación más tardía para la actividad (i-j).

$S_{ij}$ = Holgura total para la actividad i-j.

$FS_{ij}$ = Holgura libre para la actividad i-j.

$T_s$ = Tiempo programado para la terminación de un proyecto o la ocurrencia de eventos claves en un proyecto.

### CÁLCULO HACIA ADELANTE

Como se vio anteriormente el cálculo hacia adelante es el cálculo de la iniciación más temprana y más tardía para cada actividad en el proyecto apoyado en un día de trabajo específico. Complementando esto, el cálculo hacia adelante se inició en un tiempo cero dada una fecha base y todas las actividades subsecuentes empiezan lo más pronto posible, aconteciendo todos su eventos sucesores. De acuerdo a la lógica de una red, un evento final ocurre cuando todas las actividades predecesoras se han terminado y entonces, el tiempo más temprano para que ocurra un evento es igual almayor tiempo de iniciación tardía de todas las actividades que lleguen al evento en cuestión. Estas consideraciones o reglas están sumarizadas abajo.

## 5.3 REGLAS PARA EL CALCULO HACIA ADELANTE

1. - El tiempo de ocurrencia del primer evento inicial de la red se considera cero. $E_1=0$ para el primer evento.

2. - Cada actividad comienza tan pronto como sea posible hasta que ocurra su evento predecesor.

   Para una actividad arbitraria (i-j) se puede escribir:

   $ES_{ij}$ = El mayor EF de las actividades inmediatamente precedentes de la actividad (i-j).

3. - El tiempo de terminación más temprana de una actividad es la suma de su tiempo de iniciación más temprana y la duración estimada por actividad. Para una actividad arbitraria (i-j) esta se puede escribir como:

   $E_j$ = Al mayor de $EF_{ij} = ES_{ij} + T_{ij}$

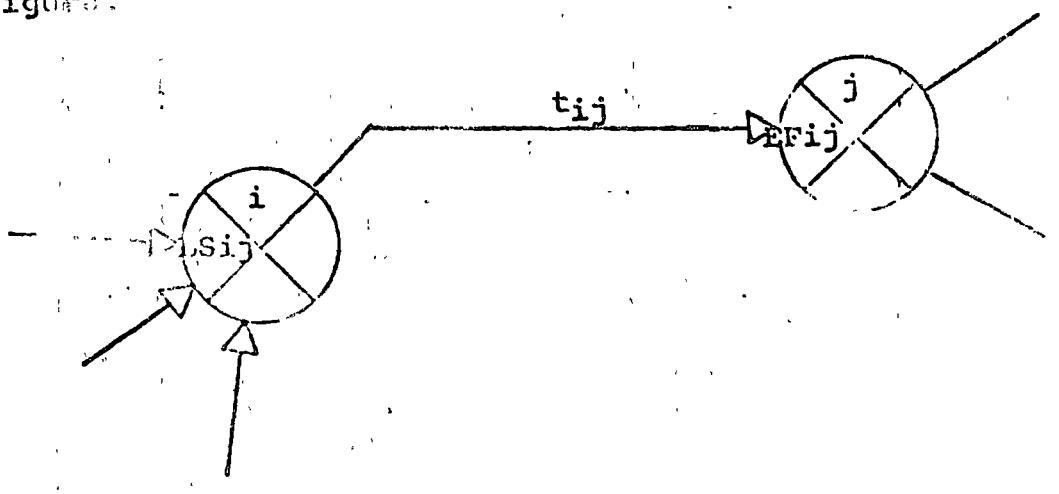
   $E_j$ = Máximo de los valores $EF_{ij}$ de las actividades que llegan al evento j.

   $E_j = \max_k t(\overline{1k})$.

   $\max_{i \in B(J)} (E_i + t_{ij}) = \max_{i \in B(i)} (EF_{ij})$ j=2,3...m

Donde

B(J) = Es el conjunto de nodos que conectan con Nj - Estas reglas las podemos representar como las mostramos en la figura.



Estas consideraciones o reglas se aplican a todo tipo de redes como la demostrada en la figura

## 5.4 CALCULO HACIA ATRAS

El propósito del cálculo hacia atrás es contabilizar los tiempos más tardíos permitidos de inicio y de terminación para cada una de las actividades lo cual permitirá que ocurra el evento terminal en su tiempo más temprano esperado, como en el cálculo hacia adelante.

Para completar esto; el cálculo hacia atrás se inicia en un solo evento terminal del proyecto y arbitrariamente se le asigna el mismo tiempo de ocurrencia más temprana y corresponde a lo máximo permitido para que ocurra. Siguiendo convencionalismos, uno lo puede interpretar como el tiempo más tardío de iniciación de una actividad, al darle el tiempo de inicio de una actividad que puede ser retrasado sin causar directamente algún incremento en el tiempo total para completar el proyecto.

Cuando $L_i = E_j$ se asigna para el evento terminal de una red, retraso permitido para que una actividad se inicie se calcula como la substracción de la duración de la actividad al último tiempo final permitido. Finalmente, de acuerdo a la lógica de la red, un evento debe ocurrir antes de que ninguna actividad sucesora comience. Por lo tanto, el último tiempo permitido para un evento es igual al menor de los tiempos de iniciación tardía permitida de las actividades que salen del evento en cuestión.

Estas consideraciones o reglas las sumarizamos aquí.

## 5.5 REGLAS PARA CALCULO HACIA ATRAS

**Regla 1.-** El tiempo de terminación más tardía para el evento final del proyecto (t) es un conjunto igual o menor que una fecha programada para terminar el proyecto, $T_s$ o debe ser igual a su tiempo de ocurrencia más temprana calculada por los calculos hacia adelante.

$$L_t = T_s \text{ o } E_t$$

Regla 2.- El tiempo de terminación más tardía para una actividad arbitraria (i-j) es igual a la más pequeña, o más temprana, de los tiempos de iniciación más tardía de sus actividades sucesoras.

$LF_{ij}$ = Mínimo de las LS de las actividades. Directamente dirigidas a la actividad (i-j)

Regla 3.- El tiempo de iniciación más tardía para una actividad arbitraria (i-j) es su tiempo de terminación más tardía menos la duración estimada de la actividad.

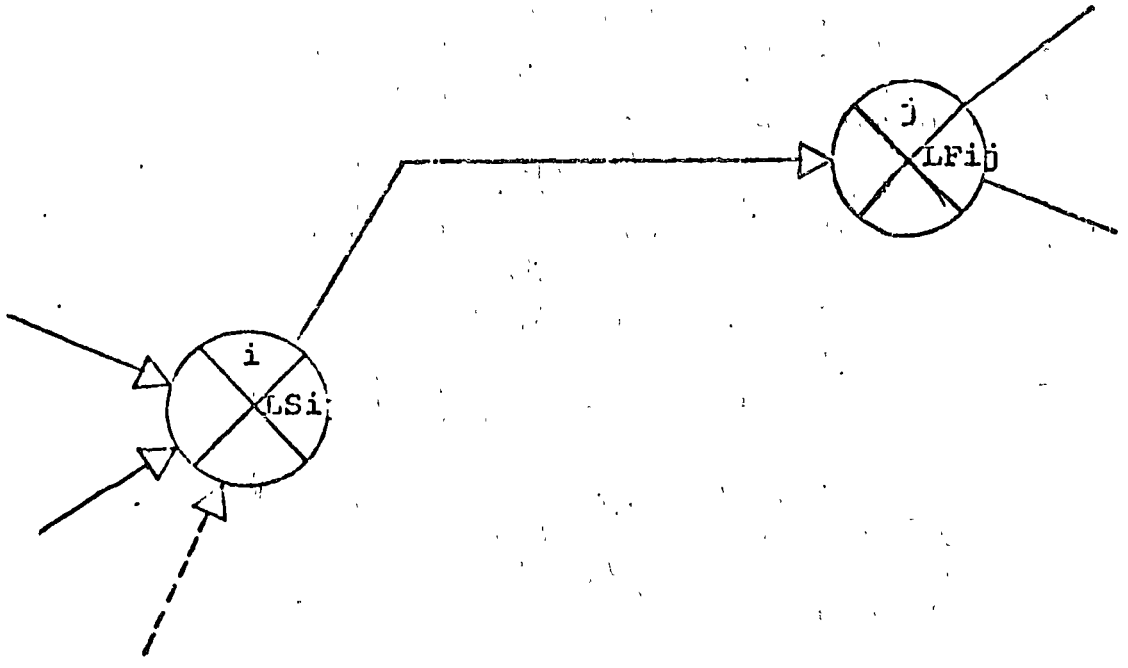
$LS_{ij} = LF_{ij} - T_{ij}$
$LF_{ij}$ = El menor de $LS_{ij}$ para un evento con n actividades que salen.

$B(i)$ = Conjunto de eventos.

$LS_{ij} = LF_j - T_{ij}$

$LF_{ij} = \min(LF_{ij} - T_{ij})$

Estas reglas se presentan en la figura.



## 5.6 DEFINICION E INTERPRETACION DE HOLGURAS

De muchos tipos de holguras definidos en literatura sobre el tema, solo discutiremos dos clases, la holgura total por actividad, o simplemente holgura total, y holgura libre por actividad, o simplemente holgura libre, cada holgura tiene una interpretación y aplicación diferente como los describiremos a continuación.

## 5.7 HOLGURA TOTAL POR ACTIVIDAD

Definición:

Holgura total por actividad es igual a la diferencia entre lo más temprano y lo más tardío permitido entre el tiempo inicial o final para una actividad en cuestión. Entonces, para la actividad (i-j), la holgura total esta dada por:

$$S_{ij} = LS_{ij} - ES_{ij}$$

ó

$$S_{ij} = LF_{ij} - EF_{ij}$$

## 5.8 HOLGURA LIBRE POR ACTIVIDAD

Definición:

Holgura libre por actividad es igual tiempo de inicio más temprano de las actividades sucesoras menos el tiempo de terminación más temprana de la actividad en cuestión. Entonces, para la actividad (i-j), la holgura libre esta dada por lo siguiente;

j - k muestra a una actividad sucesora a la actividad en cuestión.

$$FS_{ij} = ES_{jk} - EF_{ij}$$

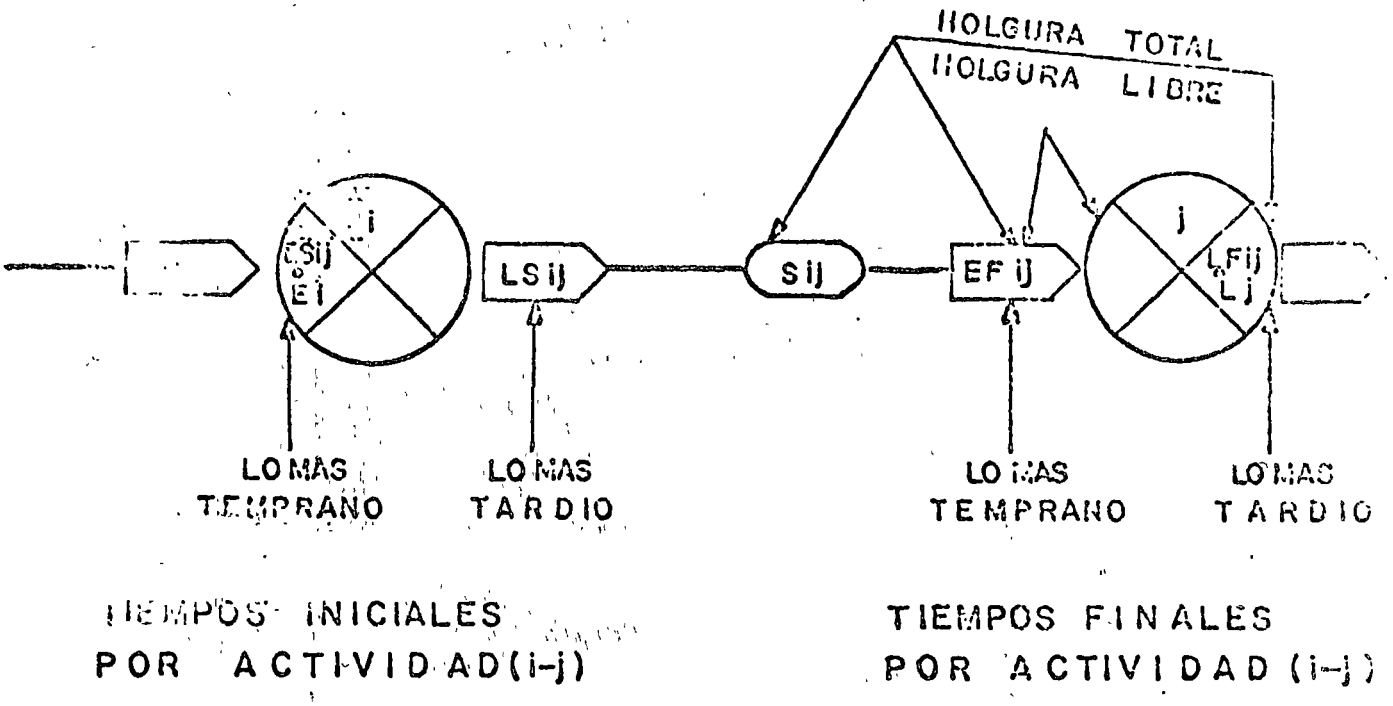
## 5.9 IDENTIFICACION DE LA RUTA CRITICA

Definición:

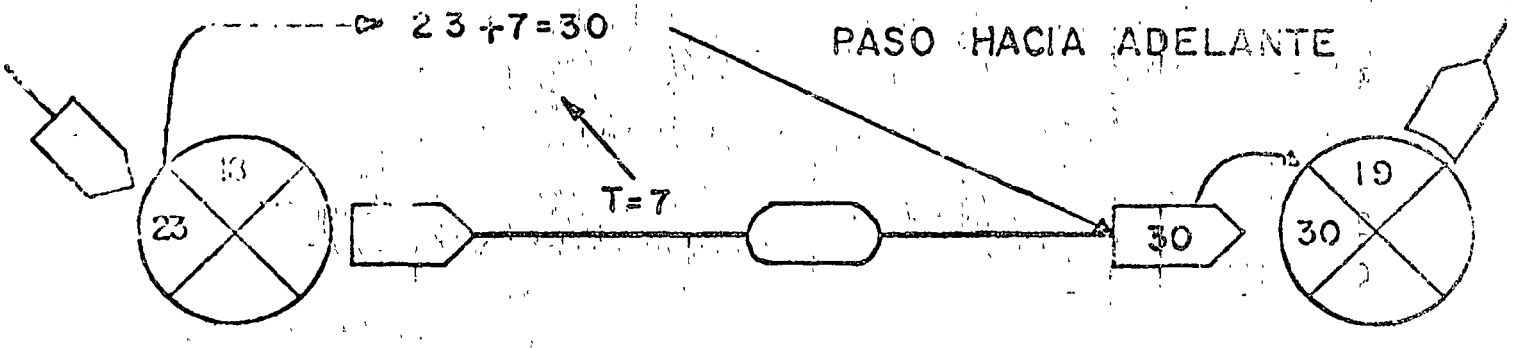
La ruta crítica es "la curva con la menor holgura total". Si la holgura cero como un convencionalismo LF para el evento final de la red presentada en la ruta crítica tendrá holgura cero; de lo contrario, la holgura sobre una ruta crítica puede ser positiva o negativa. Si la red tiene un solo evento inicial y final, y no existen fechas programadas impuestas a nodos intermedios de la red, entonces la ruta crítica es también la mayor trayectoria dentro de la red.

## 6 USO DE SIMBOLOS ESPECIALES EN CALCULOS PROGRAMADOS.

Utilizaremos unos símbolos para facilitar los cálculos de una RED. La ilustración es la siguiente:

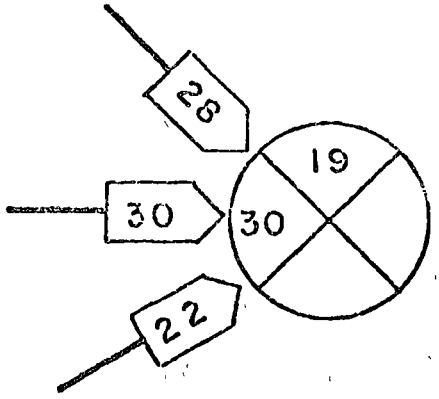



## PASOS EN LA PROGRAMACION USANDO SIMBOLOS ESPECIALES PARA ACTIVIDADES Y EVENTOS.

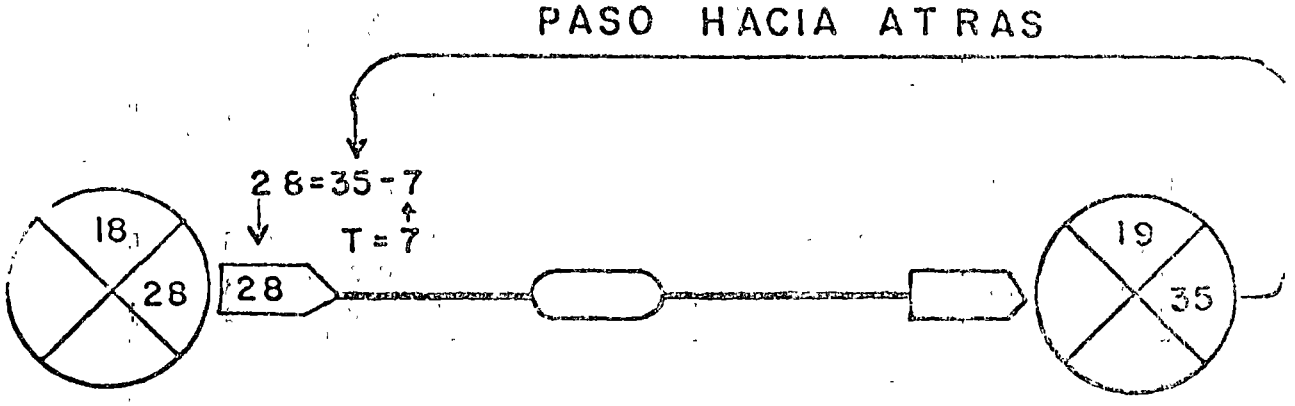


Para una actividad cualesquiera (18-19), su tiempo de iniciación más temprano (digamos 23 en este caso) en el cuadrante de la izquierda del símbolo del evento.

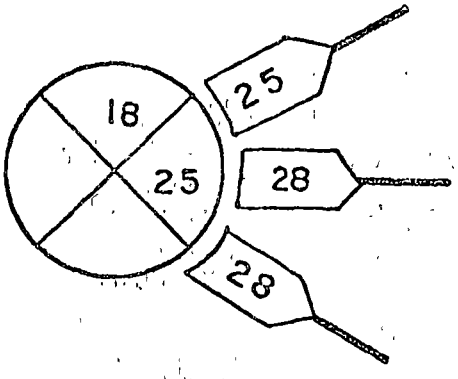
Entonces agregaremos su duración (7) a tiempo de iniciación más temprano y obtendremos su tiempo de terminación más temprano (30). Escribiremos 30 en la punta de la flecha.



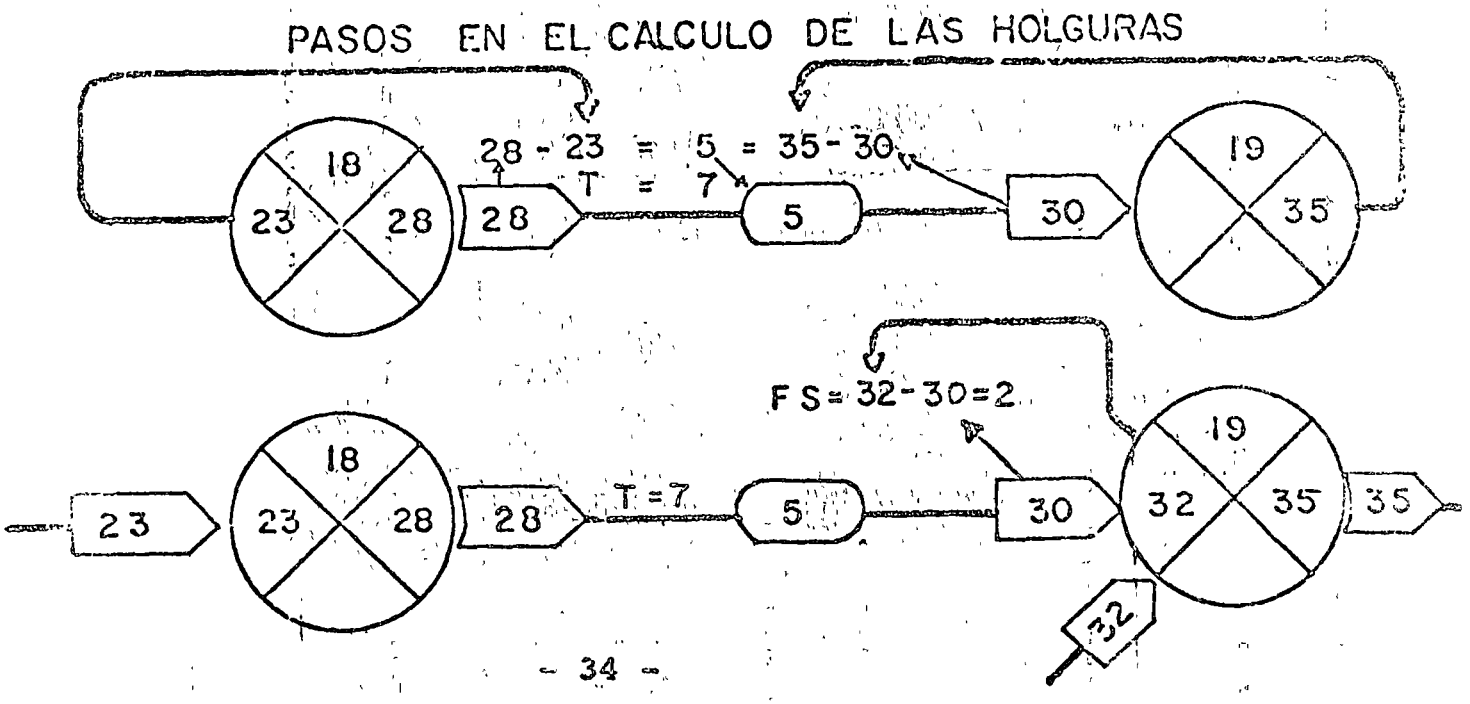

Supongamos que tres actividades llegan al Nodo 19 se anotará en el cuadrante de la izquierda al mayor tiempo de terminación más temprana.



El tiempo programado para el evento final de una actividad en cuestión (18-19) deberá ser colocado a la derecha del símbolo del evento. Para otros eventos se insertará el tiempo de ocurrencia más tardio. Para este caso especial, hay que subtraer su duración (7) del tiempo de terminación más tardio (35) para obtener el tiempo de iniciación más tardio (28). Deberá escribirse 28 al final de la flecha.




Cuando dos o más actividades salen de un evento, se debe colocar en el cuadrante de la derecha el valor más pequeño de los tiempos de iniciación más tardía para la actividad.



## 7 EJEMPLO ILUSTRATIVO

Hasta el momento tenemos información sobre las tablas de secuencias, las reglas para elaborar una red y a partir de ésta efectuar los cálculos básicos.

Con éstos elementos podemos obtener todos los datos para planificar, programar y controlar cualquier proyecto.

Si ejemplificamos un determinado proyecto que nos muestre las diferencias entre un proyecto que no utilice las técnicas CPM y PERT contra otro en el cual si se tomen en consideración, veremos, la gran diferencia que se hizo resaltar con anterioridad.

El método antiguo solo disponía de Diagramas de Gantt para coordinar proyectos, como el siguiente:

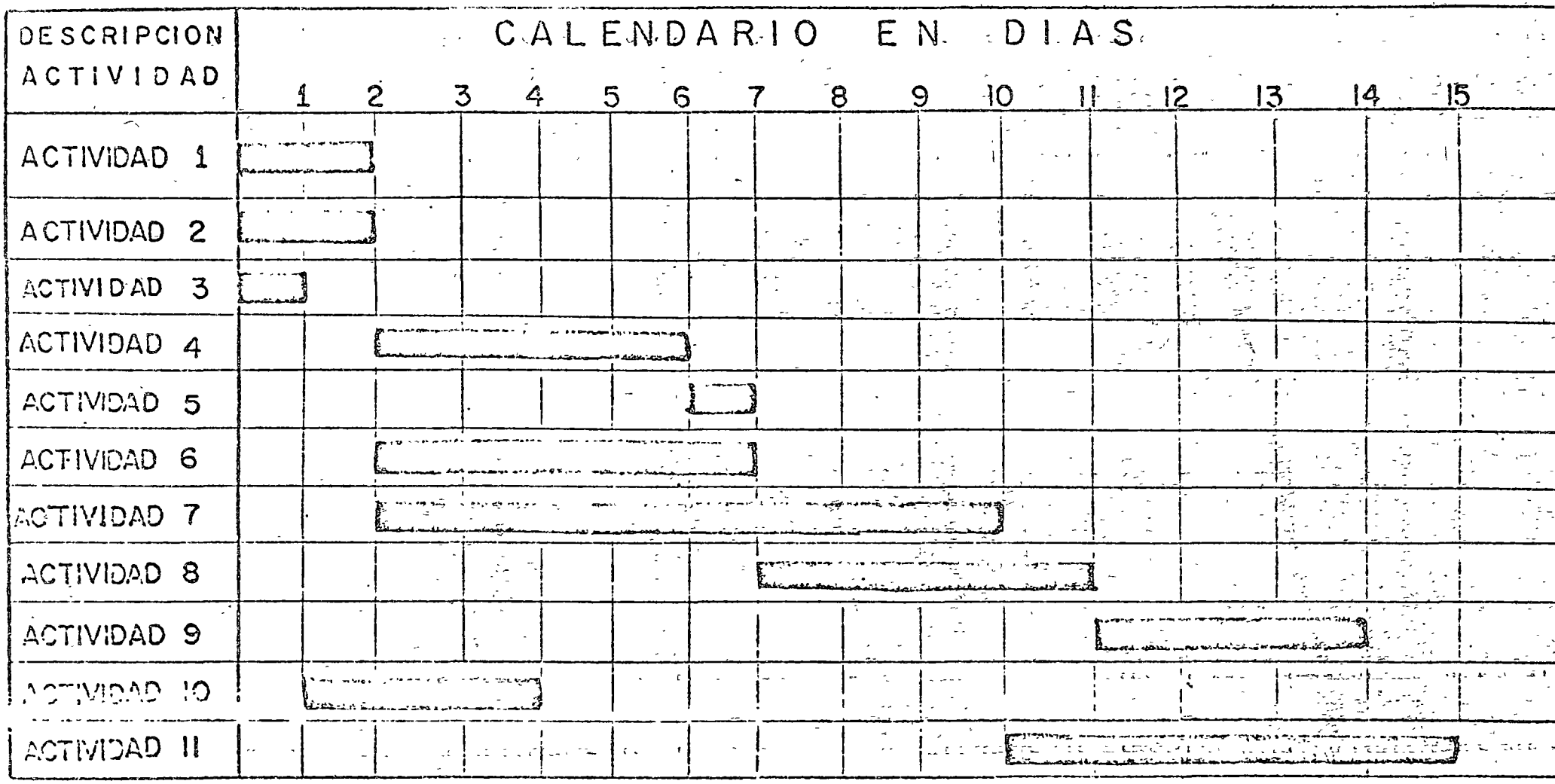



DIAGRAMA DE BARRAS

En el diagrama anterior las actividades no tienen ninguna relación entre sí y es difícil investigarlo.

El cumplimiento de un proyecto con las características mencionadas es una tarea muy árdua ya que el responsable por actividad al no tener información de otras actividades no cuenta con datos para iniciar la suya propia y en cierta medida tendrá aún problemas para terminarla y controlarla.

Hagamos uso de las técnicas antes mencionadas e interpretar correctamente lo visto hasta ahora.

Primero se diseña un listado de todas las actividades de nuestro proyecto, no importando el orden de colocación.

Lista de Actividades

Actividad 1

Actividad 2

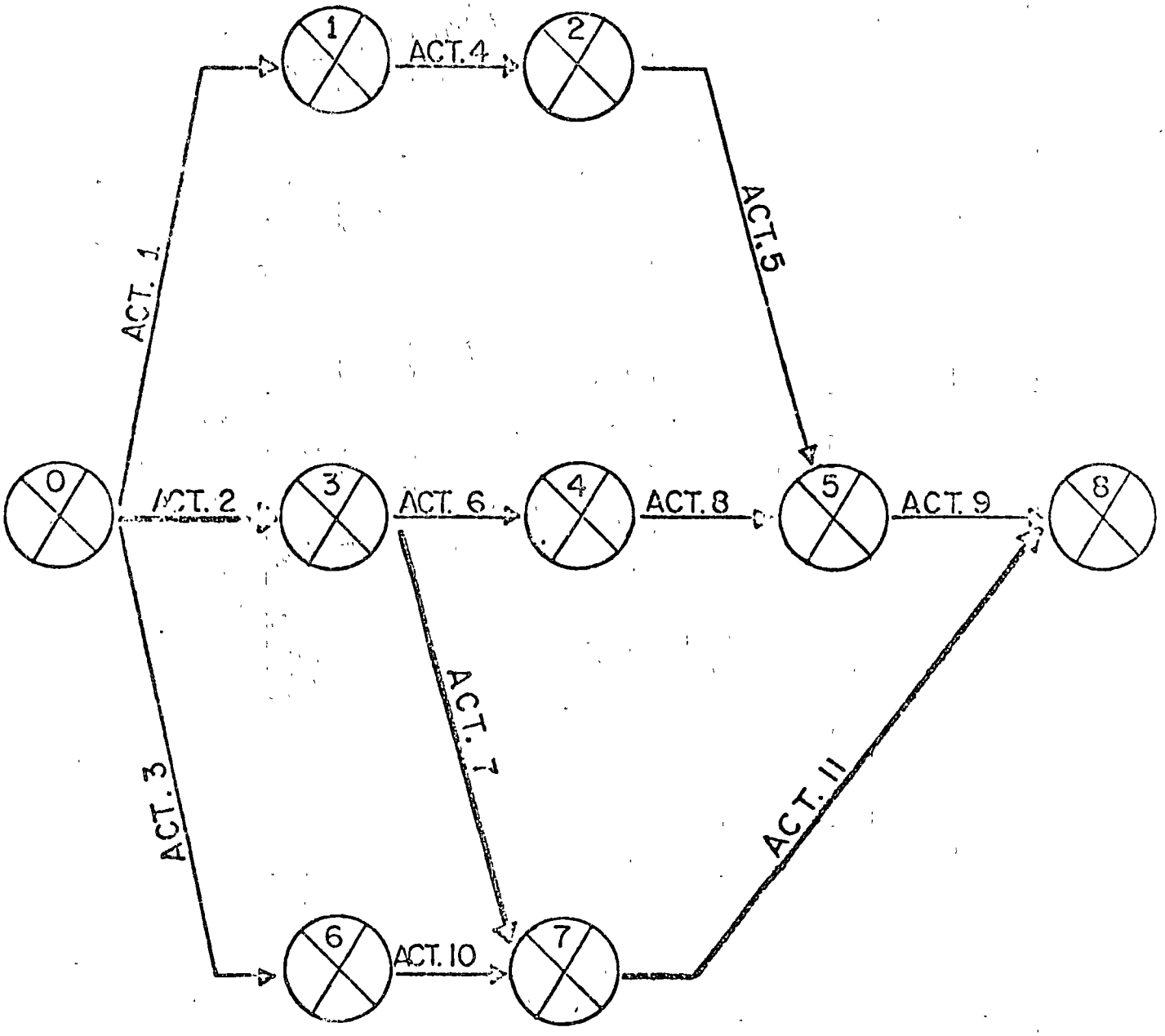
Actividad 11

El siguiente paso es elaborar la matriz de secuencias.

| ACT. INM. ANTERIOR \ ACT. INM. SIG. | ACT. 1 | ACT. 2 | ACT. 3 | ACT. 4 | ACT. 5 | ACT. 6 | ACT. 7 | ACT. 8 | ACT. 9 | ACT. 10 | ACT. 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACT. 1 | | | | X | | | | | | | |
| ACT. 2 | | | | | | X | X | | | | |
| ACT. 3 | | | | | | | | | | X | |
| ACT. 4 | | | | | X | | | | | | |
| ACT. 5 | | | | | | | | | X | | |
| ACT. 6 | | | | | | | | X | | | |
| ACT. 7 | | | | | | | | | | | X |
| ACT. 8 | | | | | | | | | X | | |
| ACT. 9 | | | | | | | | | | | |
| ACT. 10 | | | | | | | | | | | X |
| ACT. 11 | | | | | | | | | | | |

MATRIZ DE SECUENCIAS

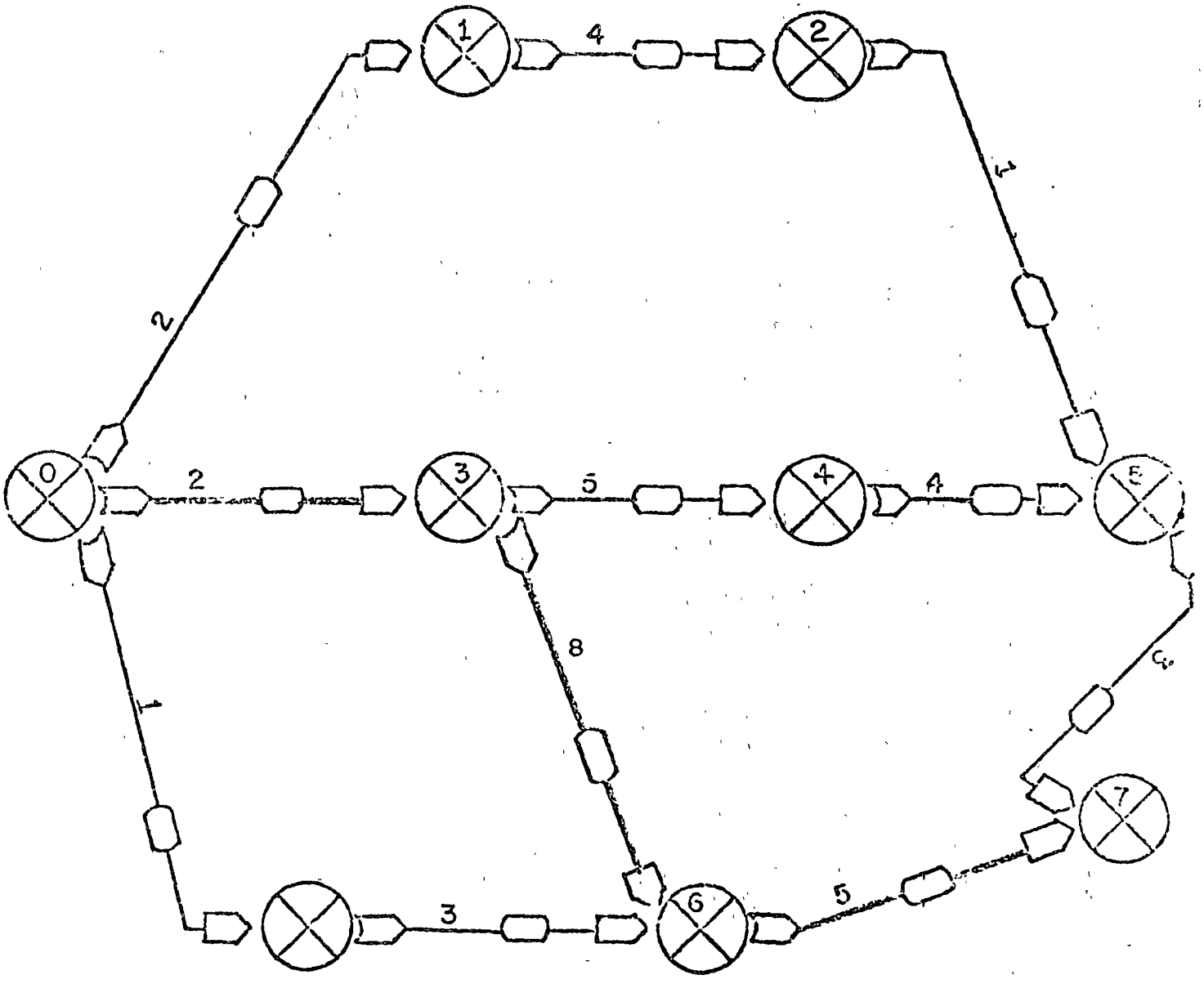
A partir de la matriz de secuencias se elabora la red y se enumeran los nodos.



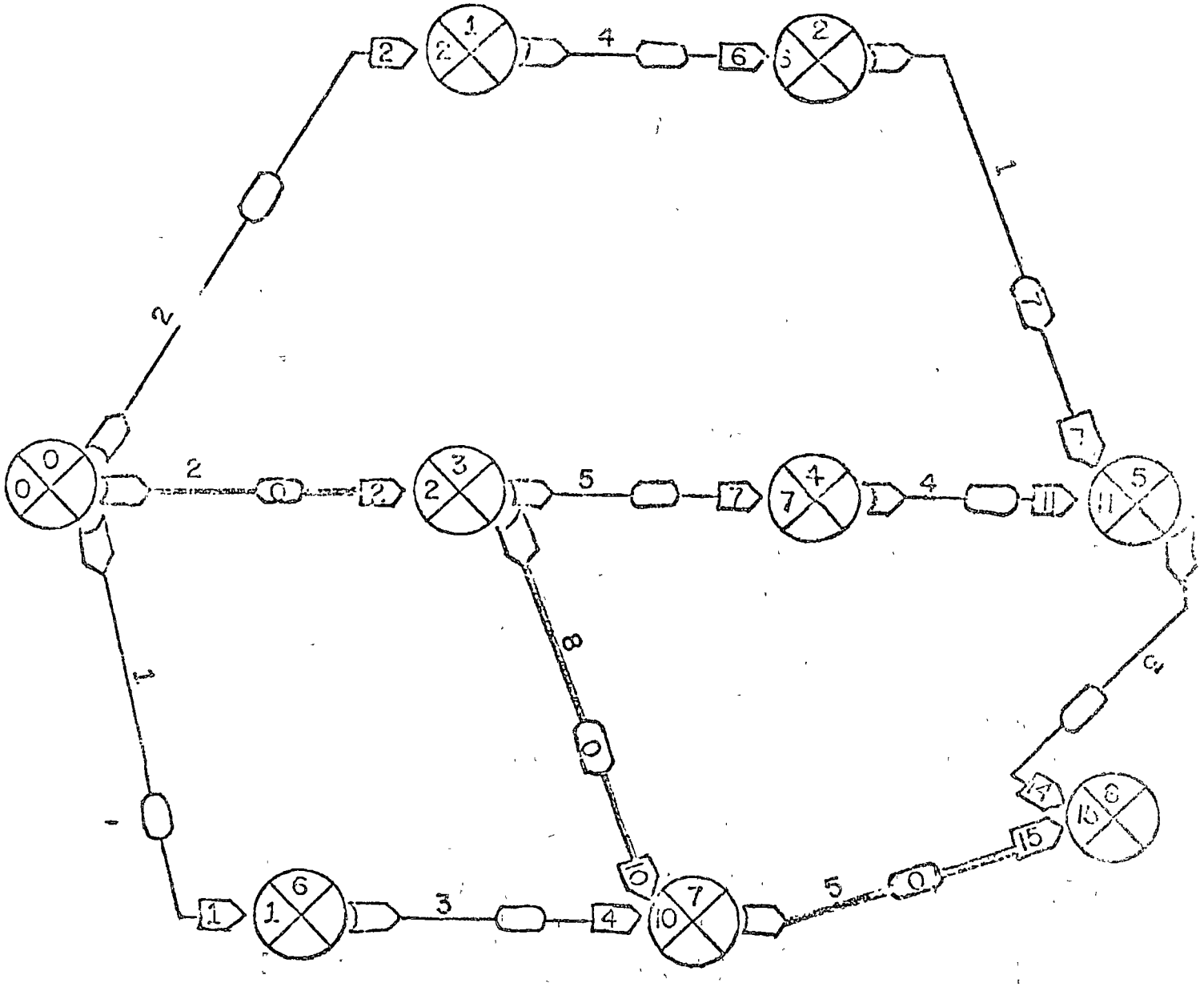
Al tener lista la red se complementa con la información adecuada,

| DESCRIPCION ACTIVIDAD | CODIGO DE IDENTIFICACION | | DURACION EN DIAS |
|---|---|---|---|
| | NODO INICIAL | NODO FINAL | |
| ACT. 1 | 0 | 1 | 2 |
| ACT. 2 | 0 | 3 | 2 |
| ACT. 3 | 0 | 6 | 1 |
| ACT. 4 | 1 | 2 | 4 |
| ACT. 5 | 2 | 5 | 1 |
| ACT. 6 | 3 | 4 | 5 |
| ACT. 7 | 3 | 7 | 8 |
| ACT. 8 | 4 | 5 | 4 |
| ACT. 9 | 5 | 8 | 3 |
| ACT. 10 | 6 | 7 | 3 |
| ACT. 11 | 7 | 8 | 5 |

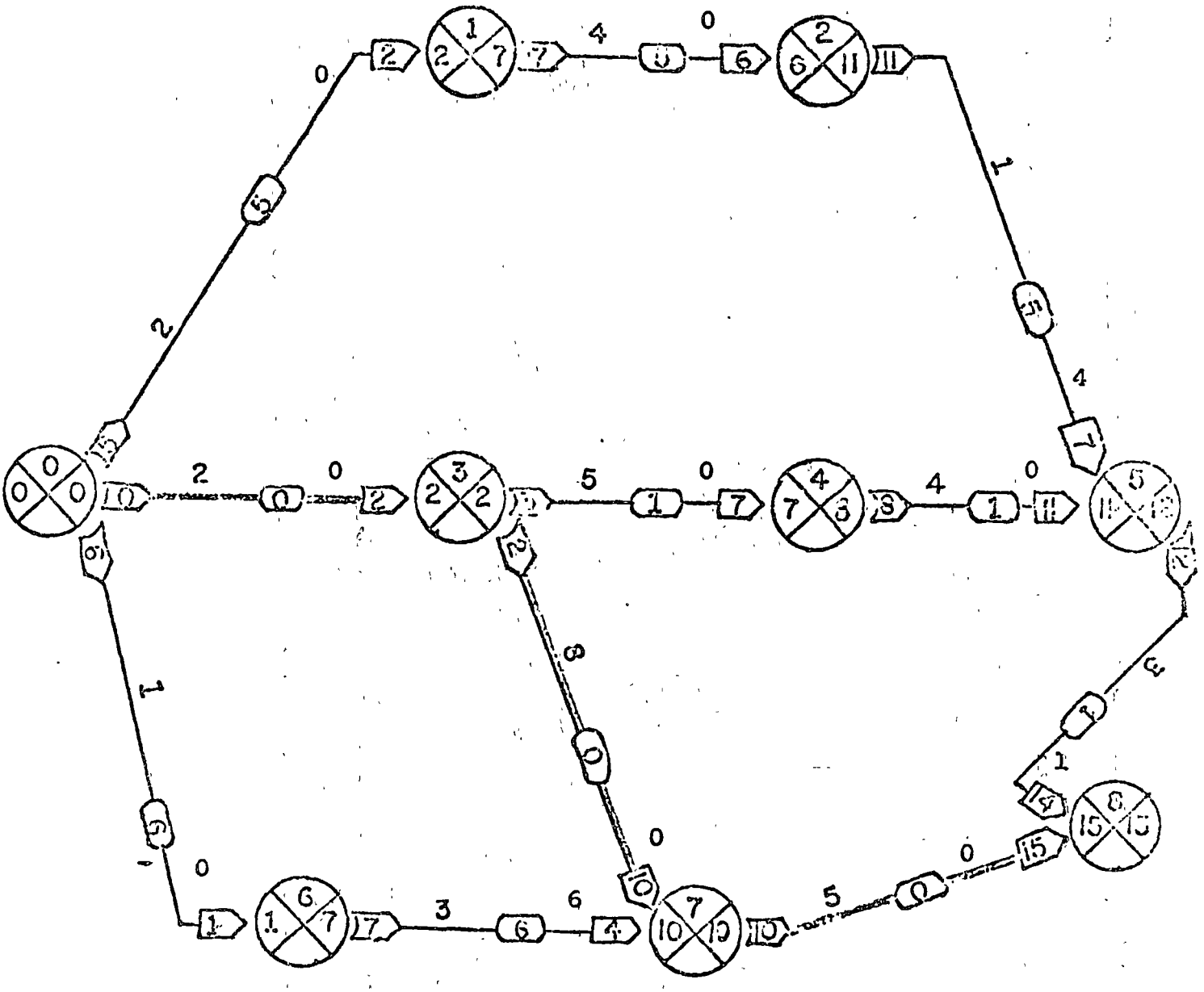
Con toda la información procedemos a calcular nuestro proyecto con los símbolos especiales.



Red ilustrativa del proyecto utilizando los símbolos especiales, para los eventos, actividades y el paso hacia - adelante.



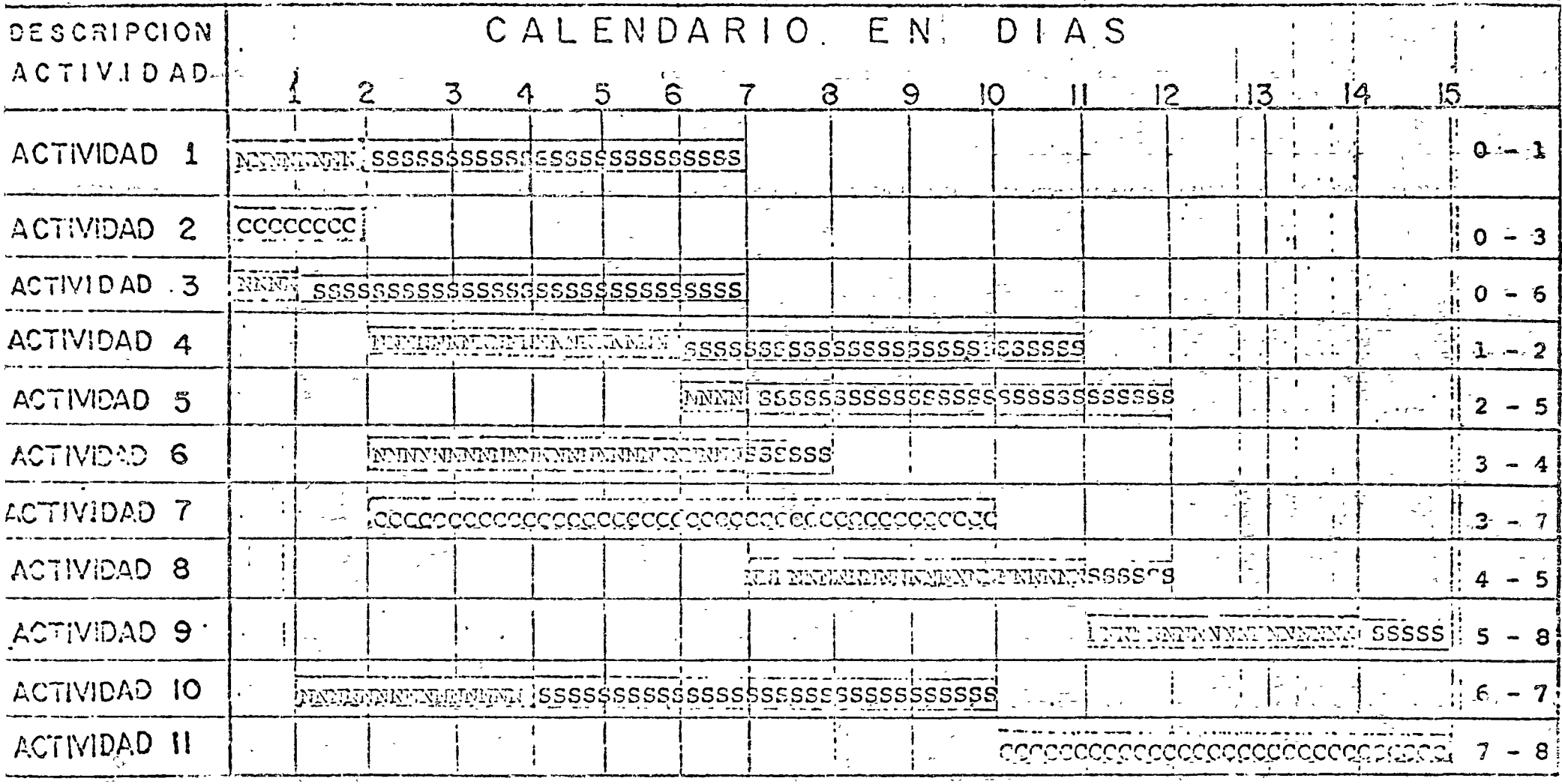
Ejemplo que ilustra la red del proyecto utilizando los símbolos especiales para los eventos, actividades, los pasos hacia adelante, hacia atrás, las holguras total y libre.



$L_8 = E_8 = 15$

Podemos resumir todos los datos contenidos en el proyecto en la lista de abajo.

| Actividad | Lo Más Temprano | | Lo Más Tardío | | Holgura Total | Criticalidad |
|---|---|---|---|---|---|---|
| | $ES_{ij}$ ó $E_i$ Tiempo de Inicio. | $EF_{ij}$ Tiempo de Terminac. | $LS_{ij}$ Tiempo de Inicio. | $LF_{ij}$ ó $L_j$ Tiempo de Terminac. | | |
| 0-3 | 0 | 2 | 0 | 2 | 0 | CRIT |
| 3-7 | 2 | 10 | 2 | 10 | 0 | CRIT |
| 7-8 | 10 | 15 | 10 | 15 | 0 | CRIT |
| 3-4 | 2 | 7 | 3 | 8 | 1 | |
| 4-5 | 7 | 11 | 8 | 12 | 1 | |
| 5-8 | 11 | 14 | 12 | 15 | 1 | |
| 0-1 | 0 | 2 | 5 | 7 | 5 | |
| 2-5 | 6 | 7 | 11 | 12 | 5 | |
| 0-6 | 0 | 1 | 6 | 7 | 6 | |
| 6-7 | 1 | 4 | 7 | 10 | 6 | |
| 1-2 | 2 | 6 | 7 | 11 | 5 | |




# DIAGRAMA DE BARRAS

N = Tiempo Normal. C = Crítica S = Holgura Total

# C O N C L U S I O N E S

La utilización de dichas técnicas en un proyecto en el cual intervenga el lector, será la experiencia más provechosa por su realidad y que considere este manual como una herramienta que se apoya en su propia motivación.

CONTROL DATA. Advanced Planning and Control Executive - (APACE) User Information Manual, Austria, Control Data Corporation, 1970.

CLARK, FRANK J. y OTROS, Procedimientos Informáticos en Sistemas Empresariales; Tr. del Inglés por Bartolomé Fabian-Frankel, España, Prentice-Hall Internacional, 1973.

CONWAY, RICHARD W. y OTROS. Theory of Scheduling. Addison-Wesley Publishing Company, 1967 Massachusetts.

ECHAVARRY GAYTAN, IRAN ZADOK. Visión General del Método de Camino Crítico. Boletín No. 1, Residencia de la Comisión de Relaciones Humanas en el Medio y Bajo Balsas. Secretaría de Recursos Hidraulicos, México, Diciembre, 1973.

ECHAVARRY GAYTAN, IRAN ZADOK y GONZALEZ-DUHART G., HORACIO. Programación y Control de la Construcción de Coches del Metro. Ponencia 12.2 Memoria de la VI - Asamblea Nacional Bienal el Desarrollo de la Tecnología en México, México, Noviembre, 1974.

ESCOLANO JUAN, RAMON. Planeación Programación y Control por el Método de la Secuencia Crítica, México, Revista Ingenieria Petrolera, Marzo 1964.

FAVRET, ANDREW G. Introduction To Digital Computer Applications. New York, Van Nostrand Reinhold Company, 1970.

FORD, L. R. y FULKERSON, D. R. Flows in Networks, New Jersey, Princeton University Press, 1962.

FULKERSON, D. R. Expected Critical Path Lengths in PERT - Networks. Operations Research, Vol. 10, No. 6, November-December, 1962.

GENERAL ELECTRIC COMPANY. Critical Path Scheduling. User's, Guide, 803211, General Electric Company, Information Service Department, 1968.

GONZALEZ M., J. y OTROS. Tres Algoritmos de Investigación de Operaciones. México, Instituto de Ingeniería, C. U., 1970.

HILLIER, FREDERICK S. y LIEBERMAN, GERALD J. Introdution to Operations Research; 2 Ed. San Francisco, Holden-Day, Inc., 1974.

IBM SYSTEM - 370, Project Analysis and Control System - (PROJACS). General Information, GH19-1055-0. Francia, IBM Corporation, 1973.

IBM SYSTEM - 370. Project Analysis and Control System (PROJACS). Operations Guide, SH19-1085-0, Francia, IBM Corporation, 1974.

IBM SYSTEM - 370, Project Analysis and Control System - (PROJACS). Program Reference Manual, SH19-1080. Francia, IBM Corporation, 1973.

IBM. Project Control System - 360. Program Description and Operations Manual, GH20-0376-3, New York, IBM Corporation, 1967.

IBM. Project Management System IV (PMS IV). Cost Processor, Program Description and Operation Manual, SH20-0898-0, New York, IBM Corporation. 1971.

IBM. Project Management System IV (PMS IV). Report Processor, Program Description and Operation Manual - SH20-0901-1, New York, IBM Corporation. 1971.

IBM. Project Management System IV(PMS IV). Resource Allocation Processor Program Description and Operation Manual, SH20-0900-0, New York, IBM Corporation, - 1971.

JAUFFRED, M. FRANCISCO J. y OTROS. Métodos de Optimización Programación Lineal-Gráficas. México, Representaciones y Servicios de Ingeniería, S. A., 1974.

LOWE, CECIL W. Critical Path Analysis By Bar Chart. New York, Brandon-System Press, 1966.

MADRID. FEDERAL ELECTRIC CORPORATION. El Método PERT; Tr. del Inglés por Miguel Díaz Galvez y Otros, Madrid, Ciencia y Técnica, S. L., 1963.

MARTINO, R. L. Determinación de la Ruta Crítica, Tr. del Inglés por Abelardo Cruz. México, Editora Técnica, S. A., 1965. N. I.

MAYNARD, C. H. B. y OTROS. Industrial Engineering Handbook. 3 ed. New York. Mc Graw-Hill Book Company, 1971.

MEXICO. CATALYTIC CONSTRUCTION COMPANY. Método Crítico; Tr. del Inglés por Manuel Arce Rincón. México, - Diana, 1970.

MILLER, ROBERT W. Schedule, Cost, and Profit Control - with PERT. Estados Unidos de Norteamérica, Mc - Graw-Hill Book Company, Inc., 1963.

MIZE, JOE H. y OTROS. Planificación y Control de Operaciones; Tr. del Inglés por Enrique Sierra Barroneche, Prentice-Hall Internacional, España, 1973.

MODER, J.J. How to Do CPM Scheduling Without a Computer. Engineering News-Record, March 14, 1963.

MODER, JOSEPH J. y PHILLIPS, CECIL R. Project Management With CPM and PERT. 2ed. New York, Van Nostrand - Reinhold Company, 1970.

MONTAÑO, G. AGUSTIN. Iniciación al Método del Camino Crítico; 2ed. México, Trillas, 1970.

O'BRIEN, JAMES J. Método de la Línea Crítica Debiera Estar Usándolo. Harvard Business Review, Agosto, 1971.

PHILLIPS, CECIL R. Fifteen Key Features of Computer Programs for CPM and PERT, Journal of Industrial Engineering, January-February, 196

PRAWDA W. JUAN. Investigación de Operaciones, México. Próxima Publicación en Limusa S. A., 1976.

RADCLIFFE, BYRON M. y OTROS. Critical Path Method. Chicago, Division of Cahners Publishing Company, Inc., - 1967.

RODRIGUEZ CABALLERO, MELCHOR. Aplicaciones en Ingeniería de Métodos Modernos de Planeación, Programación y Control de Procesos Productivos. México, Limusa-Wiley, S. A., 1970.

ROSEBOOM, J. y OTROS. Applications of a Technique for Research and Development Program Evaluation, Operations Research, Vol. 7, No. 5, September-October, 1959.

UNIVAC. PERT. Exec II and Exec 8, Programmer Reference, up-7616, UNIVAC 1100 series, U.S.A. 1968.

WAGNER, HARVEY M. Principles of Operations Research; 2ed. New Jersey, Prentice-Hall, Inc., Englewood Cliffs, 1975.

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

# APLICACIONES DE UN PAQUETE DE PROGRAMACION LINEAL

M. EN C. VERONICA CZITROM

Febrero-Marzo 1977

## TARJETAS PARA USAR EL PROGRAMA GRANM

Columna 1 ↓

| | |
|---|---|
| Tarjeta 1 | // JØB T |
| Tarjeta 2 | // XEQ GRANM 1 |
| Tarjeta 3 | *LØCALINIT, PIVØT, TABNU, RMØVE, CLEAN, TABPR |
| | - |
| | - |
| | - Tarjetas de datos (Ver página 3) |
| | - |
| | - |
| Tarjeta final | /* |

NOTAS:

- Este programa está listo para usarse en la computadora IBM 1130 de CECAFI.
- La tarjeta 1 es la tarjeta anaranjada obtenida del CECAFI.
- El número 1 que aparece en la 2a. tarjeta se perfora en la columna 17.
- El programa en la IBM, tiene una capacidad de 10 restricciones y 15 variables incluyendo de holgura y artificiales.
- Este programa también se encuentra disponible en la Burroughs del CIMASS, bajo el nombre de II/SIMPLEX. Las instrucciones para correrlo en el CIMASS aparecen en la siguiente hoja. Este admite una capacidad mayor sobre el número de restricciones y variables como se indica en la segunda hoja.
- Este programa utiliza el método de la gran M.

## TARJETAS PARA USAR EL PROGRAMA II/SIMPLEX

| | Columna 1 | | |
|---|---|---|---|
| Tarjeta 1 | # USER | clave / | _____ |
| Tarjeta 2 | # RUN | (JR82) | II/SIMPLEX |
| Tarjeta 3 | # DATA | FILE 5 | |
| | - | | |
| | - | | |
| | - Tarjetas de datos (Ver página 3) | | |
| | - | | |
| | - | | |
| Tarjeta final | # END | | |

NOTAS:

- Este programa está listo para usarse en la computadora B 6700 de CIMAS/CSC.
- La tarjeta 1 es la tarjeta roja obtenida del CIMASS.
- El símbolo "#" significa un carácter inválido. Este se obtiene presionando las teclas MULTIPUNCH Y NUMERIC simultáneamente y perforando los números 1, 2, 3, 4.
- Este programa tiene una capacidad de 30 restricciones y 40 variables incluyendo de holgura y artificiales.

# TARJETAS DE DATOS PARA EL PROGRAMA GRANM O II/SIMPLEX

La siguiente información deberá porporcionarse en lo que se indica como tarjetas de datos en las hojas anteriores.

## TARJETA DE IDENTIFICACION DEL PROBLEMA.

En esta tarjeta puede usar desde la columna 1 a la 70 para poder dar cualquier identificación que desee dar a su problema.

## TARJETA DE DIMENSION Y ETIQUETACION DEL PROBLEMA Y CONTROL PARA CORRER MAS DE UN PROBLEMA.

El usuario debe dar cuatro números enteros con formato (4I10) en la siguiente forma:

Columnas 1 - 10: Número de renglones del problema.

Columnas 11-20: Número de columnas del problema.

Columna 30 : Escriba el número 1 si desea poner etiquetas a los renglones y a las columnas.
Escriba el número 0 en caso contrario.

Columna 40 : Escriba un 1 si desea correr un problema adicional.
Escriba un 0 en caso contrario.

NOTAS:

El número de renglones no incluye la función objetivo.

Si escribe un 1 en la columna 30, el usuario, después de la tarjeta deberá dar el grupo de tarjetas para etiquetas de renglones y el grupo de tarjetas para etiquetas de columnas. Si en lugar de un 1 escribe cero deberá omitir este grupo de tarjetas y pasar a las tarjetas de coeficientes de las variables artificiales en la función objetivo.

Si escribe un 1 en la tarjeta 40 vea las notas generales.

## TARJETAS PARA ETIQUETAS DE RENGLONES.

Las etiquetas para identificar a los renglones de las restricciones, pueden tener como máximo 6 caracteres de cualquier tipo.

En una tarjeta puede escribir hasta 7 etiquetas. Estas etiquetas deben ir en las columnas 1-6, 11-16, 21-26, 31-36, 41-46, 51-56, 61-66.

## TARJETAS PARA ETIQUETAS DE COLUMNAS (VARIABLES)

Las tarjetas para identificar a las columnas o sea a las variables involucradas en el problema (incluyendo de holgura y artificiales) deberán escribirse de acuerdo a las reglas anteriores para etiquetar renglones.

## TARJETAS DE COEFICIENTES DE LAS VARIABLES ARTIFICIALES EN LA FUNCION OBJETIVO.

A cada variable artificial asígnele un 1 y a las variables no artificiales asígnele un 0. Estos números escríbalos en las columnas 10, 20, 30, 40, 50, 60, 70, de acuerdo al orden en que etiquetó a sus variables (columnas)
IMPORTANTE. Esta tarjeta es requerida aún si el problema no tiene variables artificiales.

## TARJETAS DE COEFICIENTES DE LAS VARIABLES NO ARTIFICIALES EN LA FUNCION OBJETIVO.

Escriba los coeficientes de la función objetivo con el formato (7 F 10.0). Estos coeficientes debe escribirlos de acuerdo al orden en que etiquetó sus variables (columnas). Los coeficientes de las variables de holgura y artificiales deberá ser cero.

IMPORTANTE : Los coeficientes de la función objetivo deben corresponder al problema de minimizar. Por lo tanto, si su problema es de maximizar multiplique por -1 y considere los coeficientes que resultan como los datos de entrada en este programa.

## TARJETAS DE LOS COEFICIENTES DE LA MATRIZ DE RESTRICCIONES.

Cada renglón de restricciones va en una o varias tarjetas, escribiendo los elementos sucesivamente en una tarjeta con un formato (7 F 10.0). Cada vez que proporcione un nuevo renglón debe empezarlo en otra tarjeta.

## TARJETAS DE LOS LADOS DERECHOS DE LAS RESTRICCIONES.

Los coeficientes del lado derecho de restricciones se proporcionan sucesivamente en una tarjeta o en caso de ser insuficiente use otra tarjeta. El formato es (7 F 10.0)

## TARJETAS PARA INDICAR EL CONJUNTO INICIAL DE VARIABLES BASICAS.

En una tarjeta programe sucesivamente los números de las columnas que van a ser usadas como columnas (variables) básicas iniciales. Use formato (7 I 10).

NOTAS GENERALES:

1. El orden de las tarjetas debe ser como el indicado.
2. Si en la TARJETA DE DIMENSION Y ETIQUETACION escribió un 1 en la columna 40 entonces su nuevo problema debe ir después de la TARJETA PARA INDICAR EL CONJUNTO INICIAL DE VARIABLES ARTIFICIALES. Es importante que en el nuevo problema empiece con la TARJETA DE IDENTIFICACION DEL PROBLEMA.

EJEMPLO 1. Considere el problema lineal

$$\max z = x_4 - x_5$$

s.a.

$$\begin{aligned} 2x_2 - x_3 - x_4 + x_5 &\geq 0 \\ -2x_1 + 2x_3 - x_4 + x_5 &\geq 0 \\ x_1 - 2x_2 - x_4 + x_5 &\geq 0 \\ x_1 + x_2 + x_3 &= 1 \\ x_i &\geq 0 \end{aligned}$$

Deberemos multiplicar la función objetivo por - 1 para que el problema sea de minimización y también agregar variables de holgura a las primeras tres restricciones para que lleguen a ser igualdades. Con estas observaciones el programa lineal estará en forma estandard, lo cual es una condición para aplicar el programa GRAN M. Si definimos $z' = -z$, nuestro problema en forma estandard es

$$\min z' = -x_4 + -x_5$$

$$\begin{aligned} -2x_2 + x_3 + x_4 - x_5 + s_1 &= 0 \\ 2x_1 - 2x_3 + x_4 - x_5 + s_2 &= 0 \\ -x_1 + 2x_2 + x_4 - x_5 + s_3 &= 0 \\ x_1 + x_2 + x_3 &= 1 \end{aligned}$$

$$\begin{aligned} x_i &\geq 0; \quad i = 1, 2, \ldots, 5 \\ s_i &\geq 0; \quad i = 1, 2, 3 \end{aligned}$$

Obsérvese que aunque el programa lineal ya está en forma estandard, todavía no está listo para empezar el algoritmo de la Gran M porque en la última restricción no existe una variable que aparezca en esta restricción pero no se encuentre en las otras restricciones. (ie., no se tiene una solución básica factible inmediata). Por lo tanto, deberemos agregar una variable artificial que llamaremos $t_1$, a la cuarta restricción para así completar nuestra solución básica factible en la cual se inicia el algoritmo. Sin embargo, al introducir esta variable artificial en la restricción deberemos agregarla en la función objetivo multiplicada por una cantidad positiva M muy grande. Así nuestro problema resulta ser:

$$\min z' = -x_4 + x_5 + Mt_1$$

$$\begin{aligned} -2x_2 + x_3 + x_4 - x_5 + s_1 &= 0 \\ 2x_1 - 2x_3 + x_4 - x_5 + s_2 &= 0 \\ -x_1 + 2x_2 + x_4 - x_5 + s_3 &= 0 \\ x_1 + x_2 + x_3 + t_1 &= 1 \end{aligned}$$

$$\begin{aligned} x_i &\geq 0; \quad i = 1, 2, \ldots, 5 \\ s_i &\geq 0; \quad i = 1, 2, 3 \\ t_1 &\geq 0 \end{aligned}$$

Es conveniente representar el programa lineal en un tablero (o tableau), para poder entender más fácilmente la información que deberemos proporcionar al programa de computadora GRAN M 6 II/SIMPLEX. Esta representación aparece abajo

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $s_1$ | $s_2$ | $s_3$ | $t_1$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| Función Obj. (F.O.) | 0 | 0 | 0 | -1 | 1 | 0 | 0 | 0 | M | z' |
| Renglón 1 (R.1) | 0 | -2 | 1 | 1 | -1 | 1 | 0 | 0 | 0 | 0 |
| Renglón 2 (R.2) | 2 | 0 | -2 | 1 | -1 | 0 | 1 | 0 | 0 | 0 |
| Renglón 3 (R.3) | -1 | 2 | 0 | 1 | -1 | 0 | 0 | 1 | 0 | 0 |
| Renglón 4 (R.4) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

* * * *

Var. Holgura — Var. Art.

Solución inicial básica factible

Este tablero contiene toda la información necesaria y la notación apropiada para correr el programa GRAN M 6 el II/SIMPLEX. A continuación se presenta su codificación para el GRAN M. Para correr el II/SIMPLEX la codificación es idéntica excepto por las tarjetas de control como se mencionó en la explicación de estos programas.

UNAM | HOJA DE CODIFICACION Y/O DATOS FORTRAN | FACULTAD DE INGENIERIA

NUMERO DE PROPOSICION | PROGRAMA | PROGRAMADOR | HOJA DE | FECHA | IDENTIFICACION Y SECUENCIA

```
//JOB T
//XEQ GRANM 1
*LOCALINIT,PIVOT,TABNU,RMOVE,CLEAN,TABPR
EJEMPLO CON SOLUCION ACOTADA DE INVESTIGACION DE OPERACIONES I
    4         9         1         0
R1        R2        R3        R4
X1        X2        X3        X4        X5        S1        S2
S3        T1
    0         0         0         0         0         0         0
    0         1
0.0       0.0       0.0      -1.0       1.0       0.0       0.0
0.0       0.0
0.0      -2.0       1.0       1.0      -1.0       1.0       0.0
0.0       0.0
2.0       0.0      -2.0       1.0      -1.0       0.0       1.0
0.0       0.0
-1.0      2.0       0.0       1.0      -1.0       0.0       0.0
1.0       0.0
1.0       1.0       1.0       0.0       0.0       0.0       0.0
0.0       1.0
0.0       0.0       0.0       1.0
    6         7         8         9
/*
```

EJEMPLO 2

$$\max z = x_1 + x_2$$

s.a.

$$x_1 + x_2 \geq 1$$
$$x_1 + x_2 \leq 1$$
$$-x_1 + x_2 \leq 1$$
$$x_i \geq 0$$

Expresando la función objetivo en términos de minimización e introduciendo variables de holgura, artificiales; el problema es equivalente a :

$$\min (-z) = -x_1 - x_2 + Mt_1$$
$$x_1 + x_2 - s_1 + t_1 = 1$$
$$x_1 - x_2 + s_2 = 1$$
$$-x_1 + x_2 + s_3 = 1$$

En forma de tableau:

| | $x_1$ | $x_2$ | $s_1$ | $t_1$ | $s_2$ | $s_3$ | |
|---|---|---|---|---|---|---|---|
| Func. Obj. (F.O.) | -1 | -1 | 0 | M | 0 | 0 | -z |
| Renglón 1 (R.1) | 1 | 1 | -1 | 1 | 0 | 0 | 1 |
| Renglón 2 (R. 2) | 1 | -1 | 0 | 0 | 1 | 0 | 1 |
| Renglón 3 (R.3) | -1 | +1 | 0 | 0 | 0 | 1 | 1 |
| | | | | * | * | * | |

Solución básica factible inicial.

UNAM | HOJA DE CODIFICACION Y/O DATOS FORTRAN | FACULTAD DE INGENIERIA

NUMERO DE PROPOSICION | CONTINUE | PROGRAMA | HOJA DE | IDENTIFICACION Y SECUENCIA

PROGRAMADOR | FECHA

```
//  JOB  T
//  XEQ  GRANM     1
*LOCAL INIT,PIVOT,TABNU,,RMOVE,,CLEAN,,TABPR
PROBLEMA NO ACOTADO DE INVESTIGACION DE OPERACIONES I
         3         6         1         0
R1       R2        R3
X1       X2        S1        T1        S2        S3
         0         0         0         1         0         0
-1.0     -1.0      0.0       0.0       0.0       0.0
1.0      1.0       -1.0      1.0       0.0       0.0
1.0      -1.0      0.0       0.0       1.0       0.0
-1.0     1.0       0.0       0.0       0.0       1.0
1.0      1.0       1.0
         4         5         6
/*
```

EJEMPLO 3 Resolver el dual del siguiente par de problemas primal - dual.

Primal

$$\min z = 2x_1 - 3x_2$$
$$2x_1 - x_2 - x_3 \geq 3$$
$$x_1 - x_2 + x_3 \geq 2$$
$$x_i \geq 0$$

Dual

$$\max w = 3\lambda_1 + 2\lambda_2$$
$$2\lambda_1 + \lambda_2 \leq 2$$
$$-\lambda_1 - \lambda_2 \leq -3 \rightarrow \lambda_1 + \lambda_2 \geq 3$$
$$-\lambda_1 - \lambda_2 \leq 0$$
$$\lambda_i \geq 0$$

Este dual es equivalente a

$$\min (-w) = -3\lambda_1 - 2\lambda_2 + Mt_1$$
$$2\lambda_1 + \lambda_2 + s_1 = 2$$
$$\lambda_1 + \lambda_2 - s_2 + t_1 = 3$$
$$-\lambda_1 + \lambda_2 + s_3 = 0$$

En forma de tableau; el dual está dado por

| | $\lambda_1$ | $\lambda_2$ | $s_2$ | $s_1$ | $t_1$ | $s_3$ | |
|---|---|---|---|---|---|---|---|
| F.O. | -3 | -2 | 0 | 0 | M | 0 | -w |
| R1 | 2 | 1 | 0 | 1 | 0 | 0 | 2 |
| R2 | 1 | 1 | -1 | 0 | 1 | 0 | 3 |
| R3 | -1 | 1 | 0 | 0 | 0 | 1 | 0 |
| | | | | * | * | * | |

UNAM | HOJA DE CODIFICACION Y/O DATOS FORTRAN | FACULTAD DE INGENIERIA

NUMERO DE PROPOSICION | PROGRAMA | HOJA DE | IDENTIFICACION Y SECUENCIA

PROGRAMADOR | FECHA

```
//JOB T
//XEQ GRANM 1
*LOCALINIT,PIVOT,TABNU,RMOVE,CLEAN,TABPR
PROBLEMA SIN SOLUCION DE INVESTIGACION DE OPERACTONES I
      3     6     1     0
R1    R2    R3
L1    L2    S2    S1    T1    S3
      0     0     0     0     1     0
-3.0  -2.0  0.0   0.0   0.0   0.0
2.0   1.0   0.0   1.0   0.0   0.0
1.0   1.0   -1.0  0.0   1.0   0.0
-1.0  1.0   0.0   0.0   0.0   1.0
2.0   3.0   0.0
      4     5     6
/*
```

EJEMPLO 4

$$\max\ z = x_1 - x_2 + x_3 - 3x_4 + x_5 - x_6 - 3x_7$$

s.a.

$$\begin{aligned} 3x_3 \quad &+ x_5 + x_6 \quad &= 6 \\ x_2 + 2x_3 - x_4 \quad & \quad &= 10 \\ -x_1 \quad & + x_6 \quad &= 0 \\ x_3 \quad & + x_6 + x_7 \quad &= 6 \end{aligned}$$

$$x_i \geq 0$$

$$\min\ (-z) = -x_1 + x_2 - x_3 + 3x_4 - x_5 + x_6 + 3x_7$$

s.a.

$$\begin{aligned} 3x_3 \quad &+ x_5 + x_6 \quad &= 6 \\ x_2 + 2x_3 - x_4 \quad & \quad &= 10 \\ x_1 \quad & - x_6 \quad &= 0 \\ x_3 \quad & + x_6 + x_7 \quad &= 6 \end{aligned}$$

En forma de Tableau:

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | |
|---|---|---|---|---|---|---|---|---|
| F.O. | -1 | 1 | -1 | 3 | -1 | 1 | 3 | -z |
| R1 | 0 | 0 | 3 | 0 | 1 | 1 | 0 | 6 |
| R2 | 0 | 1 | 2 | -1 | 0 | 0 | 0 | 10 |
| R3 | 1 | 0 | 0 | 0 | 0 | -1 | 0 | 0 |
| R4 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 6 |
| | * | * | | | * | | * | |

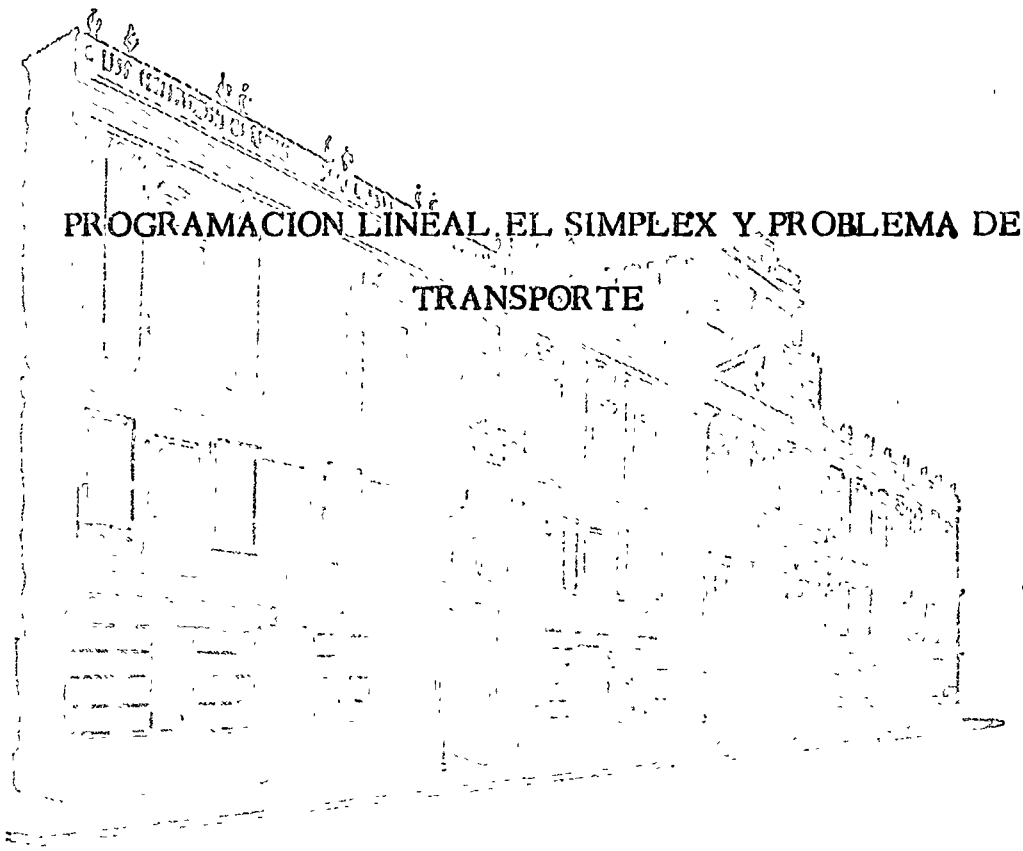
UNAM | HOJA DE CODIFICACION Y/O DATOS FORTRAN | FACULTAD DE INGENIERIA

NUMERO DE PROPOSICION | PROGRAMA | PROGRAMADOR | HOJA DE | FECHA | IDENTIFICACION Y SECUENCIA

```
// JØB T
// XEQ GRANM 1
*LØCALINIT,PIVØT,TABNU,RMØVE,CLEAN,TABPR
EJEMPLØ 4
        4       7       1       0
R1      R2      R3      R4
X1      X2      X3      X4      X5      X6      X7
        0       0       0       0       0       0       0
-1.0    1.0     -1.0    3.0     -1.0    1.0     3.0
0.0     0.0     3.0     0.0     1.0     1.0     0.0
0.0     1.0     2.0     -1.0    0.0     0.0     0.0
1.0     0.0     0.0     0.0     0.0     -1.0    0.0
0.0     0.0     1.0     0.0     0.0     1.0     1.0
6.0     10.0    0.0     6.0
        5       2       1       7
/*
```

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA



# PROGRAMACION LINEAL, EL SIMPLEX Y PROBLEMA DE TRANSPORTE

M. EN C. VERONICA CZITROM

Febrero-Marzo 1977

# MODELO 1

## EL PROBLEMA DE LA DIETA Y SU DUAL

### DEFINICION DEL PROBLEMA DE LA DIETA.

Suponga que un dietista está tratando de seleccionar una combinación de cinco tipos de alimentos (naranja, manzana, lechuga, chícharo y zanahoria) de manera que el alimento resultante de esta combinación reuna ciertos requerimientos nutricionales y tenga un costo mínimo. Los requerimientos nutricionales que debe tener el alimento resultante es de al menos 21 unidades de vitamina A y al menos 12 unidades de vitamina B. Las propiedades de los cinco elementos disponibles son:

| ALIMENTO | CONTENIDO DE VITAMINA A POR UNIDAD DE ALIMENTO. | CONTENIDO DE VITAMINA B POR UNIDAD DE ALIMENTO. | COSTO POR UNIDAD DE ALIMENTO |
|---|---|---|---|
| 1 (Naranja) | 1 | 0 | 20 |
| 2 (Manzana) | 0 | 1 | 20 |
| 3 (Lechuga) | 1 | 2 | 31 |
| 4 (Chícharo) | 1 | 1 | 11 |
| 5 (Zanahoria) | 2 | 1 | 12 |

El problema a que se enfrenta el dietista se puede modelar como un problema de programación lineal (o programa lineal), de la siguiente manera.

Sea $x_i$ la cantidad de alimento $i$ ($i=1,2,\ldots,5$) que debe estar en el alimento resultante de la combinación de los cinco alimentos. Por lo tanto, el costo de introducir el alimento $i$ en la mezcla será su costo unitario por la cantidad $x_i$ que esta presente en la mezcla. El costo total de la combinación de los cinco alimentos será la suma de los costos al combinar $x_1$, $x_2$, ... y $x_5$ unidades de cada alimento, ie. si $z$ es el costo total entonces

$$z = 20x_1 + 20x_2 + 31x_3 + 11x_4 + 12x_5$$

Ya que el objetivo del dietista es minimizar este costo total, entonces este objetivo se puede representar a través de la siguiente función objetivo

$$\min \; z = 20x_1 + 20x_2 + 31x_3 + 11x_4 + 12x_5 \qquad (1)$$

Los requerimientos nutricionales de vitamina A se pueden representar en la siguiente forma. Si el alimento nutricional $i$ está presente en una cantidad $x_i$ entonces proporciona una cantidad de vitamina A igual al producto de vitamina A que contiene una unidad de alimento por la cantidad $x_i$. La cantidad total proporcionada por los cinco alimentos será la suma de vitamina A con que controbuye cada alimento y esta deberá ser mayor que el contenido mínimo requerido que es de 21 unidades, ie.

$$x_1 + x_3 + x_4 + 2x_5 \geq 21 \qquad (2)$$

Similarmente, los requerimientos de vitamina B se pueden representar por

$$x_2 + 2x_3 + x_4 + x_5 \geq 12 \qquad (3)$$

Por último, otra restricción que debe estar presente en el problema del dietista es que la cantidad $x_i$ que interviene en la mezcla debe ser mayor o igual a cero, ie.

$$x_i \geq 0 \qquad i = 1,2,\ldots,5 \qquad (4)$$

Esta restricción es impuesta ya que no tiene sentido hablar de que una cantidad negativa $x_i$ está formando parte de la combinación de alimentos.

En resumen el problema del dietista es encontrar valores $x_1, x_2, \ldots, x_5$ para los cuales la función objetivo (1) alcance su mínimo y satisfagan las restricciones (2), (3) y (4). Reescribiendo las ecuaciones del (1) al (4), el problema del dietista está simbólicamente dado por

$$\min\ z = 20x_1 + 20x_2 + 31x_3 + 11x_4 + 12x_5 \qquad (*)$$

$$\left.\begin{aligned} x_1 \quad + x_3 + x_4 + 2x_5 &\geq 21 \\ x_2 + 2x_3 + x_4 + x_5 &\geq 12 \\ x_i \geq 0 & \end{aligned}\right\} \qquad (**)$$

La formulación anterior, (*) y (**), se acostumbra representar en un tablero (llamado también tableau) que aparecerá abajo. Esta representación es solo una abreviación de escritura ( a manera de una taquigrafía de programación lineal) que es útil en el algoritmo de solución, en el proceso convencional al procesar el problema por computadora y por una gran claridad en la formulación del problema dual que se presentará después. La representación de un programa lineal en forma de tablero consiste representar cada ecuación o desigualdad únicamente por los coeficientes de las variables omitiendo la escritura de sus correspondientes variables. Para conocer a que variable pertenece un coeficiente que aparece en este esquema se da la posición del coeficiente, escribiéndolo en la columna encabezada por la variable que le corresponde.

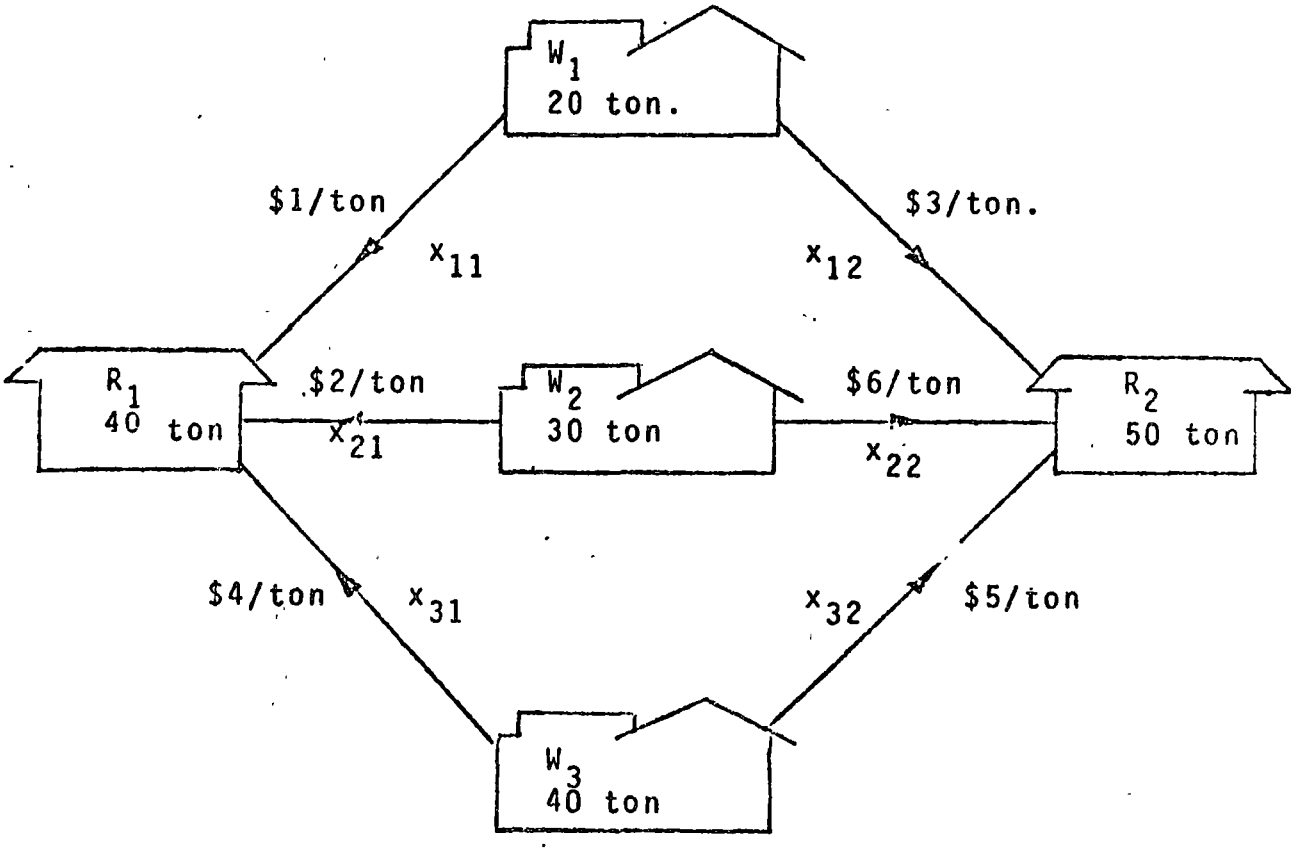
Para nuestro problema (*) y (**), la representación a través de un tablero está dada por

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | |
|---|---|---|---|---|---|
| 20 | 20 | 31 | 11 | 12 | z (min) |
| 1 | 0 | 1 | 1 | 2 | $\geq$ 21 |
| 0 | 1 | 2 | 1 | 1 | $\geq$ 12 |

$$x_i \geq 0$$

# M O D E L O  2

Una compañia tiene tres almacenes $W_1$, $W_2$, y $W_3$, y dos tiendas de ventas al por menor, $R_1$, $R_2$. Las demandas en las tiendas al por menor y el inventario en los almacenes, se muestra en las respectivas cajas de la siguiente figura. Los costos de envio por tonelada también se muestran en la figura. La compañia desea determinar la manera de realizar los envios en forma tal que minimize los costos totales de envios, satisfaga las demandas de las tiendas de menudeo, y no excedan los inventarios en los almacenes.



Sea $x_{ij}$ las toneladas del almacen $W_i$ a la tienda de menudeo $R_j$ . Entonces $x_{32}$ representa el tonelaje enviado del almacen $W_3$ a la tienda de menudeo $R_2$.

Si z representa el costo total de envios, entonces nuestro problema se puede formular por:

$$\min z = 1x_{11} + 3x_{12} + 2x_{21} + 6x_{22} + 4x_{31} + 5x_{32} \qquad (*)$$

sujeta a

$$\begin{array}{ll} \text{Restricciones sobre disponibilidad de almacenes} & x_{11} + x_{12} \leq 20 \\ & x_{21} + x_{22} \leq 30 \\ & x_{31} + x_{32} \leq 40 \\ \text{Restricciones sobre la demanda en tiendas de de menudeo} & x_{11} + x_{21} + x_{31} \geq 40 \\ & x_{12} + x_{22} + x_{32} \geq 50 \end{array} \qquad (**)$$

La formulacion anterior, (*) y (**), se acostumbra representar (por convenencia del algoritmo de solución y del proceso convencional en el procesamiento en computadora) en la siguiente tabla:

| $x_{11}$ | $x_{12}$ | $x_{21}$ | $x_{22}$ | $x_{31}$ | $x_{32}$ | |
|---|---|---|---|---|---|---|
| 1 | 3 | 2 | 6 | 4 | 5 | = z |
| 1 | 1 | 0 | 0 | 0 | 0 | $\leq$ 20 |
| 0 | 0 | 1 | 1 | 0 | 0 | $\leq$ 30 |
| 0 | 0 | 0 | 0 | 1 | 1 | $\leq$ 40 |
| 1 | 0 | 1 | 0 | 1 | 0 | $\geq$ 40 |
| 0 | 1 | 0 | 1 | 0 | 1 | $\geq$ 50 |

REPRESENTACION MATRICIAL. La formulación (*) y (**), se puede representar matricialmente como sigue:

$$\min z = \begin{bmatrix} 1 & 3 & 2 & 6 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{32} \end{bmatrix}$$

sujeta a

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{32} \end{bmatrix} \begin{matrix} \leq \\ \leq \\ \leq \\ = \\ = \end{matrix} \begin{bmatrix} 20 \\ 30 \\ 40 \\ 40 \\ 50 \end{bmatrix}$$

COMENTARIOS. El problema de programación lineal anterior ocurre tan frecuentemente en la practica, que se le ha dado un nombre especial: el problema de transporte. Los problemas de transporte en geneal, tienen tablas ralas (o matrices ralas), lo cual significa que la tabla tiene muchos ceros o sea pocos elementos distintos de cero. Dantzig y otros han desarrollado métodos especiales para la solución rápida de estos problemas.

Otro comentario importante, es la característica que presentan cada una de las columnas de la matriz de restricciones: observe que cada una tiene dos unos y los demas elementos son todos ceros.

# MODELO 4

Un inversionista tiene disponibles las actividades financieras A y B, al comienzo de cada uno de los siguientes cinco años: Cada peso invertido en A, al comienzo de un año, le regresa $ 1.40 ( una ganancia de $ 0.40) dos años más tarde (en el momento preciso para una reinversión inmediata). Cada peso invertido en B al comienzo de un año, le regresa $ 1.70 tres años después.

Además existen dos actividades financieras C y D que estarán disponibles solamente una vez en el futuro. Cada peso invertido en C en el comienzo del segundo año le regresa $ 2.00 cuatro años más tarde. Cada peso invertido en D, en el comienzo del quinto año le regresa $ 1.30 un año más tarde.

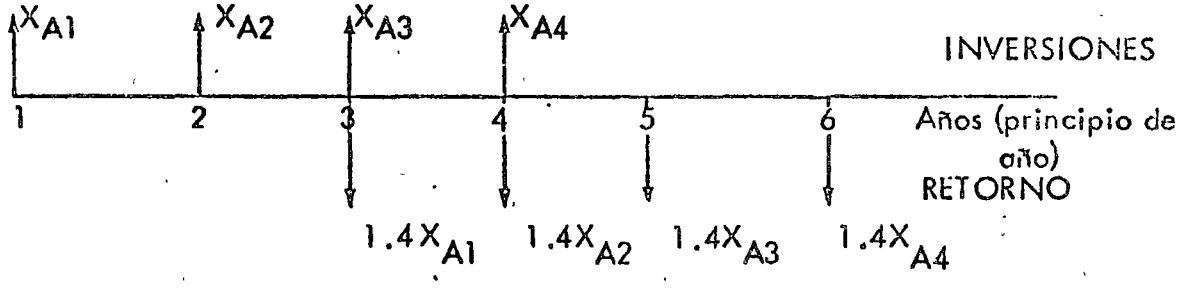
El inversionista comienza con $ 10,000.00. El desea conocer qùe plan de inversión maximiza la cantidad de dinero que el puede acumular al comienzo del sexto año. Formule un modelo de programación lineal para este problema y también expréselo en forma tabular.

SOLUCION.

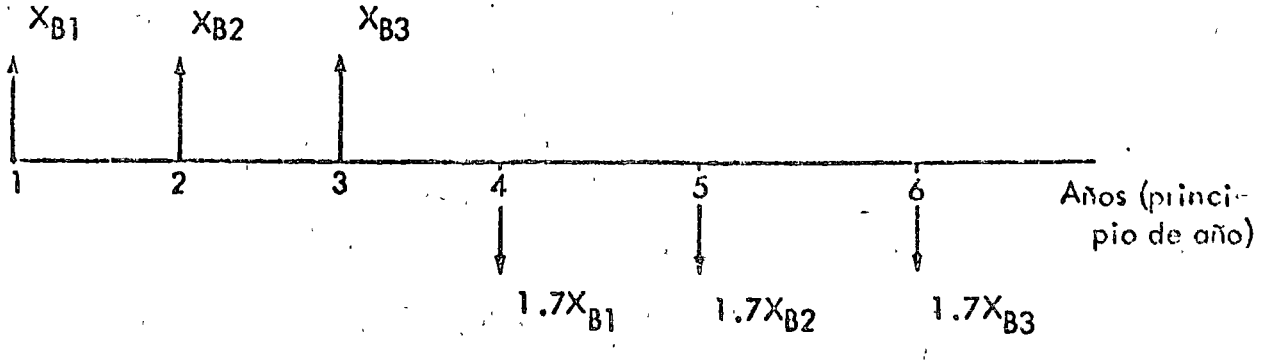
Sea $X_{ij}$ la cantidad de dinero invertida en la actividad i (i = A, B, C, D) en el año j (j = 1, 2, 3, 4, 5).

Las características dadas sobre las formas de inversión de cada una de las actividades -- A, B, C y D pueden mostrarse esquemáticamente como sigue.

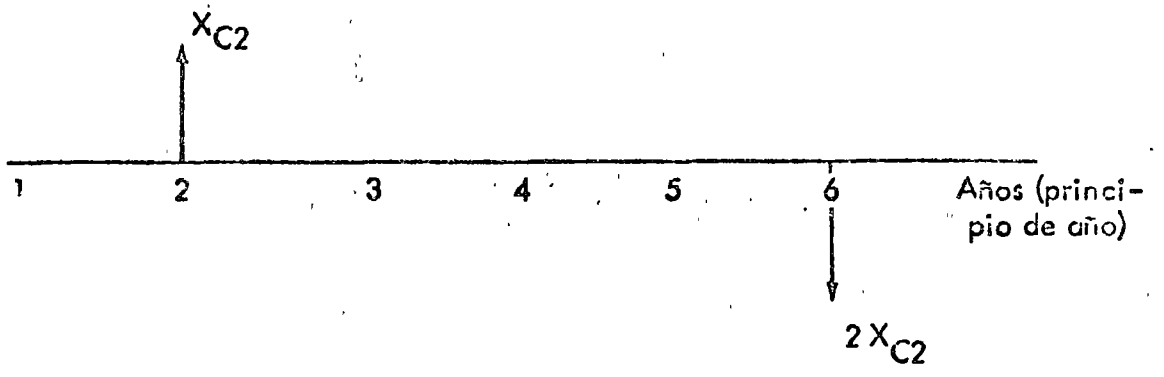
CONDICIONES DE INVERSION EN LA ACTIVIDAD A.



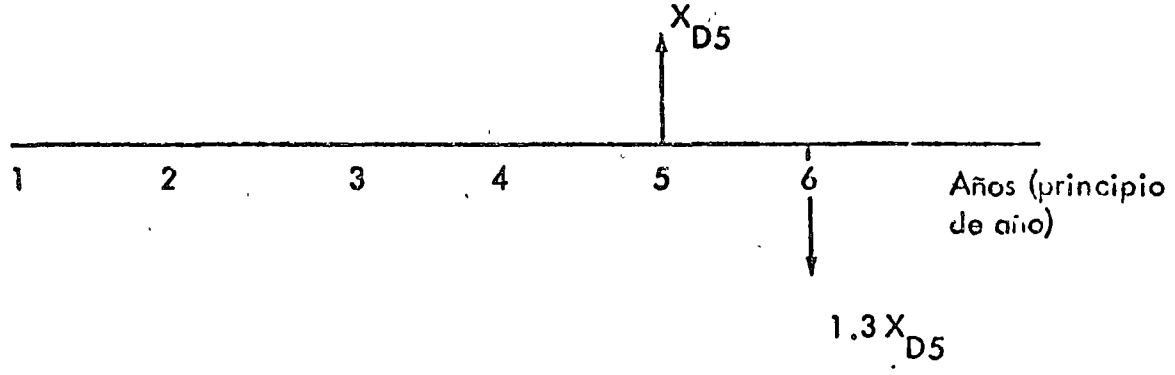

CONDICIONES DE INVERSION EN LA ACTIVIDAD B



CONDICIONES DE INVERSION EN LA ACTIVIDAD C:




CONDICIONES DE INVERSION EN LA ACTIVIDAD D:




La cantidad acumulada en el comienzo del sexto año es la cantidad original (10000) más la ganancia obtenida hasta esta fecha. Por lo tanto, el problema de maximizar la cantidad acumulada de dinero es equivalente a minimizar la ganancia, ya que la cantidad original disponible es una constante que no afecta el valor del dinero acumulado a través de cialquier plan de inversión que se siga.

Si Z es la ganancia total obtenida hasta el comienzo del sexto año, entonces la función objetivo será:

$$\max \; Z = 0.4X_{A1} + 0.4X_{A2} + 0.4X_{A3} + 0.4X_{A4} + 0.70X_{B1} + 0.70X_{B2} + X_{C2} + 0.3X_{D5}$$

Del enunciado del problema, se observa que las restricciones al problema están dadas por la cantidad disponible para invertir en cada año, y por las características de las actividades A, B, C y D. Estas restricciones sobre las inversiones anuales se determinan como sigue:

PRIMER AÑO: La cantidad de dinero invertida en el primer año debe satisfacer :

$$X_{A1} + X_{B1} \leq 10000$$

Si $U_1$ es una variable positiva o cero, que se adiciona a la desigualdad anterior, para que

esta desigualdad llegue a ser una igualdad, entonces

$$x_{A1} + x_{B1} + u_1 = 10000 \qquad (1)$$

$$u_1 \geq 0$$

NOTAS:

1. A la variable que se adiciona a una desigualdad para convertirla en igualdad se le llama una variable de holgura. Entonces $u_1$ es una variable de holgura.
2. Observe que $u_1$ representa la cantidad de dinero no invertido en el primer año, y por lo tanto también representa la cantidad disponible para invertir en el segundo año.

SEGUNDO AÑO: Las inversiones en este año deben satisfacer (observe en las figuras anteriores en que actividades financieras podemos invertir para el segundo año):

$$x_{A2} + x_{B2} + x_{C2} \leq u_1$$

Si introducimos una variable positiva $u_2$ para pasar la desigualdad anterior a igualdad, entonces

$$x_{A2} + x_{B2} + x_{C2} + u_2 = u_1 \qquad (2)$$

$$u_2 \geq 0$$

Obsérvese que la variable $u_2$ es una varible de holgura que representa la cantidad no invertida en el segundo año.

TERCER AÑO: En este año la cantidad de dinero disponible para inversiones proviene de tres fuentes:

i) cantidad no invertida en el segundo año: $u_2$

ii) ganancia obtenida de inversiones anteriores: $0.4x_{A1}$

iii) cantidad recuperada de inversiones anteriores: $x_{A1}$

Observando cada uno de los cuatro diagramas mostrados anteriormente, se tiene que para el tercer año las inversiones deben satisfacer

$$x_{A3} + x_{B3} \leq u_2 + 1.4x_{A1}$$

Introduciendo una variable de holgura $u_3$ $(u_3 \geq 0)$. se tiene que

$$x_{A3} + x_{B3} + u_3 = u_2 + 1.4x_{A1} \qquad (3)$$

$$u_3 \geq 0$$

Otra vez notese que $u_3$ representa la cantidad no invertida en el tercer año.

CUARTO AÑO: En forma similar al análisis del tercer año, se tienen tres fuentes de dinero disponibles:

i) cantidad no invertida en el tercer año: $u_3$

ii) ganancia obtenida entre el tercer y cuarto período: $0.4x_{A2}+0.7x_{B2}$

iii) cantidad recuperada de inversiones anteriores: $\dfrac{x_{A2} + x_{B1}}{u_3+1.4x_{A2}+1.7x_{B1}}$

Por lo tanto, las inversiones en el cuarto período deben satisfacer

$$x_{A4} \leq u_3 + 1.4\, x_{A2} + 1.7\, x_{B1}$$

Introduciendo la variable de holgura positiva $u_4$, se tiene:

$$x_{A4} + u_4 = u_3 + 1.4\, x_{A2} + 1.7\, x_{B1} \qquad (4)$$
$$u_4 \geq 0$$

QUINTO AÑO: La cantidad disponible en este período proviene de:

i) cantidad no invertida en el cuarto año: $u_4$

ii) ganancia entre el tercero y el cuarto período: $0.4x_{A3}+0.7x_{B2}$

iii) cantidad recuperada entre el período 3 y 4to.: $\dfrac{x_{A3} + x_{B2}}{u_4+1.4x_{A3}+1.7x_{B2}}$

Por lo tanto,

$$x_{D5} \leq u_4 + 1.4\, x_{A3} + 1.7\, x_{B2}$$

Si $u_5$ es una variable de holgura, entonces

$$x_{D5} + u_5 = u_4 + 1.4\, x_{A3} + 1.7_{B2} \qquad (5)$$
$$u_4,\ u_5 \geq 0$$

Por lo tanto, nuestro modelo de programación lineal quedaría definido por la función objetivo, dada anteriormente y el conjunto de restricciones definidas por la ecuación del (1) a la (5).

Reescribiendo las ecuaciones anteriores, nuestro modelo de programación lineal queda expresado por:

$$\max z = 0.4x_{A1}+0.4x_{A2}+0.4x_{A3}+0.4_{A4}+0.7x_{B1}+0.7x_{B2}+0.7x_{B3}+x_{C2}^{\circ}+0.3x_{D5} \qquad (0)$$

sujeto a ( s.a.)

$$x_{A1}+ x_{B1}+ u_1 = 10\,000 \qquad (1)$$

$$x_{A2}+ x_{B2}+ x_{C2} + u_2 = u_1 \qquad (2)$$

$$x_{A3}+ x_{B3} = u_2 + 1.4\, x_{A1} \qquad (3)$$

$$x_{A4}+ u_4 = u_3+ 1.4x_{A2} + 1.7x_{B1} \qquad (4)$$

$$x_{D5}+ u_5 = u_4+ 1.4x_{A3} + 1.7x_{B2} \qquad (5)$$

$$x_{Aj} \geq 0 \qquad (j= 1,2,3,4)$$

$$x_{Bj} \geq 0 \qquad (j= 1,2,3,)$$

$$x_{C2} \geq 0$$

$$x_{D5} \geq 0$$

$$u_i \geq 0 \qquad (i= 1,2,\ldots,5)$$

Este problema expresado en la forma particionada

| x | |
|---|---|
| c | z |
| A | b |

$x \geq 0$

se presenta a continuación:

MODELO 5

La Compañía aérea Aeronaves del Pacífico, necesita decidir cuántas aeromozas contratan y adiestran en los próximos 6 meses. Los requerimientos - expresados como horas-vuelo-aeromoza son:

8000 en Enero; 9000 en Febrero; 7000 en Marzo, 10 000 en Abril, 9000 - en Mayo; y 11 000 en Junio.

El entrenamiento para que una aeromoza dé servicio en un vuelo dura un mes, por tanto cada muchacha debe contratarse por lo menos un mes antes de ser necesitada.

El entrenamiento requiere de 100 horas de supervisión de aeromozas ya -- entrenadas por lo tanto disponemos de 100 horas-vuelo-aeromoza menos, durante un mes por cada aeromoza en entrenamiento.

Cada aeromoza entrenada puede trabajar hasta 150 horas en un mes y la - compañía tiene 60 aeromozas entrenadas al principio de enero.

Si el máximo tiempo disponible de las aeromozas excede al requerido en el mes (horas vuelo + supervisión) trabajarán menos de 150 horas y no es despedida ninguna. Pero en cada mes, aproximadamente el 10% de las -- aeromozas con experiencia dejan el trabajo por matrimonio u otras razones.

Cada aeromoza entrenada cuesta a la compañía $ 8000.00 al mes y cada - aeromoza en entrenamiento $ 4000.00; tomando en cuenta salarios y otros - beneficios.

a) Formule el problema de contratar y entrenar como un modelo de programación lineal haciendo $x_t$ el número de aeromozas que principian su - entrenamiento en el mes $t$, donde $x_0 = 60$ representa las aeromozas - disponibles al principio de enero. Defina cualquier símbolo adicional - que necesite para expresar las variables de decisión

b) El inciso anterior supone un horizonte de 6 meses. Suponga que se --- agregan requerimientos de julio al modelo, por ejemplo 10000 horas. (Cambiaría necesariamente la solución para los meses anteriores encontrada anteriormente? Explíquelo.

Sea $x_t$ el número de personas contratadas que principian su entrenamiento al inicio del mes t ($t = 1, 2, \ldots, 6$).

Sea $y_t$ el número de aeromozas experimentadas al final del mes t ($t = 1, 2, \ldots, 6$). Nótese que $y_t$ también representa la cantidad de aeromozas experimentadas al inicio del mes $t + 1$.

DISPONIBILIDAD DE AEROMOZAS EXPERIMENTADAS.

Observe que el número de aeromozas experimentadas $y_t$ al final del mes t, está formado por las personas contratadas al inicio de este mes(las cuales fueron entrenadas en el transcurso del mes) más el 90% de las aeromozas experimentadas que había al final del mes anterior $t-1$ (o sea al inicio del mes t), ie :

$$y_t = x_t + .9\, y_{t-1} \qquad (t = 1, 2, \ldots, 6)$$

con

$$y_o = x_o = 60 \qquad (*)$$

ó sea

$$y_1 = x_1 + .9y_o = x_1 + .9x_o \qquad (1)$$

$$y_2 = x_2 + .9y_1 \qquad (2)$$

$$y_3 = x_3 + .9y_2 \qquad (3)$$

$$y_4 = x_4 + .9y_3 \qquad (4)$$

$$y_5 = x_5 + .9y_4 \qquad (5)$$

$$y_6 = x_6 + .9y_5 \qquad (6)$$

DEMANDAS DE HORAS DE TRABAJO (VUELOS COMERCIALES Y ENTRENAMIENTO):

La demanda total de horas de vuelo por mes corresponde a la demanda de vuelos comerciales más la demanda de horas para entrenar a las nuevas personas contratadas en el inicio del mes. Para satisfacer esta demanda total en el mes t (inicio del mes t); se dispone de $y_{t-1}$ aeromozas con experiencia, las cuales pueden proporcionar 150 horas cada una de ellas. Por lo tanto, si $D_t$ es la demanda de vuelos comerciales en el mes t, entonces:

Demanda en el mes t : $150\, y_{t-1} \geq D_t + 100\, x_t \qquad (t=1, 2, \ldots, 6)$

con $y_o = x_o$

Introduciendo una variable de holgura a cada ecuación, entonces

$$150 y_{t-1} = D_t + 100 x_t + u_t \qquad (t = 1, 2, \ldots, 6)$$

$$y_o = x_o \qquad (**)$$

$$u_t \geq 0$$

Observe que $u_t$ es una variable de holgura que representa el número de horas disponible no usadas al final del período t. Expresando esta restricción para cada t se tiene que:

Demanda en el mes 1 : $150y_o = 8000 + 100\,x_1 + u_1$
Demanda en el mes 2 : $150y_1 = 9000 + 100\,x_2 + u_2$
Demanda en el mes 3 : $150y_2 = 8000 + 100\,x_3 + u_3$
Demanda en el mes 4 : $150y_3 = 10000 + 100x_4 + u_4$
Demanda en el mes 5 : $150y_4 = 9000 + 100\,x_5 + u_5$
Demanda en el mes 6 : $150y_5 = 12000 + 100x_6 + u_6$

FUNCION OBJETIVO:

Ya que el objetivo de la compañía es determinar cuantas aeromozas contratar en los próximos meses, entonces la función objetivo es minimizar los costos involucrados. Estos costos son los costos de las aeromozas experimentadas más los costos de las aeromozas que están siendo entrenadas. Por lo tanto, la función objetivo está dada por

$$\min \; Z = 8000\left[x_o + y_1 + \ldots + y_6\right] + 4000\left[x_1 + x_2 + \ldots + x_6\right]$$

ya que $x_o = y_o$,

$$\min . Z = 8000\left[y_o + y_1 + \ldots + y_6\right] + 4000\left[x_1 + x_2 + \ldots + x_6\right] \qquad (***)$$

Por lo tanto nuestro modelo de programación lineal para el problema dado, está definido por (*), (**) y (***). La representación de este problema de programación lineal en forma particionada (ó tableau ó tablero) es la siguiente.

| $y_o$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | $x_o$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 800 | 800 | 800 | 800 | 800 | 800 | 800 | | 400 | 400 | 400 | 400 | 400 | 400 | | | | | | | $Z$ (min) |
| | | | | | | | 1 | | | | | | | | | | | | | 60 |
| 1 | | | | | | | -1 | | | | | | | | | | | | | 0 |
| -.9 | 1 | | | | | | | -1 | | | | | | | | | | | | 0 |
| | -.9 | 1 | | | | | | | -1 | | | | | | | | | | | 0 |
| | | -.9 | 1 | | | | | | | 1 | | | | | | | | | | 0 |
| | | | -.9 | 1 | | | | | | | -1 | | | | | | | | | 0 |
| | | | | -.9 | 1 | | | | | | | -1 | | | | | | | | 0 |
| | | | | | -.9 | 1 | | | | | | | -1 | | | | | | | 0 |
| 150 | | | | | | | -100 | | | | | | | -1 | | | | | | 8000 |
| | -150 | | | | | | | -100 | | | | | | -1 | | | | | | 9000 |
| | | 150 | | | | | | | -100 | | | | | | -1 | | | | | 8000 |
| | | | 150 | | | | | | | -100 | | | | | | -1 | | | | 10000 |
| | | | | 150 | | | | | | | -100 | | | | | | -1 | | | 9000 |
| | | | | | 150 | | | | | | | -100 | | | | | | -1 | | 12000 |

$$\left[y_0\; y_1\; y_2\; y_3\; y_4\; y_5\; y_6\; x_0\; x_1\; x_2\; x_3\; x_4\; x_5\; x_6\; u_1\; u_2\; u_3\; u_4\; u_5\; u_6\right] \geq 0$$

| $x_{A1}$ | $x_{B1}$ | $u_1$ | $x_{A2}$ | $x_{B2}$ | $x_{C2}$ | $u_2$ | $x_{A3}$ | $x_{B3}$ | $u_3$ | $x_{A4}$ | $u_4$ | $x_{05}$ | $u_5$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| .4 | .7 | | .4 | .7 | 1 | | .4 | .7 | | .4 | | .3 | | = z (max) |
| 1 | 1 | 1 | | | | | | | | | | | | = 10 000 |
| | | -1 | 1 | 1 | 1 | 1 | | | | | | | | = 0 |
| -1.4 | | | | | | -1 | 1 | 1 | 1 | | | | | = 0 |
| | -1.7 | | -1.4 | | | | | | -1 | 1 | 1 | | | = 0 |
| | | | | -1.7 | | | -1.4 | | | | -1 | 1 | 1 | = 0 |

$$x = \left[ x_{A1}\ x_{B1}\ u_1\ x_{A2}\ x_{B2}\ x_{C2}\ u_2\ x_{A3}\ x_{B3}\ u_3\ x_{A4}\ u_4\ x_{D5}\ u_5 \right] \geq 0$$

NOTA: Las restricciones del ( 1 ) al ( 5 ) pueden expresarse sin variables de holgura, con objeto de expresar estasrestricciones como desigualdades en lugar de igualdades. El procedimiento para obtener estas igualdades es el siguiente:

Obviamente de la ecuación ( 1 ) se tiene

$$x_{A1} + x_{B1} \leq 10\ 000 \qquad (1^1)$$

Sumando ( 1 ) y ( 2 ):

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + u_2 = 10\ 000$$

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} \leq 10\ 000 \qquad (2^1)$$

Sumando ( 1 ), ( 2 ) y ( 3 ) :

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} + u_3 = 10\ 000 + 1.4x_{A1}$$

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} \leq 10\ 000 + 1.4x_{A1} \qquad (3^1)$$

Sumando (1), (2), (3) y (4) :

$$x_{A1}+x_{B1}+x_{A2}+x_{B2}+x_{C2}+x_{A3}+x_{B3}+x_{A4}+u_4 = 10\ 000+1.4x_{A1}+1.4x_{A2}+1.7x_{B1}$$

$$x_{A1}+x_{B1}+x_{A2}+x_{B2}+x_{C2}+x_{A3}+x_{B3}+x_{A4} \leq 10\ 000+1.4x_{A1}+1.4x_{A2}+1.7x_{B1} + 1.4x_{A3} + 1.7x_{B2} \qquad (4^1)$$

Sumando (1), (2), (3), (4) y (5):

$$x_{A1}+x_{B1}+x_{A2}+x_{B2}+x_{C2}+x_{A3}+x_{B3}+x_{A4}+x_{D5}+u_5 = 10\,000+1.4x_{A1}+1.4x_{A2}$$

$$+ 1.7x_{B1}+ 1.4x_{A3}+1.7x_{B2}$$

$$x_{A1}+x_{B1}+x_{A2}+x_{B2}+x_{C2}+x_{A3}+x_{B3}+x_{A4}+x_{D5} \leq 10\,000+1.4x_{A1}+1.4x_{A2}+1.7x_{B1}$$

$$+1.4x_{A3}+1.7x_{B2} \qquad (5^1)$$

Por lo tanto, las desigualdades del $(1^1)$ al $(5^1)$ son las restricciones a nuestro problema estas restricciones pueden obtenerse directamente del contexto del problema sin la introducción de variables de holgura, nuestro problema expresado a través de las restricciones de la $(1^1)$ a la $(5^1)$, queda representado en forma particionada como sigue:

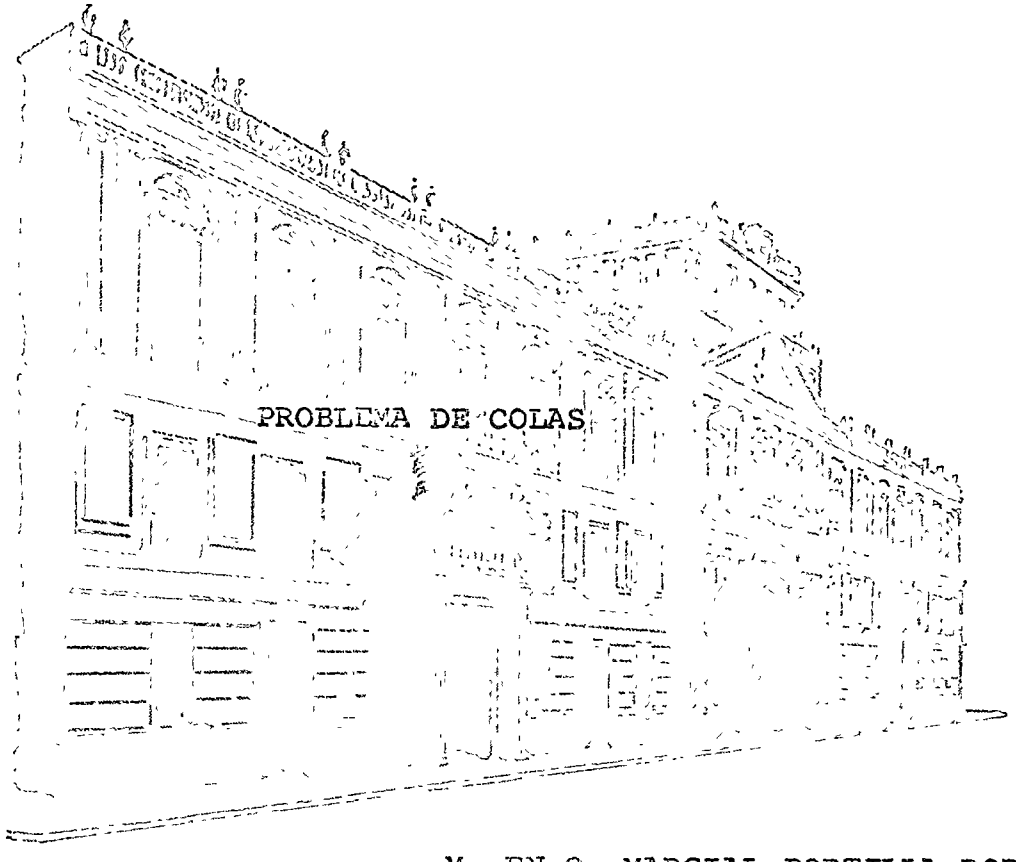
$x_{A1}$ $x_{B1}$ $x_{A2}$ $x_{B2}$ $x_{C2}$ $x_{A3}$ $x_{B3}$ $x_{A4}$ $x_{D5}$

| 4 | .7 | .4 | .7 | 1 | .4 | .7 | .4 | .3 | = z ( max ) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | | | | | | | | $\leq$ 10 000 |
| 1 | 1 | 1 | 1 | 1 | | | | | $\leq$ 10 000 |
| -.4 | 1 | 1 | 1 | 1 | 1 | 1 | | | $\leq$ 10 000 |
| -.4 | -.7 | -.4 | 1 | 1 | 1 | 1 | 1 | | $\leq$ 10 000 |
| -.4 | -.7 | -.4 | -.7 | 1 | -.4 | 1 | 1 | 1 | $\leq$ 10 000 |

$$x = \left[ x_{A1}\ x_{B1}\ x_{A2}\ x_{B2}\ x_{C2}\ x_{A3}\ x_{B3}\ x_{A4}\ x_{D5} \right] \geq 0$$

centro de educación continua
división de estudios superiores
facultad de ingeniería, unam

# INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA




M. EN C. MARCIAL PORTILLA ROBERTSON

FEBRERO-MARZO DE 1977.

# QUESIM

## a model for queueing systems

*is a simulation model for single-phase, multiple-channel queueing systems. The performance of one to nine channels may be investigated under a variety of arrival and service parameters and for a variety of associated costs.*

Queues (waiting lines) are common, everyday occurrences. Queues occur in grocery stores, in banks, in front of movie theaters, during university registration, and so on. Queues build up as a result of an interaction between customers arriving for service and a service facility. Almost everyone has experienced being frustrated by waiting in a line. The purpose of this computer exercise is to allow the user to experiment with situations in which queues occur. The model is concerned with more than waiting time in the queues, however. It allows one to look at the entire system from the operation manager's viewpoint, i.e., not only worrying about the customer but also being concerned about the utilization of facilities and the total cost of operation.

The first section provides background information on queueing concepts and how these concepts may be used by the operations manager as an aid for decision making.

The following sections present a situation in which the computer model may be used as a tool for analysis of some possible alternative management decisions. These illustrative problems include complete instructions on how to use the computer program QUESIM.

## 9.1 THE QUEUEING PROCESS

This section contains an introduction to some queueing concepts. No mathematical formulas are presented, although references are given for those readers interested in a detailed presentation of queueing theory.

**Definition of Queueing Terms**

The queueing process is centered around a *service system* which has one or more *service channels.* Customers (*arrivals*) are drawn from an *input source,* or population. In queueing models, the customer arrivals from the input source are generally characterized by a probability distribution. The symbol commonly used to represent the mean arrival rate of customers is the Greek letter *lambda* ($\lambda$). Any arrival entering the system joins a *queue*, or waiting line (a queue may be of zero length). The customer is selected from the queue for service according to a *queue discipline* or a *priority rule.* Usually, service times follow some probability distribution and the average service rate is commonly represented by the Greek letter *mu* ($\mu$). In order to have a stable queueing process the average service rate ($\mu$) must be greater than the mean arrival rate ($\lambda$). After service is completed the customer exits the system. See Figure 9-1.

Generally, a queueing system is characterized by the following properties:

1. Its arrival pattern.
2. Its service time distribution.
3. Its queue discipline.
4. Its layout, or customer-flow pattern.

**Arrival Patterns**

The feature that makes some situations into queueing situations and other situations non-queueing is the nature of the arrivals to the system. Arrivals are the customers to the service facility, since they are the people or things that need to be processed.

In the situation where all arrivals are on hand, such as a large stock of raw materials, the service center may process arrivals at will, and there is no real queueing problem as such. In other situations where appointments are made ahead of time, there is also no real, or at least visible, queueing problem.

The really interesting problems, those worthy of being studied as queueing systems, exist when arrivals are not controlled or controllable by the service center. The most common assumption made in these cases is that the time intervals between consecutive arrivals are independent random variables. Each arrival is considered to be unaffected by the time at which

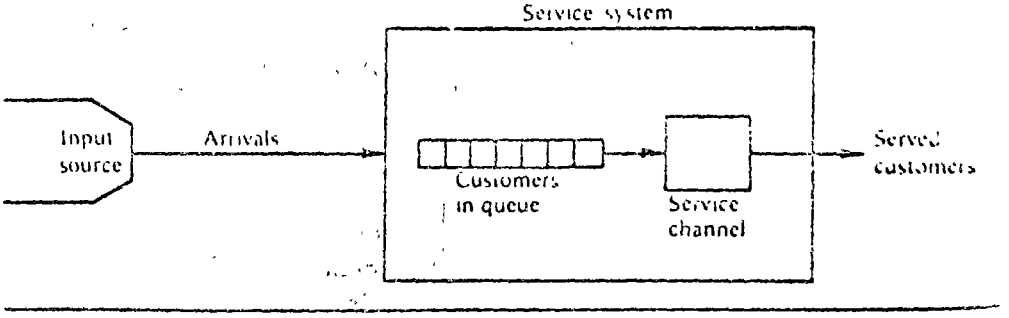



**Figure 9-1** Single queue–single channel, single-phase queueing process.

any other arrival occurs or by the number of arrivals which have already taken place. A good example is the placing of telephone calls when each customer does not really know, or care, who else is placing a call.

Studies of queueing systems have revealed that the time intervals between consecutive arrivals are often distributed according to the negative-exponential distribution. (Longer time intervals have a lower probability of occurrence.) In this case the number of arrivals expected forms a Poisson distribution. Other arrival distributions are possible, but the Poisson distribution is the most frequently occurring distribution.

**Service Time Distributions**

The simplest situation is that in which each arrival requires the same time for service as every other arrival. A vending machine, for example, is usually assumed to have a constant service time. The most commonly assumed service time density distribution, however, is the negative-exponential distribution. The service process may be further characterized as being single-phase (one operation) or multiple-phase (a series of operations).

**Queue Disciplines**

The queue discipline is the priority rule by which waiting jobs are selected from the queue for service. Because this represents a directly controllable decision variable, an extensive amount of research has been done in this area of queueing theory.

The most common priority rule is the first-come-first-served rule (FCFS). According to this rule, the first job to arrive in the queue will be the first job to be serviced. In addition, the following priority rules have received much attention: (1) the random rule selects the job which has the smallest value of a random priority assigned at the time of its arrival, and (2) the shortest operation time rule (SOT) selects from the queue the job which requires the least processing time at that service center.

**Layout or Customer-Flow Patterns**

The layout or flow pattern of a queueing system is largely determined by the specific servicing requirements of the arriving population, and by the physical limitations of the service facility.

This factor of specific servicing requirements is important when the job *must* be processed through a specific service channel or through a particular sequence of operations. If the job has no specific routing requirements, however, this factor becomes negligible. If the job requires several operations, for example, it is possible that the sequencing of these operations is of no consequence.

The physical limitations of the service facility are important, as they affect the facility layout. These physical limitations impose an additional constraint when they tend to limit waiting areas for jobs. As an example, one may prefer to have a single-queue, multiple-channel service facility, but adequate waiting space may not be available. The following arrangements are some examples of system geometry or job-flow patterns: single queue–single channel (Figure 9-1), single queue–multiple channels in parallel (Figure 9-2), multiple queue–multiple channels (Figure 9-3), service centers in tandem (Figure 9-4), and service centers in a network.

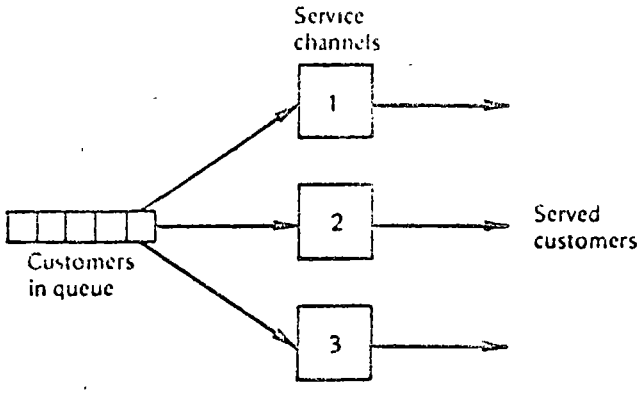



Figure 9-2 Single queue–multiple channel, single-phase.

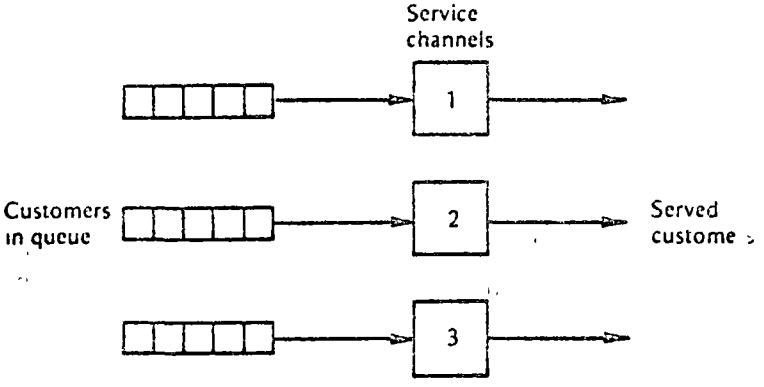



Figure 9-3 Multiple queue–multiple channel, single phase.

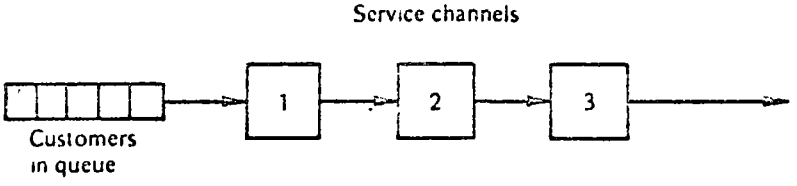



Figure 9-4 Single queue–single channel, multiple-phase.

**Analysis for Management Decision**

One of the primary purposes for studying queueing theory is its predictive capabilities. In turn, this predictive capability is relevant to the design and control of operation systems. Some of the operational characteristics of a queueing system which may be of interest to a manager are the distributions of:

1. Queue length.
2. Customer waiting time (in queue and/or in the system).
3. Idle time of service facilities.
4. Number of customers in the service system.

These distributions may be described by their mean value, standard deviation, and the probability that the variable exceeds a specific value. With

information of this nature the queueing system could then be designed (or altered) so that

1. The resulting operational characteristics are within acceptable limits.
2. An economic criterion such as cost (or profit) may be minimized (or maximized).

An economic criterion may be established if one can associate dollar values with arrivals and service. For example, if one knew the revenues per arrival and the cost of service, it would be possible to set up a profit-maximization objective. Or, if the cost of an arrival waiting in line (or in the system) and the cost of service are known, the measure of effectiveness could be a cost-minimization function.

For solving operational problems which may be characterized as a queueing process there are two methods available. First, if the arrival and service time distributions are well-known mathematical distributions, it is possible to derive formulas for describing the operational characteristics. Secondly, even if the queueing process does not possess properties of well-known distributions, one can still attempt to solve the problem by means of Monte Carlo simulation. In this approach, empirical or assumed data for arrival and service time distributions are used as bases for generating a large number of arrivals and services, on paper. This may be done by hand, but for a large simulation it is most often done on a computer.

Since QUESIM is a computer simulation model for waiting lines, the development of the analytical formulas for solving these problems will not be presented here. For readers interested in the mathematical development of these formulas see Hillier and Lieberman, Morse, and Saaty, in the references for this exercise.

## 9.2 JOHN'S ICE CREAM SHOPPE

In this section, a sample problem suitable for the application of the computer model QUESIM will be presented. The QUESIM model itself will then be described. This description will point out to the user the types of systems for which QUESIM is applicable and the control options available to the user. Following the above, detailed descriptions of both the computer input and the computer output for the sample problem will be given.

**Problem Statement.** John Entrepreneur, who is a senior at the local college of business administration, is going to open a campus ice cream shoppe. The store features seventy-eight varieties of ice cream (more than double the number of his competitor), and pretty coed's to serve them. Even though John's store will have more to offer than his competitor's store, called The Establishment, it is hypothesized that in the first few months local business will be roughly divided between both stores.

John also feels that the confection industry has a high index of substitutability and the improved availability of ice cream due to the opening of his shoppe in the near future will marginally increase the gross sales of both stores, rather than merely dividing the present sales market. John's marketing research efforts have turned up the following information:

the distribution of customer arrivals into his shoppe will most likely follow a Poisson distribution, with a mean arrival rate of 60 people per hour. John has planned that upon arrival in his shoppe each customer will take a sequential number and await his turn for service. This will facilitate his servicing of customers on a first-come-first-served basis.

John's preliminary analyses also show that one server (whose wages are $1.80 per hour) can be expected to service about 30 customers per hour, following a negative-exponential distribution. Moreover, John's marketing research shows that when the service rate is slower than the arrival rate customers will leave rather than wait in a long line. John estimates that his cost (possible opportunity cost) will be $0.05 for every minute that a customer must wait.

Thus far, however, John has not been able to ascertain from his research data the optimum number of servers to have on duty. Logically, he knows that more than one server is required, because the mean arrival rate is twice the mean service rate. Furthermore, he realizes that these rates are mean rates of probability distributions, not constant rates.

John recognizes that his problem is one of design, i.e., how many service channels to provide in the above queueing system in order to minimize total costs. He could, in fact, actually operate his shoppe with one server, two servers, three servers, etc., each for a period of time, and calculate his total costs. However, John has an alternative, and that is to simulate his ice cream shoppe operations in a compressed time period with QUESIM. Using QUESIM, John could experiment with using one, two, three servers, and so on, until the model indicates to John the optimum number of service channels to have in his service facility. Moreover, the simulation can be done in a short period of time, as opposed to waiting for weeks of empirical data from actual operations.

## The Quesim Model

QUESIM is a computer model developed for simulating single-queue, single-phase, parallel queueing systems, that is, the type of system proposed by John Entrepreneur. The model determines, through an iterative process, the optimal number of service channels to allocate to a service facility. The configuration, or design, of the service facility is under the control of the user; hence, the user must specify the following characteristics of the queueing system to be studied:

1. The arrival distribution, the mean arrival rate, and the costs associated with arrivals waiting in line.
2. The service time distribution, the mean service time, and the costs of idle servers.
3. The simulation control limits, or the initial and maximum number of service channels (up to nine) which may be considered available to the service facility during a simulation run, and the maximum length of time for the simulation to run.

The above information on arrivals, services, and simulation control limits comprises the input for the computer program, QUESIM.

The computer output includes, at the top of the page, the user's

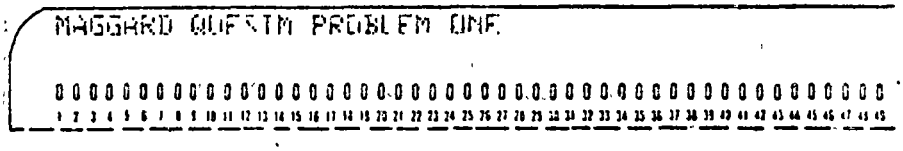
input data as a means of identifying the user's (problem) output. Following this identification information, the first twenty simulated arrivals and services are printed in a tabular form. The program then prints out statistics for the number arrived, number serviced, actual simulation-run time, maximum length of queue, mean length of queue, mean waiting time in queue, percent utilization of service facilities, waiting time (in queue) costs, idle service time costs, and total costs of operations. This data is printed out for each iterative number of service channels and the simulation is terminated when either the number of service channels has reached the maximum allowed or the total cost of the system with $N$ servers (channels) exceeds the total cost with $N-1$ servers.

An example of the application of program QUESIM to John Entrepreneur's problem is given next.

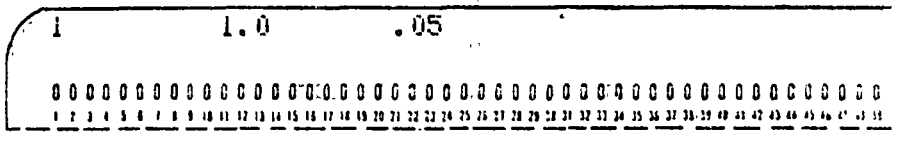
**Computer Input —John's Ice Cream Shoppe**

Shown below are the input data cards required by QUESIM for solving John's Ice Cream Shoppe Problem. In all, four data cards are required to run program QUESIM; they are the user name card, the arrival data card, the service data card, and the simulation control card. Each card will be described in turn.

*User name card* is the first card. This card may contain any identifying information (such as the user's name) which is desired. The identifying information is keypunched in the first forty card columns.

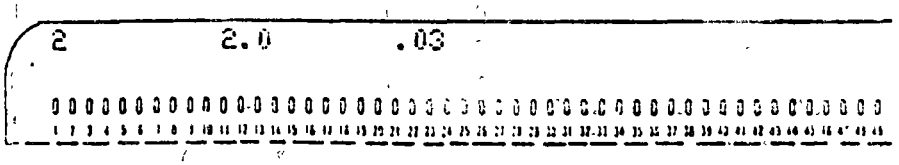



*Arrival data card* is the second data card. This card contains the arrival type (the arrival pattern) in column 1, the mean arrival rate in columns 11-15, and the cost per unit of waiting time in columns 21-25. On this and all data cards, the user *must* keypunch all decimal points. For John Entrepreneur's problem, this card is illustrated below.



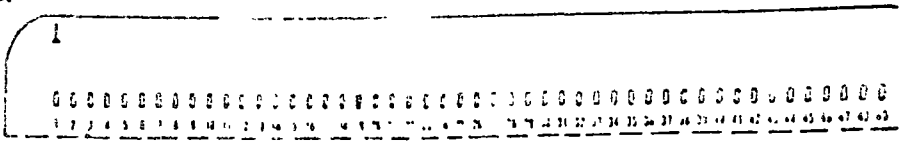

The 1 in card column one is for arrival type code 1, which specifies the Poisson arrival distribution. The 1.0 in column 11 specifies a mean arrival rate of 1.0 customers per time unit, and the .05 in column 21 represents the cost per unit of waiting time per arrival in John's Ice Cream Shoppe.

*Service data card* is the third data card. This card contains the service type (the service time distribution) in column 1, the mean service time in columns 11-15, and the cost per unit of idle service time in columns 21-25. For John Entrepreneur's problem, this data is shown below.



The 2 in column [illegible] negative exponential service time distribution, and the 2.0 [illegible] the mean service time.

The *simulation* [illegible] is the fourth data card. This card contains the beginning number of service channels in column 1, the maximum number of service channels in column 11, and the simulation-run time in columns 21-25. For John Foote, Jr.'s problem, the simulation control card is shown below. The basic unit of time for this simulation data is in minutes.




The complete listing of the four input data cards, one card per typewritten line, is shown in Figure 9-5.

```
MAGGARD QUESIM PROBLEM ONE
1          1.0          .05
2          2.0          .03
1          9            60.
```

Figure 9-5 Computer input–John's Ice Cream Shoppe.

Computer Output —John's Ice Cream Shoppe

Shown in Figure 9-6 is the computer output for the first iteration (one channel) of John's Ice Cream Shoppe Problem. At the top of the computer output, the information input on the four data cards is printed out.

Below the problem identification information, the first twenty arrivals of the actual simulation are tabulated. This does not mean that only twenty customers arrived in 60 time units. The computer program is designed to print out a table for only the first twenty arrivals, regardless of the simulation-run time specified.

In the simulation table the column headings are the customer's arrival time and his departure time at his respective channel number. The program specifies that the first customer always arrives at time zero, hence in Figure 9-6 customer 1 arrived at time 0.0. The randomly selected service time for the first arrival was 2.6 time units; thus, the first arrival departed from channel number 1 (in this case the only service channel) at time 2.6. Customer 2 also arrived at time 0.0 and required 0.7 minutes of processing time, thus exiting the system at simulation time 3.3 The arrival and departure times printed out are rounded off to the nearest one-tenth (0.1) time unit. Consequently, this rounding off makes the arrival and departure times of some customers appear to occur at the same time, for example, customer number 4 in Figure 9-6.

Below the sample simulation data, the summary queueing statistics and operation costs, which are the information of most interest, are printed out. The first line tells how many arrivals entered the system and how many were serviced by the end of the specified simulation-run time. The number of arrivals will always exceed the number served, by at least one. This is because in program QUESIM arrivals occur before departures and elapsed simulation time is checked only when departures occur. Next, the maximum

```
PROGRAM QUESIM FOR MAGGARD QUESIM PROBLEM ONE
ARRIVAL TYPE 1 RATE =      1.00  COST =      .05
SERVICE TYPE 2 TIME =      2.00  COST =      .03
NO. CHANNELS START 1 MAX 9
MAX TIME      60


FIRST TWENTY OCCURANCES FOR 001SERVICE CHANNELS
ARRIVAL      -------DEPARTURE TIME AT CHANNEL NUMBER-------
TIME---      ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE
     0       2.6
    .0       3.3
    .7       4.6
   5.1       5.1
   6.9       8.5
   8.0       8.6
   8.1       8.8
   9.1       9.4
  11.2      12.7
  11.4      12.8
  14.6      15.2
  15.1      17.0
  17.0      18.2
  17.4      21.0
  19.3      24.6
  19.5      26.0
  20.1      26.0
  20.2      27.3
  20.8      28.3
  21.4      29.4

AFTER    52 ARRIVED     36 SERVED     60 TIME UNITS
QUEUE-MAXIMUM LENGTH =      14
     -MEAN LENGTH    =     5.5
     -MEAN WAIT TIME =     6.4
SERVICE UTILIZATION  =    89.6 PERCENT
COSTS-WAIT IN QUEUE  331.1 UNITS AT $    .05 = $     16.55
      IDLE SERVICE     6.2 UNITS AT $    .03 = $       .19
      TOTAL COST OF OPERATIONS                 $     16.74
```

Figure 9-6 Computer output—John's Ice Cream Shoppe.

length of the queue, the mean number of customers in queue, and the mean waiting time in the queue are printed out. Then, the percent utilization of the service facilities is printed out. This is followed by the cost calculations. First, the total waiting time cost, which is the total waiting time of all customers in the system multiplied by the cost per unit waiting time, is given. The next line gives idle time cost, calculated as units of idle time (in the service facility) multiplied by the cost per unit of idle time.

The last line printed out is the total cost of operations, which is the idle time cost and the waiting time cost added together. It is this value that the simulator uses as a basis for comparing each iteration to the previous iteration for the purpose of determining when to terminate the simulation.

In the sample problem, the basic issue facing John Entrepreneur is the number of service channels (servers) he should have in his ice cream shoppe. The summary statistics for one channel (Figure 9-6) indicate that he

should at least have more than one server. The average length of the wait line is greater than five customers (as many as fourteen at one time), and customers must wait, on the average, more than 6 time units (minutes?) to be served. In real life, one would not expect ice cream shoppe customers to be so patient! Moreover, as one would expect with so many customers waiting, the server is busy almost 90 percent of the time.

For the cost structure given in the problem, the waiting time cost of the customers far exceeds the service time cost of the single server. The total cost of $16.74 consists almost entirely of waiting time costs.

The *simulation control card* for this problem requested that program QUESIM initially simulate John Entrepreneur's system with one service channel and continue to simulate his sytem, adding one additional service channel each run, until the simulation terminated. The simulation will terminate either when the total cost of the system with $N$ servers exceeds the total cost with $N - 1$ servers or when the maximum number of allowable service channels has been reached.

For John's problem, the effect of adding additional servers in his shoppe is shown below. The cost figures in Figure 9-7 come directly from the computer printouts for the solution to John's problem. These figures, like all simulation results, are a function of a random number generator which may be different for each computer. Hence, these figures may be slightly different on different computers.

| *Number of channels* | *Waiting cost ($)* | *Idle time cost ($)* | *Total cost of operations ($)* |
|---|---|---|---|
| 1 | 16.55 | 0.19 | 16.74 |
| 2 | 7.14 | 0.49 | 7.63 |
| 3 | 0.26 | 2.61 | 2.87 |
| 4 | 0.96 | 3.71 | 4.67 |

**Figure 9-7** The total cost of operations for one to four service channels.

Figure 9-7 shows that adding a second service channel dramatically reduces the cost of customer waiting from $16.55 to $7.14. Adding a third service channel (server) continues to decrease waiting time cost and is the number of servers which minimizes the total cost of operations. Thus the solution to John Entrepreneur's problem, as given by QUESIM, is for John to provide three servers in his shoppe operating under his expected steady state conditions. Keep in mind, however, that if we treat each simulation time unit as one minute, we have simulated only one hour of operations. Thus, John may want better data and/or a longer simulation run before coming to a definite conclusion about his problem.

## 9 JOHN'S ICE CREAM SHOPPE REVISITED

**John's New Data**

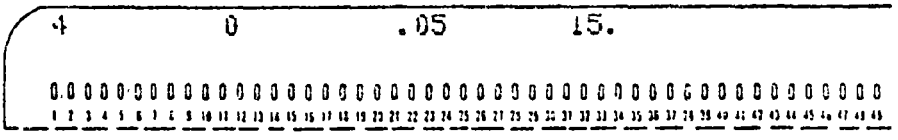
Grand Opening!! John Entrepreneur's ice cream shoppe is now open. He is so excited about testing out his previous conclusions on the number of servers to have in his shoppe that after observing only the first fifteen arrivals

he rushes over to the local computer center to simulate QUESIM with his empirical data. In recording his data, John kept track of both the time that each arrival entered the shoppe and the time it took to serve each customer. To John's surprise, he found that all of the service times were a constant, three minutes. John now wants to use program QUESIM with the data that he has obtained. The required input and the resulting computer output for John's new data are described next.

Computer Input —John's Ice Cream Shoppe Revisited

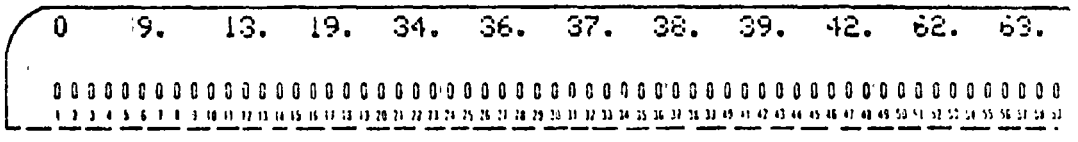
The *user name card* is always the first card in the data deck.

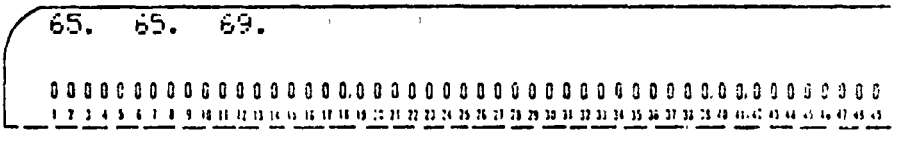
The *arrival data card* is the second card in the data deck. For this problem it reads as follows:



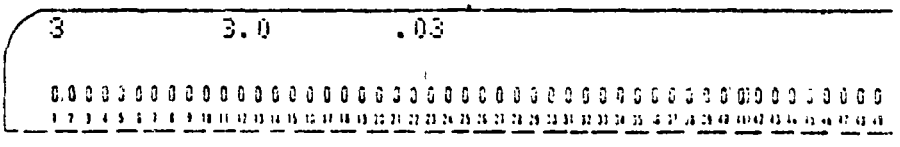

Note that in column 1 of the *arrival data card* the code number is 4, which specifies that all arrival times are to be "read in" from additional data cards. The cost of waiting is punched in columns 21-25. The number of arrivals to be read in is indicated by the value specified in columns 31-33. Although this value was not required in the previous problem, here it has a specified value of 15.

*Read-in arrivals* are keypunched twelve per card and immediately follow the arrival data card. Since John has fifteen data values he needs two read-in arrival data cards, with twelve data points on the first card and three on the second card. These two cards have a format of 12F5.0 and are shown below for Problem Two. Notice that it is the actual arrival times that are recorded, not the elapsed time between arrivals.



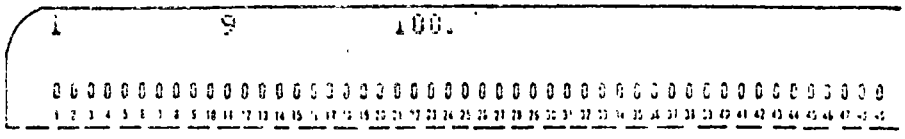

The *service data card* follows the arrival data cards. John noted that each service required exactly three minutes, i.e., a constant service time. Code 3 keypunched in column 1 specifies a constant service time. The service data card now reads as follows:




This card requests a constant service time of 3.0 time units per arrival at a cost of $0.03 for each unit of idle service time.

The *simulation control card* is again the last card in the data deck.

John now specifies a run with from one to nine channels and a simulation time limit of 100 units. These parameters are shown below.




```
MAGGARD QUESIM PROBLEM TWO
4              0              .05            15.
0     9.      13.     19.     34.     36.     37.     38.     39.     42.     62.     63.
65.   65.     69.
3              3.              .03
1              9.             100.
```

Figure 9-8 Computer input—John's Ice Cream Shoppe Revisited.

```
PROGRAM QUESIM FOR MAGGARD QUESIM PROBLEM TWO
ARRIVAL TYPE 4 RATE =          0  COST =       .05
  15 ARRIVALS READ IN AS FOLLOWS
    0     9    13    19    34    36    37    38    39    42    62    63
   65    65    69
SERVICE TYPE 3 TIME =      3.00  COST =       .03
NO. CHANNELS START 1 MAX 9
MAX TIME    100


FIRST TWENTY OCCURANCES FOR ***1***SERVICE CHANNELS
ARRIVAL      -------DEPARTURE TIME AT CHANNEL NUMBER-------
TIME---      ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE
     0       3.0
   9.0      12.0
  13.0      16.0
  19.0      22.0
  34.0      37.0
  36.0      40.0
  37.0      43.0
  38.0      46.0
  39.0      49.0
  42.0      52.0
  62.0      65.0
  63.0      68.0
  65.0      71.0
  65.0      74.0
  69.0      77.0

***WARNING****OUT OF DATA BEFORE TIME LIMIT****

AFTER     17 ARRIVED     15 SERVED    100 TIME UNITS
QUEUE-MAXIMUM LENGTH =       3
     -MEAN LENGTH    =      .4
     -MEAN WAIT TIME =     2.3
SERVICE UTILIZATION  =    45.0 PERCENT
COSTS-WAIT IN QUEUE     39.0 UNITS AT $   .05 = $       1.95
      IDLE SERVICE      55.0 UNITS AT $   .03 = $       1.65
      TOTAL COST OF OPERATIONS                $       3.60
```

Figure 9-9 Computer output—John's Ice Cream Snoppe Revisited.

The complete input data deck, one card per typewritten line, now appears as shown in Figure 9-8.

**Computer Output —John's Ice Cream Shoppe Revisited**

Shown in Figure 9-9 is the computer output for the results of this problem for one channel. On the computer output, the identification information at the top of the page is changed to reflect the changes in the input data. The arrival type, service type, and simulation-run time are different from the previous situation. Furthermore, the simulation table is for only fifteen arrivals and services, as that is the total number of arrivals read in. Notice that on this output a warning message (***WARNING****OUT OF DATA BEFORE TIME LIMIT****) has been printed out. This message is printed out by QUESIM because the last arrival occurred at time 69, which was before the simulation time limit of 100 units. Moreover, the service facility was idle from time 77 to time 100; thus, the actual idle time in the service facility is overstated in the summary statistics. When this message appears, the user of QUESIM must exercise some care in the interpretation of the results or go back and rerun with a lower, more realistic, time limit. The summary statistics and cost information printed out below the simulation table is the same for all computer printouts.

## REFERENCES

Bhatia, A., and A. Garg, "Basic Structure of Queueing Problems," *The Journal of Industrial Engineering*, vol. 14, no. 1 (January-February 1963), pp. 13-17.

Buffa, E. S., *Modern Production Management*, 3d ed. New York: Wiley, 1969.

, *Operations Management: Problems and Models*, 2d ed., New York: Wiley, 1969.

Gavett, J. W., *Production and Operations Management*, New York: Harcourt, Brace & World, 1968.

Hillier, F. S., and G. J. Lieberman, *Introduction to Operations Research*, San Francisco, Calif.: Holden-Day, 1967.

Lee, A. M., *Applied Queueing Theory*, London: Macmillan, 1966.

McMillan, C., and R. F. Gonzalez, *Systems Analysis*, rev. ed., Homewood, Ill.: Irwin, 1968.

Morse, P. M., *Queues, Inventories and Maintenance*, New York: Wiley, 1958.

Panico, J. A., *Queueing Theory*, Englewood Cliffs, N.J. : Prentice-Hall, 1969.

Richmond, S. B., *Operations Research for Management Decisions*, New York: Ronald, 1968.

Saaty, T. L., *Elements of Queueing Theory with Applications*, New York: McGraw-Hill, 1961.

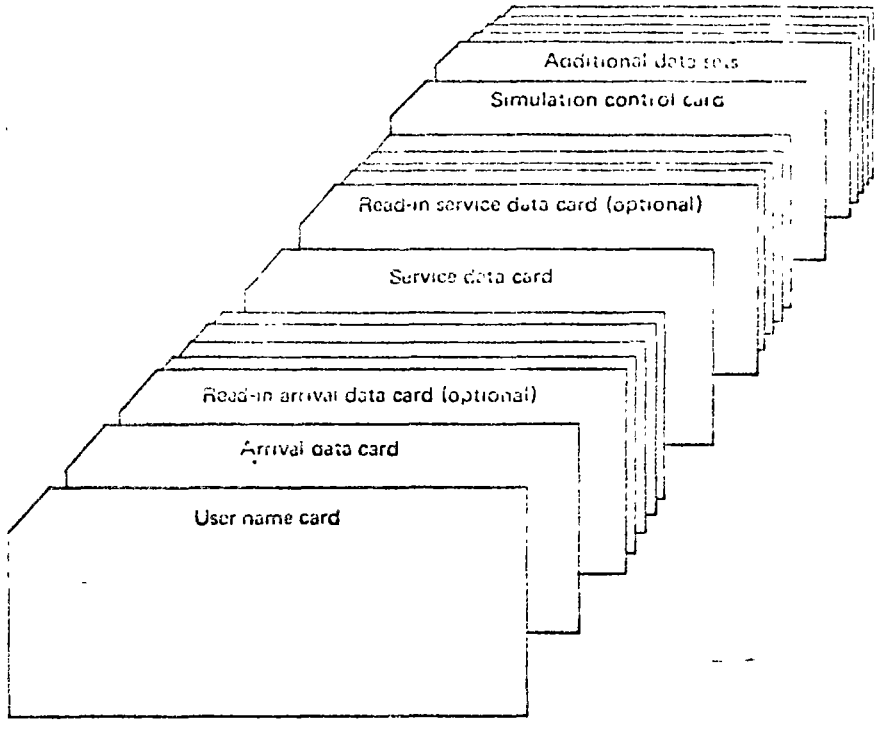
## 9.4 QUESIM DATA DECK STRUCTURE

**Arrival Cards**

The arrival data card must always be included

| Columns | Format | Item |
|---|---|---|
| 1 | I1 | code* for distribution |
| 2-11 | F5.0 | mean arrival rate per unit time |
| 12-21 | F5.0 | cost per unit of waiting time |
| 22-33 | I3.0 | no. of arrivals to be read in |

* The second arrival data card is optional, depending upon the data. Its format is 12F5.0, so that arrival times are punched, 12 to a card, in columns 1-5, 6-10, 11-15, 16-20, etc.




3. Service data card must always be included.

| *Card columns* | *Format* | *Item* |
|---|---|---|
| 1 | I1 | code* for distribution |
| 11-15 | F5.0 | mean service time |
| 21-25 | F5.0 | cost per unit of idle time |
| 31-33 | F3.0 | no. of service times to be read in |

4. Read-in service data card is optional, depending upon the data. Its format is 12F5.0, so that service times are punched, 12 to a card, in columns 1-5, 6-10, 11-15, 16-20, etc.

5. Control card must always be included.

| *Card columns* | *Format* | *Item* |
|---|---|---|
| 1 | I1 | beginning no. of service channels |
| 11 | I1 | maximum no. of service channels |
| 21-25 | F5.0 | simulation run time |

*Code for distribution
1 the Poisson distribution
2 the negative-exponential distribution
3 a constant rate
4 historical input data, read-in

## 6.5 QUESIM PROGRAM LISTING

```
C      PROGRAM QUESIM                                                IBM   1
C      COPYRIGHT JUNE 1970 ROY D HARRIS                               A    2
C      THIS VERSION FOR THE IBM/360                                  IBM   3
C     DICTIONARY OF VARIABLES                                         A    4
C     AITIME          THE HOURS OF SYSTEM IDLE TIME - TOTAL           A    5
C     ANUM,SNUM       NUMBER OF READ IN ARRIVALS AND SERVICE          A    6
C     AR(500)         AN ARRAY OF READ IN ARRIVAL TIMES               A    7
C     ARR RT,ARR TM THE ARRIVAL RATE AND MEAN TIME                    A    8
C     CH              AN ARRAY OF ARRIVAL AND SERVICE ON FIRST 20 CUSTOMERS  A  9
C     CIDLE           THE COST OF SYSTEM IDLE TIME - TOTAL            A   10
C     CUMQUE(100)     AN ARRAY WHICH STORES IDLE CUSTOMER HOURS       A   11
C     COSTS           THE COST PER TIME UNIT OF IDLE SERVICE          A   12
C     COSTA           THE COST PER TIME UNIT OF IDLE CUSTOMERS        A   13
C     CUSERV,KCUS     THE NUMBER OF THE CUSTOMER BEING SERVED         A   14
C     CWAIT            THE COST OF CUSTOMERS HOURS IN QUEUE- TOTAL    A   15
C     DEP RT,DEP TM THE SERVICE RATE AND MEAN DURATION                A   16
C     HRSNQ           THE HOURS OF CUSTOMER TIME IN QUEUE- TOTAL      A   17
C     I,J             THE CHANNEL NUMBER BEING PROCESSED              A   18
C     IZ              NUMBER OF ARRIVALS WHICH HAVE OCCURED           A   19
C     KA,KS           OPTION CODES FOR ARRIVALS AND SERVICE           A   20
C     N,MAXS          BEGINNING,MAXIMUM NUMBER CHANNELS               A   21
C     PCUTIL          THE PERCENT UTILIZATION OF THE SERVICE FACILITY  A   22
C     QUEUE           THE NUMBER OF CUSTOMERS IN QUEUE AT ANY POINT   A   23
C     SR(500)         AN ARRAY OF READ IN SERVICE TIMES               A   24
C     TCOOP           THE TOTAL COST OF THE SYSTEM                    A   25
C     TIME,TTIME      CLOCK TIME,MAX SIMULATION TIME                  A   26
C     TNARV           THE LATEST ARRIVAL TIME                         A   27
C     TNDPR           THE DEPARTURE TIME OF THE LATEST DEPARTURE      A   28
C                     MAY BE SET ARTIFICALLY FOR PROGRAM EFFICIENCY   A   29
C     XMNTN,XMNTX     MEAN WAIT TIME,MEAN NUMBER IN QUE               A   30
      COMMON ALPHA(1 ), ANUM, AR(500), ARRRT, ARRTM, CH(20, 10), CUMQUE(  A   31
     1100), CUMUTL, CUSERV, DEPRT, DEPTM, I, IUSERV, IZ, KA, KCUS, KS, N  A   32
     2, NFLAG,QUEUE,SNUM, SR(500), STATUS(9), T, TIME, TTIME, TNARV, TND  A   33
     3PR(9)                                                            A   34
      NIN = 5                                                         IBM   35
      NOUT = 6                                                        IBM   36
C     READ AND PRINT STUDENT NAME CARD                                A   37
    1 READ (NIN,23) ALPHA                                             A   38
      WRITE (NOUT,24) ALPHA                                           A   39
C     READ AND PRINT ARRIVAL DATA CARD                                A   40
      READ (NIN,25) KA, ARRRT, COSTA, ANUM                            A   41
      WRITE (NOUT,26) KA, ARRRT, COSTA                                A   42
C     READ ARRIVAL STATISTICS                                         A   43
      IF (ANUM .LE.  .0) GO TO 2                                      A   44
      NUM = ANUM                                                      A   45
      ARRRT = 1.234                                                   A   46
      KA = 4                                                          A   47
      READ (NIN,27) (AR(I), I = 1, NUM)                               A   48
      WRITE (NOUT, 8) NUM                                             A   49
      WRITE (NOUT,29) (AR(I), I = 1, NUM)                             A   50
C     SET ARRIVAL TIME AT INVERSE OF ARRIVAL RATE                     A   51
    2 ARRTM = 1.0/ARRRT                                               A   52
C     READ AND PRINT SERVICE DATA CARD                                A   53
      READ (NIN,25) KS, DEPTM, COSTS, SNUM                            A   54
      WRITE (NOUT,30) KS, DEPTM, COSTS                                A   55
      IF (SNUM .LE. 0.0) GO TO 3                                      A   56
      NUM = SNUM                                                      A   57
      DEPTM = 1.234                                                   A   58
      KS = 4                                                          A   59
      READ (NIN,27) (SR(I), I = 1, NUM)                               A   60
      WRITE (NOUT,31) NUM                                             A   61
```

```
      WRITE (NOUT,29) (SR(I), I = 1, NUM)
C     SET SERVICE RATE AT INVERSE OF SERVICE TIME
      DEPRT = 1.0/DEPTM
C     READ SIMULATION CONTROL CARD AND PRINT
      READ (NIN,32) N, MAXS, TTIME
      WRITE (NOUT,33) N, MAXS
      WRITE (NOUT,34) TTIME
C     CHECK SIMULATION RUN LIMITS
      IF (TARRRT .LE. 0.0) GO TO 7
      IF (DEPTM .LE. 0.0) GO TO 7
      IF (TTIME .LE. 0.0) GO TO 7
      IF (N .EQ. 0) GO TO 7
      IF (MAXS .LT. N) GO TO 7
      IF (KA .EQ.  ) GO TO 7
      IF (KA .GT. 4) GO TO 7
      IF (KS .EQ. 0) GO TO 7
      IF (KS .GT. 4) GO TO 7
      IF (ANUM .EQ. 0.0) GO TO 5
      IF (ANUM .GT. 500.0) GO TO 7
      NUM = ANUM
      DO 4 I = 2, NUM
      J = I-1
      IF (AR(J) .GT. AR(I)) GO TO 7
 4    CONTINUE
 5    IF (SNUM .EQ. 0.0) GO TO 8
      IF (SNUM .GT. 500.0) GO TO 7
      NUM = SNUM
      DO 6 I = 1, NUM
      IF (SR(I) .LT. 0.0) GO TO 7
 6    CONTINUE
      GO TO 8
C     PRINT OUT DATA ERROR MESSAGE
 7    WRITE (NOUT,35)
      WRITE (NOUT,36)
      GO TO 1
C     END OF INPUT DATA CHECK
 8    BCOOP = 999999.9
C     SIMULATION OF A GIVEN NUMBER OF CHANNELS (N) BEGINS HERE
C     INITALIZE SYSTEM FOR NEXT SIMULATION RUN
 9    TIME = 0.0
      TNARV = 0.0
      QUEUE = 0.0
      CUMUTL = 0.0
      CUSERV = 0.0
      IZ = 0
      KCUS = 0
      NFLAG = 0
      IUSERV = 0
      SET = RAND(1234567)
      DO 10 M = 1, 100
 10   CUMQUE(M) = 0.0
      DO 11 L = 1, N
      TNDPR(L) = 999999.9
 11   STATUS(L) = 0.0
      DO 12 I = 1, 20
      DO 12 J = 1, 10
 12   CM(I, J) = 0.0
C     PRINT HEADINGS FOR RESULTS
      WRITE (NOUT,37)
      WRITE (NOUT,38) N
      WRITE (NOUT,39)
      WRITE (NOUT,40)
```

```
C     SET FIRST ARRIVAL OCCURANCE AT TIME ZERO                                  A 124
      TNARV = 0.0                                                               A 125
      CH(1, 1) = 0.0                                                            A 126
      IZ = IZ+1                                                                 A 127
C     MAIN SIMULATION BRANCH POINT                                              A 128
C     CHECK EACH CHANNEL IN TURN FOR POSSIBLE DEPARTURE                         A 129
C     IF ALL CHANNELS ARE IDLE (TNDPR = 999999.9) THEN GO TO ARRIVE             A 130
C     IF ALL CHANNELS ARE BUSY (TNARV IS .GE. TNDPR) THEN GO TO ARRIVE          A 131
C     IF A DEPART IS NEXT (TNDPR IS .GE. TNARV) THEN GO TO DEPART               A 132
C     SET AND IVALUE KEEP MULTIPLE DEPARTURES IN CORRECT TIME SEQUENCE          A 133
 13   SET = 888888.                                                             A 134
      DO 14 I = 1, N                                                            A 135
      IF (TNDPR(I) .GT. TNARV) GO TO 14                                         A 136
      IF (TNDPR(I) .GT. SET) GO TO 14                                           A 137
      SET = TNDPR(I)                                                            A 138
      IVALUE = I                                                                A 139
 14   CONTINUE                                                                  A 140
      I = IVALUE                                                                A 141
      IF (SET .LT. 888888.) GO TO 15                                            A 142
      CALL ARRIVE                                                               A 143
      GO TO 13                                                                  A 144
 15   CALL DEPART                                                               A 145
C     ON RETURN FROM DEPART CHECK SIMULATION TIME LIMIT                         A 146
      IF (ITIME .GT. TIME) GO TO 13                                             A 147
C     END OF SIMULATION RUN---PRINT FIRST TWENTY TRIALS                         A 148
      N1 = N+1                                                                  A 149
      IF (CUSERV .GE. 20.0) GO TO 16                                            A 150
      NXX = CUSERV                                                              A 151
      GO TO 17                                                                  A 152
 16   NXX = 20                                                                  A 153
 17   DO 18 I = 1, NXX                                                          A 154
 18   WRITE (NOUT,41) (CH(I, J), J = 1, N1)                                     A 155
C     COMPUTE HOURS IN QUEUE FOR SUMMARY PRINTOUT                               A 156
      HRSNQ = 0.0                                                               A 157
      MAXQUE = 0                                                                A 158
      DO 20 M = 2, 100                                                          A 159
      IF (CUMQUE(M) .EQ. 0.0) GO TO 19                                          A 160
      MAXQUE = M-1                                                              A 161
 19   XM = M-1                                                                  A 162
 20   HRSNQ = HRSNQ+(XM*CUMQUE(M))                                              A 163
      IF (MAXQUE .LT. 99) GO TO 21                                              A 164
      WRITE (NOUT,42)                                                           A 165
 21   IF (INFLAG .NE. 76) GO TO 22                                              A 166
      WRITE (NOUT,43)                                                           A 167
 22   XN = N                                                                    A 168
      XIZ = IZ                                                                  A 169
C     PRINT SUMMARY STATISTICS FOR THIS NUMBER (N) CHANNELS.                    A 170
      WRITE (NOUT,44) IZ, CUSERV, TIME.                                         A 171
      WRITE (NOUT, 5) MAXQUE.                                                   A 172
      XMNTX = HRSNQ/TIME                                                        A 173
      WRITE (NOUT, 6) XMNTX                                                     A 174
      XMNTM = HRSNQ/XIZ                                                         A 175
      WRITE (NOUT, 7) XMNTM                                                     A 176
      PCUTIL = ((CUMUTL/TIME)*100.)/XN                                          A 177
      WRITE (NOUT,48) PCUTIL                                                    A 178
      CWAIT = HRSNQ*COSTA                                                       A 179
      WRITE (NOUT,49) HRSNQ, COSTA, CWAIT                                       A 180
      AITIME = (TIME*XN)-CUMUTL                                                 A 181
      CIDLE = AITIME*COSTS                                                      A 182
      WRITE (NOUT,50) AITIME, COSTS, CIDLE                                      A 183
      TCOOP = CIDLE+CWAIT                                                       A 184
      WRITE (NOUT,51) TCOOP                                                     A 185
```

```
C       STOP RUN IF TCOOP INCREASED FROM LAST RUN                          A 192
        IF (TCOOP .GE. OCOOP) GO TO 1                                      A 193
C       STOP RUN IF MAXIMUM NUMBER OF SERVERS REACHED                      A 194
        IF (N .GE. MAXS) GO TO 1                                           A 195
C       UPDATE NUMBER OF CHANNELS AND CURRENT TOTAL COST                   A 196
        N = N+1                                                            A 197
        OCOOP = TCOOP                                                      A 198
C       RETURN FOR NEXT RUN WITH MORE CHANNELS (N)                         A 199
        GO TO 9                                                            A 200
C                                                                          A 201
 23     FORMAT   (10A6)                                                    A 202
 24     FORMAT   (20H1PROGRAM QUESIM FOR ,10A6)                            A 203
 25     FORMAT   (I1,9X,F5.0,5X,F5.0,5X,F3.0)                              A 204
 26     FORMAT   (16H ARRIVAL TYPE ,I1,8H RATE = ,F8.2,8H  COST =,F8.2)    A 205
 27     FORMAT   (12F5.0)                                                  A 206
 28     FORMAT   (1H ,I4,26H ARRIVALS READ IN AS FOLLOWS)                  A 207
 29     FORMAT   (1H ,12F5.0)                                              A 208
 30     FORMAT   (14H SERVICE TYPE ,I1,8H TIME = ,F8.2,8H  COST =,F8.2)    A 209
 31     FORMAT   (1H ,I4,28H SERVICES READ IN AS FOLLOWS)                  A 210
 32     FORMAT   (I1,9X,I1,9X,F5.0)                                        A 211
 33     FORMAT   (20H NO. CHANNELS START ,I1,5H MAX ,I1)                   A 212
 34     FORMAT   (10H MAX TIME ,F6.0)                                      A 213
 35     FORMAT   (35H ****ERROR IN QUESIM DATA CARDS****)                  A 214
 36     FORMAT   (35H ****CORRECT DATA AND TRY AGAIN****)                  A 215
 37     FORMAT   (1H0)                                                     A 216
 38     FORMAT   (31H FIRST TWENTY OCCURANCES FOR **,I1,18H**SERVICE CHANNE A 217
     1LS)                                                                  A 218
 39     FORMAT   (8H ARRIVAL,4X,44H------DEPARTURE TIME AT CHANNEL NUMBER--A 219
     1----)                                                                A 220
 40     FORMAT   (8H TIME---,4X,44HONE TWO THREE FOUR FIVE SIX SEVEN EIGHT A 221
     1NINE)                                                                A 222
 41     FORMAT   (1H ,F6.1,3X,9F5.1)                                       A 223
 42     FORMAT   (50H0****WARNING****QUE EXCEEDED PROGRAM LIMIT OF 99****) A 224
 43     FORMAT   (48H ****WARNING****OUT OF DATA BEFORE TIME LIMIT****)    A 225
 44     FORMAT   (6H0AFTER,I6,8H ARRIVED,F6.0,7H SERVED,F6.0,11H TIME UNITS A 226
     1)                                                                    A 227
 45     FORMAT   (23H QUEUE-MAXIMUM LENGTH =,I7)                           A 228
 46     FORMAT   (23H      -MEAN LENGTH    =,F7.1)                         A 229
 47     FORMAT   (23H      -MEAN WAIT TIME =,F7.1)                         A 230
 48     FORMAT   (23H SERVICE UTILIZATION  =,F7.1,8H PERCENT)              A 231
 49     FORMAT   (20H COSTS-WAIT IN QUEUE,F7.1,11H UNITS AT $,F6.2,4H = $, A 232
     1 F9.2)                                                               A 233
 50     FORMAT   (7X,13HIDLE SERVICE ,F7.1,11H UNITS AT $,F6.2,4H = $,F9.2) A 234
 51     FORMAT   (7X,24HTOTAL COST OF OPERATIONS,16X,1H$,F9.2)             A 235
        END                                                                A 236

        SUBROUTINE ARRIVE                                                  S   1
        COMMON ALPHA(10), ANUM, AR(500), ARRT, ARRTM, CH(20, 10), CUMQUE(  S   2
       1100), CUMUTL, CUSERV, DEPRT, DEPTM, I, IUSERV, IZ, KA, KCUS, KS, N S   3
       2, NFLAG, QUEUE, SNUM, SR(500), STATUS(9), T, TIME, TTIME, TNARV, TND S   4
       3PR(9)                                                              S   5
C       THIS SUBROUTINE CALLED WHEN AN ARRIVAL IS THE NEXT OCCURANCE       S   6
C       IT UPDATES THE TIME SPENT IN QUEUE                                 S   7
C       IT UPDATES THE CLOCK TO THE TIME OF THE NEW ARRIVAL (PREVIOUSLY    S   8
C          SELECTED)                                                       S   9
C       IT CHECKS EACH CHANNEL TO SEE IF THE NEW ARRIVAL CAN BEGIN SERVICE S  10
C       IF A CHANNEL IS AVAILABLE IT DOES THE FIRST PART OF THE            S  11
C          DEPART PROCESSING OTHERWISE IT ADDS ONE TO THE QUEUE            S  12
C       LASTLY, IT SELECTS THE TIME FOR THE NEXT ARRIVAL TO OCCUR          S  13
        M = QUEUE                                                          S  14
C       CHECK LENGTH OF QUEUE, IF OVER 99 HOLD AT 99                       S  15
        IF (M .LE. 99) GO TO 1                                             S  16
```

```
      M = 99                                                            A  17
C     UPDATE HOURS SPENT IN QUEUE                                       A  18
 1    CUMQUE(J,1) = CUMQUE(M,1)+TNARV-TIME                              A  19
C     UPDATE CLOCK TIME TO NEXT ARRIVAL                                 A  20
      TIME = TNARV                                                      A  21
C     CHECK EACH CHANNEL. IF STATUS = 0 IT IS AVAILABLE                 A  22
      DO 12 J = 1, N                                                    A  23
      IF (STATUS(J) .GT. 0.0) GO TO 11                                  A  24
C     DO FIRST PART OF DEPART PROCESSING                                A  25
      STATUS(J) = 1.0                                                   A  26
      GO TO (2,3,4,5), KS                                               A  27
C     POISSON SERVICE RATE                                              A  28
 2    R = RAND(0)                                                       A  29
      T = ABS(DEPRT*ALOG(R))                                            A  30
      GO TO 7                                                           A  31
C     NEGATIVE EXPONENTIAL SERVICE TIME                                 A  32
 3    R = RAND(0)                                                       A  33
      T = ABS(DEPTM*ALOG(R))                                            A  34
      GO TO 7                                                           A  35
C     CONSTANT SERVICE TIME                                             A  36
 4    T = DEPTM                                                         A  37
      GO TO 7                                                           A  38
C     READ-IN SERVICE TIME                                              A  39
 5    IUSERV = KCUS+1                                                   A  40
      NUM = SNUM                                                        A  41
      IF (IUSERV .LE. NUM) GO TO 6                                      A  42
      NFLAG = 76                                                        A  43
      T = 666666.                                                       A  44
      GO TO 7                                                           A  45
 6    T = SR(IUSERV)                                                    A  46
C     SET TIME OF NEXT DEPARTURE                                        A  47
 7    TNDPR(J) = TIME+T                                                 A  48
C     STORE FIRST TWENTY DEPARTURE TIMES IN CH                          A  49
      KCUS = KCUS+                                                      A  50
      IF (KCUS .GT. 20) GO TO 8                                         A  51
      II = J+1                                                          A  52
      CH(KCUS, II) = TNDPR(J)                                           A  53
 8    CONTINUE                                                          A  54
C     CHECK IF OUT OF SIMULATION TIME                                   A  55
      IF (TTIME .LT. TNDPR(J)) GO TO 9                                  A  56
C     ACCUMULATE PROCESSING TIME                                        A  57
      CUMUTL = CUMUTL+T                                                 A  58
      CUSERV = CUSERV+1.0                                               A  59
      GO TO 14                                                          A  60
C     END OF SIMULATION UPDATING                                        A  61
C     LAST DEPARTURE FORCED OUT AT TTIME                                A  62
 9    TK = T-(TNDPR(J)-TTIME)                                           A  63
      IF (TK .GT. 0.0) GO TO 10                                         A  64
      TK = 0.                                                           A  65
 10   TNDPR(J) = TTIME                                                  A  66
      CUMUTL = CUMUTL+TK                                                A  67
      GO TO 14                                                          A  68
 11   IF (J .GE. N) GO TO 13                                            A  69
 12   CONTINUE                                                          A  70
C     ALL CHANNELS ARE BUSY. ADD ONE TO THE QUEUE                       A  71
 13   QUEUE = QUEUE + 1                                                 A  72
C     SELECT ARRIVAL TIME (STORE FIRST TWENTY IN CH)                    A  73
 14   GO TO (15,16,17,18), KA                                           A  74
C     POISSON ARRIVAL TIME DISTRIBUTION                                 A  75
 15   R = RAND(0)                                                       A  76
      TNARV = ABS(ARRTM*ALOG(R))                                        A  77
      TNARV = TNARV+TIME                                                A  78
```

```
      IZ = IZ+1                                                          B  79
      IF (IZ .GT. 20) GO TO 20                                           B  80
      CH(IZ, I) = TNARV                                                  B  81
      GO TO 20                                                           B  82
C     NEGATIVE EXPO ARRIVAL TIME DISTRIBUTION                            B  83
 16   R = RAND(0)                                                        B  84
      TNARV = ABS(ARRRT*ALOG(R))                                         B  85
      TNARV = TNARV+TIME                                                 B  86
      IZ = IZ+1                                                          B  87
      IF (IZ .GT. 20) GO TO 20                                           B  88
      CH(IZ, 1) = TNARV                                                  B  89
      GO TO 20                                                           B  90
C     CONSTANT ARRIVAL TIMES                                             B  91
 17   TNARV = TIME+ARRTM                                                 B  92
      IZ = IZ+1                                                          B  93
      IF (IZ .GT. 20) GO TO 20                                           B  94
      CH(IZ, 1) = TNARV                                                  B  95
      GO TO 20                                                           B  96
C     READ IN ARRIVAL TIMES                                              B  97
 18   IZ = IZ+1                                                          B  98
      NUM = ANUM                                                         B  99
      IF (IZ .LL. NUM) GO TO 19                                          B 100
      AR(IZ) = 777777.                                                   B 101
      NFLAG = 76                                                         B 102
 19   TNARV = AR(IZ)                                                     B 103
      IF (IZ .GT. 20) GO TO 20                                           B 104
      CH(IZ, I) = TNARV                                                  B 105
 20   RETURN                                                             B 106
      END                                                                B 107

      SUBROUTINE DEPART                                                  C   1
      COMMON ALPHA(10), ANUM, AR(500), ARRRT, ARRTM, CH(20, 10), CUMQUE( C   2
     1100), CUMUTL, CUSERV, DEPRT, DEPTM, I, IUSERV, IZ, RA, KCUS, KS, N C   3
     2, NFLAG, QUEUE, SNUM, SR(500), STATUS(9), T, TIME, TTIME, TNARV, TND C   4
     3PR(9)                                                              C   5
C     THIS SUBROUTINE PROCESSES THE DEPARTURE OF EVERY CUSTOMER          C   6
C     IT UPDATES THE HOURS SPENT IN THE QUEUE                            C   7
C     IT UPDATES THE CLOCK TO THE NEXT DEPARTURE TIME (PREVIOUSLY        C   8
C        SELECTED)                                                       C   9
C     IT CHECKS THE LENGTH OF THE QUEUE                                  C  10
C     IF NO ONE IN QUEUE IT SETS THE CHANNEL AT AN IDLE STATUE (THIS     C  11
C        DEPARTURE WAS PREVIOUSLY PARTIALLY PROCESSED EITHER AT          C  12
C        ARRIVE OR BY A PRIOR PASS THROUGH DEPART)                       C  13
C     IF A QUEUE EXISTS THEN TAKE ONE FROM THE QUEUE, SET ITS DEPARTURE  C  14
C     TIME, SET THE CHANNEL AT A BUSY STATUS AND RETURN                  C  15
      M = QUEUE                                                          C  16
C     CHECK LENGTH OF QUEUE, IF OVER 99 HOLD AT 99                       C  17
      IF (M .LE. 99) GO TO 1                                             C  18
      M = 99                                                             C  19
C     UPDATE THE HOURS SPENT IN QUEUE                                    C  20
 1    CUMQUE(M+1) = CUMQUE(M+1)+TNDPR(I)-TIME                            C  21
C     UPDATE THE CLOCK TO NEXT DEPARTURE TIME                            C  22
      TIME = TNDPR(I)                                                    C  23
      IF (QUEUE.GE.1.0) GO TO 2                                          C  24
C     THIS SECTION COMPLETES THE PROCESSING OF A CUSTOMER                C  25
C     WHEN NO ONE IS WAITING IN THE QUEUE                                C  26
      STATUS(I) = 0.0                                                    C  27
      TNDPR(I) = 999999.9                                                C  28
      RETURN                                                             C  29
C     THIS SECTION DOES THE DEPART PROCESSING                            C  30
C     WHEN THE CHANNEL HAS BEEN BUSY                                     C  31
 2    QUEUE = QUEUE - 1.0                                                C  32
```

```
      GO TO (3,4,5,6), KS                                          C  33
C     SELECT NEXT DEPARTURE TIME                                   C  34
C     POISSON SERVICE RATE                                         C  35
 3    R = RAND(0)                                                  C  36
      T = ABS(DEPRT*ALOG(R))                                       C  37
      GO TO 8                                                      C  38
C     NEGATIVE EXPONENTIAL SERVICE TIME                            C  39
 4    R = RAND(0)                                                  C  40
      T = ABS(DEPTM*ALOG(R))                                       C  41
      GO TO 8                                                      C  42
C     CONSTANT SERVICE TIME                                        C  43
 5    T = DEPTM                                                    C  44
      GO TO 8                                                      C  45
C     READ-IN SERVICE TIME                                         C  46
 6    IUSERV = KCUS+1                                              C  47
      NUM = SNUM                                                   C  48
      IF (IUSERV .LE. NUM) GO TO 7                                 C  49
      NFLAG = 76                                                   C  50
      T = 666666.                                                  C  51
      GO TO 8                                                      C  52
 7    T = SR(IUSERV)                                               C  53
 8    TNDPR(I) = TIME+T                                            C  54
C     STORE FIRST TWENTY DEPARTURE TIMES IN CH                     C  55
      KCUS = KCUS+1                                                C  56
      IF (KCUS .GT. 20) GO TO 9                                    C  57
      II = I                                                       C  58
      CH(KCUS, II) = TNDPR(I)                                      C  59
 9    CONTINUE                                                     C  60
C     CHECK IF OUT OF SIMULATION TIME                              C  61
      IF (TTIME .LT. TNDPR(I)) GO TO 10                            C  62
C     RESET STATUS BACK TO BUSY AND RETURN                         C  63
      CUMUTL = CUMUTL+T                                            C  64
      CUSERV = CUSERV+1.0                                          C  65
      STATUS(I) = 1.0                                              C  66
      RETURN                                                       C  67
C     ADJUST T AND CUMUTL AT TERMINATION OF SIMULATION             C  68
C     LAST CUSTOMER FORCED TO DEPART AT TTIME                      C  69
 10   TK = T-(TNDPR(I)-TTIME)                                      C  70
      IF (T .GT. 0.0) GO TO 11                                     C  71
      TK = 0.                                                      C  72
 11   TNDPR(I) = TTIME                                             C  73
      CUMUTL = CUMUTL+TK                                           C  74
      STATUS(I) = 1.0                                              C  75
      RETURN                                                       C  76
      END                                                          C  77-

      FUNCTION RAND (K)                                            D   1
C     MACHINE DEPENDENT RANDOM NUMBER GENERATOR (0 TO 1)           D   2
C     THIS VERSION IS FOR 32 BIT WORD (IBM 360)                  IBM   3
C     K SET AT POSITIVE ODD INTEGER TO INITALIZE                   D   4
C     K SET AT ZERO TO CONTINUE STRING OF RANDOM NUMBERS           D   5
C     SEE NAYLOR,COMPUTER SIMULATION TECHNIQUES,WILEY +SONS,1966   D   6
      IF (K) 2,2,1                                                 D   7
 1    N = K                                                        D   8
 2    N = N*16807                                                IBM   9
      IF (N) 3,4,4                                                 D  10
 3    N = N+2147483647+1                                         IBM  11
 4    XN = N                                                       D  12
      RAND = XN/2147483647.                                      IBM  13
      RETURN                                                       D  14
      END                                                          D  15-
```

```
YOUR NAME = A1 PREMIUM TRUCK
1           .05         .1667       2
2           15.         1000.       .133
1           1.          6000.
YOUR NAME = A2 DELUX TRUCK
1           .05         .1667       2
2           12.         1400.       .167
1           1           6000.
YOUR NAME = A3 TWO PREIUM TRUCKS
1           .05         .1667       2
2           15.         1000.       .133
2           2           6000.
YOUR NAME = B1 NEW PREIUM TRACK
1           .05         .1667       2
2           15.         800.        .10
1           1.          6000.
YOUR NAME = B2 ONE DELUX TRUCK
1           .05         .1667       2
2           12.         1400.       .167
1           1.          6000.
YOUR NAME = B3 TWO NEW PREIUM TRUCKS
1           .05         .1667       2
2           15.         800.        .10
2           2           6000.
YOUR NAME = C1 PREIUM / 4 PER HOUR
1           .0007       .1667       2
2           15.         1000.       .133
1           1.          6000.
YOUR NAME = C2 DELUX / 4 PER HOUR
1           .0007       .1667       2
2           12.         1400.       .167
1           1.          6000.
YOUR NAME = C3 TWO PREIUM / 4 PER HOUR
1           .0007       .1667       2
2           15.         1000.       .133
2           2           6000.
YOUR NAME = D1 SOT
1           .05         .1667       3
2           12.         1400.       .167
1           1.          6000.
YOUR NAME = D2 SOT
1           .05         .1667       1
2           15.         1000.       .133
2           2           6000.
YOUR NAME = D3 SOT
1           .05         .1667       3
2           15.         1000.       .133
2           2           6000.
YOUR NAME = D1 RANDOM
1           .05         .1667       1
2           15.         1000.       .133
1           1.          6000.
YOUR NAME = D2 RANDOM
1           .05         .1667
2           15.         1000.       .133
1           1.          6000.
YOUR NAME = D3 RANDOM
1           .05         .1667       1
2           12.         1400.       .167
1           1.          6000.
STOP
```

PROGRAM QUEUES FOR YOUR NAME - AT PREMIUM TRUCK

ARRIVAL TYPE 1 RATE = .16 COST = .17 SCHEDULE RULE 2
(FCFS)

SERVICE TYPE 2 TIME = 15.00
FIXED COST $ 1000.00 , VARIABLE COST $ .13

NO. CHANNELS START 1 MAX 1 MAX TIME 6000

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS

| ARRIVAL TIME--- | ------DEPARTURE TIME AT CHANNEL NUMBER------ ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.4 | | | | | | | | |
| 26.0 | 45.5 | | | | | | | | |
| 26.2 | 45.5 | | | | | | | | |
| 40.6 | 56.4 | | | | | | | | |
| 128.2 | 193.4 | | | | | | | | |
| 135.6 | 199.4 | | | | | | | | |
| 148.0 | 206.7 | | | | | | | | |
| 148.0 | 206.7 | | | | | | | | |
| 184.8 | 236.2 | | | | | | | | |
| 200.1 | 247.4 | | | | | | | | |
| 221.3 | 263.7 | | | | | | | | |
| 223.0 | 265.0 | | | | | | | | |
| 243.3 | 280.2 | | | | | | | | |
| 244.4 | 281.1 | | | | | | | | |
| 246.7 | 282.8 | | | | | | | | |
| 249.7 | 285.0 | | | | | | | | |
| 292.2 | 324.0 | | | | | | | | |
| 306.8 | 335.0 | | | | | | | | |
| 310.9 | 338.1 | | | | | | | | |
| 374.1 | 421.5 | | | | | | | | |

AFTER 276 ARRIVED 272 SERVED 6000 TIME UNITS

| | |
|---|---|
| QUEUE-MAXIMUM LENGTH | 7 |
| -MEAN LENGTH | 1.5 |
| -MEAN WAIT TIME | 21.5 |
| SERVICE UTILIZATION | 74.2 PERCENT |
| AVERAGE ARRIVALS PER TIME UNIT | .1460 |
| AVERAGE SERVICE TIME | 16.3633 |
| MEAN NO. IN THE SYSTEM | 1.7396 |
| MEAN TIME IN THE SYSTEM | 37.8181 |

COST INFORMATION OF OPERATIONS

| | | | | |
|---|---|---|---|---|
| COSTS-WAIT IN QUEUE | 5921.5 UNITS AT $ | .17 | = $ | 987.12 |
| SERVICE COST VARIABLE | 272.0 UNITS AT $ | .13 | = $ | 36.18 |
| SERVICE COST FIXED | 1000.00 WITH 1 CHANNELS | | = $ | 1000.00 |

TOTAL COST OF OPERATIONS $ 2023.29

SPECIAL QUEUES FOR YOUR NAME - PI MEN PREFIUM TRACK

ARRIVAL TYPE 1 RATE = .08 COST = .17 SCHEDULE RULE 2
(FCFS)

SERVICE TYPE 2 TIME = 15.00
FIXED COST $ 800.00 VARIABLE COST $ .10

NO. CHANNELS START 1 MAX 1 MAX TIME 6000

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS

| ARRIVAL TIME--- | ------DEPARTURE TIME AT CHANNEL NUMBER------ ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.4 | | | | | | | | |
| 26.0 | 45.5 | | | | | | | | |
| 26.2 | 45.6 | | | | | | | | |
| 40.6 | 56.4 | | | | | | | | |
| 128.2 | 143.4 | | | | | | | | |
| 139.6 | 199.4 | | | | | | | | |
| 148.0 | 208.7 | | | | | | | | |
| 148.0 | 208.7 | | | | | | | | |
| 184.6 | 230.1 | | | | | | | | |
| 200.1 | 247.4 | | | | | | | | |
| 221.3 | 263.7 | | | | | | | | |
| 223.0 | 265.0 | | | | | | | | |
| 243.3 | 286.2 | | | | | | | | |
| 244.4 | 281.1 | | | | | | | | |
| 246.7 | 282.5 | | | | | | | | |
| 249.7 | 285.5 | | | | | | | | |
| 292.2 | 324.0 | | | | | | | | |
| 306.8 | 335.5 | | | | | | | | |
| 310.9 | 338.1 | | | | | | | | |
| 374.1 | 421.5 | | | | | | | | |

AFTER 276 ARRIVED 272 SERVED 6000 TIME UNITS

| | |
|---|---|
| QUEUE-MAXIMUM LENGTH | 7 |
| -MEAN LENGTH | 1.0 |
| -MEAN WAIT TIME | 21.5 |
| SERVICE UTILIZATION | 74.2 PERCENT |
| AVERAGE ARRIVALS PER TIME UNIT | .0460 |
| AVERAGE SERVICE TIME | 16.3633 |
| MEAN NO. IN THE SYSTEM | 1.7396 |
| MEAN TIME IN THE SYSTEM | 37.8181 |

COST INFORMATION OF OPERATIONS

| | | | |
|---|---|---|---|
| COSTS-WAIT IN QUEUE | 5921.5 UNITS AT $ .17 | = $ | 987.12 |
| SERVICE COST VARIABLE | 272.1 UNITS AT $ .10 | = $ | 27.21 |
| SERVICE COST FIXED | 800.00 WITH 1 CHANNELS | = $ | 800.00 |
| TOTAL COST OF OPERATIONS | | $ | 1814.32 |

```
PROGRAM QUEUES FOR YOUR NAME - C1 PRETIUM / 4 PER HOUR

ARRIVAL TYPE 1 RATE =      .17  COST =      .17  SCHEDULE FIL
                                                 (FCFS)

SERVICE TYPE 2 TIME =    15.6
       FIXED COST =  1,000.00          VARIABLE COST =      .13

NO. CHANNELS START 1 MAX 1          MAX TIME   6000

FIRST TWENTY OCCURRENCES FOR  1  SERVICE CHANNELS
ARRIVAL     ------DEPARTURE TIME AT CHANNEL NUMBER------
TIME---     ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE
     0       2.4
  19.5      39.0
  19.6      39.1
  30.4      49.9
  96.1     161.6
 101.6     107.3
 110.9     176.6
 110.9     176.7
 138.5     204.2
 150.0     215.8
 165.9     231.6
 167.2     232.9
 182.4     248.1
 183.2     249.0
 184.9     250.7
 187.2     252.9
 219.0     284.8
 230.0     295.7
 233.1     298.8
 280.5     346.3

AFTER   388 ARRIVED   374 SERVED  6000 TIME UNITS
QUEUE-MAXIMUM LENGTH                17
     -MEAN LENGTH                    5.3
     -MEAN WAIT TIME                31.8
SERVICE UTILIZATION                 98.  PERCENT
AVERAGE ARRIVALS PER TIME UNIT        .0647
AVERAGE SERVICE TIME                15.7217
MEAN NO. IN THE SYSTEM               6.342
MEAN TIME IN THE SYSTEM             97.4871

COST INFORMATION OF OPERATIONS
COSTS-WAIT IN QUEUE  31725.  UNITS AT $   .17 = $   5288.55
SERVICE COST VARIABLE  374.0 UNITS AT $   .13 = $     49.74
SERVICE COST FIXED   1,000.00 WITH 1 CHANNELS = $   1000.00

TOTAL COST OF OPERATIONS                        $   6338.30
```

PROGRAM QUEUES FOR YOUR NAME = DI SOT

ARRIVAL TYPE 1 RATE = .05 COST = .17 SCHEDULE RULE 3
(SOT)

SERVICE TYPE 2 TIME = 12.00
FIXED COST $ 1400.00 VARIABLE COST $ .17

NO. CHANNELS START 1 MAX 1 MAX TIME 6000

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS

| ARRIVAL TIME--- | ------DEPARTURE TIME AT CHANNEL NUMBER------ ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.0 | | | | | | | | |
| 26.0 | 41.5 | | | | | | | | |
| 26.2 | 41.7 | | | | | | | | |
| 40.6 | 56.1 | | | | | | | | |
| 128.2 | 143.7 | | | | | | | | |
| 135.6 | 185.2 | | | | | | | | |
| 148.0 | 185.2 | | | | | | | | |
| 148.0 | 192.6 | | | | | | | | |
| 184.8 | 200.4 | | | | | | | | |
| 200.1 | 223.9 | | | | | | | | |
| 221.3 | 236.8 | | | | | | | | |
| 223.0 | 237.7 | | | | | | | | |
| 243.3 | 255.4 | | | | | | | | |
| 244.4 | 256.1 | | | | | | | | |
| 246.7 | 257.5 | | | | | | | | |
| 249.7 | 259.4 | | | | | | | | |
| 292.2 | 317.0 | | | | | | | | |
| 306.8 | 326.4 | | | | | | | | |
| 310.9 | 328.9 | | | | | | | | |
| 374.1 | 412.1 | | | | | | | | |

AFTER 276 ARRIVED 272 SERVED 6000 TIME UNITS

| | |
|---|---|
| QUEUE-MAXIMUM LENGTH | 6 |
| -MEAN LENGTH | .5 |
| -MEAN WAIT TIME | 13.8 |
| SERVICE UTILIZATION | 59.4 PERCENT |
| AVERAGE ARRIVALS PER TIME UNIT | .0460 |
| AVERAGE SERVICE TIME | 13.1065 |
| MEAN NO. IN THE SYSTEM | 1.0973 |
| MEAN TIME IN THE SYSTEM | 23.8643 |

COST INFORMATION OF OPERATIONS

| | | | | |
|---|---|---|---|---|
| COSTS-WAIT IN QUEUE | 2968.8 UNITS AT $ | .17 | = $ | 494.89 |
| SERVICE COST VARIABLE | 272.0 UNITS AT $ | .17 | = $ | 45.42 |
| SERVICE COST FIXED | 1400.00 WITH 1 CHANNELS | | = $ | 1400.00 |
| TOTAL COST OF OPERATIONS | | | $ | 1940.32 |

PROGRAM QUEUES FOR YOUR NAME - D) RANDOM

ARRIVAL TYPE 1 RATE = .16 COST = .17 SCHEDULE RULE 1
(RANDOM)

SERVICE TYPE 2 TIME = 15.10
FIXED COST $ 1000.00 VARIABLE COST $ .13

NO. CHANNELS START 1 MAX 1 MAX TIME 6000

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS

| ARRIVAL TIME--- | ------DEPARTURE TIME AT CHANNEL NUMBER------- ONE |
|---|---|
| 0 | 2.4 |
| 25.0 | 45.5 |
| 29.2 | 45.6 |
| 40.6 | 56.4 |
| 178.2 | 193.9 |
| 189.5 | 199.4 |
| 191.0 | 208.7 |
| 200.0 | 220.3 |
| 18[illegible] | 247.8 |
| 200.1 | 247.9 |
| 221.3 | 263.7 |
| 223.0 | 278.9 |
| 243.3 | 281.1 |
| 244.4 | 313.0 |
| 246.7 | 314.7 |
| 249.7 | 317.8 |
| 292.2 | 318.6 |
| 306.8 | 329.6 |
| 310.9 | 330.9 |
| 374.1 | 421.5 |

AFTER 276 ARRIVED 275 SERVED 6000 TIME UNITS

| | |
|---|---|
| QUEUE-MAXIMUM LENGTH | 7 |
| -MEAN LENGTH | 1.6 |
| -MEAN WAIT TIME | 22.3 |
| SERVICE UTILIZATION | 74.8 PERCENT |
| AVERAGE ARRIVALS PER TIME UNIT | .0460 |
| AVERAGE SERVICE TIME | 16.3202 |
| MEAN NO. IN THE SYSTEM | 1.7779 |
| MEAN TIME IN THE SYSTEM | 38.6374 |

COST INFORMATION OF OPERATIONS

| | | | | |
|---|---|---|---|---|
| COSTS-WAIT IN QUEUE | 6159.5 UNITS AT $ | .17 | = $ | 1026.80 |
| SERVICE COST VARIABLE | 275.0 UNITS AT $ | .13 | = $ | 36.57 |
| SERVICE COST FIXED | 1000.00 WITH 1 CHANNELS | | = $ | 1000.00 |

TOTAL COST OF OPERATIONS $ 2063.37

ANALYSIS OF RESULTS

1. The three alternatives available to Rio Valley Electric Co-op (herein referred to as Rio) are:

| Alt. | Mean service rate | Variable cost/ unit | Fixed cost/ simulation run | Description |
|---|---|---|---|---|
| A-1 | 15 | $.13 | $1000 | one premium truck |
| A-2 | 12 | .17 | 1400 | one deluxe truck |
| A-3 | 15 | .13 | 2000 | two premium trucks |

For all the alternatives the FCFS priority rule was used and the simulated runs were for 6000 minutes. The results for the runs are:

| Alt. | Mean length Queue | Mean wait time | Service util. | Cost wait | Var. cost | Fix cost | Total cost |
|---|---|---|---|---|---|---|---|
| A-1 | 1.0 | 21.5 | 74.2% | $987.12 | $36.18 | $1000 | $2023.29 |
| A-2 | .5 | 11.5 | 59.4 | 529.42 | 45.42 | 1400 | 1975.22 |
| A-3 | .1 | 1.1 | 37.2 | 51.25 | 36.44 | 2000 | 2087.69 |

From the runs Rio should choose alternative A-2 as it has the lowest total cost. However, if the Rio Company is planning to grow it might prefer to choose alternative A-3 since this alternative has the lowest service utilization and thus would allow for the most growth (note the very low waiting cost as compared to alternatives A-1 and A-2).

2. The lower cost of the premium truck definitely changes the optimal cost alternative of question 1. The alternatives now available are:

| Alt. | Mean service rate | Variable cost/ unit | Fixed cost/ simulation run | Description |
|---|---|---|---|---|
| B-1 | 15 | $.10 | $ 800 | new premium truck |
| B-2 | same as alternative A-2 | | | |
| B-3 | 15 | .10 | 1600 | two new premium trucks |

The results from the computer runs are:

| Alt. | Mean length Queue | Mean wait time | Service util. | Cost wait | Var. cost | Fix cost | Total cost |
|---|---|---|---|---|---|---|---|
| B-1 | 1.0 | 21.5 | 74.2% | $987.12 | $27.12 | $ 800 | $1814.32 |
| B-2 | .5 | 11.5 | 59.4 | 529.42 | 45.42 | 1400 | 1975.22 |
| B-3 | .1 | 1.1 | 37.2 | 51.25 | 27.40 | 1600 | 1678.65 |

Clearly Rio should select B-3 as it has by far the lowest cost and because of the low service utilization will allow for the greatest expansion.

3. The alternatives C-1, C-2, and C-3 are the same as A-1, A-2, and A-3. In these runs the arrival rate was increased from 3 to 4 per hour (Poisson). The results of the runs are as follows:

| Alt. | Mean length Queue | Mean wait time | Service util. | Cost wait | Var. cost | Fix cost | Total cost |
|---|---|---|---|---|---|---|---|
| C-1 | 5.3 | 81.8 | 98.0% | $5288.35 | $1974 | $1000 | $6336.[illegible] |
| C-2 | 1.3 | 19.8 | 79.8 | 1285.83 | 64.[illegible] | 1400 | 2720.62 |
| C-3 | .1 | 2.1 | 49.9 | 137.50 | 51.60 | 2000 | 2189.10 |

Alternative C-3 is the best selection. This is due, as was mentioned in part 1, to the fact that the alternative C-3 had room for expansion while the others were already being highly utilized. There is still quite a difference between the waiting time cost.

4. In this part we are asked to compare the result of part 1 using the other two priority rules, SOT (D-1S, D-2S, and D-3S) and RANDOM (D-1R, D-2R, and D-3R). The results of the computer runs are shown below:

| Alt. | Mean length Queue | Mean wait time | Service util. | Cost wait | Var. cost | Fix cost | Total cost |
|---|---|---|---|---|---|---|---|
| A-1 | 1.0 | 21.5 | 74.2% | $987.12 | $36.18 | $1000 | $2023.29 |
| D-1S | .9 | 19.0 | 74.2 | 873.53 | 36.18 | 1000 | 1909.70 |
| D-1R | 1.0 | 22.3 | 74.2 | 1026.80 | 36.57 | 1000 | 2063.37 |
| A-2 | .5 | 11.5 | 59.4 | $529.42 | $45.42 | $1400 | $1975.22 |
| D-2S | .5 | 10.8 | 59.4 | 494.89 | 45.42 | 1400 | 1940.32 |
| D-2R | .5 | 10.7 | 59.4 | 493.38 | 45.42 | 1400 | 1938.81 |
| A-3 | .1 | 1.1 | 37.2 | $ 51.25 | $36.44 | $2000 | $2087.69 |
| D-3S | .1 | 1.1 | 37.2 | 51.01 | 36.44 | 2000 | 2087.45 |
| D-3R | .1 | 1.1 | 37.2 | 52.84 | 36.44 | 2000 | 2089.29 |

In the first set of alternatives, D-1S is the best selection. This is due primarily to the fact that the mean waiting time is less for the SOT rule since the shorter jobs are worked on first and gotten out of the system quickly.

The runs using the deluxe truck, D-2R gave the best result. Here the range was not nearly as wide as for the single premium truck.

In the case of the two premium trucks (A-3, D-3S, and D-3R) the run using the SOT rule gave the best results. The best explanation for this is that because of the large amount of slack in the system (35% utilization) it really doesn't make too much difference which rule is used.

D-2R has the lowest cost of all nine runs. This indicates that under present conditions the deluxe truck would be the best selection and that the RANDOM priority rule should be used. In actuality the company should probably select the deluxe truck and use a modified SOT rule (such SOT between 8am and 8pm, all jobs in QUEUE at 8pm on FCFS until 8am following morning).

## CONCLUDING REMARKS

From the results of the run, the following would be recommended to Rio:

1. If Rio expects the arrival rate of calls to remain about 3/hr, they should buy the deluxe truck, provided the premium is not available at the reduced rate.
2. If arrival rate is expected to increase, buy the two premium trucks.

As is suggested, the two premium trucks are probably the best selection all around. One major reason for this would be that if one of the trucks broke down, the other would still be able to make service calls.

| Alt. | Mean length Queue | Mean wait time | Service util. | Cost wait | Var. cost | Fix cost | Total cost |
|---|---|---|---|---|---|---|---|
| C-1 | 5.3 | 81.8 | 98.0% | $5268.55 | $4974 | $1000 | $6338.30 |
| C-2 | 1.3 | 19.8 | 79.8 | 1285.83 | 64.80 | 1400 | 2750.62 |
| C-3 | .1 | 2.1 | 49.9 | 137.50 | 51.60 | 2000 | 2189.10 |

Alternative C-3 is the best selection. This is due, as was mentioned in part 1, to the fact that the alternative C-3 had room for expansion while the others were already being highly utilized. There is still quite a difference between the waiting time cost.

4. In this part we are asked to compare the result of part 1 using the other two priority rules, SOT (D-1S, D-2S, and D-3S) and RANDOM (D-1R, D-2R, and D-3R). The results of the computer runs are shown below:

| Alt. | Mean length Queue | Mean wait time | Service util. | Cost wait | Var. cost | Fix cost | Total cost |
|---|---|---|---|---|---|---|---|
| A-1 | 1.0 | 21.5 | 74.2% | $987.12 | $36.18 | $1000 | $2023.29 |
| D-1S | .9 | 19.0 | 74.2 | 873.53 | 36.18 | 1000 | 1909.70 |
| D-1R | 1.0 | 22.3 | 74.2 | 1026.80 | 36.57 | 1000 | 2063.37 |
| A-2 | .5 | 11.5 | 59.4 | $529.42 | $45.42 | $1400 | $1975.22 |
| D-2S | .5 | 10.8 | 59.4 | 494.89 | 45.42 | 1400 | 1940.32 |
| D-2R | .5 | 10.7 | 59.4 | 493.38 | 45.42 | 1400 | 1938.81 |
| A-3 | .1 | 1.1 | 37.2 | $ 51.25 | $36.44 | $2000 | $2087.69 |
| D-3S | .1 | 1.1 | 37.2 | 51.01 | 36.44 | 2000 | 2087.45 |
| D-3R | .1 | 1.1 | 37.2 | 52.84 | 36.44 | 2000 | 2089.29 |

In the first set of alternatives, D-1S is the best selection. This is due primarily to the fact that the mean waiting time is less for the SOT rule since the shorter jobs are worked on first and gotten out of the system quickly.

The runs using the deluxe truck, D-2R gave the best result. Here the range was not nearly as wide as for the single premium truck.

In the case of the two premium trucks (A-3, D-3S, and D-3R) the run using the SOT rule gave the best results. The best explanation for this is that because of the large amount of slack in the system (35% utilization) it really doesn't make too much difference which rule is used.

D-2R has the lowest cost of all nine runs. This indicates that under present conditions the deluxe truck would be the best selection and that the RANDOM priority rule should be used. In actuality the company should probably select the deluxe truck and use a modified SOT rule (such SOT between 8am and 8pm, all jobs in QUEUE at 8pm on FCFS until 8am following morning).

CONCLUDING REMARKS

From the results of the run, the following would be recommended to Rio:

1. If Rio expects the arrival rate of calls to remain about 3/hr, they should buy the deluxe truck, provided the premium is not available at the reduced rate.
2. If arrival rate is expected to increase, buy the two premium trucks.

As is suggested, the two premium trucks are probably the best selection all around. One major reason for this would be that if one of the trucks broke down, the other would still be able to make service calls.

```
      PROGRAM QUEUES                                                    A    1
C     MARCH 1972 M J MAGGARD                                            A    2
C     THIS VERSION FOR CDC 3100                                         A   2A
C     DICTIONARY OF VARIABLES                                           A    3
C     ATTIME          THE HOURS OF SYSTEM IDLE TIME - TOTAL             A    4
C     ANUM,SNUM       NUMBER OF READS IN ARRIVALS AND SERVICE           A    5
C     AR(500.         AN ARRAY OF READ IN ARRIVAL TIMES                 A    6
C     ARR RT,ARR TM THE ARRIVAL RATE AND TIME                           A    7
C     CH              AN ARRAY OF ARRIVAL AND SERVICE ON FIRST 20 CUSTOMERS A 8
C     CIDLE           THE COST OF SYSTEM IDLE TIME - TOTAL              A    9
C     CUMQUE(100)     AN ARRAY WHICH STORES IDLE CUSTOMER HOURS         A   10
C     COSTS           THE COST PER TIME UNIT OF IDLE SERVICE            A   11
C     COSTA           THE COST PER TIME UNIT OF IDLE CUSTOMERS          A   12
C     CUSERV,KCUS     THE NUMBER OF CUSTOMERS BEING SERVED              A   13
C     CWAIT           THE COST OF CUSTOMERS HOURS IN QUEUE - TOTAL      A   14
C     DEP RT,DEP TM THE SERVICE RATE AND MEAN DURATION                  A   15
C     HRSNQ           THE HOURS OF CUSTOMER TIME IN QUEUE - TOTAL       A   16
C     II,J            THE CHANNEL NUMBER BEING PROCESSED                A   17
C     IIZ             NUMBER OF ARRIVALS WHICH HAVE OCCURED             A   18
C     KA,KS           OPTION CODES FOR ARRIVALS AND SERVICE             A   19
C     MN,MAXS         BEGINNING,MAXIMUM NUMBER CHANNELS                 A   20
C     PCUTIL          THE PERCENT UTILIZATION OF THE SERVICE FACILITY   A   21
C     QUE             THE NUMBER OF CUSTOMERS IN QUEUE AT ANY POINT     A   22
C     SR(500)         AN ARRAY OF READ IN SERVICE TIMES                 A   23
C     TCOOP           THE TOTAL COST OF THE SYSTEM                      A   24
C     TIME,TTIME      CLOCK TIME,MAX SIMULATION TIME                    A   25
C     TNARV           THE LATEST ARRIVAL TIME                           A   26
C     TNDPR           THE DEPARTURE TIME OF THE LATEST DEPARTURE        A   27
C                     MAY BE SET ARTIFICALLY FOR PROGRAM EFFICIENCY     A   28
C     XMNTN,XMNTX     MEAN WAIT TIME,MEAN NUMBER IN QUE                 A   29
C     AVARRA          AVERAGE ARRIVALS PER TIME UNIT                    A   30
C     AVSERV          AVERAGE SERVICE TIME                              A   31
C     AMNIS           MEAN NO. IN THE SYSTEM                            A   32
C     AMTIS           MEAN TIME IN THE SYSTEM                           A   33
C     VCOST           TOTAL VARIABLE COST OF OPERATIONS                 A   34
C     VCOSTS          VARIABLE COST PER UNIT                            A   35
C     FCOST           TOTAL FIXED COST OF OPERATIONS                    A   36
C     FCOSTS          FIXED COST PER UNIT                               A   37
C     QSVCT           AN ARRAY OF QUEUED SERVICE TIMES                  A   38
C     KRULE           SCHEDULE RULE CODE (1=RANDOM,2=FCFS,3=SOT)        A   39
      COMMON ILPHA(10),ANUM,ARRRT,ARRTM,CH(20,10),CUMUTL,CUSERV,DEPRT,  A   40
     1       DEPTM,I,IUSERV,IZ,KA,KCUS,KS,N,NFLAG,SNUM,STATUS(9),T,     A   41
     2       TIME,TTIME,TNARV,TNDPR(9),AVARRA,AVSERV,AMNIS,AMTIS,       A   42
     3       VCOST,VCOSTS,FCOST,FCOSTS,KRULE,IQUE,CUMQUE(1,1),          A   43
     4       QSVCT(101),AR(500),SR(500)                                 A   44
      COMMON /DATA/ IEND                                                A   45
      DATA (IEND=4HSTOP)                                                A   46
      IISTOP=IEND                                                       A   47
      MIN=60                                                            A   48
      MOUT=61                                                           A   49
C     READ AND PRINT USER NAME CARD                                     A   50
   11 CONTINUE                                                          A   51
      READ (MIN,311) ILPHA                                              A   52
      WRITE (MOUT,321) ILPHA                                            A   53
          IF (ILPHA(1).EQ.ISTOP) GO TO 301                              A   54
C     READ AND PRINT ARRIVAL DATA CARD AND SCHEDULE RULE CODE           A   55
      READ (MIN,331) KA,ARRRT,COSTA,KRULE,ANUM                          A   56
      WRITE (MOUT,341) KA,ARRRT,COSTA,KRULE                             A   57
          IF (KA.LT.1) GO TO 241                                        A   58
          IF (KA.GT.6) GO TO 241                                        A   59
          IF (KA.LE.LT.1) GO TO 241                                     A   60
          IF (KA.LE.GT.6) GO TO 241                                     A   61
          GO TO (21,31,41), KA.LE                                       A   62
   21 CONTINUE                                                          A   63
      WRITE (MOUT,351)                                                  A   64
```

```
      GO TO 91                                                          A  65
 31   CONTINUE                                                          A  66
      WRITE (NOUT,361)                                                  A  67
      GO TO 61                                                          A  68
 41   CONTINUE                                                          A  69
      WRITE (NOUT,371)                                                  A  70
C     READ ARRIVAL STATISTICS                                           A  71
 51   CONTINUE                                                          A  72
      IF (ANUM.LE.0.0) GO TO 71                                         A  73
      NUM=ANUM                                                          A  74
      ARRRT=1.234                                                       A  75
      KA=4                                                              A  76
      WRITE (NOUT,391) NUM                                              A  77
      IF (ANUM.GT.500.0) GO TO 291                                      A  78
      READ (NIN,381) (AR(I),I=1,NUM)                                    A  79
      WRITE (NOUT,401) (AR(I),I=1,NUM)                                  A  80
      DO 61 I=2,NUM                                                     A  81
      J=I-1                                                             A  82
      IF (AR(J).GT.AR(I)) GO TO 291                                     A  83
 61   CONTINUE                                                          A  84
C     SET ARRIVAL TIME AT INVERSE OF ARRIVAL RATE                       A  85
 71   CONTINUE                                                          A  86
      IF (ARRRT.LE.0.0) GO TO 291                                       A  87
      ARRTM=1.0/ARRRT                                                   A  88
C     READ AND PRINT SERVICE DATA CARD                                  A  89
      READ (NIN,411) KS,DEPTM,FCOSTS,VCOSTS,SNUM                        A  90
      WRITE (NOUT,421) KS,DEPTM                                         A  91
      WRITE (NOUT,431) FCOSTS,VCOSTS                                    A  92
      IF (KS.EQ.0) GO TO 291                                            A  93
      IF (KS.GT.4) GO TO 291                                            A  94
      IF (SNUM.LE.0.0) GO TO 91                                         A  95
      NUM=SNUM                                                          A  96
      DEPTM=1.234                                                       A  97
      KS=4                                                              A  98
      WRITE (NOUT,441) NUM                                              A  99
      IF (SNUM.GT.500.0) GO TO 291                                      A 100
      READ (NIN,381) (SR(I),I=1,NUM)                                    A 101
      WRITE (NOUT,401) (SR(I),I=1,NUM)                                  A 102
      DO 81 I=1,NUM                                                     A 103
      IF (SR(I).LT.0.0) GO TO 291                                       A 104
 81   CONTINUE                                                          A 105
C     SET SERVICE RATE AT INVERSE OF SERVICE TIME                       A 106
 91   CONTINUE                                                          A 107
      IF (DEPTM.LE.0.0) GO TO 291                                       A 108
      DEPRT=1.0/DEPTM                                                   A 109
C     READ SIMULATION CONTROL CARD AND PRINT                            A 110
      READ (NIN,451) N,MAXS,TTIME                                       A 111
      WRITE (NOUT,461) N,MAXS,TTIME                                     A 112
C     CHECK SIMULATION RUN LIMITS                                       A 113
      IF (N.EQ.0) GO TO 291                                             A 114
      IF (MAXS.LT.N) GO TO 291                                          A 115
      IF (TTIME.LE.0.0) GO TO 291                                       A 116
C     END OF INPUT DATA CHECK                                           A 117
      BCOOP=999999.9                                                    A 118
C     SIMULATION OF A GIVEN NUMBER OF CHANNELS (N) BEGINS HERE          A 119
C     INITIALIZE SYSTEM FOR NEXT SIMULATION RUN                         A 120
101   CONTINUE                                                          A 121
      TIME=0.0                                                          A 122
      TNARV=0.0                                                         A 123
      IQUE=1                                                            A 124
      CUMUTL=0.0                                                        A 125
      CUSERV=0.0                                                        A 126
      IZ=0                                                              A 127
      KCUS=0                                                            A 128
      NFLAG=0                                                           A 129
      IUSERV=0                                                          A 130
```

```
      SET=RAN(1234567.0)                                                A 131
      DO 111 I=1,100                                                    A 132
      QSVCT(I)=0.0                                                      A 133
      CUMQUE(I)=0.0                                                     A 134
  111 CONTINUE                                                          A 135
C       DETERMINE SERVICE TIME FOR THE 1ST ARRIVAL                      A 136
        GO TO (121,131,141,151), KS                                     A 137
C       POISSON SERVICE RATE                                            A 138
  121 CONTINUE                                                          A 139
      R=RAND(0.9)                                                       A 140
      T=ABS(DEPRT*ALOG(R))                                              A 141
        GO TO 161                                                       A 142
C       NEGATIVE EXPONENTIAL SERVICE TIME                               A 143
  131 CONTINUE                                                          A 144
      R=RAND(0.9)                                                       A 145
      T=ABS(DEPTM*ALOG(R))                                              A 146
        GO TO 161                                                       A 147
C       CONSTANT SERVICE TIME                                           A 148
  141 CONTINUE                                                          A 149
      T=DEPTM                                                           A 150
        GO TO 161                                                       A 151
C     READ IN SERVICE TIME                                              A 152
  151 CONTINUE                                                          A 153
      T=SR(1)                                                           A 154
  161 CONTINUE                                                          A 155
      QSVCT(1)=T                                                        A 156
      DO 171 L=1,N                                                      A 157
      TNDPR(L)=999999.9                                                 A 158
      STATUS(L)=0.0                                                     A 159
  171 CONTINUE                                                          A 160
      DO 181 I=1,20                                                     A 161
      DO 181 J=1,10                                                     A 162
      CH(I,J)=0.0                                                       A 163
  181 CONTINUE                                                          A 164
C     PRINT HEADING FOR RESULTS                                         A 165
      WRITE (NOUT,471) N                                                A 166
      WRITE (NOUT,481)                                                  A 167
      WRITE (NOUT,491)                                                  A 168
C     SET FIRST ARRIVAL OCCURANCE AT TIME ZERO                          A 169
      TNARV=0.0                                                         A 170
      CH(1,1)=0.0                                                       A 171
      IZ=IZ+1                                                           A 172
C     MAIN SIMULATION BRANCH POINT                                      A 173
C       CHECK EACH CHANNEL IN TURN FOR POSSIBLE DEPARTURE               A 174
C     IF ALL CHANNELS ARE IDLE (TNDPR = 999999.9) THEN GO TO ARRIVE     A 175
C     IF ALL CHANNELS ARE BUSY (TNARV IS .GE. TNDPR) THEN GO TO ARRIVE  A 176
C     IF A DEPART IS NEXT (TNDPR IS .GE. TNARV) THEN GO TO DEPART       A 177
C     SET AND IVALUE KEEP MULTIPLE DEPARTURES IN CORRECT TIME SEQUENCE  A 178
  191 CONTINUE                                                          A 179
      SET=888888.                                                       A 180
      DO 201 I=1,N                                                      A 181
        IF (TNDPR(I).GT.TNARV) GO TO 201                                A 182
        IF (TNDPR(I).GT.SET) GO TO 201                                  A 183
      SET=TNDPR(I)                                                      A 184
      IVALUE=I                                                          A 185
  201 CONTINUE                                                          A 186
      I=IVALUE                                                          A 187
        IF (SET.LT.888888.) GO TO 211                                   A 188
      CALL ARRIVE                                                       A 189
        GO TO 191                                                       A 190
  211 CONTINUE                                                          A 191
      CALL DEPART                                                       A 192
C     ON RETURN FROM DEPART CHECK SIMULATION TIME LIMIT                 A 193
        IF (TTIME.GT.TIME) GO TO 191                                    A 194
C     END OF SIMULATION RUN---PRINT FIRST TWENTY TRIALS                 A 195
      N1=N+1                                                            A 196
```

```
      IF (CUSERV.GE.20.0) GO TO 221                                    A 197
      NAX=CUSERV                                                       A 198
      GO TO 231                                                        A 199
  221 CONTINUE                                                         A 200
      NAX=20.                                                            201
  231 CONTINUE                                                         A 202
      DO 241 I=1,NAX                                                   A 203
      WRITE (NOUT,501) (CH(I,J),J=1,N)                                 A 204
  241 CONTINUE                                                         A 205
      COMPUTE HOURS IN QUEUE FOR SUMMARY PRINTOUT                      A 206
      HRSNQ=0.0                                                        A 207
      MAXQUE=0                                                         A 208
      DO 261 M=2,100                                                   A 209
         IF (CUMQUE(M).EQ.0.0) GO TO 25.                               A 210
      MAXQUE=M-1                                                       A 211
  251 CONTINUE                                                         A 212
      XM=M-1                                                           A 213
      HRSNQ=HRSNQ+(XM*CUMQUE(M))                                       A 214
  261 CONTINUE                                                         A 215
         IF (MAXQUE.LT.99) GO TO 271                                   A 216
      WRITE (NOUT,511)                                                 A 217
  271 CONTINUE                                                         A 218
         IF (NFLAG.NE.76) GO TO 281                                    A 219
      WRITE (NOUT,521)                                                 A 220
  281 CONTINUE                                                         A 22.
      XN=N                                                             A 222
      IZ=IZ-1                                                          A 223
      XIZ=IZ                                                           A 224
C     PRINT SUMMARY STATISTICS FOR THIS NUMBER (N) CHANNELS            A 225
      WRITE (NOUT,531) IZ,CUSERV,TIME                                  A 226
      WRITE (NOUT,541) MAXQUE                                          A 227
      XMNTX=HRSNQ/TIME                                                 A 228
      WRITE (NOUT,551) XMNTX                                           A 229
      XMNTM=HRSNQ/XIZ                                                  A 230
      WRITE (NOUT,561) XMNTM                                           A 231
      PCUTIL=((CUMUTL/TIME)*100.)/XN                                   A 232
      WRITE (NOUT,571) PCUTIL                                          A 233
      CWAIT=HRSNQ*COSTA                                                A 234
      TITIME=(TIME*XN)-CUMUTL                                          A 235.
C     COMPUTE AVERAGE ARRIVALS PER TIME UNIT                           A 236
      IZ=IZ                                                            A 237
      AVARRA=IZ/TIME                                                   A 238
      WRITE (NOUT,581) AVARRA                                          A 239
C     COMPUTE AVERAGE SERVICE TIME                                     A 240
      AVSERV=CUMUTL/CUSERV                                             A 241
      WRITE (NOUT,591) AVSERV                                          A 242
C     COMPUTE MEAN NO. IN THE SYSTEM                                   A 243
      AMNIS=XMNTX*(AVARRA/(1.0/AVSERV))                                A 244
      WRITE (NOUT,601) AMNIS                                           A 245
C     COMPUTE MEAN TIME IN THE SYSTEM                                  A 246
      AMTIS=XMNTM*AVSERV                                               A 247
      WRITE (NOUT,611) AMTIS                                           A 248
      WRITE (NOUT,621)                                                 A 249
      WRITE (NOUT,631) HRSNQ,COSTA,CWAIT                               A 250
C     COMPUTE TOTAL VARIABLE COST                                      A 251
      VCOST = CUSERV*VCOSTS                                            A 252
C     COMPUTE TOTAL FIXED COST                                         A 253
      FCOST=XN*FCOSTS                                                  A 254
C     PRINT SUMMARY COST INFORMATION                                   A 255
      WRITE (NOUT,641) CUSERV,VCOSTS,VCOST                             A 256
      WRITE (NOUT,651) FCOSTS,N,FCOST                                  A 257
      TCOOP=FCOST+VCOST+CWAIT                                          A 258
      WRITE (NOUT,661) TCOOP                                           A 259
C     STOP RUN IF TCOOP INCREASED FROM LAST RUN                        A 260
         IF (TCOOP.GE.BCOOP) GO TO 11                                  A 261
C     STOP RUN IF MAXIMUM NUMBER OF SERVERS REACHED                    A 262
```

```
      IF (N.GE.MAXS) GO TO 11                                          A 263
C     UPDATE NUMBER OF CHANNELS AND CURRENT TOTAL COST                 A 264
      N=N+1                                                            A 265
      BCOOP=TCOOP                                                      A 266
C     RETURN FOR NEXT RUN WITH MORE CHANNELS (N)                       A 267
      GO TO 101                                                        A 268
C     PRINT OUT DATA ERROR MESSAGE                                     A 269
  291 CONTINUE                                                         A 270
      WRITE (NOUT,671)                                                 A 271
      WRITE (NOUT,681)                                                 A 272
  301 CONTINUE                                                         A 273
      WRITE (NOUT,691)                                                 A 274
C                                                                      A 275
C                                                                      A 276
  311 FORMAT  (10A4)                                                   A 277
  321 FORMAT  (20H1PROGRAM QUEUES FOR ,10A4)                           A 278
  331 FORMAT  (I1,9X,F5.0,5X,F5.0,5X,I1,9X,F3.0)                       A 279
  341 FORMAT  (14H0ARRIVAL TYPE ,I1,8H RATE = ,F8.2,8H  COST =,F8.2,16H  A 280
     1 SCHEDULE RULE ,I1)                                              A 281
  351 FORMAT  (54X,8H(RANDOM))                                         A 282
  361 FORMAT  (55X,6H(FCFS))                                           A 283
  371 FORMAT  (55X,5H(SOT))                                            A 284
  381 FORMAT  (12F5.0)                                                 A 285
  391 FORMAT  (1H ,I4,28H ARRIVALS READ IN AS FOLLOWS)                 A 286
  401 FORMAT  (1H ,12F5.0)                                             A 287
  411 FORMAT  (I1,9X,F5.0,5X,F10.0,F10.0,F3.0)                         A 288
  421 FORMAT  (14H0SERVICE TYPE ,I1,8H TIME = ,F8.2)                   A 289
  431 FORMAT  (8X,13H FIXED COST $,F8.2,8X,16H VARIABLE COST $,F8.2)   A 290
  441 FORMAT  (1H ,I4,27H SERVICE READ IN AS FOLLOWS)                  A 291
  451 FORMAT  (I1,9X,I1,9X,F5.0)                                       A 292
  461 FORMAT  (20H0NO. CHANNELS START ,I1,5H MAX ,I1,8X,16H MAX TIME ,F6 A 293
     1.0)                                                              A 294
  471 FORMAT  (31H0FIRST TWENTY OCCURRENCES FOR    I1,18H  SERVICE CHANNE A 295
     1LS)                                                              A 296
  481 FORMAT  (8H ARRIVAL,4X,44H-------DEPARTURE TIME AT CHANNEL NUMBER-- A 297
     1----)                                                            A 298
  491 FORMAT  (9H TIME---,4X,44HONE TWO THREE FOUR FIVE SIX SEVEN EIGHT  A 299
     1NINE)                                                            A 300
  501 FORMAT  (1H ,F6.1,3X,9F5.1)                                      A 301
  511 FORMAT  (40H0***WARNING****QUE EXCEEDED PROGRAM LIMIT OF 99****) A 302
  521 FORMAT  (48H0***WARNING****OUT OF DATA BEFORE TIME LIMIT*****)   A 303
  531 FORMAT  (6H0AFTER,I6,8H ARRIVED,F6.0,7H SERVED,F6.0,11H TIME UNITS A 304
     1)                                                                A 305
  541 FORMAT  (22H QUEUE-MAXIMUM LENGTH ,8X,I7)                        A 306
  551 FORMAT  (22H      -MEAN LENGTH    ,8X,F9.1)                      A 307
  561 FORMAT  (22H      -MEAN WAIT TIME ,8X,F9.1)                      A 308
  571 FORMAT  (22H SERVICE UTILIZATION  ,8X,F9.1,8H PERCENT)           A 309
  581 FORMAT  (32H AVERAGE ARRIVALS PER TIME UNIT ,F10.4)              A 310
  591 FORMAT  (32H AVERAGE SERVICE TIME           ,F10.4)              A 311
  601 FORMAT  (32H MEAN NO. IN THE SYSTEM         ,F10.4)              A 312
  611 FORMAT  (32H MEAN TIME IN THE SYSTEM        ,F10.4)              A 313
  621 FORMAT  (31H0COST INFORMATION OF OPERATIONS)                     A 314
  631 FORMAT  (20H COSTS-WAIT IN QUEUE,  F9.1,11H UNITS AT $,F6.2,4H = $ A 315
     1,F9.2)                                                           A 316
  641 FORMAT  (22H SERVICE COST VARIABLE,F7.1,11H UNITS AT $,F6.2,4H = $ A 317
     1,F9.2)                                                           A 318
  651 FORMAT  (22H SERVICE COST FIXED   ,F7.2,6H WITH ,I1,14H CHANNELS  A 319
     1= $,F9.2)                                                        A 320
  661 FORMAT  (20H0TOTAL COST OF OPERATIONS                    $,F9.   A 321
     12)                                                               A 322
  671 FORMAT  (35H0****ERROR IN GUESIM DATA CARDS****)                 A 323
  681 FORMAT  (35H ****CORRECT DATA AND TRY AGAIN****)                 A 324
  691 FORMAT  (22 GUESIM RUN TERMINATED)                               A 325
      END                                                              A 326

      SUBROUTINE ARRIVE                                                B
```

```
      COMMON ALPHA(10),ANUM,ARRPT,ARRTM,CH(20,10),CUMUTL,CUSERV,DEPRT,
     1       DEPTM,I,IUSERV,IZ,KA,KCU,KS,N,NFLAG,SNUM,STATUS(9),T,
     2       TIME,TTIME,TNARV,TNDPR(9),AVAIPA,AVSERV,AMTIS,
     3       VCOST,VCOSTS,FCOST,FCOSTS,KROLE,IQUE,CUMQUE(101),
     4       QSVCT(101),AR(500),SR(500)
C     THIS SUBROUTINE CALLED WHEN AN ARRIVAL IS THE NEXT OCCURANCE
C     IT UPDATES THE TIME SPENT IN QUE
C     IT UPDATES THE CLOCK TO THE TIME OF THE NEW ARRIVEL (PREVIOUSLY
C        SELECTED)
C     IT CHECKS EACH CHANNEL TO SEE IF THE NEW ARRIVEL CAN BEGIN SERVICE
C     IF A CHANNEL IS AVAILABLE IT DOES THE FIRST PART OF THE
C        DEPART PROCESSING OTHERWISE IT ADDS ONE TO THE QUE
C     LASTLY, IT SELECTS THE TIME FOR THE NEXT ARRIVAL TO OCCUR
      IF (IQUE.LT.100) GO TO 11
C     CHECK LENGTH OF QUE, IF OVER 99 HOLD AT 99
      IQUE=100
C     UPDATE HOURS SPENT IN QUEUE
   11 CONTINUE
      CUMQUE(IQUE)=CUMQUE(IQUE)+TNARV-TIME
C     UPDATE CLOCK TIME TO NEXT ARRIVAL
      TIME=TNARV
C     CHECK EACH CHANNEL, IF STATUS = 0 IT IS AVAILABLE
      DO 61 J=1,N
      IF (STATUS(J).GT.0.0) GO TO 51
C     DO FIRST PART OF THE DEPART PROCESSING
      STATUS(J)=1.0
C     SET TIME OF NEXT DEPARTURE
      TNDPR(J)=TIME+QSVCT(1)
C     STORE FIRST TWENTY DEPARTURE TIMES IN CH
      KCUS=KCUS+1
      IF (KCUS.GT.20) GO TO 21
      II=J+1
      CH(KCUS,II)=TNDPR(J)
   21 CONTINUE
C     CHECK IF OUT OF SIMULATION TIME
      IF (TTIME.LT.TNDPR(J)) GO TO 31
C     ACCUMULATE PROCESSING TIME
      CUMUTL=CUMUTL+QSVCT(1)
      CUSERV=CUSERV+1.0
      GO TO 81
C     END OF SIMULATION UPDATING
C     LAST DEPARTURE FORCED OUT AT TTIME
   31 CONTINUE
      TK=QSVCT(1)-(TNDPR(J)-TTIME)
      IF (TK.GT.0.0) GO TO 41
      TK=0
   41 CONTINUE
      TNDPR(J)=TTIME
      CUMUTL=CUMUTL+TK
      GO TO 81
   51 CONTINUE
      IF (J.GE.N) GO TO 71
   61 CONTINUE
C     ALL CHANNELS ARE BUSY, ADD ONE TO THE QUE
   71 CONTINUE
      IQUE=IQUE+1
C     SELECT ARRIVAL TIME (STORE FIRST TWENTY IN CH)
   81 CONTINUE
      GO TO (91,101,111,121), KA
C     POISSON ARRIVAL TIME DISTRIBUTION
   91 CONTINUE
      R=RAND(0.0)
      TNARV=ABS(ARRTM*ALOG(R))
      TNARV=TNARV+TIME
      GO TO 141
C     NEGATIVE EXPO ARRIVAL TIME DISTRIBUTION
```

```
 101  CONTINUE                                                          68
      R=RAND(0.0)                                                       69
      TNARV=ABS(ARRRT*ALOG(R))                                          70
      TNARV=TNARV+TIME                                                  71
         GO TO 141                                                      72
C     CONSTANT ARRIVAL TIME                                             73
 111  CONTINUE                                                          74
      TNARV=TIME+ARRTM                                                  75
         GO TO 141                                                      76
C     READ IN ARRIVAL TIMES                                             77
 121  CONTINUE                                                          78
      NUM=ANUM                                                          79
      IZQQ=IZ+1                                                         80
         IF (IZQQ.LE.NUM) GO TO 131                                     81
      AR(IZQQ)=777777.                                                  82
      NFLAG=76                                                          83
 131  CONTINUE                                                          84
      TNARV=AR(IZQQ)                                                    85
 141  CONTINUE                                                          86
      IZ=IZ+1                                                           87
         IF (IZ.GT.20) GO TO 151                                        88
      CH(IZ,1)=TNARV                                                    89
C       BEGIN LOGIC TO STORE ARRIVAL AND SERVICE TIMES IN QUEUE ARRAYS  90
C       FOR EACH WAITING CUSTOMER/PRODUCT                               91
C       DETERMINE SERVICE TIME FOR THE NEW ARRIVAL                      92
 151  CONTINUE                                                          93
         GO TO (161,171,181,191), KS                                    94
C     POISSON SERVICE RATE                                              95
 161  CONTINUE                                                          96
      R=RAND(0.0)                                                       97
      T=ABS(DEPRT*ALOG(R))                                              98
         GO TO 211                                                      99
C     NEGATIVE EXPONENTIAL SERVICE TIME                                 100
 171  CONTINUE                                                          101
      R=RAND(0.0)                                                       102
      T=ABS(DEPTM*ALOG(R))                                              103
         GO TO 211                                                      104
C     CONSTANT SERVICE TIME                                             105
 181  CONTINUE                                                          106
      T=DEPTM                                                           107
         GO TO 211                                                      108
C     READ-IN SERVICE TIME.                                             109
 191  CONTINUE                                                          110
      NUM=SNUM                                                          111
         IF (IZ.LE.NUM) GO TO 201                                       112
      NFLAG=76                                                          113
      T=666666.                                                         114
         GO TO 211                                                      115
 201  CONTINUE                                                          116
      T=SR(IZ)                                                          117
 211  CONTINUE                                                          118
      QSVCT(IQUE)=T                                                     119
C       IF ONLY ONE IN QUE - NO SCHEDULING NECESSARY                    120
         IF (IQUE.LE.2) GO TO 241                                       121
      IQQ=IQUE                                                          122
C       USE SCHEDULE RULE TO REORDER THE QUEUE FOR PROCESSING           123
C       KRULE = 1 FOR RANDOM                                            124
C       KRULE = 2 FOR FCFS                                              125
C       KRULE = 3 FOR SOT                                               126
         GO TO (231,241,221), KRULE                                     127
C     C        SOT SCHEDULE RULE                                       128
 221  CONTINUE                                                          129
         IF (IQQ.LE.2) GO TO 241                                        130
         IF (QSVCT(IQQ).GE.QSVCT(IQQ-1)) GO TO 241                      131
      TS=QSVCT(IQQ-1)                                                   132
      QSVCT(IQQ-1)=QSVCT(IQQ)                                           133
```

```
      QSVCT(IQQ)=QTS                                                    B  134
      IQUE=IQUE+1                                                       B  135
         GO TO 221                                                      B  136
C      RANDOM SCHEDULE RULE                                            B  137
  231 CONTINUE                                                         B  138
      QI=IQUE                                                           B  139
      IQQ=(RAND(0.99)*QI)                                               B  140
      IQQ=IQQ+1                                                         B  141
         IF (IQQ.LE.1) GO TO 241                                        B  142
      QTS=QSVCT(IQUE)                                                   B  143
      QSVCT(IQUE)=QSVCT(IQQ)                                            B  144
      QSVCT(IQQ)=QTS                                                    B  145
C     QUEUE IS SCHEDULED - RETURN                                       B  146
  241 CONTINUE                                                         B  147
      RETURN                                                            B  148
      END                                                               B  149-

      SUBROUTINE DEPART                                                 C    1
      COMMON ALPHA(10),ANUM,ARRRT,APRTM,CH(20,10),CUMUTL,CUSERV,DEPRT,  C    2
     1       DEPTM,I,IUSERV,IZ,KA,KCUS,KS,N,NFLAG,SNUM,STATUS(9),T,     C    3
     2       TIME,TTIME,TNARV,TNDPR(9),AVARRA,AVSERV,AMNIS,AMTIS,       C    4
     3       VCOST,VCOSTS,FCOST,FCOSTS,KRULE,IQUE,CUMQUE(101),          C    5
     4       QSVCT(101),AR(500),SR(500)                                 C    6
C     THIS SUBROUTINE PROCSSSES THE DEPARTURE OF EVERY CUSTOMER         C    7
C     IT UPDATES THE HOURS SPENT IN THE QUE                             C    8
C     IT UPDATES THE CLOCK TO THE NEXT DEPARTURE TIME (PREVIOUSLY       C    9
C        SELECTED)                                                      C   10
C     IT CHECKS THE LENGTH OF THE QUE                                   C   11
C     IF NO ONE IN THE QUE IT SETS THE CHANNEL AT AN IDLE STATUS (THIS  C   12
C        DEPARTURE WAS PREVIOUSLY PARTIALLY PROCESSED EITHER AT         C   13
C        ARRIVE OR BY A PRIOR PASS THROUGH DEPART)                      C   14
C     IF A QUE EXIST THEN TAKE ONE FROM QUE, ITS DEPARTURE TIME,        C   15
C        SET THE CHANNEL AT A BUSY STATUS AND RETURN                    C   16
C     CHECK LENGTH OF QUE, IF OVER 99 HOLD AT 99                        C   17
         IF (IQUE.LT.100) GO TO 11                                      C   18
      IQUE=100                                                          C   19
C     UPDATE THE HOURS SPENT IN QUEUE                                   C   20
   11 CONTINUE                                                          C   21
      CUMQUE(IQUE)=CUMQUE(IQUE)+TNDPR(I)-TIME                           C   22
C     UPDATE THE CLOCK TO NEXT DEPARTURE TIME                           C   23
      TIME=TNDPR(I)                                                     C   24
         IF (IQUE.GT.1) GO TO 21                                        C   25
C     THIS SECTION COMPLETES THE PROCESSING OF A CUSTOMER               C   26
C     WHEN NO ONE IS WAITING IN THE QUE                                 C   27
      STATUS(I)=0.0                                                     C   28
      TNDPR(I)=999999.9                                                 C   29
      RETURN                                                            C   30
C     THIS SECTION DOES THE DEPART PROCESSING                           C   31
C     WHEN THE CHANNEL HAS BEEN BUSY                                    C   32
C     SET NEXT DEPARTURE TIME                                           C   33
   21 CONTINUE                                                          C   34
      TNDPR(I)=TIME+QSVCT(I)                                            C   35
C     STORE FIRST TWENTY DEPARTURE TIMES IN CH                          C   36
      KCUS=KCUS+1                                                       C   37
         IF (KCUS.GT.20) GO TO 31                                       C   38
      II=I+1                                                            C   39
      CH(KCUS,II)=TNDPR(I)                                              C   40
   31 CONTINUE                                                          C   41
C     CHECK IF OUT OF SIMULATION TIME                                   C   42
         IF (TTIME.LT.TNDPR(I)) GO TO 51                                C   43
C     RESET STATUS BACK TO BUSY AND RETURN                              C   44
      CUMUTL=CUMUTL+QSVCT(I)                                            C   45
      CUSERV=CUSERV+1.0                                                 C   46
      STATUS(I)=1.0                                                     C   47
C     SHIFT SERVICE QUEUE UP ONE POSITION                               C   48
      DO 41 I=1,IQUE                                                    C   49
```

```
      [illegible]                                                       C  51
      QSVCT(III)=QSVCT(III)                                             C  52
   61 CONTINUE                                                          C  53
      QSVCT(IQUE)=0.0                                                   C  54
      SUBRTACT ONE FROM QUE                                             C  55
      IQUE=IQUE-1                                                       C  56
      RETURN                                                            C  57
C     ADJUST TK AND CUMUTL AT TERMINATION OF SIMULATION                 C  58
C     LAST CUSTOMER FORCED TO DEPART AT TTIME                           C  59
    5 CONTINUE                                                          C  60
      TK=QSVCT(1)-(TNDPR(I)-TTIME)                                      C  61
         IF (TK.GT.0.0) GO TO 61                                        C  62
      TK=0.                                                             C  63
   61 CONTINUE                                                          C  64
      TNDPR(I)=TTIME                                                    C  65
      CUMUTL=CUMUTL+TK                                                  C  66
      STATUS(I)=1.0                                                     C  67
      RETURN                                                            C  68
      END

      FUNCTION RAND (K,NN)                                              D   1
C     MACHINE DEPENDENT RANDOM NUMBER GENERATOR (0 TO 1)                D   2
C     THIS VERSION FOR CDC 3100                                         D   3
C     K SET AT POSITIVE ODD INTEGER TO INITIALIZE                       D   4
C     K SET AT ZERO TO CONTINUE STRING OF RANDOM NUMBERS                D   5
C     SEE NAYLOR,COMPUTER SIMULATION TECHNIGUES,WILEY +SONS,1966        D   6
         IF (K) 21,21,11                                                D   7
   11 CONTINUE                                                          D   8
      N=K                                                               D   9
      NN=K                                                              D  10
      NNN=K                                                             D  11
   21 CONTINUE                                                          D  12
         IF (KK) 31,31,61                                               D  13
   31 CONTINUE                                                          D  14
      N=N*2051                                                          D  15
         IF (N) 41,51,51                                                D  16
   41 CONTINUE                                                          D  17
      N=N+8388607+1                                                     D  18
   51 CONTINUE                                                          D  19
      XN=N                                                              D  20
      RAND=XN/8388607.                                                  D  21
      RETURN                                                            D  22
C     POSITIVE KK RUNS SECOND STRING OF RANDOM NUMBERS                  D  23
   61 CONTINUE                                                          D  24
         IF (KK-50) 71,71,101                                           D  25
   71 CONTINUE                                                          D  26
      NN=NN*2051                                                        D  27
         IF (NN) 81,91,91                                               D  28
   81 CONTINUE                                                          D  29
      NN=NN+8388607+1                                                   D  30
   91 CONTINUE                                                          D  31
      XNN=NN                                                            D  32
      RAND=XNN/8388607.                                                 D  33
      RETURN                                                            D  34
C     KK OVER 5  RUNS THIRD STRING OF RANDOM NUMBERS                    D  35
  101 CONTINUE                                                          D  36
      NNN=NNN*2051                                                      D  37
         IF (NNN) 111,121,121                                           D  38
  111 CONTINUE                                                          D  39
      NNN=NNN+8388607+1                                                 D  40
  121 CONTINUE                                                          D  41
      NNN=NNN                                                           D  42
      RAND=XNNN/8388607.                                                D  43
      RETURN                                                            D  44
      END                                                               D  45
```