Anexo 1

Temario de los Cursos

Curso: Programación Estructurada

Temas:

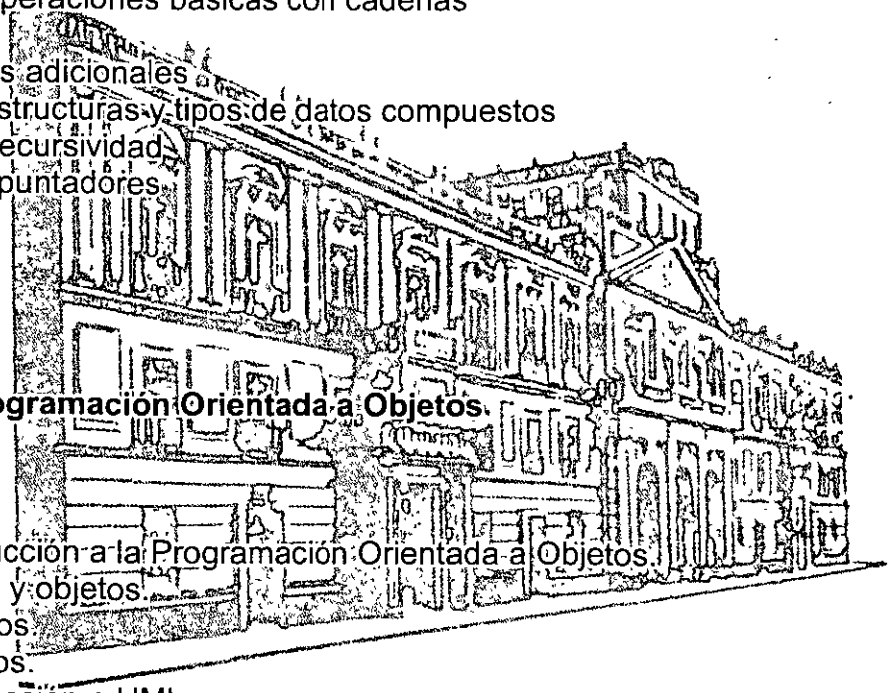
1. Introducción
   - 1.1 Historia de la programación
   - 1.2 Elaboración de un programa
2. Solución de problemas
   - 2.1 Uso de programas
   - 2.2 Algoritmos
   - 2.3 Pseudo-código
3. Elementos básicos de programación
   - 3.1 Datos y tipos de datos
   - 3.2 Operaciones primitivas
   - 3.3 Variables y expresiones
   - 3.4 Concepto de asignación
4. Modularidad
   - 4.1 Descomposición de un problema
   - 4.2 Concepto de función
   - 4.3 Paso de parámetros
   - 4.4 Estructura de programas
5. Estructuras de decisión
   - 5.1 If-then-else
   - 5.2 If anidados
   - 5.3 Condiciones compuestas
   - 5.4 Estructuras de tipo CASE
6. Estructura de repetición
   - 6.1 Estructura básica de un ciclo
   - 6.2 Ciclos anidados
   - 6.3 Contadores y terminación de ciclos
   - 6.4 For
   - 6.5 While y do-while

Palacio de Minería, Calle de Tacuba No. 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Centro Histórico, México D.F,
APDO Postal M-2285 • Tels: 5521.4021 al 24, 5623.2910 y 5623.2971 • Fax. 5510.0573

Anexo 1

7. Datos estructurados
   - 7.1 Vector como estructura de datos
   - 7.2 Índices
   - 7.3 Operaciones
   - 7.4 Búsquedas
   - 7.5 Ordenamientos
   - 7.6 Arreglos bidimensionales
   - 7.7 Cadenas de caracteres
   - 7.8 Vectores y cadenas
   - 7.9 Operaciones básicas con cadenas

8. Tópicos adicionales
   - 8.1 Estructuras y tipos de datos compuestos
   - 8.2 Recursividad
   - 8.3 Apuntadores



Curso: Programación Orientada a Objetos.

Temas:

1. Introducción a la Programación Orientada a Objetos.
2. Clases y objetos.
3. Atributos.
4. Métodos.
5. Introducción a UML.
6. Encapsulamiento.
7. Modelado y diseño orientado a objetos.
8. Restricciones.
9. Herencia.
10. Polimorfismo.
11. Paquetes.
12. Diseño de aplicaciones orientas a objetos.
13. Objetos típicos en un diseño.
14. Modelado dinámico.
15. Excepciones.
16. Otros diagramas UML.

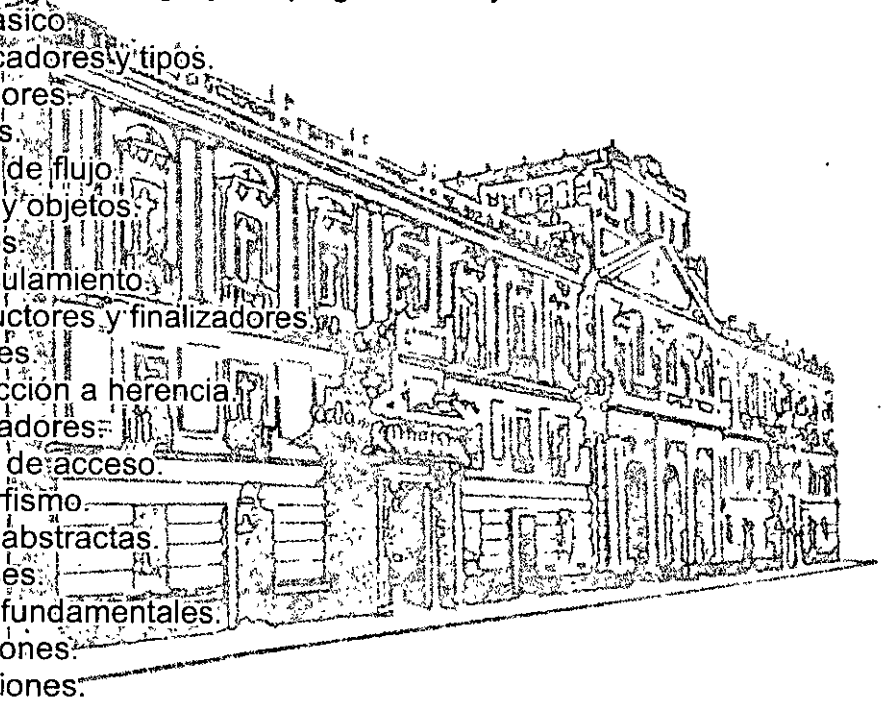
Palacio de Minería, Calle de Tacuba No 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Centro Histórico, México D.F.
APDO Postal M-2285 • Tels: 5521.4021 al 24, 5623.2910 y 5623.2971 • Fax 5510.0573

Anexo 1

Curso: Introducción al Desarrollo en Lenguaje Java.

Temas:

1. Introducción a programación orientada a objetos.
2. Introducción al lenguaje de programación java.
3. Java básico.
4. Identificadores y tipos.
5. Operadores.
6. Arreglos.
7. Control de flujo.
8. Clases y objetos.
9. Métodos.
10. Encapsulamiento.
11. Constructores y finalizadores.
12. Paquetes.
13. Introducción a herencia.
14. Modificadores.
15. Control de acceso.
16. Polimorfismo.
17. Clases abstractas.
18. Interfases.
19. Clases fundamentales.
20. Colecciones.
21. Excepciones.
22. Estilo.



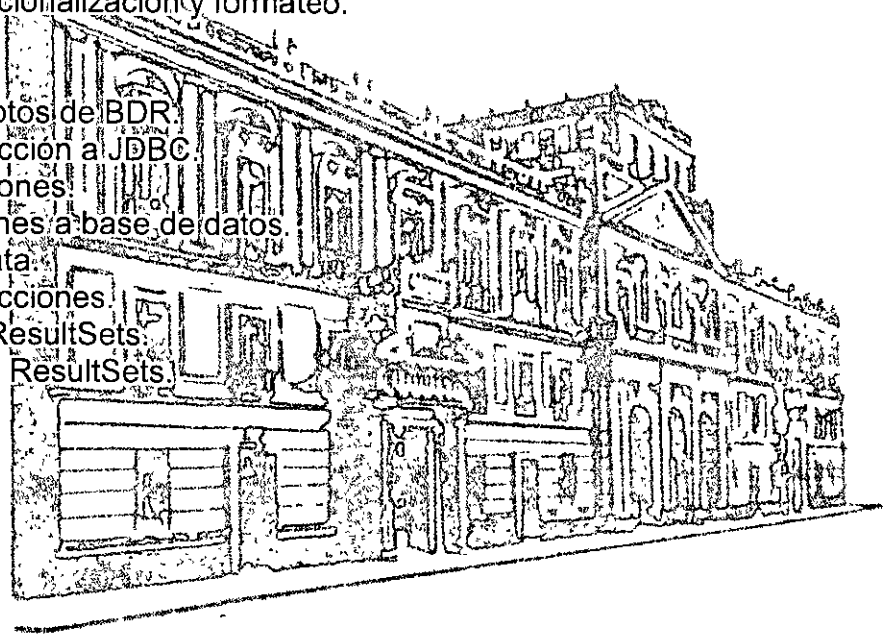
Palacio de Minería, Calle de Tacuba No 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Centro Histórico, México D.F.
APDO Postal M-2285 ■ Tels. 5521.4021 al 24, 5623 2910 y 5623.2971 ■ Fax: 5510.0573

Anexo 1

Curso: Características Avanzadas del Lenguaje Java.

Temas:

1. JVM
2. Java I/O.
3. Introducción a java Threads.
4. Internacionalización y formateo.
5. RMI.
6. JNDI.
7. Conceptos de BDR.
8. Introducción a JDBC.
9. Conexiones.
10. Peticiones a base de datos.
11. Metadata.
12. Transacciones.
13. Scroll ResultSets.
14. Update ResultSets.



Palacio de Minería, Calle de Tacuba No 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Centro Histórico, México D F,
APDO Postal M-2285 • Tels: 5521.4021 al 24, 5623.2910 y 5623.2971 • Fax: 5510 0573

FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

CURSO INSTITUCIONAL

MATERIAL DIDACTICO

PROGRAMACION ESTRUCTURADA

CURSO DIRIGIDO A PERSONAL DE LA COMPAÑÍA DE LUZ Y FUERZA DEL CENTRO

SEPTIEMBRE 26, 2005

Palacio de Minería, Calle de Tacuba No. 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Centro Histórico, México D.F.
APDO Postal M-2285 ■ Tels. 5521.4021 al 24, 5623.2910 y 5623 2971 ■ Fax: 5510 0573

Programación Estructurada

1. Introducción

1.1. Historia de la programación

Durante los años 60, el desarrollo de software se encontró con severas dificultades. Los programas de entrega del software se retrasaban, sus costos excedían en gran medida los presupuestos, y los productos terminados no eran confiables. La actividad de investigación de los años 60 dio como resultado la evolución de la programación estructurada- un método disciplinado de escribir programas que sean claros, que se demuestre que son correctos y fáciles de modificar.

Uno de los resultados más tangibles de esta investigación, fue el desarrollo en 1971 hecho por el profesor Nicklaus Wirth del lenguaje de programación Pascal. Pascal fue diseñado par4a la enseñanza de la programación estructurada en entorno académico, y se convirtió con rapidez en el lenguaje introductorio de programación de la mayor parte de las universidades.

ADA fue desarrollado bajo el patrocinio del Departamento de Defensa de los Estados Unidos (DOD) durante los años 70 y principios de los 80.

DOD deseaba un solo lenguaje que pudiera llenar sus objetivos. Pascal fue seleccionado como base, pero el lenguaje final ADA.

A Lady Lovelace se le da por lo general crédito de haber escrito el primer programa de computación del mundo a principios de 1800. Una capacidad importante de Ada se conoce como multitareas.

Otros lenguajes muy utilizados de alto nivel que hemos analizado incluyendo C y C++ permiten al programador escribir programas que solo ejecuten una actividad a la vez.

2. Solución de problemas

2.1. Elaboración de un programa

Un programa puede considerarse como una secuencia de acciones (instrucciones) que manipulan un conjunto de objetos (datos). Contendrá, por lo tanto, dos bloques para la descripción de los aspectos citados.

- Bloque de declaraciones: En él se especifican todos los objetos que utiliza el programa (variables, constantes, tablas, registros, archivos, etcétera).
- Bloque de instrucciones: Constituido por el conjunto de operaciones que se han de realizar para la obtención de los resultados deseados.

Fases para la elaboración de un programa

Programar es un proceso mental complejo, dividido en varias etapas. La finalidad de la programación es comprender con claridad el problema que va a resolverse o simularse por medio de la computadora, y entender también con detalle cual será el procedimiento mediante el cual la máquina llegará a la solución deseada. La actividad de programar es conceptual y su finalidad es intentar definir, cada vez con mayor precisión, acercamientos que resuelvan el problema de manera virtual, es decir, que efectúen una especie de experimentos mentales sobre el problema por resolver o simular. El resultado constituirá una descripción de los pasos necesarios para encontrar la solución.
La importancia de la programación consiste en que este lenguaje funciona a la vez como vehículo descriptor y como modelo de la representación dada a la solución; el lenguaje es neutro y completo (independiente a la máquina y capaz de expresar cualquier idea).

Definición y análisis del programa

Definición: Es necesario comprender el problema que va a resolverse. Si se toma en cuenta que los sistemas de programación reales son largos y complejos, que a veces implican la participación de varias personas, se podrá comprender la importancia de entender con claridad el problema antes de tratar de llevar a una solución.

Análisis: Un análisis completo del problema o sistema existente se realiza con la finalidad de proponer un modelo para su solución. Un sistema está formado por un conjunto estructurado de elementos interrelacionados entre sí de modo que es posible tener dos sistemas diferentes con componentes iguales.

La función de un analista de sistemas consiste en describir el modelo que mejor se adapte a la estructura del modelo que se estudia. Un enfoque funcional puede ser adecuado a muchos casos (se hace el análisis partiendo de la función que cada componente desempeña en el sistema como un todo). En otro tipo de problemas puede emplearse un análisis dirigido por los datos que maneja un sistema o por algún otro aspecto que pueda servir de guía. Es fundamental que esta recopilación de datos se documente para poder realizar la etapa de diseño.

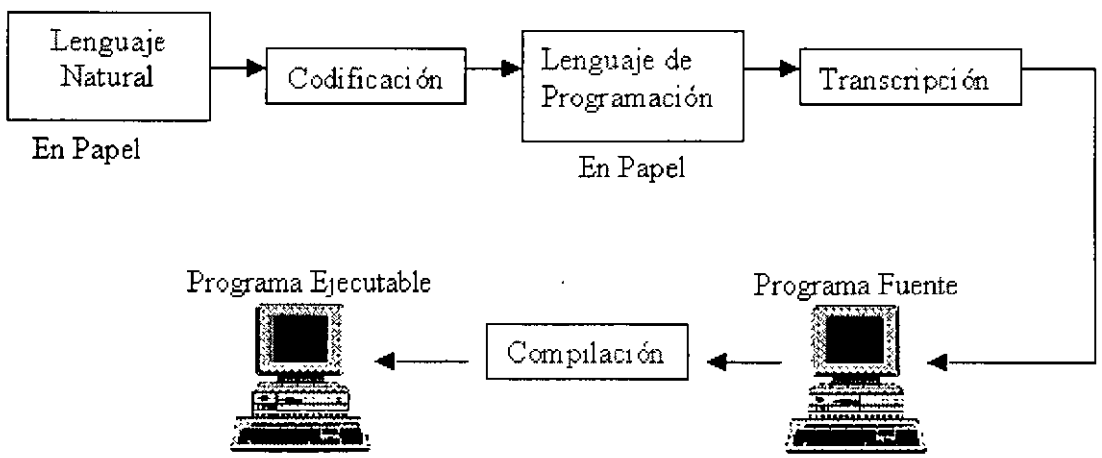
Diseño: El diseño del software es un proceso enfocado en cuatro atributos distintos del programa. La estructura de los datos, la arquitectura del software, los procedimientos que se llevarán a cabo y el diseño de la interfaz. Al diseñar el programa se establece la calidad requerida al representar la información obtenida (modelo) en la etapa de análisis. El diseño debe documentarse y formar parte de la configuración del software.

Pruebas de escritorio.

Cuando se tiene el modelo (algoritmo) con el que se pretende resolver el problema, se llevan a cabo pruebas en papel, o sea, se hace una comprobación utilizando varios datos que nos permitan establecer el correcto desempeño del modelo.

Codificación.

Una vez terminada la fase de programación, se habrá producido una descripción del modelo propuesto, escrita en pseudo código. El proceso mediante el cual se llega a un programa esencialmente correcto recibe el nombre de refinamientos progresivos. Un modelo no es ejecutable por medio de una computadora, el objetivo del refinamiento consiste en acercar el programa escrito en pseudo código a un programa escrito en lenguaje de programación.



Compilación o interpretación.

Una vez obtenido el pseudo código, se elige un lenguaje de programación y se traduce el pseudo código al lenguaje seleccionado, a ese programa se le conoce como programa fuente. La traducción puede llevarse a cabo por medio de un intérprete o un compilador. El intérprete lee el programa fuente, lo traduce y a continuación lo ejecuta. El compilador lee el archivo fuente, crea un archivo objeto, lo traduce a ensamblador y luego a lenguaje máquina. A este último se le conoce como programa ejecutable.

Validación.

Tras la codificación del programa, deberá ejecutarse en una computadora y a continuación comprobar los resultados obtenidos con el fin de verificar que los datos arrojados por el programa sean correctos, en caso de encontrar errores, el programador revisar nuevamente el algoritmo o el código de su programa.

Documentación y mantenimiento.

La documentación de un programa es el conjunto de información interna y externa al programa que facilitará su posterior mantenimiento.
La documentación pueden ser interna y externa.

Documentación interna: Está constituida por:

- Comentarios dentro del código del programa
- La presentación. Es la manera de estructurar el código para facilitar su lectura y comprensión.

Documentación externa: No está contenida dentro del código del programa. Está constituida por:

- Manual del Usuario
- Manual del Programador
- Manual de Mantenimiento del Programa
- Especificaciones del Programa
- Lista de Datos de Prueba (test) y Resultados
- Historia del desarrollo del programa, modificaciones posteriores
- Diseño descendente con detalle en módulos y sub módulos
- Versiones en uno y diferencia entre sí.

2.2. *Algoritmos*

Definición de Algoritmo

Conjunto específico de procedimientos matemáticos y lógicos simples y bien definidos que pueden seguirse para resolver un problema en un número determinado de pasos.

El conjunto de instrucciones que especifican la secuencia de operaciones a realizar en orden, para resolver un sistema específico o clase de problemas, se denomina algoritmo.

Las características fundamentales que debe cumplir un algoritmo son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso
- Debe estar definido (siempre se debe llegar al mismo resultado)
- Debe ser finito (debe terminar en algún momento, o sea tener un número finito de pasos)

Algoritmos Cotidianos

Se refiere a todos aquéllos algoritmos que nos ayudan a resolver problemas diarios, y que los hacemos casi sin darnos cuenta de que estamos siguiendo una metodología para resolverlos.

Algunos ejemplos son:

Diseñar un algoritmo para cambiar una llanta a un coche.

1. Inicio.
2. Traer gato.
3. Levantar el coche con el gato.
4. Aflojar tornillos de las llantas.
5. Sacar los tornillos de las llantas.
6. Quitar la llanta.
7. Poner la llanta de repuesto.
8. Poner los tornillos.
9. Apretar los tornillos.
10. Bajar el gato.

Fin

Un cliente ejecuta un pedido a una fábrica. La fábrica examina en su banco de datos la ficha del cliente, si el cliente es solvente entonces la empresa acepta el pedido, en caso contrario rechazar el pedido.

Pasos del algoritmo:

Inicio

Leer el pedido

Examinar ficha del cliente

Si el cliente es solvente aceptar pedido, en caso contrario rechazar pedido

Fin

Determinar el mayor de tres números enteros.

Pasos del algoritmo:

1.- Comparar el primero y el segundo entero, deduciendo cuál es el mayor.

2.- Comparar el mayor anterior con el tercero y deducir cuál es el mayor. Este será el resultado.

Los pasos anteriores se pueden descomponer en otros pasos más simples en los que se denomina refinamiento del algoritmo.

1.- Obtener el primer número (entrada), denominado NUM1

2.- Obtener el segundo número (entrada), denominado NUM2

3.- Compara NUM1 con NUM2 y seleccionar el mayor; si los dos enteros son iguales, seleccionar NUM1. Llamar a este número MAYOR.

4.- Obtener el tercer número (entrada), y se denomina NUM3.

5.- Compara MAYOR con NUM3 y seleccionar el mayor; si los dos enteros son iguales, seleccionar el MAYOR. Denominar a este número MAYOR.

6.- Presentar el valor MAYOR (salida).

7.- Fin

Definición de Lenguajes para Algoritmos

Los algoritmos pueden describirse utilizando diversos lenguajes. Cada uno de estos lenguajes permiten describir los pasos con mayor o menor detalle.

La clasificación de los lenguajes para algoritmos puede enunciarse de la siguiente manera:

- Lenguaje Natural.

- Lenguaje de Diagrama de Flujo.
- Lenguaje Natural de Programación.
- Lenguaje de Programación de Algoritmos.

Lenguaje Natural

Es aquél que describe en español, para nuestro caso, los pasos a seguir utilizando un vocabulario cotidiano. Se le conoce como lenguaje jerga cuando se utilizan términos especializados de una determinada ciencia, profesión o grupo.

Lenguaje de Diagrama de Flujo

Es aquél que se vale de diversos símbolos para representar las ideas o acciones a desarrollar. Es útil para organizar las acciones o pasos de un algoritmo pero requiere de etapas posteriores para implementarse en un sistema de cómputo.

Lenguaje Natural de Programación

Son aquéllos que están orientados a la solución de problemas que se definen de una manera precisa. Generalmente son aplicados para la elaboración de fórmulas o métodos científicos.

Tiene las siguientes características:

- Evita la ambigüedad (algo confuso que se puede interpretar de varias maneras).
- Son precisos y bien definidos.
- Utilizan términos familiares al sentido común.
- Elimina instrucciones innecesarias.

Lenguaje de Programación de Algoritmos

Es aquél que se utiliza para introducir en la computadora un algoritmo específico. Se les conoce también como Lenguaje de Programación.

Lenguaje de Programación:

Es un conjunto de palabras, símbolos y reglas sintácticas mediante los cuales puede indicarse a la computadora los pasos a seguir para resolver un problema.

Los lenguajes de programación pueden clasificarse por diversos criterios, siendo el más común su nivel de semejanza con el lenguaje natural, y su capacidad de manejo de niveles internos de la máquina.

Los principales tipos de lenguajes utilizados son tres:

- Lenguaje Máquina.
- Lenguaje de bajo Nivel (ensamblador).
- Lenguajes de Alto Nivel.

Lenguaje Máquina

Son aquéllos que están escritos en lenguajes directamente inteligibles por la máquina (computadora), ya que sus instrucciones son cadenas binarias (cadenas o series de caracteres de dígitos 0 y 1) que especifican una operación y las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario.

Las instrucciones en lenguaje máquina dependen del hardware de la computadora y, por tanto, diferirán de una computadora a otra.

Ventajas del Lenguaje Máquina

- Posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior, lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Desventajas del Lenguaje Máquina

- Dificultad y lentitud en la codificación.
- Poca fiabilidad.
- Gran dificultad para verificar y poner a punto los programas.
- Los programas solo son ejecutables en el mismo procesador (CPU).

En la actualidad, las desventajas superan a las ventajas, lo que hace prácticamente no recomendables a los lenguajes máquinas.

Lenguajes de Bajo Nivel

Son más fáciles de utilizar que los lenguajes máquina, pero al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como nemotécnicos. Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés: ADD, SUB, DIV, etc.; en español: SUM, RES, DIV, etc.

Una instrucción típica de suma sería:

ADD M, N, P

Esta instrucción significa "sumar el contenido en la posición de memoria M al número almacenado en la posición de memoria N y situar el resultado en la posición de memoria P". Evidentemente es más sencillo recordar la instrucción anterior con un nemotécnico que su equivalente en código máquina.

0110 1001 1010 1011

Un programa escrito en lenguaje ensamblador, requiere de una fase de traducción al lenguaje máquina para poder ser ejecutado directamente por la computadora.

El programa original escrito en lenguaje ensamblador se denomina programa fuente y el programa traducido en lenguaje máquina se conoce como programa objeto, el cual ya es directamente entendible por la computadora.

Ventajas del lenguaje ensamblador frente al lenguaje máquina

- Mayor facilidad de codificación y, en general, su velocidad de cálculo.

Desventajas del lenguaje ensamblador

- Dependencia total de la máquina lo que impide la transportabilidad de los programas (posibilidad de ejecutar un programa en diferentes máquinas. El lenguaje ensamblador del PC es distinto del lenguaje ensamblador del Apple Machintosh.

La formación de los programadores es más compleja que la correspondiente a los programadores de alto nivel, ya que exige no solo las técnicas de programación, sino también el conocimiento del interior de la máquina.

Los lenguajes ensamblador tienen sus aplicaciones muy reducidas, se centran básicamente en aplicaciones de tiempo real, control de procesos y de dispositivos electrónicos.

Lenguajes de Alto Nivel

Estos lenguajes son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Un programa escrito en lenguaje de alto nivel es independiente de la máquina (las instrucciones no dependen del diseño del hardware o de una computadora en particular), por lo que estos programas son portables o transportables. Los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras.

Ventajas de los lenguajes de alto nivel

- El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes.
- La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos. Nombres de las instrucciones tales como READ, WRITE, PRINT, OPEN, etc.
- Las modificaciones y puestas a punto de los programas son más fáciles.
- Reducción del coste de los programas.
- Transportabilidad.

Desventajas de los lenguajes de alto nivel

- Incremento del tiempo de puesta a punto al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo.
- No se aprovechan los recursos internos de la máquina que se explotan mucho mejor en lenguajes máquina y ensambladores.
- Aumento de la ocupación de memoria.
- El tiempo de ejecución de los programas es mucho mayor.

Para una mejor compresión de este tema se definirá el concepto de programa, por ser este un término muy utilizado en el diseño estructurado de algoritmos.

2.3. Pseudocódigo.

Es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil.

El pseudocódigo nació como un lenguaje similar al inglés y era un medio representar básicamente las estructuras de control de programación estructurada. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. Cabe señalar que el pseudocódigo no puede ser ejecutado por una computadora.

La ventaja del pseudocódigo es que en su uso en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa, además de todo esto es fácil su traducción a lenguajes como pascal, COBOL, C, FORTRAN o BASIC.

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés (similares a sus homónimos en los lenguajes de programación), tales como ***star,begin, end, stop, if-then-elsė, while, repeat-until....***etc

Secuencia

Inicio
    acción1
    acción2
    :
    acción n
Fin

Decisión

Simple

si **condición** entonces
        acción1
        acción2
:
acción n

Doble

si **condición** then
        acción1
        acción2
:
en caso contrario
        acción1
        acció2

Iteracción

Fija

para **var. Entera inicial** hasta **final** hacer
        acción1
        acción2
            :

acción n

Condicional al inicio

mientras **condición** hacer

acción1

acción2

:

acción n

Condicional al final

Repetir

acción1

acción2

:

acción n

Hasta que **condición**

Selección

casos ***selector*** de

valor1 : acción1

acción2

valor2 : acción1

acción2

valor n : acción1

acción2

Ejercicio:

Se requiere preguntar dos valores, y a continuación ofrecer un menú con las operaciones básicas (+, -, *, /). Después de presentar el resultado se ofrecerá la posibilidad de una nueva operación.

Declaración de variables:

```
Real : X, Y, RESPUESTA
Entero : OPCION
Carácter : OP
Inicio
   Repetir
escribir('Primer valor : ' )
leer(X)
escribir('Segundo valor : ')
leer(Y)
escribir('1) Suma ')
escribir('2) Resta ')
escribir('3) Multiplicación ')
escribir('4) División ')
escribir('Qué operación deseas realizar ? : ')
leer(OPCION)
casos OPCION de
1 : RESULTADO ← X+Y
2 : RESULTADO ← X-Y
3 : RESULTADO← X*Y
4 :   si Y=0 entonces
 escribir(' Error ')
 RESULTADO← 0
en caso contrario
RESULTADO← X/Y
escribir ('Resultado : ',RESULTADO)
escribir('Deseas otro cálculo : [S/N] ')
leer(OP)
   Hasta que RES = 'N'
Fin
```

Ejercicio:

Preguntar un nombre y repetirse en pantalla tantas veces como se desee. (preguntar cantidad de repeticiones), indicando el número de repetición.

Declaración de variables

Cadena : nom

entero : x, n

Inicio

escribir('Nombre : ')

leer(nom)

escribir('Cuántas veces quieres repetirlo ? : ')

leer(n)

para x ← 1 **hasta** n **hacer**

  escribir(x'.- ', nom)

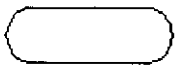
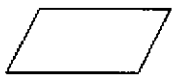
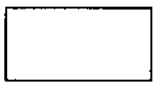
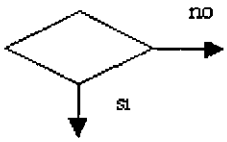
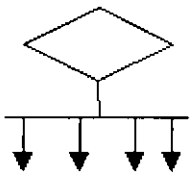
Fin

2.4. *Diagrama de flujo*

Se basan en la utilización de diversos símbolos para representar operaciones específicas. Se les llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación.

La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a un patrón definido previamente.

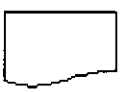
SIMBOLOGIA UTILIZADA EN LOS DIAGRAMAS DE FLUJO

Símbolo	Función
	Terminal (representa el Inicio y el Final, de un programa, puede representar también una parada o interrupción programada que sea necesario realizar en un programa.
	Entrada/Salida (cualquier tipo de introducción de datos)
	Proceso (cualquier tipo de operacion que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas)
	Desición (indica operaciones lógicas o de comparación entre datos -normalmante dos- y en función del resultado de la misma determina cual de los distintos caminos alternativos del programa se debe seguir)
	Desición múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).
	Conector (Sirve para enlazae dos partes cualesquiera de un organigrama a través de un conector en la salida y otro conector en la salida)
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones)
	Línea conectora (sirve de unión entre dos símbolos).

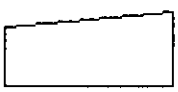
Conector (Conexión entre dos puntos del organigrama situado en páginas diferentes)

Llamada a subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del progama)

Pantalla (se utiliza en ocasiones en lugar del símbolo de E/S)



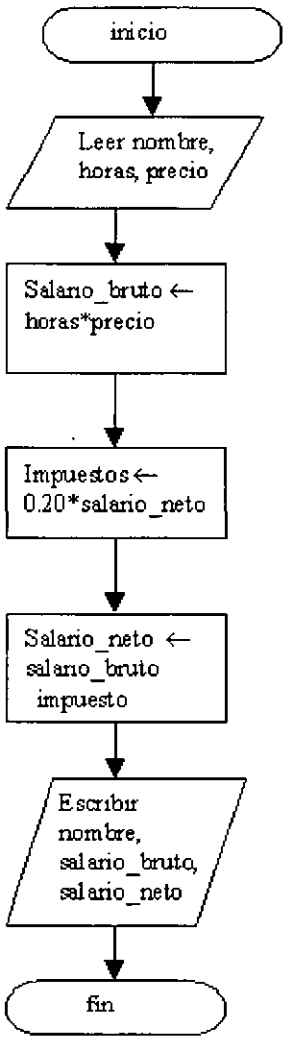
Impresora (se utiliza en ocasiones en lugar del símbolo de E/S).



Teclado (Se utiliza en ocasiones en lugar del símbolo de E/S).

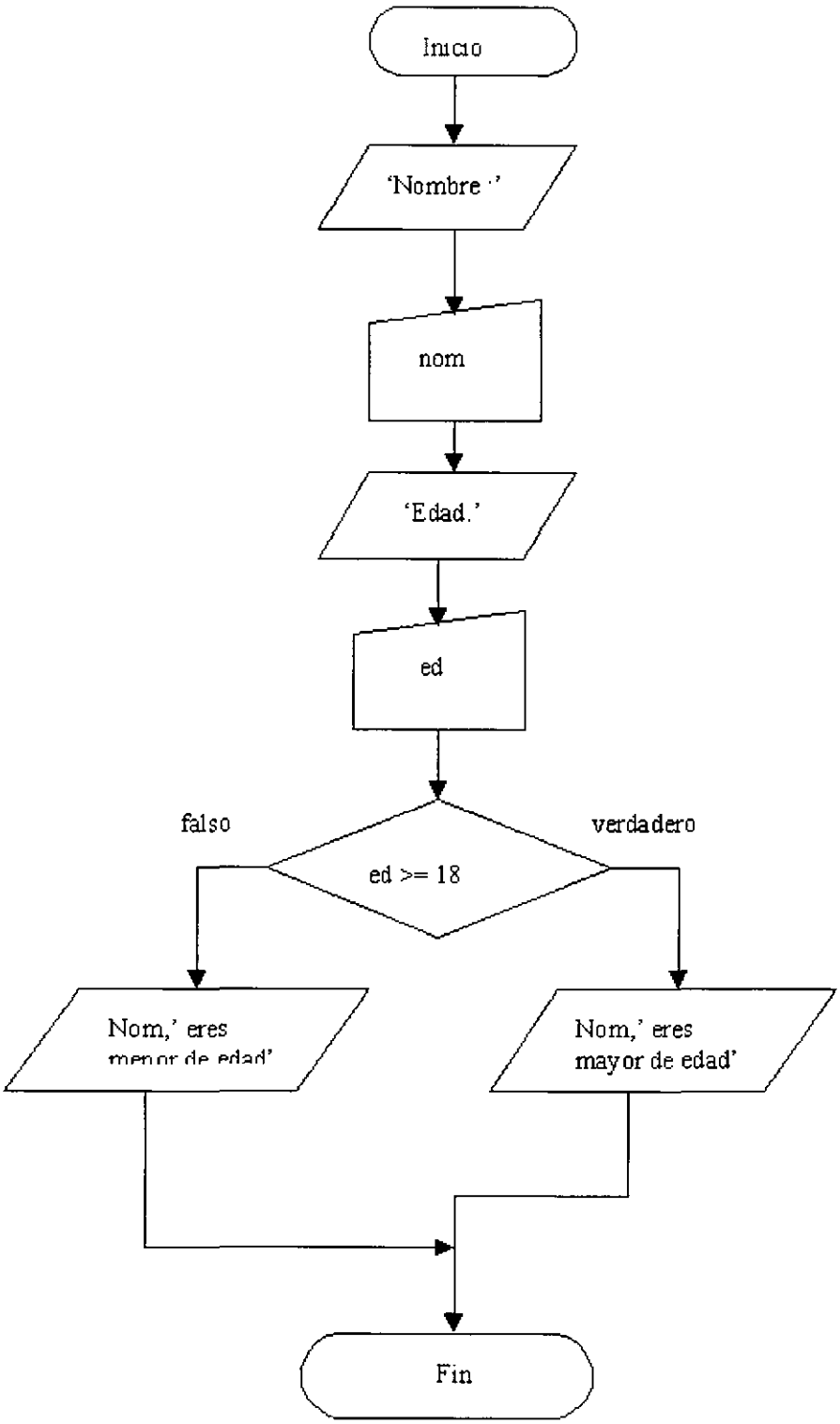
Ejemplo:

Calcular el salario neto de un trabajador en función del número de horas trabajadas, precio de la hora de trabajo y considerando unos descuentos fijos al salario bruto en concepto de impuestos (20 por 100).



Ejemplo:

Realizar un diagrama de flujo que permita mostrar en pantalla un mensaje de mayoría o minoría de edad según sea el caso para un nombre específico.



3. Elementos básicos de programación

3.1. Datos y tipos de datos

Un **dato** se define como la expresión general que describe los objetos con los cuales opera una computadora. Los datos de entrada se transforman por el programa, después de las etapas intermedias, en datos de salida.

Los datos se clasifican en diversas categorías, según el tipo de máquina o del lenguaje en uso. Generalmente podemos encontrar las siguientes categorías:

- Numéricos
- Lógicos
- Cadenas

Datos Numéricos

Son aquéllos que representan una cantidad o valor determinado. Su representación se lleva a cabo en los formatos ya conocidos (enteros, punto y fracciones decimales si estas existen).

Estos pueden representarse en dos formas distintas:

- Tipo Numérico Entero (integer).
- Tipo Numérico Real (real).

Enteros

Es un conjunto finito de los números enteros. Los enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser negativos y positivos.

Algunos ejemplos son :

- 3 7
- -10 9
- 15.25
- 50

Reales

Consiste en un subconjunto de los números reales. Estos números siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consiste de un número entero y una parte decimal. Algunos ejemplos son:

- 0.52 664.32
- 6.579 8.0
- -9.3 -47.23

Cadenas

Son los datos que representan información textual (palabras, frases, símbolos, etc). No representan valor alguno para efectos numéricos. Pueden distinguirse porque son delimitados por apóstrofes o comillas.

Se clasifica en dos categorías:

- Datos tipo carácter (char)
- Datos tipo Cadena (string)

Datos Tipo Carácter

Es un conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato de este tipo contiene solo un carácter.

Reconoce los siguientes caracteres:

- Caracteres Alfabéticos (A,B,C,...Z,a,b,c...z)
- Caracteres Numéricos (0,1,2,...9)
- Caracteres Especiales (+, -, *, /, ^, . , ;, <, >, $, ...)

Datos Tipo Cadena (string)

Es un sucesión de caracteres que se encuentran delimitados por una comilla (apóstrofe) o dobles comillas, según el tipo de lenguaje de programación. La longitud de una cadena de caracteres es el número de ellos comprendidos entre los separadores o delimitadores.

Ejemplos:

- 'Hola Mortimer'
- '12 de octubre de 1496'
- 'Enunciado cualquiera'

Nota: Los símbolos disponibles para la formulación de caracteres y de cadenas son aquéllos que se encuentran en el código ASCII.

ASCII (American Standard Code for Information Interchange).

Lógicos

También se le denomina Booleano, es aquél dato que solo puede tomar uno de dos valores : Falso y verdadero.

Se utiliza para representar las alternativas (si/no) a determinadas condiciones. Por ejemplo, cuando se pide si un valor entero sea primo, la respuesta será verdadera o falsa, según sea.

Las categorías y tipos que se mencionaron anteriormente se conocen como Tipos **Simples**, puesto que no poseen una estructura compleja.

En forma adicional, cada lenguaje puede proporcionar la utilización de ***Tipos Compuestos***, siendo estos, datos que tienen una estructura predeterminada.

Tipos Compuestos

Entre los principales tipos compuestos se encuentran los siguientes:

a.- **SUBRANGO**: Son aquéllos en los que se especifica con precisión el intervalo de valores válidos para un dato.

Ejemplos:

0..100 (son enumerativos de tipo entero)

'A'..'Z' (son enumerativos de tipo cadena)

Los ***Reales*** no son válidos para crear enumerativos, ya que su intervalo no está definido.

b.- **ENUMERATIVOS** : Son aquéllos en los que se definen individualmente los valores para un dato.

Ejemplos:

(0,25,40,52) Siempre deben ponerse entre paréntesis.

c.- **DEFINIDOS POR EL USUARIO:** Son aquéllos que el programador crea para satisfacer las necesidades del programa en diseño.

10.1. Operaciones primitivas

3.2. Constantes y variables

Una ***Constante*** es aquélla que no cambia de valor durante la ejecución de un programa (o comprobación de un algoritmo en este caso). Se representa en la forma descrita para cada categoría.

Las ***Variables*** son aquéllas que pueden modificar su valor durante la ejecución de un programa.

Su representación se da a través de letras y símbolos generalmente numéricos a los que se les asigna un valor.

Ejemplos:

	Constantes	Variables
Numéricos	36 450.35 0.58	A Nom Edad
Cadena	'A' 'Juan' 'La Paz'	Ciudad Estatura
Lógicos	Falso Verdadero	

3.3. Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operadores, paréntesis y nombres de funciones especiales. Las mismas ideas son utilizadas en notación matemática tradicional;

por ejemplo :

a + b (b+2) Aquí los paréntesis indican el orden de cálculo.

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Una expresión consta de operadores y operándoos. Según sea el tipo de objetos que manipulan, las expresiones se clasifican en :

- Aritméticas
- Relacionales
- Lógicas
- Carácter

El resultado de la expresión numérica es de tipo numérico; el resultado de una expresión relacional y de una expresión lógica es de tipo lógico ; el resultado de una expresión carácter es de tipo carácter.

Expresiones Aritméticas

Las expresiones aritméticas son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (real o entera) y las operaciones son las aritméticas.

+	suma
-	resta
*	multiplicación
/	división
**, ^	exponenciales
div	división entera
mod	módulo (resto)

Los cálculos que implican tipos de datos reales y enteros suelen dar normalmente resultados del mismo tipo si los operándoos lo son también. Por ejemplo, el producto de operándoos reales produce un real.

Ejemplo :

4 x 6se representa por 4 * 6

3^9 se representa por 3 ^ 9

18 div 6 se representa por 18/6

Expresiones Lógicas (booleanas)

Es una expresión que solo pueden tomar los valores de : verdadero y falso. Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones y otras expresiones lógicas utilizando los operadores lógicos not, and y or, y los operadores relacionales (de relación o comparación) =, >, <, <=, >=, <>.

3.4. Operadores

Un operador es el símbolo que determina el tipo de operación o relación que habrá de establecerse entre los operándoos para alcanzar un resultado.

Los operadores se clasifican en tres grupos:

- Aritméticos.
- Relaciónales.
- Lógicos.

Operadores Aritméticos

Son aquéllos que permiten la realización de cálculos aritméticos. Utilizan operandos numéricos y proporcionan resultados numéricos.

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División real
Div	División entera
Mod	Residuo
^	Exponenciación

Ejemplos:

7+3 = 10 10 Div 4 = 2

7-3 = 4 20 Mod 3 = 2

7*3 = 21 5 Mod 7 = 5

10/4= 2.5 4 ^ 2 = 16

En la expresión 7+3, los valores 7 y 3 se denominan ***operándoos***. El valor de la expresión 7+3 se conoce como ***resultado*** de la expresión.

Todos los operadores aritméticos no existen en todos los lenguajes de programación, por ejemplo, en Fortran no existen Div y mod.

Operadores Div y Mod

El símbolo / se utiliza para la división real, y el operador Div representa la división entera.

Expresión	Resultado	Expresión	Resultado
10.5/3.0	3.5	10 Div 3	3
1/4	0.25	18 Div 2	9
2.0/4.0	0.5	30 Div 30	1
30/30	1.0	10 Mod 3	1
6/8	0.75	10 Mod 2	0

Operadores Relaciónales

Permiten realizar comparaciones de valores de tipo numérico o carácter. Estos operadores sirven para expresar las condiciones en los algoritmos. Proporcionan resultados lógicos.

Operador	Significado
<	Menor que
>	Mayor que
=	Igual que
<=	Menor o igual que
>=	Mayor o igual que
<>	Diferente de

El formato general para las comparaciones es:

expresión1 operador de relación *expresión2*

El resultado de la operación será Verdadero o Falso.

Así por ejemplo, si

A=4 y B=3, entonces:

A>B Es Verdadero

(A-2) < (B-4) Es Falso

Los operadores de relación se pueden aplicar a cualquiera de los cuatro tipos de datos estándar: enteros, real, lógico y carácter.

'A' < 'K' = Verdadero

'A' > 'a' = Falso

'MARIA' < 'JUAN' = Falso (se considera la primera letra)

'JAIME' > 'JORGE' = Falso

Nota: La comparación de cadenas se rige por el código ASCII.

Prioridad De Operadores Aritméticos y Relacionales

Determina el orden en que habrán de realizarse las operaciones en una expresión determinada. Para obtener la prioridad se deben conocer las siguientes reglas:

- Las operaciones que están encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.
- Las operaciones aritméticas dentro de una expresión suelen seguir el siguiente orden de prioridad.

Operador	Prioridad
^	Alta
*, /, Div	
+, -, Mod	
Relaciónales	Baja

En caso de coincidir varios operadores de igual prioridad en una expresión o sub expresión encerrada entre paréntesis, el orden de prioridad en este caso es de izquierda a derecha.

Cuando se desea realizar una operación con baja prioridad por adelantado, debe agruparse a los operándoos involucrados.

4 + 12 /2 = 10 (sin agrupar)

(4 + 12) /2 = 8 (con agrupador)

Ejemplo:

Obtener los resultados de las expresiones:

-4 * 7 + 2 ^ 3 / 4 - 5

Solución:

-4 *7 + 2 ^ ¾

Resulta:

-4 * 7+ 8/4 -5

-28 + 8/4 -5

-28 + 2 - 5

-26 - 5

-31

Los paréntesis tienen prioridad sobre el resto de las operaciones.

A * (B+3)	La constante 3 se suma primero al valor de B, después este resultado se multiplica por el valor de A.
(A*B) +3 A y B	Se multiplican primero y a continuación se suma 3.
A + (B/C) + D	Esta expresión equivale a A+ B/C + D

Operadores Lógicos

Son aquéllos que permiten la combinación de condiciones para formar una sola expresión lógica. Utilizan operándoos lógicos y proporcionan resultados lógicos también.

Operador	Relación
not	Negación (No)
and	Conjunción (Y)
or	Disyunción (O)
xor	Disyunción Exclusiva (O/SOLO)

Se obtiene Verdadero si:	
NOT	El operando es falso
AND	Ambos operándoos son verdaderos
OR	Al menos un operando es verdadero
XOR	Solo uno de lòs operándoos son verdadero

X	Y	NOT(X)	NOT(Y)	X	X OR	X

				AND Y	Y	XOR Y
F	**F**	V	V	F	F	F
V	**F**	F	V	F	V	V
F	**V**	V	F	F	V	V
V	**V**	F	F	V	V	F

Prioridad De Los Operadores Lógicos

Los operadores aritméticos seguían un orden específico o de prioridad cuando existían más de un operador en las expresiones. De modo similar los operadores lógicos y relacionales tienen un orden de prioridad.

Ejemplos:

Not 4 > 6 Produce un error, ya que el operador **not** se aplica a 4.

Not (4 > 14) Produce un valor verdadero.

(1.0<x) And (x<z +7.0) Si x vale 7 y z vale 4, se obtiene un valor falso.

3.5. Asignación

La operación de asignación es el modo de darle valores a una variable. La operación de asignación se representa por el símbolo u operador . La operación de asignación se conoce como instrucción o sentencia de asignación cuando se refiere a un lenguaje de programación.

A fin de manejar datos por medio de variables, estos pueden recibir valores determinados. El tipo de los valores que pueden recibir dependen de la declaración previa de tales variables.

En una asignación se resuelve, primeramente la expresión (al lado derecho del símbolo de asignación) y se asigna el resultado en la variable.

El formato general de asignación es:

Nom_variable Expresión

Donde **Expresión** puede ser una variable o constante, operación, función.

Ejemplo:

$$A \leftarrow 9$$

Significa que la variable A se le ha asignado el valor 9. La acción de asignar es destructiva, ya que el valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor. Así en la secuencia de operaciones:

$A \leftarrow 30$

$A \leftarrow 189$

$A \leftarrow 9$

Cuando se ejecutan, el último valor que toma A será 9, ya que los valores anteriores a este han desaparecido.

Las acciones de asignación se clasifican según sea el tipo de expresiones: **Aritméticas, Lógicas y de Caracteres.**

Asignación Aritmética

Las expresiones en las operaciones de asignación son aritméticas:

Suma $\leftarrow 5+10+2$

Se evalúa la expresión 5+10+2 y se asigna a la variable Suma, es decir, 17 será el valor que toma Suma.

Asignación Lógica

La expresión que se evalúa en la operación de asignación es lógica. Supóngase que M, N, y P son variables de tipo lógico.

$M \leftarrow 8 < 5$

$N \leftarrow M$ o $(7 <= 12)$

$P \leftarrow 7 > 6$

Tras ejecutar las operaciones anteriores, las variables M,N,P toman los valores, **falso, verdadero, verdadero** respectivamente.

Asignación de caracteres

La operación que se evalúa es de tipo carácter.

x ← '3 de Mayo de 1999'

La acción de asignación anterior asigna la cadena de caracteres '3 de Mayo de 1999' a la variable de tipo carácter x.

4. Modularidad

4.1. Descomposición de un problema

Diseño modular.

El programa se divide en módulos (partes independientes), cada una de las cuales ejecuta una única actividad o tarea y se codifican independientemente de otros módulos. Cada uno de estos módulos se analiza, codifican y se refinan por separado. Cada programa tiene un módulo principal que controla a los demás sub módulos (subprogramas).

Diseño descendente.

Es el proceso mediante el cual un problema se descompone de una serie de niveles o pasos sucesivos de refinamiento (Stepwise). Consisten en efectuar una relación entre las sucesivas etapas de estructuración, de modo que se relacionen unas con otras mediante Entradas y Salidas de información. El problema se descompone en etapas o estructuras jerárquicas (lo que hace y como lo hace).

Diseño secuencial.

Es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la Salida de una es la Entrada de la siguiente y así sucesivamente hasta el fin del proceso.

4.2. Concepto de función

Programa principal.

Cada sistema tiene un módulo principal que controla a los demás sub módulos (subprogramas).

Procedimientos.

Cuando se requiere que un subprograma calcule varios resultados en vez de uno solo es necesario utilizar procedimientos (o subrutinas). Es un programa que ejecuta un proceso específico. Ningún valor está asociado con el procedimiento.

Funciones.

Matemáticamente una función es una operación que toma uno o varios valores llamados argumentos y produce un valor denominado resultado (valor de la función para los argumentos dados). Las funciones se mandan llamar utilizando su nombre en una expresión con los argumentos actuales o reales. Las funciones incorporadas al sistema se denominan funciones internas, y las funciones definidas por el usuario se denominan funciones externas.

4.3. Paso de parámetros

Existen diferentes métodos para el paso de parámetros a subprogramas, un mismo programa puede producir diferentes resultados bajo diferentes sistemas de paso de parámetros.

Los parámetros se clasifican de la siguiente manera:

1. **Entradas**: Las entradas proporcionan valores desde el programa que llama y que se utilizan dentro de un procedimiento. En los subprogramas función las entradas son los argumentos en el sentido tradicional.
2. **Salidas**: Las salidas producen los resultados del subprograma; de nuevo si se utiliza el caso una función, mientras que con procedimientos pueden calcularse cero, una o varias salidas.
3. **Entradas/Salidas**: Un solo parámetro se utiliza para mandar argumentos a un programa y para devolver resultados.

Los métodos más empleados para realizar el paso de parámetros son:

- Paso por valor (parámetro valor).
- Paso por referencia o dirección (parámetro variable).

PASO POR VALOR

Se utiliza en muchos lenguajes de programación (pascal, basic, modula-2, algol, etc), debido a su analogía con los argumentos de una función, donde los valores se proporcionan en el orden de cálculo de resultados. Los parámetros se tratan como variables locales y los valores iniciales se proporcionan copiando los valores de los correspondientes argumentos.

Los parámetros formales (locales a la función), reciben como valores iniciales los valores de los parámetros actuales y con ello se ejecutan las acciones descritas en el subprograma.

La llamada por valor no devuelve información al programa que llama.

PASO POR REFERENCIA

Se utiliza cuando se requiere que ciertos parámetros sirvan como parámetros de salida, es decir, devuelvan los resultados a la unidad o programas que llama. La unidad que llama pasa a la unidad llamada la dirección del parámetro actual (que está en el ámbito de la unidad llamante). Una referencia al correspondiente parámetro formal se tratra como una referncia a la posición de memoria, cuya dirección se ha pasado. Entonces una variable pasada como parámetro real es compartida, es decir, se puede modificar directamente por el subprograma. Si el parámetro actual es una expresión, el subprograma recibe la dirección de la posición temporal que contiene el valor de la expresión.

4.4. Estructura de programas

Estructura general de un programa

El programador debe establecer el conjunto de especificaciones que debe contener el programa: Entrada, Salida y Algoritmos de resolución que incluirán las técnicas para obtener las Salidas a partir de las Entradas.
Para saber donde como estructurar las diferentes secciones de un programa es necesario conocer el tipo de programación que maneja el lenguaje a utilizar (Programación Modular, Estructurada, Orientada a Objetos, Tipo de Datos Abstractos, No Estructurada).

Sección de identificación.

Para encontrar la solución óptima de un problema es necesario hacer un análisis para comprender totalmente cada una de las partes que integran al problema.

Sección de datos.

Es todo aquello que nos permite identificar cada componente que forma parte del problema y que nos permite encontrar la solución del mismo.

Sección de procedimientos.

Es el diseño de los pasos que se seguirán para resolver el problema utilizando una secuencia lógica, la cual permitirá la ordenada ejecución de comandos (instrucciones).

5. Estructuras de decisión

La especificación formal de algoritmos tiene realmente utilidad cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición.

Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también ***estructuras de decisión o alternativas.***

En las estructuras selectivas se evalúa una condición y en función del resultado la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (if, then, else o bien en español si, entonces, sino), con una figura geométrica en forma de rombo o bien con un triángulo en el interior de una caja rectangular.

Las estructuras selectivas o alternativas pueden ser:

Simples

Múltiples

5.1. if-then-else

Selectivas simples.

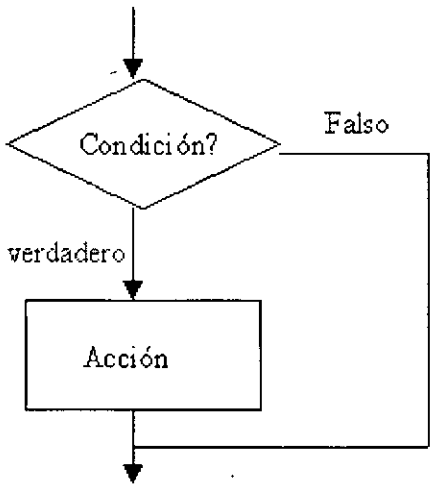
Ejecuta una determinada acción cuando se cumple una determinada condición. La selección if – then (si – entonces) evalúa la condición y si es verdadera ejecuta la acción de lo contrario no hará nada.

Pseudocódigo en español

```
Si   <condición>  Entonces
     <acción S1>
Fin_si
```

Pseudocódigo en inglés

```
If   <condición>  then
     <acción S1>
end_if
```



Selectiva doble.

Permite elegir entre dos opciones posibles en función del cumplimiento o no de una determinada condición. Si la condición es verdadera, se ejecuta la acción 1, si es falsa, se ejecuta la acción 2. La selectiva en pseudo código es if – then – else.

5.2. if anidados

Se tiene una estructura de decisión anidada cuando una estructura if – then – else contiene otra dentro de sí, y ésta a su vez contiene otra dentro de sí. Estas estructuras contendrán varios si - entonces dentro de otros. Debido a que este tipo de estructuras pueden ser confusas, se implementa la identación para evitar perder el hilo de dónde comienza y dónde termina cada estructura.

5.3. Estructuras de tipo CASE

La estructura de selección múltiple (case en pseudocódigo) evaluará una expresión que podrá tomar *n* valores distintos 1, 2, 3, 4, ...n. Según sea el valor en la condición, se realizará una de las *n* acciones.

Pseudocódigo

En inglés la estructura de decisión múltiple se representa

```
Case expresión of
                [e1]: acción S1
                [e2]: acción S2
                :
                [en]: acción Sn

    else
                acción Sx

            end_case
```

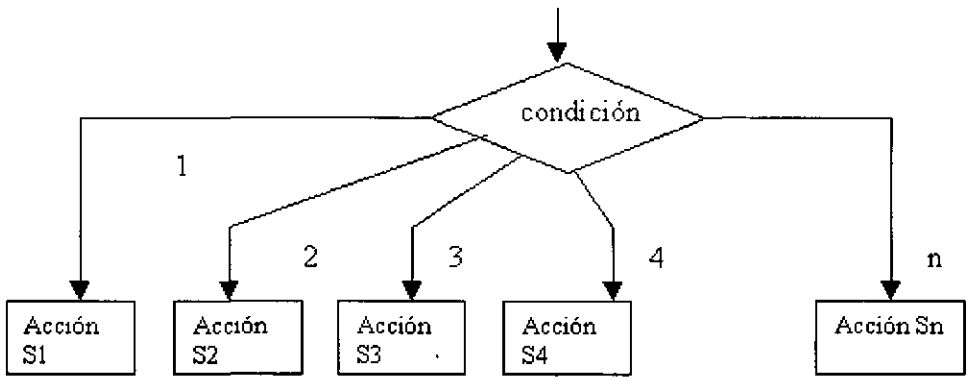
Ejemplo:

Se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable DIA introducida por teclado.

Los días de la semana son 7; por consiguiente, el rango de valores de DIA será 1..7, y caso de que DIA tome un valor fuera de este rango se deberá producir un mensaje de error advirtiendo la situación anómala.

```
Inicio
Leer DIA
Según_sea DIA hacer
                1: escribir('Lunes')
                2: escribir('Martes')
                3: escribir('Miércoles')
                4: escribir('Jueves')
                5: escribir('Viernes')
                6: escribir('Sabado')
                7: escribir('Domingo')
else
                escribir('Error')
                fin_según
```

fin


condición
1
2
3
4
n
Acción S1
Acción S2
Acción S3
Acción S4
Acción Sn

6. Estructuras de repetición

6.1. *Estructura básica de un ciclo*

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan ***Bucles*** y se denomina ***Iteración*** al hecho de repetir la ejecución de una secuencia de acciones. Entre las estructuras repetitivas se encuentran:

- Mientras (while)
- Repetir (repeat/do-while)
- Desde (for)

6.2. *Estructuras desde/para (for)*

Se utilizan las estructuras for cuando se conocen con certeza el número de veces que desea repetir un bucle, es decir, cuando es un número fijo de veces.

Pseudocódigo en Español **Pseudocódigo en Inglés**

Desde variable(v)= vi **Hasta** vf **hacer** **For** variable (v)= vi **To** vf **Do**

<acciones><acciones>

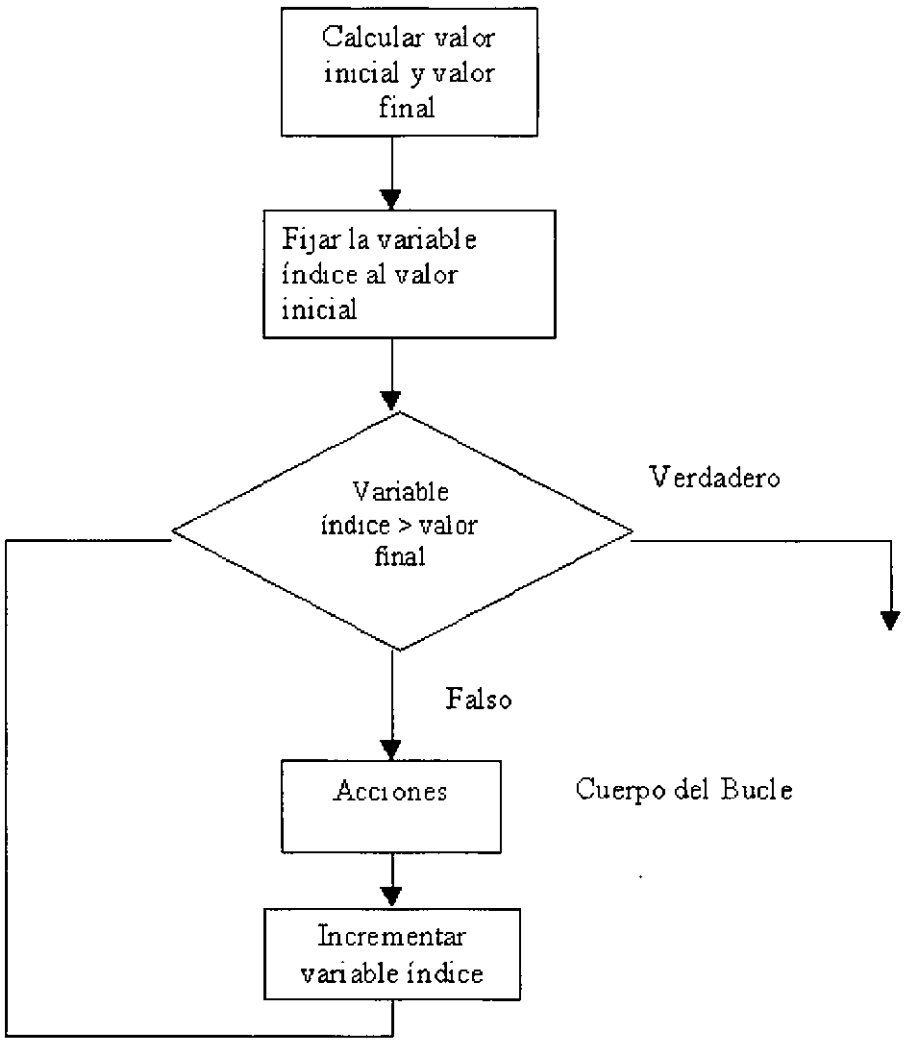
:

Fin_desde

Donde:

v: Variable índice

vi, vf: Valores inicial y final de la variable

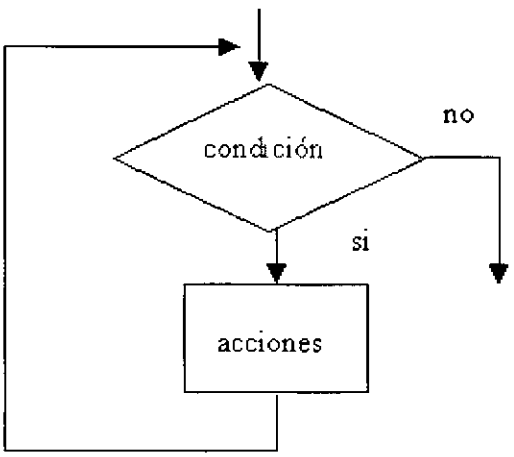


6.3. *while y repeat (do-while)*

Estructura mientras (while).

La estructura repetitiva mientras es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta esta instrucción, la primera cosa que sucede es que se evalúa la condición. Si la expresión es verdadera, entonces se ejecuta el cuerpo del bucle. Este proceso se repite una y otra vez mientras la condición sea verdadera.

Pseudocódigo en español	Pseudocódigo en inglés
Mientras condición **hacer**	**while** condición **do**
Acción S1<Acciones>	
Acción S2:	
:	**End_while**
acción Sn	
Fin_mientras	



Estructura repetir hasta (do/while).

Se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle, esto permite que la iteración se ejecute al menos una vez antes de que la condición sea evaluada.

Pseudocódigo en Español	Pseudocódigo en Inglés
Repetir	**Repeat**
<acciones>	<acciones>
: :	
Hasta que <*condición*>	**Until** <*condición*>

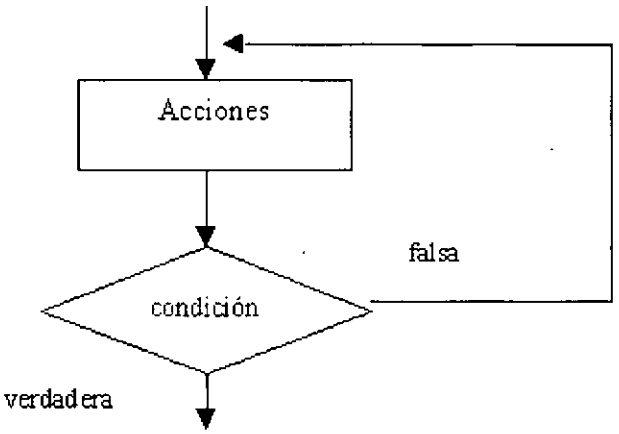
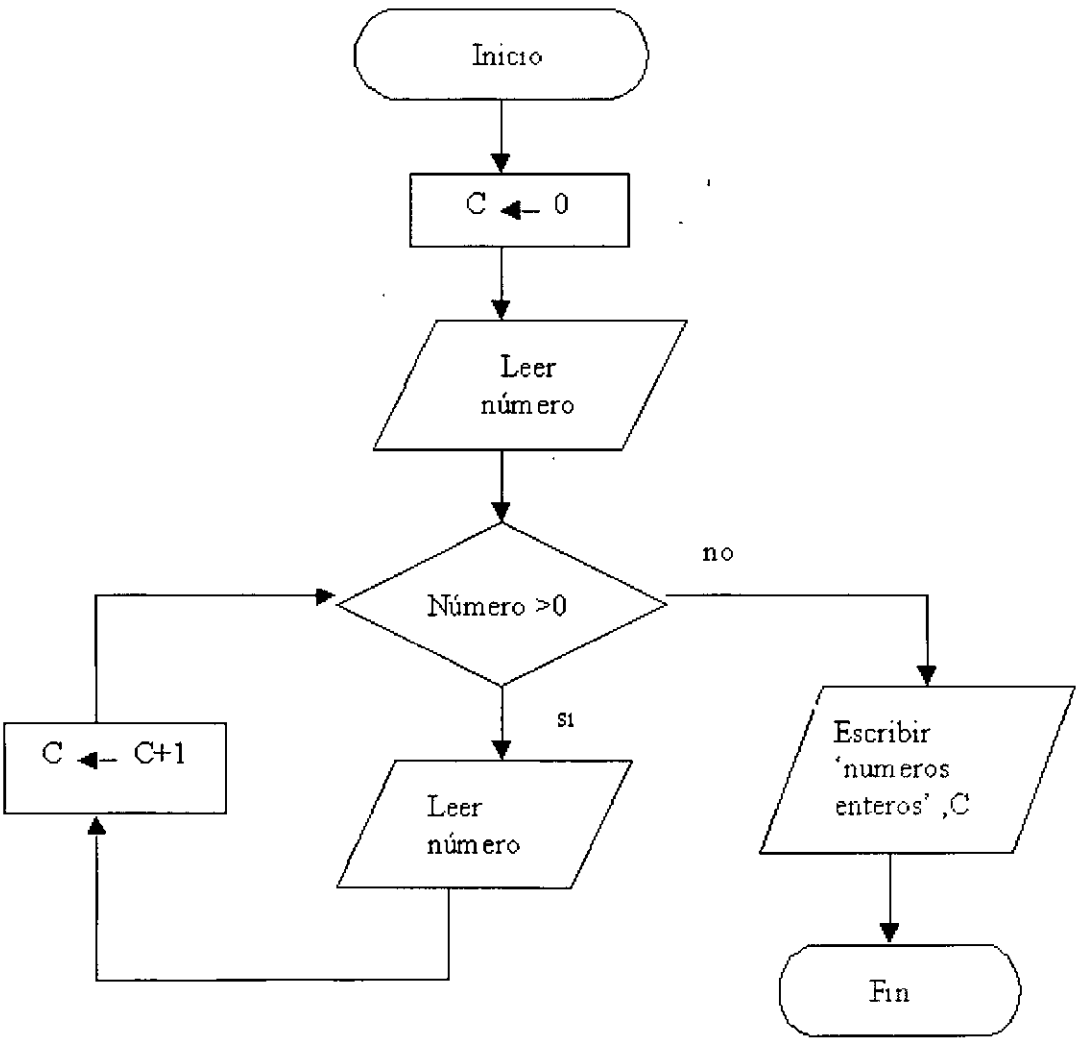


Diagrama de Flujo



7. Datos estructurados

7.1. Arreglos

Las variables que hemos utilizado hasta ahora nos permiten el almacenamiento de un solo valor a la vez.

Para resolver cierto tipo de problemas con datos múltiples en forma eficiente, se requiere almacenamiento en conjunto. A esta organización de elementos se le conoce con el nombre de arreglo.

Otra definición de arreglo más completa (Luis Joyanes A.), es un conjunto finito y ordenado de elementos homogéneos. La propiedad "ordenado" significa que el elemento primero, segundo, tercero...n-ésimo de un arreglo puede ser identificado. Los elementos de una arreglo deberán ser homogéneos, es decir, del mismo tipo de datos. Por ejemplo un arreglo puede estar compuesto de todos sus elementos de tipo cadena, otro puede tener sus elementos de tipo entero, etc.

Al tratar el tema de arreglos es necesario conocer el término de dimensión.

Dimensión	Descripción
0	Un solo punto.
1	(vector o lista) Una recta. Contiene largo.
2	(matriz o tabla) Contiene largo y ancho.
3	(cubo) Tiene largo, ancho y fondo.

Declaración de un Arreglo

Nom_variable : Arreglo [dimensión] de Nom_tipo

En donde dimensión especifica :

<subíndice inferior> ...<subíndice superior>

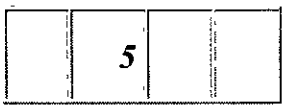
En el orden : Fila, Columna, Fondo.

Ejemplo de una declaración :

***0 Dimensión* :**

x 20 donde x, es de tipo Entero.

1 Dimensión :



1 2 3 4

x[2] 5

x : arreglo [1..4] de Enteros.

***2 Dimensiones* :**

1		3	4
5	6	7	

x[2,4] 8

x : arreglo [1..3, 1..4] de Enteros.

3 Dimensiones :

x[1,4,2] 3

x : arreglo[1..3, 1..4, 1..2] de Enteros.

Las operaciones que se pueden realizar con arreglos durante el proceso de resolución de un problema son :

- Asignación

- Lectura/Escritura
- Recorrido (acceso secuencial)
- Actualizar (añadir, borrar, insertar)
- Ordenación
- Búsqueda

7.2. *Vectores*

Son aquéllos de una sola dimensión, por lo que también son llamados arreglos Unidimensionales.

Ejemplo :

Un vector de una dimensión llamado CALIF, que consta de n elementos.

calif(1)	*calif(2)*	...	*calif(n-2)*	*calif(n-1)*	*calif(n)*

El subíndice o índice de un elemento (1, 2 ...n) designa su posición en la ordenación del vector. Otras posibles notaciones del vector son :

a1, a2,...,an En matemáticas y algunos lenguajes(BASIC)

A(1), A(2),...A(n)

A[1], A[2],...A[n] En programación (Pascal)

Los vectores se almacenan en memoria central de la computadora en un orden adyacente. Así, un vector de 50 elementos denominado NUMEROS se representa gráficamente por 50 posiciones de memoria sucesivas.

Memoria

NUMEROS(1) ← Dirección x

NUMEROS(2) ← Dirección x+1

NUMEROS(3) ← Dirección x+2

:

:

NUMEROS(50) ← Dirección x+49

Cada elemento de un vector se puede procesar como si fuese una variable simple al ocupar una posición de memoria. Así :

NUMEROS(25) 75 (almacena el valor 75 en la posición 25a del vector NUMEROS y la instrucción de salida : escribir NUMEROS(25).

Declaración

nom _arreglo = arreglo[liminf..limsup] de tipo

donde :

nom_arreglo : nombre válido del arreglo.

liminf..limsup : límites inferior y superior del rango del arreglo.

tipo : tipo de datos de los elementos del arreglo : entero, real, carácter.

NOMBRES= arreglo [1..10] de carácter

Significa que NOMBRES es un arreglo (array) unidimensional de 10 elementos (1 a 10) de tipo carácter.

Asignación

NOMBRES(8) 'Ana'

Asigna el valor 'Ana' al elemento 8 del vector NOMBRES

Acceso Secuencial al Vector (recorrido)

Se puede acceder a los elementos de un vector para introducir datos (escribir) en el o bien para visualizar su contenido (leer). Estas operaciones se realizan utilizando estructuras repetitivas, cuyas variables de control (por ejemplo I) se utilizan como subíndices del vector (por ejemplo, X(I). El incremento del contador del bucle producirá el tratamiento sucesivo de los elementos del vector.

Ejemplo :

Lectura de 15 valores enteros de un vector denominado TOTAL.

```
TOTAL= array [1..15] de entero
desde i ← 1 hasta 15 hacer
   leer TOTAL(i)
fin _desde
```

Si se cambian los limite inferior y superior, por ejemplo, 5 y 12, el bucle de lectura sería :

```
desde i ← 5 hasta 12 hacer
   leer TOTAL(i)
fin_desde
```

La salida o escritura de vectores se representa de un modo similar.

```
desde i← 1 hasta 15 hacer
   escribir TOTAL(i)
fin_desde
```

Visualiza todo el vector completo (un elemento en cada línea independiente).

Actualización de un Vector

Puede constar de tres operaciones más elementales:

a) Añadir elementos (añade un nuevo elemento al final del vector)

Un arreglo A se ha dimensionado a 6 elementos, pero solo se han asignado 4 valores a los elementos A(1), A(2), A(3), A(4), se podrán añadir dos elementos más con una simple acción de asignación.

A(5) ← 15

A(6) ← 9

b) Insertar elementos (introduce un elemento en el interior de un vector)

Ejemplo:

Se tiene un arreglo NOM de 6 elementos de nombres de personas, en orden alfabético y se desea insertar un nuevo nombre.

{Calcular la posición ocupada por el elemento a insertar} P

{Inicializar contador de inserciones} i n.

mientras i >= P hacer

{transferir el elemento actual hacia abajo, a la posición i+1}

NOM(i+1) ← NOM(i)

{decrementar contador}

i ← i-1

fin_mientras

{Insertar el elemento en la posición P}

NOM(P) ← 'nuevo elemento'

{Actualizar el contador de elementos del vector}

n ← n+1

fin

c) Borrar elementos (Elimina elementos de un vector)

Algoritmo de Borrado

Inicio

{se utilizará una variable auxiliar AUX, que contendrá el valor del elemento que se desea borrar}

```
AUX       NOM(i)
desde i     j hasta N-1 hacer
{llevar elemento j+1 hacia arriba}
   NOM(i)          NOM(i+1)
fin_desde
{actualizar contador de elementos}
{ahora tendrá un elemento menos, N-1}
N       N-1
Fin
```

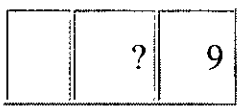
Referencia a un elemento de Arreglo

variable de arreglo [subíndice]

Ejemplo :

xx[3] ← 9

escribir (x[2])



Ejercicio :

Se desea la lectura y desplegado de 5 nombres. Resuelva el problema por cada uno de los siguientes criterios :

a) Lectura y desplegado alternados.

b) Todas las lecturas, todos los desplegados.

a) variables :

string : nom

entero : x

```
Inicio
Para x← 1 hasta 5 hacer
  escribir('Nombre ? ')
  leer(nom)
  escribir( 'La persona número', x, ' se llama : ',nom)
Fin
```

Nota : No se utilizaron arreglos porque no se requería de almacenamiento múltiple.

b) variables :

```
nom : Arreglo[1..5] de string
x : Entero

Inicio
para x      1 hasta 5 hacer
escribir('Dame el nombre número',x,' ?')
leer(nom[x])
para x      1 hasta 5 hacer
  escribir('La persona número ', x,' se llama : ',nom[x])
Fin
```

Ordenación de Arreglos

Existen diversos métodos para ordenar los elementos de un arreglo. El más conocido de ellos (no el mejor) es el Método de la Burbuja.

El método consiste en hacer un recorrido por el arreglo comparando parejas de elementos ; si estos no están en el orden deseado, se procede a intercambiarlos.

Al finalizar el recorrido se verifica la cantidad de intercambios, si esta es 0 se asume que el arreglo está ordenado ; en caso contrario se inicia nuevamente el recorrido.

Las parejas de elementos que se comparan deben ser contiguos (elemento1 y elemento2, elemento2 y elemento3, etc). El número total de comparaciones es n-1 (donde n es la cantidad de elementos).

Ejemplo :

Se requiere la ordenación de una lista con 5 valores enteros previamente introducidos.

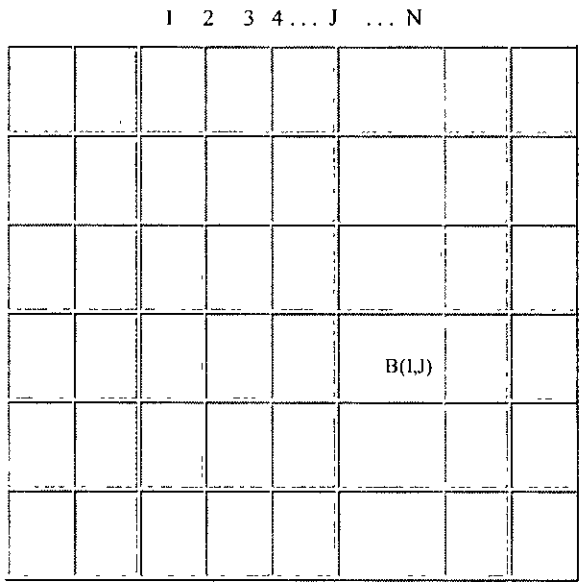
```
Variables :
LISTA : arreglo[1..5] de entero
x, aux : entero
cambio : boleano

Inicio
Para x      1 hasta 5 hacer
escribir('Dame el valor',x,' :')
leer( LISTA[x])
repetir
  cambio falso
  para x      1 hasta 4 hacer
si LISTA[x] > LISTA[x+1] entonces
aux      LISTA[x]
LISTA[x]      LISTA[x+1]
LISTA[x+1]      aux
cambio      verdadero
  fin_si_entonces
hasta cambio = falso
escribir('Lista ordenada')
para x      1 hasta 5 hacer
   escribir('Elemento número',x,' es',LISTA[x])
Fin
```

7.3. Matrices

Se puede considerar como un vector de vectores. Es, por consiguiente, un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo y en el que se necesitan especificar dos subíndices para poder identificar a cada elemento del arreglo.

También se les llama arreglos Bidimensionales, ya que una tabla será utilizada cuando se requiere de establecer relaciones por renglones y columnas entre datos de un tipo común.



Se considera que este arreglo tiene dos dimensiones (una dimensión por cada subíndice) y necesita un valor para cada subíndice, y poder identificar un elemento individual. En notación estándar, normalmente el primer subíndice se refiere a la fila del arreglo, mientras que el segundo subíndice se refiere a la columna del arreglo. Es decir, B(I,J), es el elemento de -b que ocupa la Ia y la Ja columna como se muestra en la figura anterior.

Un ejemplo típico de un arreglo Bidimensional es un tablero de ajedrez. Se puede representar cada posición o casilla del tablero mediante un arreglo, en el que cada elemento es una casilla y en el que su valor será un código representativo de cada figura del juego.

Ejemplo:

Se desea registrar las edades de 4 grupos de personas, cada uno de ellos con 5 elementos. Los datos deberán ser mostrados en forma posterior.

variables:

edad: arreglo(1..4,1..5) de entero

```
g,p: enteros

Inicio
para g      1 hasta 4 hacer
  escribir('Grupo : ',g)
  para p      1 hasta 5 hacer
    escribir('Edad de la persona ',p)
leer(edad[g,p])
          para g      1 hasta 4 hacer
            escribir('Grupo ',g)
            para p      1 hasta 5 hacer
              escribir('persona ',p,' Tiene',edad[g,p],'años')
Fin
```

Ejemplo:

Se tienen 4 fábricas cada una de ellas con 6 empleados. Se desea registrar los salarios y mostrarlos posteriormente. En forma alternada deberá leerse también el nombre de cada empleado. Durante el desplegado de los gastos se indicará el nombre y salario del empleado mejor pagado de cada fábrica así como el total de nómina (de cada fábrica).

```
variables:
Nom: arreglo[1..4,1..6] de string
salario: arreglo[1..4,1..6] de entero
fab, emp, total, cont, lug, zuc: entero


Inicio
  para fab      1 hasta 4 hacer
    para emp      1 hasta 6 hacer
          escribir('Empleado : ',emp)
escribir('Nombre del empleado :')
leer(nom[fab,emp])
escribir('Salarios del empleado : ')
```

```
leer(sal[fab,emp])
        para fab      1 hasta 4 hacer
total     0
cont      sal[fab,1]
escribir('Fábrica', fab)
para emp      1 hasta 6 hacer
escribir('El empleado ',emp,' se llama',nom[fab,emp],' y gana ',salario[fab,emp])
                              total     total + salario[fab,emp]
                              si salario[fab,emp] >= cont entonces
                                   cont      salario[fab,emp]
                                   lug       fab
                                   zuc       emp
    finEntonces
escribir('El mejor pagado es ',nom[lug,zuc],'y gana ',salario[lug,zuc])
                         escribir('El total es : ',total)

    Fin
```

7.4. Cadenas

Definición

Una cadena (string) de caracteres es un conjunto de caracteres (incluido el blanco), que se almacenan en un área contigua de la memoria. Pueden ser entradas o salidas a/desde una computadora.

La longitud de una cadena es el número de caracteres que contiene. La cadena que no contiene ningún carácter se le denomina cadena vacía o nula. y su longitud es cero; no se debe confundir con una cadena compuesta de solo espacios en blanco.

La representación de las cadenas suele ser con comillas simples o dobles, las comillas actúan como separadores, algunos ejemplos:

'12 de Octubre de 1492'

'Hola, como estás?'

‘ ’

‘MEXICO ES GRANDE’

La cadena puede contener entre sus separadores, cualquier carácter válido del código aceptado por el lenguaje y la computadora; el blanco es uno de los caracteres más utilizado.

Una **Subcadena** es una cadena de caracteres que ha sido extraída de otra de mayor longitud.

’12 de’	**es una subcadena de ’12 de Octubre de 1492’**
‘como estás’	**es una subcadena de ‘Hola, como estás?’**
‘XI’	**es una subcadena de ‘MEXICO ES GRANDE’**

Datos Tipo Carácter

Anteriormente se analizaron los diferentes tipos de datos y entre ellos existían el dato tipo carácter (char) que se incorpora en diferentes lenguajes de programación, bien con este nombre, o bien como datos tipo cadena.

Constantes

Una constante tipo carácter es un conjunto de caracteres válidos encerrados entre comillas, para evitar confundirlos con nombres de variables, operadores, enteros, etc

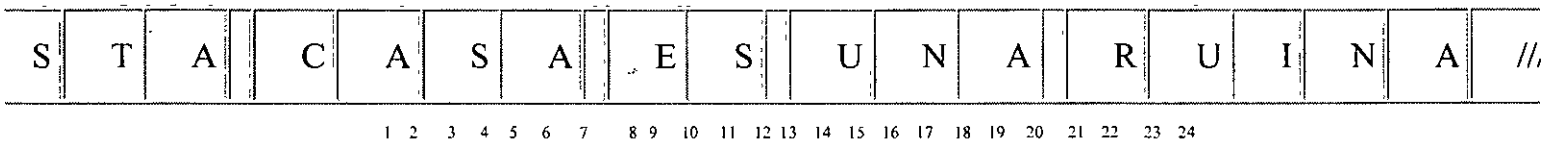
Variables

Una variable de cadena o tipo carácter es una variable cuyo valor es una cadena de caracteres. Las variables de tipo carácter se deben declarar en el algoritmo y según el lenguaje tendrán una notación u otra.

var NOMBRE, DIRECCION, PAIS: carácter

Cadenas de Longitud Fija

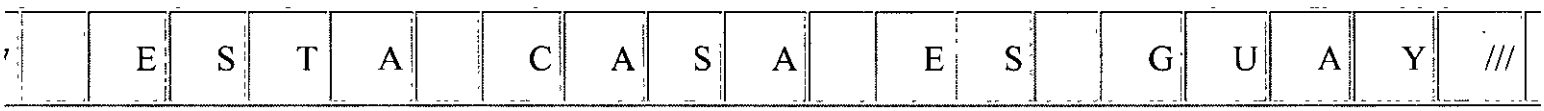
Se consideran vectores de la longitud declarada, con blancos a izquierda o derecha si la cadena no tiene la longitud declarada. Así el ejemplo siguiente:



Se declaró con una dimensión de 24 caracteres y los dos último se rellenan con blancos.

Cadenas de Longitud Variable con un Máximo

Se considera un puntero, con dos campos que contienen la longitud máxima y la longitud actual.



Donde:

20 = Longitud máxima

17 = Longitud actual

Instrucciones Básicas con Cadenas

Las instrucciones básicas : asignar y entrada/salida (leer/escribir) se realizan de un modo similar al tratamiento de dichas instrucciones con datos numéricos.

Asignación

Si la variable NOMBRE ha sido declarada como carácter

var NOMBRE : carácter

La instrucción de asignación debe contener en el lado derecho de la asignación una constante tipo carácter (una cadena) o bien otra variable tipo carácter. Así:

NOMBRE □ 'Juan González'

Significa que la variable NOMBRE toma por valor la cadena 'Juan González'

Función de cadenas de caracteres

El tratamiento de cadenas es un tema importante, debido esencialmente a la gran cantidad de información que se almacena en ellas. Según el tipo de lenguaje de programación elegido se tendrá mayor o menor facilidad para la realización de operaciones.

Las operaciones con cadenas más usuales son:

- Cálculo de la longitud
- Comparación
- Concatenación
- Extracción de subcadenas
- Búsqueda de información

1.- CALCULO DE LA LONGITUD DE UNA CADENA

La longitud de una cadena, como ya se ha comentado, es el número de caracteres de la cadena. Así:

'Hola' Tiene 4 caracteres

La operación de determinación de la longitud de una cadena se representará por la función Longitud, cuyo formato es:

Longitud (cadena)

La función longitud tiene como argumento una cadena, pero su resultado es un valor numérico entero.

Ejemplo:

longitud('Como estás?')	Proporciona 11.
longitud(' ')	Cadena de un blanco, proporciona 1.
longitud(' Hola')	Cadena 'Hola' relleanda de un blanco a la izquierda para tener longitud de 5.

2.- COMPARACION

Es una operación muy importante sobre todo en la clasificación de datos tipo carácter, que se utiliza con mucha frecuencia en aplicaciones de proceso de datos (clasificaciones de listas, tratamiento de textos, etc).

Los criterios de comparación se basan en el orden numérico del código o juego de caracteres que admite la computadora o el propio lenguaje de programación. En nuestro lenguaje algorítmico utilizaremos el código ASCII como código numérico de referencia.

En la comparación de cadenas se pueden considerar dos operaciones más elementales: igualdad y desigualdad.

Ejemplos:

'LUIS' < 'LUISITO'	verdadera
'ANA' < 'MARTA'	verdadera
'BARTOLO' >= 'BARTOLOME'	falsa

3.- CONCATENACION

La concatenación es la operación de reunir varias cadenas de caracteres en una sola, pero conservando el orden de los caracteres en cada una de ellas. El símbolo que representa la concatenación varía de unos lenguajes a otros. Los más utilizados son:

\+ // & o

Las cadenas para concatenarse pueden ser constantes o variables.

Ejemplos:

'Mayo ' + 'de 1999' = 'Mayo de 1999'

x 'Lenguaje'

y 'Ensamblador'

z x+' '+y z= 'Lenguaje Ensamblador'

4.- SUBCADENAS

Esta operación permite la extracción de una parte específica de una cadena: subcadena.

Subcadena (cadena, inicio, longitud)

Cadena: Es la cadena de la que debe extraerse una subcadena.

Inicio: Es un número o expresión numérica entera que corresponde a la posición inicial de la subcadena.

Longitud: Es la longitud de la subcadena.

Ejemplos:

subcadena ('abcdefgh',3,6)	Equivale a	'cdef'
subcadena ('Tecnológico',1,3)	Equivale a	'Tec'
subcadena ('México',5,2)	Equivale a	'co'

5.- BUSQUEDA

Esta operación localiza si una determinada cadena forma parte de otra cadena más grande o buscar la posición en que aparece un determinado carácter o secuencia de caracteres en un texto.

El resultado de la función es un valor entero

Posición (cadena, subcadena)

subcadena: Es el texto que se trata de localizar.

Ejemplo:

nombre 'Marielena'

Posición (nombre,'elena') Produce 5.

La función Posición, al tomar también un valor numérico entero se puede utilizar en expresiones aritméticas o en instrucciones de asignación a variables numéricas.

Otras funciones de cadenas

a) Insertar Cadenas

Si se desea insertar una cadena C dentro de un texto o cadena más grande, se debe indicar la posición. El formato es:

Insertar (t, p, s)

t Texto o cadena donde se va a insertar.

p Posición a partir dc la cual se va a insertar.

s Subcadena que se va a insertar.

Ejemplo:

Insertar ('Tecnológico',4,'XXX') = 'TecXXXnológico'

Insertar ('Juan O',5,'de la')= 'Juan de la O'

b) Borrar

Elimina una subcadena que comienza en la posición p y tiene una longitud l.

Borrar (t, p ,l)

t Texto o cadena de donde se va a eliminar una subcadena.

p Posición a partir del cual se va a insertar.

l Longitud de la subcadena a eliminar.

Ejemplo:

Borrar ('Tecnológico',4,2) = 'Teclógico'

c) Conversión Cadenas/Números

Existen funciones o procedimientos en los lenguajes de programación (val y str) que permiten convertir un número en una cadena y viceversa. En nuestros algoritmos los denotaremos por valor y cadena.

Valor (cadena) = Convierte la cadena en un número; siempre que la cadena fuese de dígitos numéricos.

Cadena (valor) = Convierte un valor numérico en una cadena.

Ejemplos:

Valor ('12345') = 12345

Cadena(12345)= '12345'

Funciones internas

Las operaciones que se requieren en los programas exigen en numerosas ocasiones, además de las operaciones aritméticas básicas, ya tratadas, un número determinado de operaciones especiales que se denominan funciones internas, incorporadas o estándar. Por ejemplo la función ln se puede utilizar para determinar logaritmo de un n se puede utilizar para determinar logaritmo de un número y la función sqrt calcula la raíz cuadrada de un número positivo. Existen otras funciones que se utilizan para determinar las funciones trigonométricas.

La siguiente tabla muestra las funciones internas más usuales, siendo x el argumento de la función.

Función	Descripción	Tipo_Argumento	Resultado
abs(x)	Valor absoluto de x	Entero o real	igual que argumento
arctan(x)	Arcotangente de x	Entero o real	real
cos(x)	Coseno de x	Entero o real	real
exp(x)	Exponencial de x	Entero o real	real
ln(x)	Logaritmo de x	Entero o real	real
log10(x)	Logaritmo decimal de x	Entero o real	real

round(x)	Redondeo de x	Real	entero
sin(x)	Seno de x	Entero o real	real
sqr(x)	Cuadrado de x	Entero o real	igual que argumento
sqrt(x)	Raíz cuadrada de x	Entero o real	real
trunc(x)	Truncamiento de x	real	entero

Manipulación de cadenas

Ejercicio 1:

Contar el número de letras 'i' de una frase terminada en un punto. Se supone que las letras pueden leerse independientemente.

```
Algoritmo letras_i
var
N: entera
Letra: carácter
        Inicio
        N      0
        repetir
           leer(Letra)
           si Letra = 'i'  entonces
              N      N+1
           fin si
        hasta que Letra = '.'
```

```
    escribir('La frase tiene :', N,' letras i')
  Fin
```

Ejercicio 2:

Aceptar un nombre y proporcionar la cantidad de caracteres que contiene.

```
Inicio
  escribir('Cuál es el nombre : ')
  Leer(nom)
  escribir('Tu nombre tiene ', Longitud(nom),' letras')
Fin
```

Ejercicio 3:

Pedir una cadena que contenga una vez la letra F ('F'). Deberá informarse la posición donde fue encontrada, el carácter previo y el siguiente.

```
Inicio
  escribir('Dame cadena que contenga [F] ')
  leer(cad)
  x     Posición('F', cad)
  escribir('La letra 'F' se encuentra en la posición : ',x)
  escribir('El carácter anterior es : ' , Subcadena(cad, x-1,1))
  escribir('El carácter siguiente es : ',  Subcadena(cad, x+1, 1))
Fin
```

Ejercicio 4:

Leer dos caracteres y deducir si están en orden alfabético.

```
variables:
    CAR1, CAR2 : carácter.
Algoritmo Comparación
Inicio
  Leer CAR1, CAR2
```

```
Si CAR1 <= CAR2 entonces
   escribir('Si se encuentran en orden')
sino
   escribir('Desordenados')
Fin_si
Fin
```

- **Ejercicio 5:**

Se desea eliminar los blancos de una frase dada terminada en un punto. Se supone que es posible leer los caracteres de la frase de uno a uno.

Análisis del problema

Para poder efecturar la lectura de la frase almacena ésta en un arreglo de caracteres (F), de modo que F(1) contiene el primer carácter i-ésimo del a frase dada. Construiremos una frase nueva sin blancos en otro arreglo G.

Algoritmo

Los pasos a dar para la realización del algoritmo son:

Inicializar contador de letras de la nueva frase G.

Leer el primer carácter.

Repetir

Si el primer carácter no es en blanco, entonces escribir en el lugar siguiente del arreglo F leer carácter siguiente de la frase dada.

Hasta que el último carácter se encuentre.

Escribir la nueva frase (G), ya sin blancos.

Variables:

F : array de caracteres de la frase dada.

G : array de caracteres de la nueva frase dada.

i : Constador del array F.

j : Constador del array G.

Algoritmo Blanco

Inicio

```
  i     1
  j     0
  leer F(i)
  Repetir
    Si F(i) <> ' ' entonces
      j     j+1
      G(j)      F(i)
    Fin si
    i     i+1
    leer F(i)
  Hasta que F(i) = '.'
  {escritura de la nueva frase G}
  Desde i     1 a j hacer
      escribir G(i)
  Fin Desde
Fin
```

- **Ejercicio 6:**

Encontrar un espacio en blanco en una cadena.

```
variables:
      x: entero
      nom,a,b : cadena
        Inicio
leer(nom)
x     Posición (' ',nom)
a     Subcadena (nom,1,x-1)
b     Subcadena (nom, x+1, x)
        Fin
```

- **Ejercicio 7:**

Solicitar un nombre y mostrarlo con la inicial en mayúscula.

```
variables:
      nom,a,b: cadena
      x,y     : carácter
```

```
    z       : entero
Inicio
  escribir('Introduce nombre : ')
  leer(nom)
  x     Subcadena(nom,1,1)
  y     MAYUSCULA(x)
  z     Longitud (nom)
  a     Subcadena (nom, 2, z-1)
  Concatenación     y+a
  Escribir('Te llamas : ' , concatenación)
Fin
```

Nota: MAYUSCULA es una función para convertir letras minúsculas a mayúsculas.