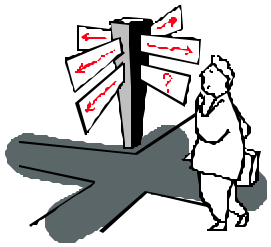
# II

# ALGORITMOS



La primera fase en la construcción de programas la determina el algoritmo a utilizar, que nos indica una serie de pasos ordenados y lógicos para resolver un problema dado.

## II.1. REQUISITOS, DISEÑO, PROGRAMACIÓN Y PRUEBAS.

Pueden ser identificadas dos etapas en el proceso de resolución de problemas :

1. Fase de solución
2. Fase de implementación (realización) en algún lenguaje de programación.

La fase de solución incluye, a su vez el análisis del problema, el diseño y la verificación del algoritmo.

El primer paso es el análisis del problema, aquí se debe examinar cuidadosamente el problema a fin de obtener una idea clara de lo que se quiere hacer y determinar cuales son los datos que se necesitan para solucionar el mismo.

Este primer paso se conoce como "Análisis de Requerimientos", en este proceso la persona que plantea el problema (cliente) expone sus necesidades a quien realizará el programa (programador), esto se lleva a cabo por medio de diferentes técnicas, tales como: entrevistas, cuestionarios y observación.

Una vez que se tienen los requerimientos, se pasa al proceso de "Diseño", el cual es una actividad creativa. Es la forma mediante la que se pueden traducir con precisión los requerimientos del cliente a un producto o programa terminado.

En este proceso se obtiene el "Algoritmo", el cual puede ser definido como la secuencia ordenada de pasos, no ambiguos, que conducen a la solución del problema planteado.

Todo algoritmo debe ser :

**Preciso**. Indicando el orden de realización de cada uno de los pasos.
**Definido**. Si se sigue el algoritmo varias veces proporcionándole los mismos datos, se deben obtener siempre los mismos resultados.
**Finito.** Al seguir el algoritmo, éste debe terminar en algún momento, es decir, tener un número finito de pasos.

Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes y disponerlas en el orden en que han de llevarse a cabo. Los pasos en esta primera descripción de actividades deberán ser refinados, añadiendo más detalles a los mismos e incluso, algunos de ellos pueden requerir un refinamiento adicional antes de que se pueda obtener un algoritmo claro, preciso y completo. Este método de diseño de los algoritmos en etapas, donde se va de los conceptos generales a los detalles a través de refinamientos sucesivos, se conoce como método descendente (**Top-down**).

Existe otro método, que no es recomendable y al contrario del Top-down, consiste en ir de lo particular hacia lo general. Este método se conoce como **Bottom-up**.

Una vez que se tiene el algoritmo concluido, se pasa a la Fase de Implementación, en ésta, se lleva a cabo la "Codificación" del mismo (traducción del algoritmo a algún lenguaje de programación), la ejecución y comprobación del programa.

El paso de comprobación es muy importante, en este, se ejecuta el programa varias veces, con distintos datos, para verificar que se obtengan los resultados que se esperaban.

## II.2. TÉCNICAS DE PROGRAMACIÓN ALGORITMICA PROGRAMACIÓN ESTRUCTURADA

La programación estructurada es un conjunto de técnicas para desarrollar algoritmos fáciles de escribir, verificar, leer y modificar. Utiliza:

- Diseño descendente. Consiste en diseñar los algoritmos en etapas, partiendo de los conceptos generales hacia los detalles. El diseño descendente se verá completado y ampliado con el modular.

- Recursos abstractos. En cada descomposición de una acción compleja se supone que todas las partes resultantes están ya resueltas, posponiendo su realización para el siguiente refinamiento.

- Estructuras básicas. Los algoritmos deberán ser escritos utilizando únicamente tres tipos de estructuras básicas: secuenciales, decisión e iteración, las cuales se describen más adelante.

### TEOREMA DE BÖHM Y JACOPINI

Para que la programación sea estructurada, los programas han de ser ***propios***. Un programa se define como propio si cumple las siguientes características:

- Tiene un solo punto de entrada y uno de salida
- Toda acción del algoritmo es accesible, es decir, existe al menos un camino que va desde el inicio hasta el fin del algoritmo, se puede seguir y pasa a través de dicha acción.
- No posee lazos o bucles infinitos.

El teorema de Böhm y Jacopini dice que: <<un programa ***propio*** puede ser escrito utilizando únicamente tres tipos de estructuras: secuencial, selectiva y repetitiva >>. De este teorema se deduce que se han de diseñar los algoritmos empleando exclusivamente dichas estructuras, las cuales, como tiene un único punto de entrada y único punto de salida, harán que nuestros programas sean propios.

## II.3. ELEMENTOS BÁSICOS

Un algoritmo puede ser escrito en lenguaje natural, pero esta descripción puede ser ambigua, por lo que se utilizan diferentes métodos de representación, que permiten evitar dicha ambigüedad y permitir al mismo tiempo que sea fácilmente codificable. Los métodos más usuales para la representación de algoritmos son:

- Descripción narrada
- Diagrama de flujo
- Pseudocódigo

### DESCRIPCIÓN NARRADA

Es la forma más sencilla de describir o expresar un algoritmo. Consiste en dar un relato de la solución en lenguaje natural.

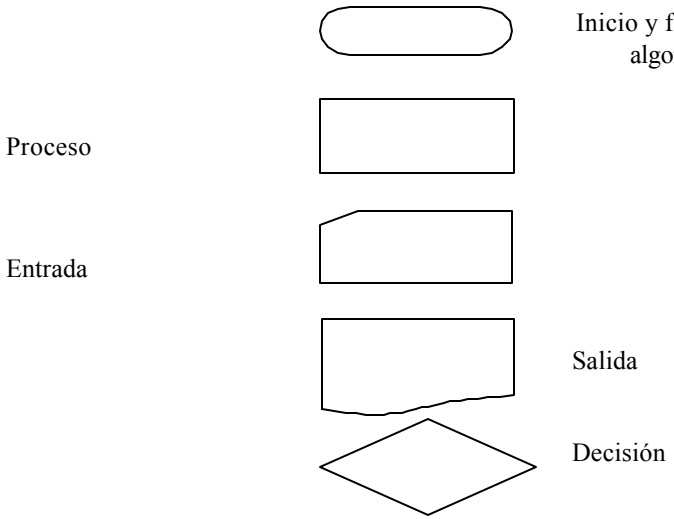
Por ejemplo: Algoritmo en descripción narrada para la suma de 2 números.

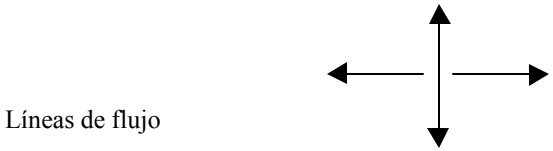
1. obtener los números a sumar
2. sumar los números
3. anotar el resultado

El uso del lenguaje natural provoca frecuentemente que la descripción sea imprecisa y poco confiable, por lo que este tipo de representación no es recomendable.

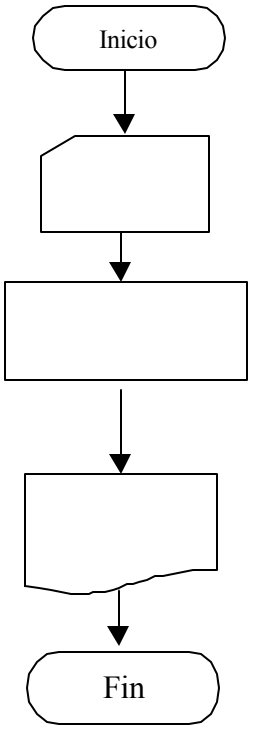
**DIAGRAMA DE FLUJO**

Es la representación gráfica de un algoritmo. Utiliza símbolos normalizados, con los pasos del algoritmo escritos en el símbolo adecuado y los símbolos unidos por flechas, denominadas "líneas de flujo", que indican el orden en que los pasos deben ser ejecutados. Los símbolos principales son:




De manera general un diagrama de flujo esta constituido de la siguiente manera: inicia, recibe datos, realiza el procesamiento, muestra resultados y finaliza.



**PSEUDOCÓDIGO**

El pseudocódigo en un lenguaje de especificación de algoritmos que utiliza palabras reservadas y exige la *indentación*, o sea, sangría en el margen izquierdo, de algunas líneas. Se concibió para superar las dos principales desventajas de los diagramas de flujo: Lento de crear y difícil de modificar sin un nuevo redibujo. Es una herramienta muy efectiva para el seguimiento de la lógica de un algoritmo y para transformar con facilidad los algoritmos a programas.

Las distintas estructuras de control se representan de la siguiente forma:

**Secuenciales**: **Leer** (Lista_de_variables)

**Escribir** (Lista_de_resultados)

**Decisión**

**simple:**
```
si <condición > entonces
        acción_si
fin_si
```

**doble:**
```
si <condición> entonces
        acción_si
    si_no
        acción_no
fin_si
```

**múltiple:**
```
según <expresión> sea
    <valor1> : acción1
    <valor2>: acción2
    .....
    <valorn>: acción n
    [si_no
        acción_sino]
fin_según
```

**Repetitivas:**

```
mientras <condición>hacer
        acciones
fin_mientras

repetir
        acciones
hasta   <condición>

para <Variable de control> ← <valor_inicial> hasta
                                <valor_final>  hacer
                                    acciones
fin_para
```

## II.4. DATOS Y EXPRESIONES

Dato es la expresión general que describe a los elementos con los cuales opera un programa. Los datos pueden estar expresados como **Variables** o como **Constantes**.

**CONSTANTES**

Las constantes son elementos cuyo valor no cambia durante todo el desarrollo del algoritmo. Pueden ser ***literales*** o ***simbólicas***. Las constantes simbólicas tienen un valor asignado y se identifican por un nombre. Una constante literal es un valor que se utiliza como tal.

Ejemplo de constantes simbólicas:

PHI (tiene el valor asignado 3.1416)

Ejemplos de constante literal:

2.0
0
etc.

Las constantes pueden ser:

- **Numéricas enteras:** En el rango de los números enteros positivos o negativos.
  Ejemplos: 0, 2, -3, -8, etc.
- **Numéricas reales:** En el rango de los números reales positivos o negativos.

  Ejemplos: 3.1416, 0.5, -4.3, etc.

- **Lógicas:** con valores **True** o **False** únicamente.

- **Carácter:** Alfabético (´a´, ´b´, ..., ´z´) en mayúscula o minúscula, Numérico (´0´,´1´,...,´9´) o Carácter especial (´+´,´?´, etc).

- **Cadena:** Sucesión de caracteres encerrados entre apostrofes.

  Ejemplo: 'Cadena de ejemplo'

**VARIABLES**

Las variables son elementos cuyo valor puede cambiar durante el desarrollo del algoritmo. Se identifican por un ***nombre*** y un ***tipo.*** El ***tipo*** determina el conjunto de valores que la variable puede tomar.

Ejemplos:

| **Variable** | **Valor** | **Tipo** |
|---|---|---|
| A | 2 | Entero |
| B | 5 | Entero |
| Radio | 2.5 | Real |
| Carac1 | 'a' | carácter |
| Carac2 | 'c' | carácter |

Las variables pueden ser de tipo: entero, real, carácter, lógico o cadena.

Las constantes y variables se utilizan para la formar **expresiones**.

**EXPRESIONES**

Una **expresión** es una combinación de operadores y operandos. Los operandos pueden ser constantes, variables u otras expresiones. Los operadores pueden aritméticos, lógicos o relacionales.

Los operadores aritméticos son :

| **Operador** | **Significado** | **Prioridad** |
|---|---|---|
| - | operador unario menos | 3 |
| * | multiplicación | 2 |
| / | división | 2 |
| + | suma | 1 |
| - | resta | 1 |

La evaluación de las expresiones se realiza de izquierda a derecha cuidando la prioridad de los operadores, los de prioridad mayor se evalúan primero. La evaluación de los operadores con la misma prioridad se realiza siempre de izquierda a derecha. Si una expresión contiene subexpresiones encerrada entre paréntesis, dichas expresiones se evalúan primero.

Ejemplo:

Si A=2, B=3 y C=4,

A+B*C=14 ≠ (A+B)*C=20

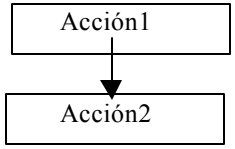
Lo referente a los operadores lógicos y relacionales será tratado más adelante.

## II.5. ESTRUCTURAS SECUENCIALES

Se caracterizan porque una acción se ejecuta detrás de la otra. El flujo del programa coincide con el orden físico en el que se han ido poniendo las instrucciones. Es decir, es una secuencia de acciones, donde se ejecuta primero la acción uno, después la dos, luego la tres, etc. Dichas acciones pueden consistir en acciones simples tales como:

- como leer datos
- realizar operaciones
- escribir resultados

Estas estructuras se representan de la siguiente forma:

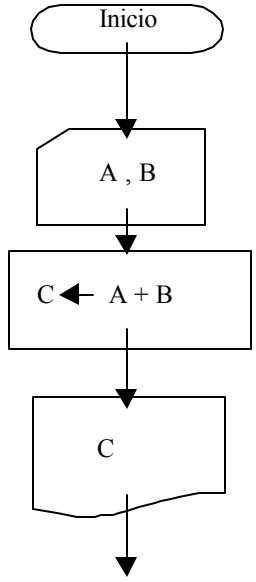
| Diagrama de Flujo | Pseudocódigo |
|---|---|
|  | Acción1<br>Acción2 |



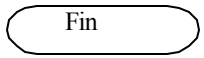
**Funcionamiento**

1. Se lleva a cabo la Acción 1
2. Se lleva a cabo la Acción 2

**Ejemplo:** Para sumar 2 números.

| Diagrama de Flujo | Pseudocódigo |
|---|---|
|  | **Inicio**<br>**Leer** (A,B)<br>C ← A+B<br>**Escribir** (C)<br>**Fin** |







Se leen los datos a sumar A y B, se suman y el resultado se coloca en C **(note el uso del símbolo ← que significa *asignación* )**. Finalmente se muestra C.

**Ejemplo:** Para calcular e imprimir el sueldo de un empleado.

| Diagrama de Flujo | Pseudocódigo |
|---|---|

**Inicio**
**Leer** (*nombre,horas,cuota*)
*sueldo* ← horas x cuota
**Escribir** (*nombre,sueldo*)
**Fin**

Se lee el *nombre* del trabajador, las *horas* laboradas y la *cuota* por hora, se obtiene el *sueldo* y finalmente se muestra el *nombre* y el *sueldo* calculado. Es importante observar que el *nombre* solo es empleado para mostrarse junto con el resultado y no ha sido empleado para el cálculo lo cual es válido.

## II.6. ESTRUCTURAS SELECTIVAS

Permiten controlar la ejecución de acciones que requieran ciertas condiciones para su realización, es decir, se ejecutan unas acciones u otras según se cumpla o no una determinada condición.

Estas estructuras son utilizadas cuando:

- Se tienen acciones que son "excluyentes", es decir, que sólo tiene que ejecutarse una o la otra, pero no ambas.

- Cuando es necesario elegir la acción a realizar de entre un conjunto de acciones existentes.
- Cuando es necesario verificar que lo datos sean válidos para la aplicación en cuestión, por ejemplo: no es posible dividir entre 0, la cuota por hora que se le paga a un trabajador no puede ser negativa, etc.

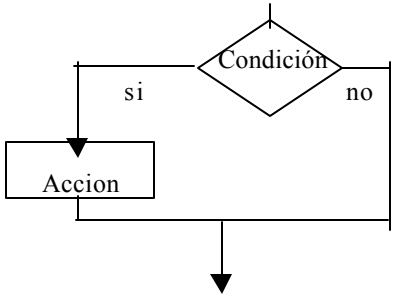
Las estructuras selectivas pueden ser: simples, dobles o múltiples.

**SIMPLES.**

Se evalúa la condición y si ésta da como resultado verdad se ejecuta una determinada acción o grupo de acciones; en caso contrario no se ejecuta dicho grupo de acciones y se continua con el flujo.

Esta estructura se representa de la siguiente forma:

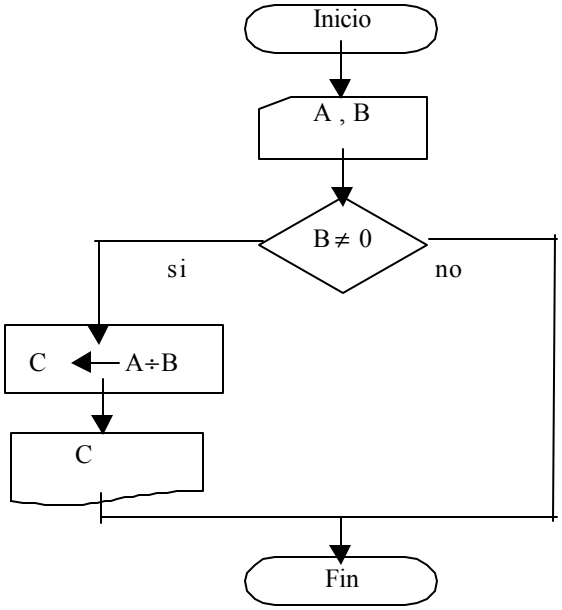
| Diagrama de flujo | Pseudocódigo |
|---|---|




**Si** <condición> **entonces**
acción
**Fin_si**

**Ejemplo:** Para dividir 2 números considerando que el divisor no puede ser 0.

| Diagrama de Flujo | Pseudocódigo |
|---|---|




**Inicio**
**Leer** (A,B)
**Si** B ≠ 0 entonces
C ← A÷B
Escribir (C)
**Fin_si**
**Fin**

Se leen los datos a emplear A y B, y como se va a dividir A ÷B, se "válida" el valor de B a través de la condición B≠0, si la condición se cumple (si la evaluación de la condición da verdad) entonces se realiza la división y se muestra el resultado, en otro caso se va al final sin efectuar acción alguna.

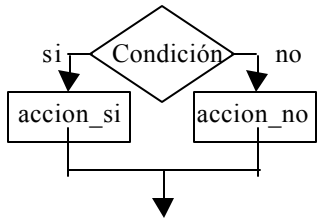
**DOBLES.**

Cuando el resultado de evaluar la condición es verdad se ejecutará una determinada acción o grupo de acciones y si el resultado es falso se

ejecutará otra acción o grupode acciones diferentes. En ambos casos las sentencias podrán ser simples o compuestas.

Esta estructura se representa de la siguiente forma:

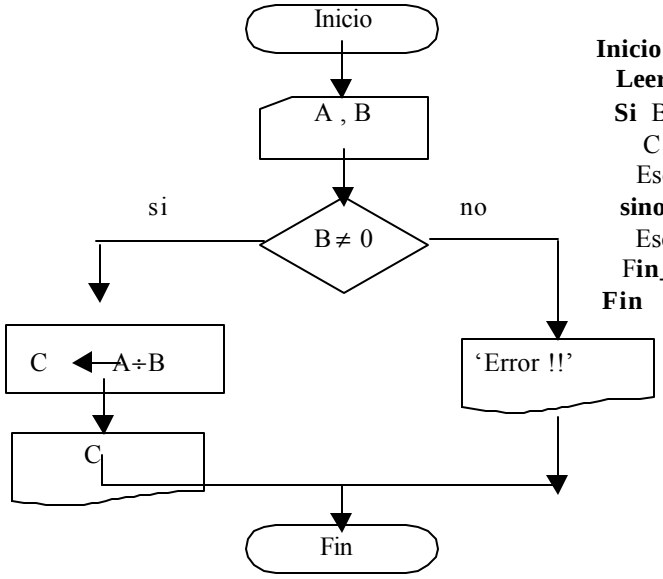
| Diagrama de flujo | Pseudocódigo |
|---|---|




**Si** <condición > **entonces**
acción_si
**sino**
acción_no
**Fin_si**

**Ejemplo 1:** Para dividir 2 números considerando que el divisor no puede ser 0.

| Diagrama de Flujo | Pseudocódigo |
|---|---|

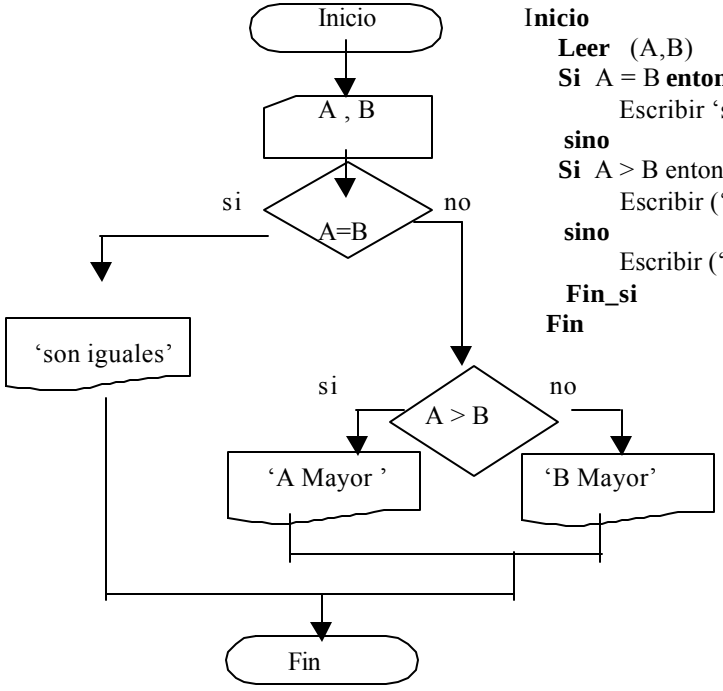



**Inicio**
**Leer** (A,B)
**Si** B ≠ 0 entonces
C ← A÷B
Escribir (C)
**sino**
Escribir ('Error !!')
**Fin_si**
**Fin**

Se leen los datos a emplear A y B y como se va a dividir A ÷B, se "válida" el valor de B a través de la condición B≠0, si la condición se cumple (si la evaluación de la condición da verdad) entonces se realiza la división y se muestra el resultado, en otro caso se escribe un mensaje de Error (note el uso de '' para colocar el mensaje a escribir) y se va al final sin efectuar acción alguna.

**Ejemplo 2:** Para determinar el mayor de dos números, considerando el hecho de que sean iguales.

| Diagrama de Flujo | Pseudocódigo |
|---|---|




**Inicio**
**Leer** (A,B)
**Si** A = B **entonces**
Escribir 'son iguales'
**sino**
**Si** A > B entonces
Escribir ('A Mayor')
**sino**
Escribir ('B Mayor')
**Fin_si**
**Fin**

Se leen A y B para verificar si son iguales, si es asi, entonces se escribe el mensaje 'Son iguales' y se va al final, si no son iguales, entonces se verifica si A > B, si es asi entonces se escribe 'A Mayor', si no es asi, entonces B es mayor, se escribe el mensaje correspondiente y se va al final. Es importante resaltar en este ejemplo la existencia de estructuras selectivas anidadas, introduciendo unas dentro de otras, en este caso introduciendo otra estructura selectiva dentro de la rama del **no** de la primera estructura selectiva (A=B).

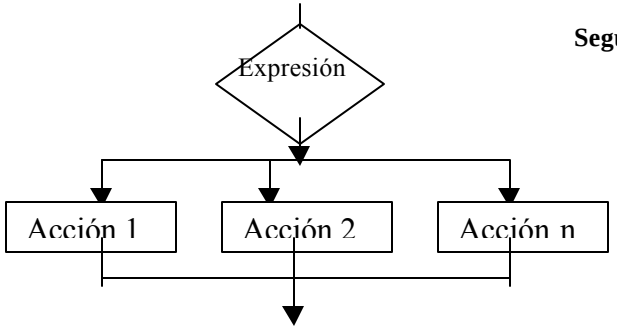
**MÚLTIPLES.**

Las estructuras selectivas múltiples permiten controlar la ejecución de acciones cuando se tienen más de dos opciones alternativas de selección. Aquí se ejecutarán unas acciones u otras según el resultado que se obtenga al evaluar una expresión.

Cada grupo de acciones se encuentra ligado con un valor o una serie de valores expresados mediante: una constante, varias constantes separadas por comas, un rango expresado como valor_inicial..valor_final, o una mezcla de constantes y rangos.

Cuando el valor obtenido al evaluar la expresión no está presente en ninguna lista de valores se ejecutaran las acciones establecidas en la cláusula **sino** (si existiese dicha cláusula).

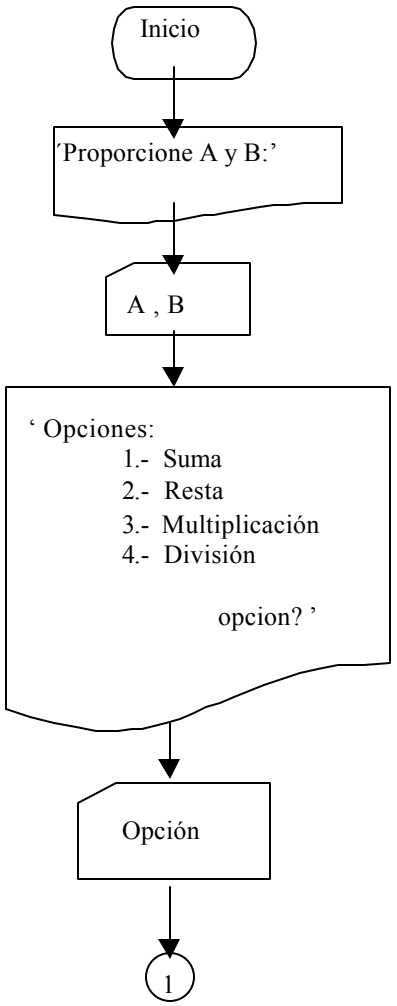
| Diagrama de flujo | Pseudocódigo |
|---|---|



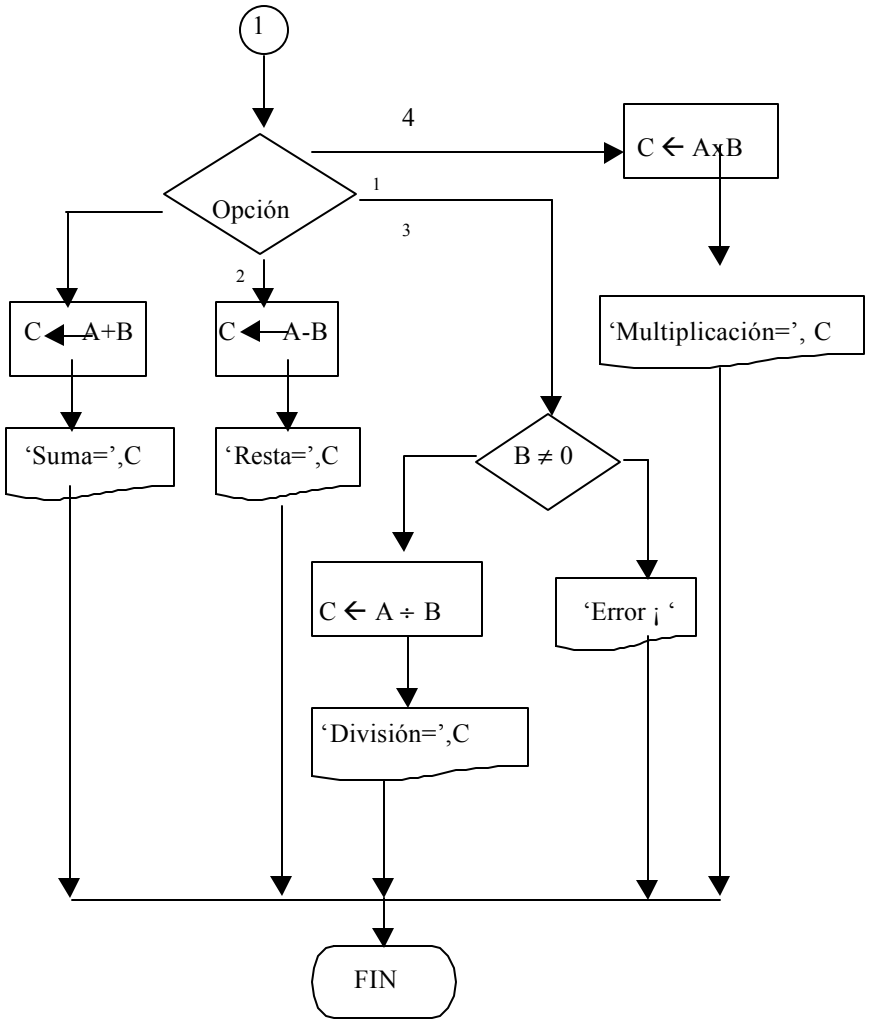

**Según** <expresión> **sea**
<valorl>**:**Acción1
<valor2>**:**Acción2
.
.
[**sino**
Acción n]
**fin_según**

**Ejemplo:** Para realizar alguna de las siguientes operaciones: suma, resta, multiplicación o división, según la elección del usuario.

| Diagrama de flujo | Pseudocódigo |
|---|---|




**Inicio**
**Escribir** 'Proprcione A y B'
**Leer** (A y B)
**Escribir** 'Opciones:'
**Escribir** ' 1.- Suma'
**Escribir** ' 2.- Resta'
**Escribir** ' 3.- Multiplicación'
**Escribir** ' 4.- División'
**Escribir** ' Opción ?'
**Leer** Opción
**Según** Opción **sea**
1: C ← A+B
**Escribir** 'Suma=',C
2: C ← A-B
**Escribir** 'Resta=',C
3: **Si** B ≠ 0 **entonces**
C ← A ÷B
**Escribir** 'División=',C
**sino**
**Escribir** 'Error !!'
**Fin_si**
4: C ← A x B
**Escribir** 'Multiplicación ='
**Fin_según**
**Fin**




Para terminar con esta sección, es importante mencionar que las estructuras selectivas simples y dobles emplean **Expresiones Lógicas** que sirven para plantear condiciones, que dan como resultado un valor booleano verdadero o falso, es decir se cumple o no se cumple la condición.

Las expresiones lógicas se pueden clasificar en : Simples y Compuestas.

Las Expresiones Lógicas Simples se forman relacionando operandos (variables y/o constantes) mediante ***operadores relacionales***. Así, tienen la siguiente forma:

Operando1 Operador Relacional Operando2

Los operadores relacionales son:

| **Operador** | **Significado** |
|---|---|
| > | Mayor que |
| < | Menor que |
| = | Igual a |
| ≠ | diferente de |
| >= | Mayor o igual que |
| <= | Menor o igual que |

Ejemplos:

A > B

B ≠ 0

Las Expresiones Lógicas Compuestas se forman relacionando operandos booleanos (expresiones lógicas que proporcionan un valor verdadero o falso) con ***operadores lógicos***. Tienen la siguiente forma:

Operando Booleano1 Operador Lógico Operando Booleano2

Los operadores lógicos son:

| **Operador** | **Significado** |
|---|---|
| Not (no) | Negación |
| And (y) | Conjunción |
| Or (o) | Disyunción |

Ejemplo:

(A ≠ B) y (B > C)

## II.7. ESTRUCTURAS DE REPETICIÓN

*Iterar* o *ciclar* es *repetir* una tarea: hacer algo y luego regresar y hacerlo una y otra vez hasta terminar la tarea. Las aplicaciones típicas de la computadora que requieren iteración son:

- La introducción de muchos datos, uno tras otro, para efectuar diversos cálculos (por ejemplo obtener el promedio de calificaciones de un alumno).
- La clasificación periódica de una gran colección de datos (por ejemplo la clasificación de cheques procesados por sucursal bancaria, y para cada sucursal por número de cuenta del cliente cada día de la semana).
- La búsqueda de un dato en una gran colección de ellos(por ejemplo encontrar el precio actual de un articulo o el estado de una cuenta de depósito).
- Y muchas formulas científicas que sólo se pueden calcular por aproximaciones sucesivas (reduciendo el intervalo de la respuesta con cada ciclo).

Existen tres clases de mecanismos de iteración:

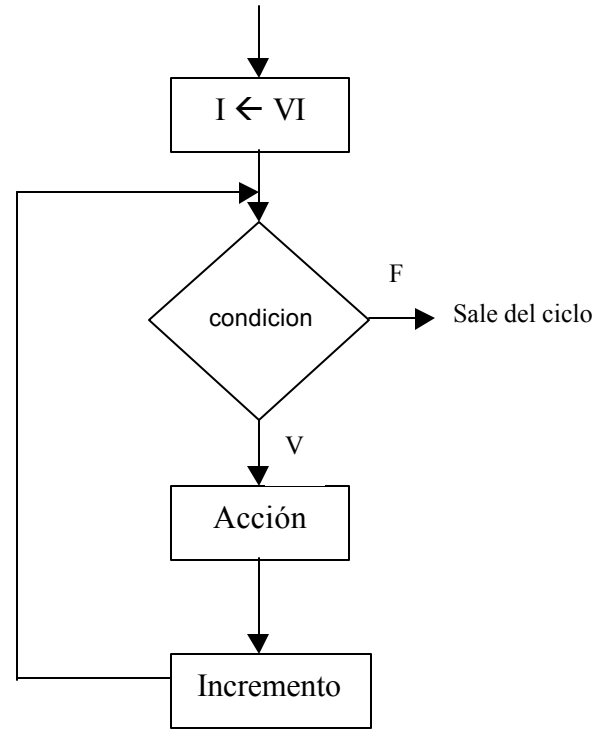
1. **Para** todos los valores de la progresión.
2. **Mientras** se valida una condición.
3. **Repetir- hasta** que se satisfaga una condición.

Veamos cada uno de ellos:

**PARA.**

Es usado *cuando se conoce de antemano*, el número de veces que debe repetirse una instrucción o conjunto de ellas. Es un ciclo *incondicional*, que abarca todos los valores de una progresión, empieza con el primer valor y termina con el ultimo de ellos, los valores de la progresión deben ser asignados a una variable, la cual se denomina *variable de control*.

| Diagrama de Flujo y Pseudocódigo |
|---|

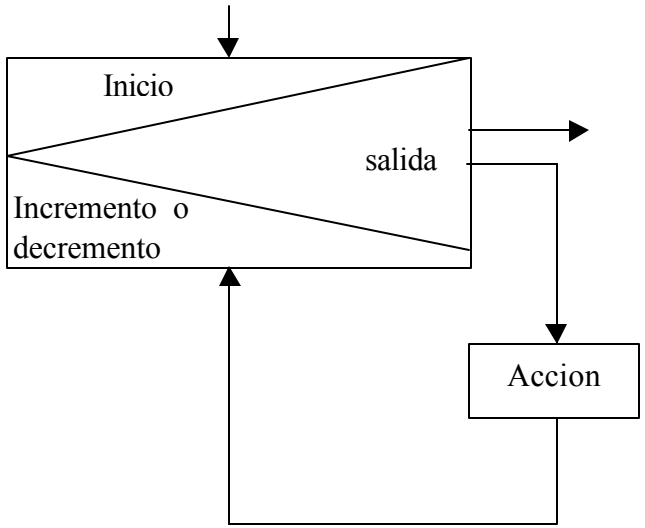



para <Variable de control> ← <valor_inicial> hasta <valor_final> hacer

acción 1
.
.
.
acción n

fin_para

Otra representacion para el Diagrama de Flujo




Funcionamiento

1. Se inicia la condición de control, se verifica la condición de paro si no se cumple entra al ciclo y ejecuta la acción.
2. Al llegar al fin regresa el control al encabezado de ciclo, actualizando el valor del contador de acuerdo al incremento o decremento.
3. Al volver el control del encabezado se pregunta si la variable de control llegó al valor final:
    a. Si se cumple, entonces se sale del ciclo, dirigiéndose a la siguiente instrucción después del fin.
    b. Si no ha tomado el valor final, entra al ciclo a ejecutar la instrucción.

Después de lo anterior, llega al fin el cual remite el control al inicio del para, actualizando el valor del contador de acuerdo con el incremento o decremento.

**Ejemplo.** Para sumar los 5 primero enteros positivos:

```
Inicio
        suma ← 0
        para j ← 1 a 5 hacer
                suma ← suma + j
        fin_para
        escribir(suma)
Fin
```

Aquí a la variable de control J se le asigna originalmente el valor 1. La variable SUMA, que antes de entrar al ciclo tenía un 0, aumentará por el valor de J. En la segunda pasada la J tendrá un 2, y SUMA, aumentara a 3. En la tercera vuelta J llegará a 3, y esto incrementará SUMA a 6, y así sucesivamente hasta que J se le asigne el último valor de la progresión, 5, después de los cual SUMA contendrá la suma de los 5 primeros enteros positivos, es decir, 15.

**CICLO PARA ANIDADO.**

Al igual que todas las estructuras de control, es posible que un ciclo PARA contenga anidado otro ciclo y éste a otro; veamos, el siguiente ciclo:

**Ejemplo**. Algoritmo ilustra el uso de un PARA anidado.

```
inicio
    para i ← 1 a 5 hacer
escribir ('i =',i)
                para j ← 1 a 3 hacer
                        escribir ('j =',j)
                fin_para
    fin_para
fin
```

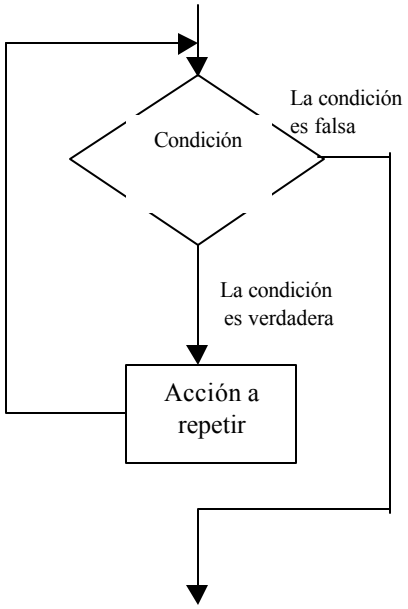
Se trata de un ciclo controlado por **I**, dentro del cual se imprime el valor de I; además, contiene anidado un ciclo **Para** controlado por **J**, donde se imprime el valor de **J**.
Por cada una de las veces que entre en el primer ciclo **Para** (el más externo), entrará 3 veces al ciclo más interno; esto significa que por 5 veces que entrará en **I**, lo hará 3 veces en **J**.

Obsérvese el par **inicio-fin** porque hay más de una instrucción dentro del ciclo.

**MIENTRAS.**
La instrucción *Mientras... hacer* continuará repitiéndose mientras la condición continué siendo válida (es decir, su valor de verdad sea *verdadero*).

| Diagrama de Flujo y Pseudocódigo |
|---|

MIENTRAS...HACER




mientras <condición>hacer
<acciones>
fin_mientras

**Ejemplo**: Elaborar un algoritmo que calcule e imprima el sueldo de varios empleados utilizando *MIENTRAS*

**inicio**
**escribir** ( '¿hay empleado (s/n)?')
**leer** (otro)
**mientra**s otro = 's' **hacer**
**escribir** ('proporcione nombre, número de horas trabajadas y cuota')
**leer** (nombre,hrstrab,cuotahr)
sueldo ← hrstrab*cuotahr
escribir (nombre, sueldo)
**escribir**( '¿desea procesar otro empleado (s/n)?')
**leer** (otro)
**fin_mientras**
**fin**

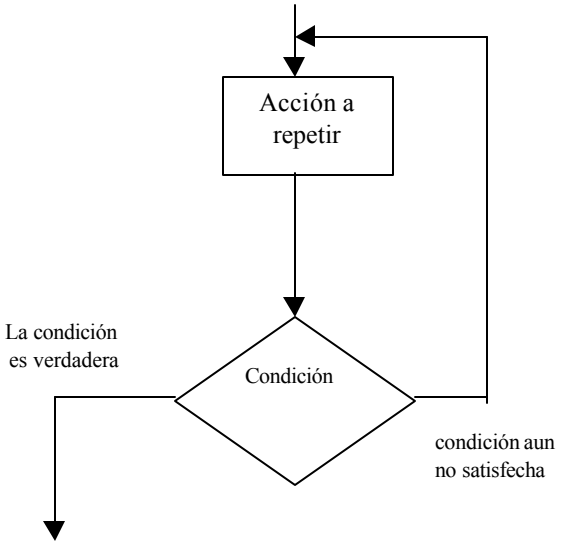
**REPETIR**

La instrucción *Repitir...hasta* continuará repitiéndose mientras no se satisfaga la condición (su valor de vedad sea *falso*).

Esta instrucción tiene el siguiente diagrama de flujo:

| Diagrama de Flujo y Pseudocódigo |
|---|

REPETIR ... HASTA




repetir
<accion>
hasta <condición>

Vemos que en el MIENTRAS...HACER la condición se evalúa primero, y si la prueba falla (el valor de verdad de la condición es FALSO), entonces el ciclo no se lleva a cabo de ninguna manera. En ciclo REPITIR … HASTA que la prueba se realiza al final (es decir la condición se evalúa al final) y si el valor de verdad de la condición es VERDADERO, entonces se abandona el ciclo después de realizarlo por lo menos una vez.

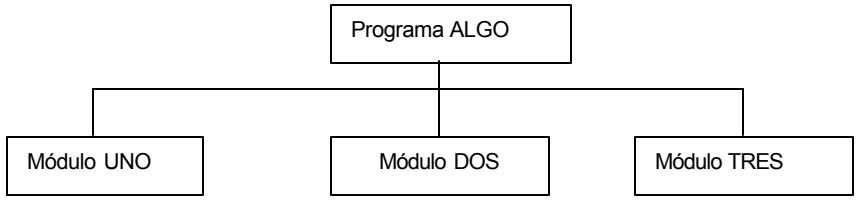
**Ejemplo**: Elaborar un algoritmo que calcule e imprima el sueldo de varios empleados utilizando REPETIR

```
inicio
    repetir
            escribir ('proporcione nombre, número de horas trabajadas y
            cuota ')
            leer (nombre,hrstrab,cuotahr)
            sueldo ←hrstrab*cuotahr
            escribir (nombre, sueldo)
            escribir ( '¿desea procesar otro empleado (s/n)?')
            leer (desea)
   hasta desea = 'n'
fin
```

## II.8. DISEÑO DESCENDENTE

El diseño descendente es la técnica que permite diseñar la solución de un problema con base en la modularización, o segmentación dándole un enfoque de arriba abajo (Top Down Design). Esta solución se divide en módulos que se estructuran e integran jerárquicamente, como si fuera el organigrama de una empresa.

**Ejemplo:**



En el diagrama anterior se muestra la estructura del programa ALGO, que se auxilia de tres módulos subordinados, cada uno de los cuales ejecuta una tarea especifica. En su momento, el módulo principal ALGO invocará o llamará a los módulos subordinados, es decir, dirigirá su funcionamiento.

¿QUE ES UN MÓDULO?

Es un segmento, rutina, subrutina, subprograma que puede ser definido dentro de un programa con el propósito de ejecutar una tarea específica, pudiendo ser llamado o invocado desde el programa principal cuando se requiera.

¿CUÁNDO ES ÚTIL LA MODULARIZACIÓN?

Este enfoque de segmentación o modularización es útil en dos casos:

1. Cuando existe un grupo de instrucciones o una tarea especifica que debe ejecutarse en más de una ocasión.
2. Cuando un problema es complejo o extenso, la solución se "divide" o "segmenta" en módulos que ejecuta "partes" o tareas específicas

Las principales razones de la estructura de módulos se deben a que los programas son:

- Mas fáciles de escribir
- Mas fáciles de leer y comprender
- Mas fáciles de corregir y modificar
- Mas fáciles de usar.

Dicha solución es organizada de forma similar a como lo hacen las empresas cuando se estructuran con base en las funciones para realizar sus actividades; en otras palabras, el trabajo se divide en partes que sean fácilmente manejables y que, lógicamente, pueden ser separadas; así, cada una de estas partes se dedica a ejecutar una determinada tarea, lo que redundará en una mayor concentración, entendimiento y capacidad de solución a la hora de diseñar la lógica de cada una de estas. Dichas partes son los módulos o segmentos del programa, algunos de ellos son módulos directivos, o de control, que son los que se encargarán de distribuir el trabajo de los demás módulos. De esta manera se puede diseñar un organigrama que indique la estructura general de un programa.

En el diagrama anterior se tiene un módulo directivo llamada ALGO, que dirige el funcionamiento de tres módulos subordinados que son: MODULO UNO, MODULO DOS Y MODULO TRES.

**PROCESO DE MODULARIZACIÓN.**

El proceso de modularización consiste en hacer una abstracción de un problema, del cual se tiene inicialmente un panorama general, enseguida se procede a "desmenuzar" o "dividir". El problema en partes pequeñas y simples, como se muestra:

**Ejemplo:** Se necesita elaborar un programa que calcule el sueldo de varios empleados, similar a los algoritmos vistos en estructuras de repetición, sólo que ahora utilizando descomposición modular y controlando el salto de página.

El proceso es el siguiente:

1. El proceso es el siguiente: leer el monto del salario mínimo, para cada empleado se solicitaran sus datos, se calculará su salario y se escribirán estos datos y salario. Además, cada 57 empleados se debe imprimir el encabezado que ocupa tres lineas. Este proceso termina cuando ya no se desean agregar empleados.

*Aplicación*:

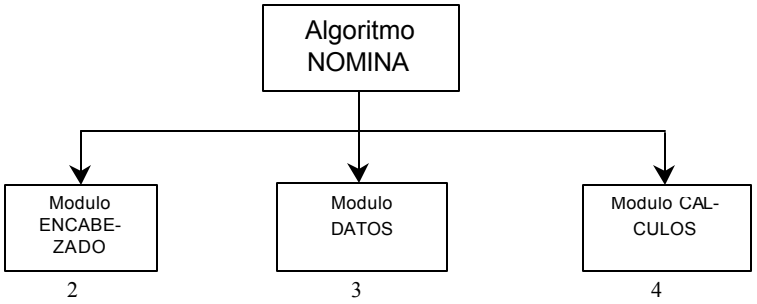
Aplicar lo anunciado en este punto; y, si se considera que se trata de un programa que ayudara a calcular la nomina tenemos el módulo principal siguiente:

| PROGRAMA NOMINA |
|---|

2. Se toma este módulo y se busca la forma de dividirlo en otros módulos más pequeños, que ejecuten tareas o funciones específicas. Las mismas funciones que se desea que ejecute el programa, nos darán la pauta para definir los módulos y así hacer una segmentación de la solución del problema en partes más manejables.

*Aplicación*:

Si nos referimos al ejemplo anterior, tenemos que se deben llevar a cabo tres tareas o funciones claramente definidas: encabezado, datos, cálculos de ahí que necesitemos de un módulo para cada una de las tareas, las cuales deberán estar subordinadas al módulo principal. La estructura que se tiene, entonces, es la siguiente:



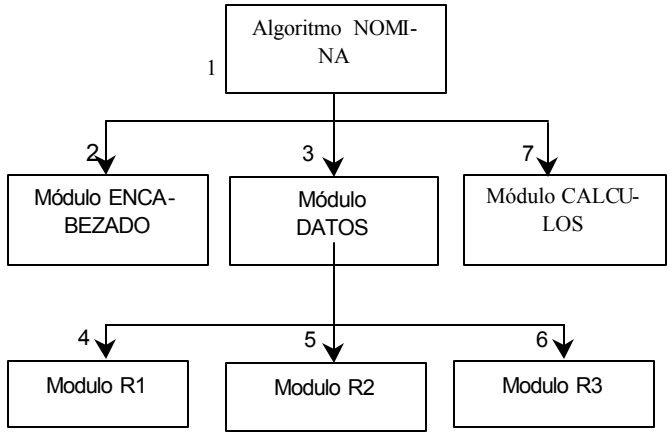

Nota: Los módulos se enumeran de arriba hacia abajo y de izquierda a derecha

**3.** Se repite el paso 2 para cada módulo nuevo definido, hasta llegar a un nivel de detalle adecuado, es decir, hasta hacer que cada módulo ejecute una tarea específica, que esté claramente definida y que el diseño y la codificación de la lógica del mismo, resulte fácil.

*Aplicación*:

En el problema que estamos analizando, se revisan los módulos encabezado, datos y cálculos. Se considera que estos módulos tienen un nivel de detalle adecuado, porque cada módulo hace una tarea muy simple, clara y específica y, en consecuencia, se pueden diseñar fácilmente.

En caso de no ser así, es decir, que el módulo datos, por ejemplo, requiera otros módulos subordinados, la estructura general del programa sería la siguiente:




Como se puede ver, la numeración sufre ciertas alteraciones, es decir, cuando se ejecuta el módulo datos, se enumera todo lo que dependa de éste, para después continuar con él de cálculos, que es el siguiente en el orden. En este caso, el módulo de datos se convierte a su vez en un módulo directivo que controla la ejecución de sus módulos subordinados.

## II.9. PASOS PARA PROGRAMAR

Desarrollar un programa es un proceso lógico lineal. Si se considera el tiempo requerido y sigue el proceso de principio a fin, se conseguirá tener éxito en la programación.

1. Diseñar el programa
2. Escribir el programa
3. Compilar el programa
4. Ligar el programa
5. Probar el programa

**DISEÑAR EL PROGRAMA**

Esta es la parte más importante, pues aquí se definirán los rasgos y características del programa a desarrollar, lo primero es hacer un bosquejo del programa con tanto detalle como sea posible. La mayoría de los programas siguen un patrón llamado IPO, para Input (Entrada), Processing (Procesamiento), Output (Salida).

Por ejemplo se requiere un programa que obtenga una media de una muestra aleatoria. ¿Qué necesita hacer el programa?. Considere las entradas. Se requieren los datos que proporcionan información acerca de las características de lo censado, por ejemplo el costo, la durabilidad, o simplemente se desea obtener un promedio de calificaciones de un semestre, en este caso se emplean las calificaciones como entradas.

A continuación se determina el proceso. En este caso se requiere calcular la suma de las calificaciones y posteriormente dividirlas entre el número de estas. Finalmente, considere la salida. El resultado de los cálculos deberá ser desplegado o impreso para presentar un reporte o hacer un análisis.

Entrada. Indicar al usuario del programa que ingrese el número de calificaciones. Aceptar por el teclado cada una de las calificaciones
Proceso. Sumar las calificaciones y el total dividirlo en el número de datos solicitados.
Salida. Mostrar el promedio en pantalla o impreso en papel

**ESCRIBIR EL PROGRAMA**

Se utiliza un editor para escribir el programa. Un editor es un programa similar a un procesador de texto, excepto que no requiere la capacidad de dar formato a los caracteres o a los párrafos. En realidad, el archivo del código fuente no deberá contener ningún código de formato especialel compilador no los entenderá y los tratará como errores.

**COMPILAR EL PROGRAMA**

Después de grabar el archivo con el código fuente, utilice el compilador para crear el archivo objeto inmediato. Aquellas instrucciones que el compilador no puede entender generan avisos preventivos del compilador o mensajes de error. Un aviso preventivo (WARNING) significa que hay un problema potencial pero que el compilador puede continuar generando el código objeto. Un mensaje de error (ERROR normalmente detendrá el proceso de compilación. Si aparece un mensaje de error, se requiere volver a cargar, por medio de un editor, el archivo con el programa fuente, y corregir el error. Por lo general estos son errores de sintaxis, equivocaciones en la escritura, puntuación o en la redacción de un comando C o una función.

**ENCADENAR EL PROGRAMA**

Una vez que no haya errores de compilación, se encadena el archivo objeto con las bibliotecas para crear un programa ejecutable. Se obtendrán mensajes de error si el encadenador no puede encontrar la información requerida en las bibliotecas. Deberá analizarse el código fuente para estar seguros de que se están empleando los archivos de biblioteca correctos.

**PROBAR EL PROGRAMA**

Ahora se puede proceder a ejecutar el programa. Si todo se realizo correctamente, el programa correrá sin problemas. Sin embargo, podrán presentarse dos tipos de errores.

Error en el tiempo de ejecución
Errores de lógica

Un error en tiempo de ejecución se presenta cuando un programa incluye una instrucción que no puede ser ejecutada. Aparecerá un mensaje en la pantalla y el programa se detendrá. Los errores de

tiempo de ejecución con frecuencia están asociados con archivos o con dispositivos del equipo

Los errores de lógica ocurren cuando el programa puede continuar con la ejecución de las instrucciones pero estas son incorrectas; esto es, cuando producen resultados erróneos. Estos son los problemas más difíciles de detectar porque puede ser que se ignore su existencia. Se deberá analizar los resultados y las salidas del programa para verificar su
exactitud.

## II.10 ARREGLOS

Un arreglo es un conjunto finito y ordenado de elementos homogéneos, que se referencian por un identificador común (nombre). La propiedad ordenado significa que el elemento primero, segundo, hasta el n-ésimo de un arreglo puede ser identificado. La propiedad homogéneo significa que todos los elementos de un arreglo son del mismo tipo de datos.

Los arreglos se clasifican en:

- Unidimensionales (Vectores).
- Bidimensionales (Tablas o Matrices)
- Multidimensionales.

### *ARREGLOS DE UNA SOLA DIMENSIÓN O VECTORES*

Un arreglo de una dimensión o vector es un conjunto finito y ordenado de elementos homogéneos.

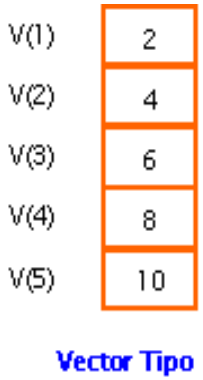
El *subíndice* o *índice* de un elemento [1, 2, ..,i, i+1, ,, n] designa su posición en el orden del vector. Por ejemplo un arreglo denominado Coches, que consta de 7 elementos, se puede representar con un vector de longitud 7 de la siguiente manera:

**Vector Coches**

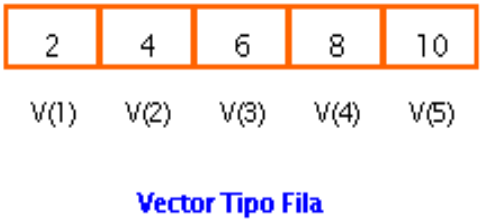
| Jetta | Pontiac | Seat | Chevy | Jeep | Peugeot | Mustang |
|---|---|---|---|---|---|---|
| Primer Elemento | Segundo Elemento | Tercer Elemento | Cuarto Elemento | Quinto Elemento | Sexto Elemento | Séptimo Elemento |

Los vectores se pueden representar como filas de datos o como columnas de datos.

Los elementos de un arreglo unidimensional o vector **V** de 5 elementos se muestran a continuación:




Vector Tipo




Vector Tipo Fila

Los elementos del vector V se representan con la siguiente notación:

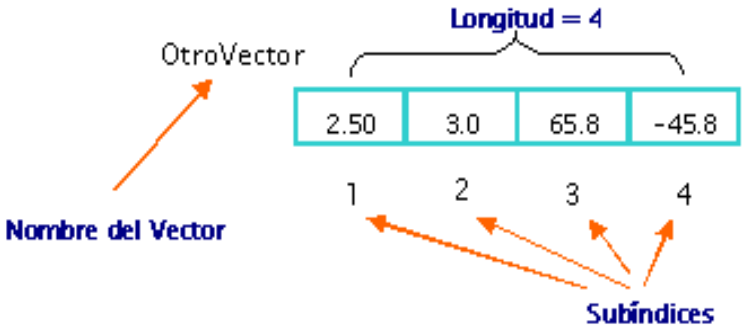
V[1], V[2], V[3], ..., V[N]

Es decir, todos los elementos tienen el mismo nombre (V) y se referencian a través del subíndice que indica la posición que ocupan en el arreglo. Dichos elementos se manipulan como variables individuales. De esta manera se le puede asignar un valor a un elemento, por ejemplo V[1]= 3 u obtener el

valor de un elemento y almacenarlo en otra variable, por ejemplo, Aux = V[4].

En un arreglo unidimensional o vector se distinguen tres elementos:

- El Nombre del vector
- La Longitud o rango del vector
- Los Subíndices del vector




## OPERACIONES BÁSICAS CON VECTORES

Las operaciones que se pueden realizar con vectores son:

- Asignación
- Lectura/Escritura
- Actualización
- Recorrido o acceso secuencial
- Ordenamiento
- Búsqueda

En general las operaciones con vectores implican el procesamiento de los elementos individuales del vector.

## ASIGNACIÓN

La asignación de valores a un elemento del vector se realiza con la instrucción de asignación:

<NombreVector>[subíndice] ← <Valor>

Así la instrucción V[1] = 8, le asigna el valor 8 al elemento 1 del vector V.

Ejemplos:

V[1] ← 8

MiVector[4] ← 16

Nombres[12] ← 'México'

MiVector[1] ← MiVector[1] + 1

MiVector[2] ← MiVector[1] / 2

V[2] ← V[3] / 4 + V[4] * 3

Los subíndices de un vector pueden ser enteros, variables o expresiones enteras. Esto es,

Si i ← 2 entonces

| | |
|---|---|
| V[ i ] | representa el 2do. elemento del vector |
| V[ i + 1 ] | representa el 3er. elemento del vector |
| V[ i + 3 ] | representa el 5to. Elemento del vector |

Así, si el objetivo es asignar valores a todos los elementos de un vector, entonces se pueden utilizar las estructuras repetitivas (Para-Hasta-Hacer, Mientras-Hacer, Repite-Hasta)

Ejemplo: Asignar el valor 0 a todos los elementos del vector A de longitud 100.

```
Para x ← 1 hasta 100 hacer
   A(X) ← 0
Fin_para
```

**LECTURA/ESCRITURA**

La lectura y escritura de datos de arreglos se realizan con estructuras repetitivas, o utilizando los elementos individuales si ese es el caso.

Ejemplo: Leer los datos del arreglo M de longitud 50 y mostrarlos por pantalla, uno a uno.

```
Escribir('Los valores del arreglo son : ')
        Para i ← 1 Hasta 50 Hacer
           Leer(Mi]
           Escribir(M[i])
        Fin_para
```

Los elementos de un vector se pueden leer también con ciclos *Mientras* o *Repite.*

Ejemplo:

```
i ← 1
Mientras i <= 50 Hacer
        Leer(M[i])
        Escribir(M[i])
        i ← i +1
Fin_mientras
```

**RECORRIDO O ACCESO SECUENCIAL**

Se accede a los elementos de un vector para introducir datos en él (asignar valores ó leer) y para ver su contenido (escribir). A la operación de efectuar una acción general sobre todos los elementos de un vector se le denomina recorrido del vector. Estas operaciones se realizan utilizado estructuras repetitivas, cuyas variables de control (por ejemplo, i) se utilizan como subíndices del vector (por ejemplo, V(i)). El incremento de la variable de control produce el tratamiento sucesivo e individual de los elementos del vector.

Ejemplo: Cálculo de la suma y promedio de los primeros 10 elementos enteros de un vector W

```
Inicio
        Suma ← 10
        Para i ← 1 Hasta 10 Hacer
                Leer(W[i])
        Fin_para
        Para i ← 1 Hasta 10 Hacer
                Suma ← Suma + W[i]
        Fin_para
        Promedio ← Suma/10
        Escribir(Suma, Promedio)
Fin
```

**ACTUALIZACIÓN**

La actualización del vector consiste en la asignación de valores a los elementos del vector ya sea de manera individual o a través del procesamiento general del vector.

Ejemplo 1: Almacenar en el arreglo M de longitud 20, los valores leídos por teclado.

```
Inicio
        Para x ← 1 hasta 20 hacer
                Leer(valor)
                M[x] ← valor
        Fin-para
Fin
```

**ORDENAMIENTO**

El ordenamiento de un vector es el acomodo de los valores de los elementos del vector de acuerdo a un criterio específico.

Ejemplo:

- Ordenar los valores del arreglo NUM de 20 elementos de tipo entero de menor a mayor.

- Ordenar los elementos de arreglo ALUMNOS en orden alfabético

Para resolver estos problemas, se puede utilizar el método de ordenamiento por selección, el cual consiste en encontrar el elemento más pequeño de la lista y colocarlo en la primera posición, luego se encuentra el siguiente elemento más pequeño y se lleva a la segunda posición. Se continúa el proceso hasta llegar a la posición última del vector.
Para obtener el elemento más pequeño se compara el primer valor con cada uno de los restantes y cada vez que se encuentra un valor más pequeño se intercambian los valores.

Ejemplo:

```
Inicio
        // Encontrar la posición del menor valor
        // Se utilizan dos índices: J para el elemento fijo y
        //  K para recorrer los elementos que se comparan.
        Para J ← 1 Hasta 4  Hacer
                Para  K ← J + 1 Hasta 5 Hacer
                        Si  Num[K] < Num[ PosMenor] Entonces
                                // Intercambiar valores en el
                        vector
                                aux ← Num[J]
                                Num[J] ← Num[k]
                                Num[k] ← aux
                        Fin_si
                Fin_ Para
        Fin_Para
Fin
```

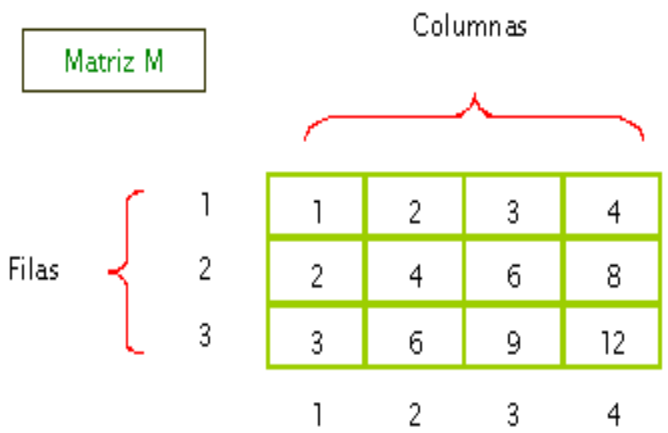
**BÚSQUEDA**

Es el proceso que localiza la posición (subíndice) del elemento que contiene el valor deseado o requerido. Se puede utilizar para determinar si un valor dado se encuentra o no en un vector.

Ejemplo 1: Búsqueda del elemento “Manzana” dentro del vector FRUTAS. Imprimir si se encuentra o no y dado el caso su posición.

```
Inicio
        Encontrado = Falso
        // Lectura del valor a buscar
        Leer(fruta)
        //Búsqueda del valor en el vector
        Para i ← 1 Hasta 10 Hacer
                Si frutas[i] = fruta entonces
                        Encontrado ← Verdadero
```

```
                    Posición ← i
                Fin_si
            Fin_para
            // Notificación de los resultados de la búsqueda
            Si  Encontrado = Verdadero Entonces
                Escribir('Fruta encontrada en la posición: ', posi-
            cion)
            Sino
                Escribir('Fruta no lo encontrada')
            Fin_si
Fin
```




## *ARREGLOS BIDIMENSIONALES O MATRICES*

Un arreglo bidimensional (de dos dimensiones) es un vector de vectores. Es un conjunto de elementos, todos del mismo tipo, en los que el orden de los componentes es significativo y en el que se necesitan dos subíndices para definir cualquier elemento. Un arreglo bidimensional se denomina también tabla o matriz.

Los arreglos bidimensionales se referencian con dos subíndices. El primero se refiere a la fila y el segundo se refiere a la columna. Así en una matriz M de dimensión 3 x 4 (3 filas por 4 columnas), una representación sería:

Donde:
El elemento M[1,1] se refiere al elemento situado en la fila 1 columna 1

El elemento M[3,2] se refiere al elemento situado en la fila 3 columna 2

El elemento M[2,3] se refiere al elemento situado en la fila 2 columna 3

La matriz M almacena hasta 12 [3x4] elementos.

## OPERACIONES CON MATRICES

Las operaciones que se pueden realizar sobre las matrices son básicamente las mismas que con los vectores, es decir:

- Asignación
- Lectura/Escritura
- Actualización
- Recorrido o acceso secuencial
- Ordenamiento.

Ejemplo 1: Recorrido de una matriz A de 3x4 de tipo entero para inicializar sus elementos con el valor 0.

```
Inicio
        para i ← 1 hasta 3 hacer
           para J ← 1 hasta 4 hacer
              A[i,j] ← 0
           Fin_para
        Fin_para
 Fin
```

Ejemplo 2: Inicializar los elementos de una matriz M de 5x7 de tipo entero con el valor de la suma de los subíndices que determinan su posición, es decir, M[i,j] = i + j.

```
Inicio
        Para i ← 1 hasta 5 hacer
           Para j ← 1 hasta 7 hacer
              M[i,j] ←  i + j
           Fin_para
        Fin_para
Fin
```

Ejemplo 3: Recorrer una Tabla de 100x200 de tipo real para determinar la posición del elemento más grande.

```
Inicio
        Max ←  Tabla[1,1]
        imax ← 1
        jmax ← 1
        Para i ← 1 hasta 100 hacer
                Para j ← 1 hasta 200 hacer
                        Si Tabla[i,j] > Max entonces
                                Max ← Tabla[i,j]
                                imax ← i
                                jmax ← j
                        fin_si
                Fin_para
        Fin_para
        Escribir('El elemento mayor es : ')
        Escribir('Tabla(',imax, ',', imax,')= ',Max)
Fin
```

## *ARREGLOS MULTIDIMENSIONALES*

Los arreglos multidimensionales son aquellos que tienen tres o mas dimensiones. En general un arreglo multidimensional es un conjunto de elementos, todos del mismo tipo, en los que el orden de los componentes es significativo y en el que se necesitan tantos subíndices como dimensiones tenga para definir cualquier elemento.

### OPERACIONES CON ARREGLOS MULTIDIMENSIONALES

Las operaciones que se pueden realizar sobre los arreglos multidimensionales son básicamente las mismos que con los vectores y las matrices, es decir:

- Asignación
- Lectura/Escritura
- Actualización
- Recorrido o acceso secuencial
- Ordenación
-

Ejemplo: Inicializar un arreglo multidimensional MulDim de 3x4x7x12 de tipo entero asignando a cada elemento el valor 0.

```
Inicio
        Para i ← 1 hasta 3 hacer
                Para j ← 1 hasta 4 hacer
                        Para k ← 1 hasta 7 hacer
                                Para l ← 1 hasta 17 hacer
                                        MultiDim(i,j,k,l) = 0
                                Fin_para
                        Fin_para
                Fin_para
        Fin_para
Fin
```