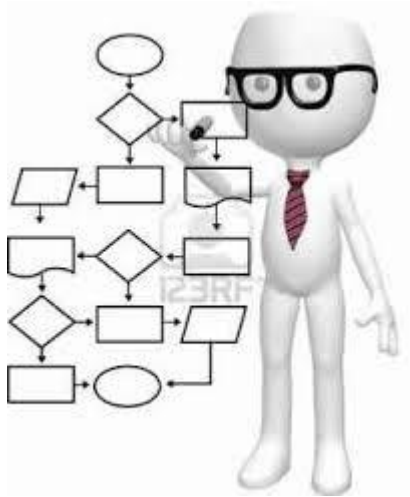


Lógica y Estructura de Datos

Algoritmos



```
Inicio
  aprobados ← 0
  reprobados ← 0
  resultado ← 0
  estudiantes ← 1
  Mientras estudiantes <= 10
    Leer resultado
    Si resultado == 1 entonces
      aprobados ← aprobados + 1
    SiNo
      reprobados ← reprobados + 1
    FinSi
    estudiantes ← estudiantes + 1
  FinMientras
  Mostrar aprobados
  Mostrar reprobados
  Si aprobados > 8 entonces
    Mostrar "Aumentar la colegiatura"
  FinSi
Fin
```

Técnico Superior en Desarrollo de Software

Esc. Superior N° 49 "Cap. Gral. J.J. Urquiza"

Año 2017

Autor: Ing. Álvaro Hergenreder

1. Introducción a la Programación

Para realizar un proceso en la computadora se le debe suministrar al procesador un algoritmo adecuado, por ejemplo al cocinero debe dársele una receta, al pianista una partitura, y así sucesivamente considerando al cocinero y al pianista como procesadores.

En la computadora el algoritmo ha de expresarse de una forma que recibe el nombre de programa, un programa se escribe en un lenguaje de programación, y a la actividad de expresar un algoritmo en forma de programa se llama Programación.

1.1 Algoritmo

Un algoritmo se puede definir como una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas. Esas instrucciones son las operaciones que debe realizar la computadora.

Ese grupo de instrucciones realizadas en orden conducen a obtener la solución de un problema. En esencia, todo problema se puede describir por medio de un algoritmo.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa. El diseño de algoritmos requiere creatividad y conocimientos profundos de la técnica de programación (Joyanes, 1990). Luis Joyanes, programador experto y escritor de muchos libros acerca de lógica y programación dice que en la ciencia de la computación y en la programación los algoritmos son más importantes que los lenguajes de programación o las computadoras. "Un lenguaje de programación es sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo".

Los algoritmos son independientes de los lenguajes de programación. En cada problema el algoritmo puede escribirse y luego ejecutarse en un lenguaje diferente de programación. El algoritmo es la infraestructura de cualquier solución, escrita en cualquier lenguaje. Así por ejemplo en una analogía con la vida diaria, una receta de un plato de comida se puede expresar en español, inglés o francés, pero en cualquiera que sea el lenguaje, los pasos para la elaboración de él, se realizara sin importar el idioma.

Un lenguaje de programación es tan solo un medio para expresar un algoritmo y una computadora es solo un procesador para ejecutarlo.

Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y efectúe el proceso correspondiente.

Dentro de los objetivos planteados para esta asignatura aparecen palabras tales como: algoritmo, programa, lenguaje de programación, etc., con las cuales no estamos familiarizados; para entenderlos mejor veamos algunas definiciones:

ALGORITMO: Secuencia de acciones o pasos que permite resolver un problema. Un mismo problema puede ser resuelto con distintos algoritmos.

PROGRAMA: Traducción o codificación de un algoritmo a un lenguaje de programación; o sea, una secuencia de instrucciones que indica las acciones que han de ejecutarse.

LENGUAJE DE PROGRAMACION: Conjunto de reglas, símbolos y palabras especiales utilizadas para construir un programa.

LENGUAJE DE MAQUINA: Lenguaje usado directamente por la computadora y compuesto de instrucciones codificadas en sistema binario.

COMPILADOR: Programa que traduce un programa escrito en lenguaje de alto nivel a lenguaje de máquina. No todos los lenguajes requieren de compiladores, en cuyo caso se los denomina **INTÉRPRETES**. Por ejemplo lenguajes como C o JAVA requieren compiladores para generar código máquina y correr en una PC, y otros lenguajes como JAVASCRIPT son interpretados directamente por la computadora.

1.1.1 Características de los algoritmos

- Preciso. Definirse de manera rigurosa, sin dar lugar a ambigüedades.
- Definido. Si se sigue un algoritmo dos veces, se obtendrá el mismo resultado.
- Finito. Debe terminar en algún momento.
- Debe tener cero o más elementos de entrada, es decir, debe tener por lo menos una instrucción que ordene averiguar el dato o los datos.
- Debe producir un resultado. Los datos de salida serán los resultados de efectuar las instrucciones. Los datos de entrada pueden ser ninguno, pero los de la salida deben ser alguno o algunos.
- Se concluye que un algoritmo debe ser suficiente y breve, es decir, no exceder en las instrucciones ni quedarse corto. Entre dos algoritmos que lleven a un mismo objetivo, siempre será mejor el más corto.

1.1.2 Etapas para la solución de un problema por medio del computador

- Análisis del problema, definición y delimitación (macroalgoritmo). Considerar los datos de entrada, el proceso que debe realizar el computador y los datos de salida.
- Diseño y desarrollo del algoritmo. Pseudocódigo o escritura natural del algoritmo, diagramas de flujo, Diagramas rectangulares.
- Prueba de escritorio. Seguimiento manual de los pasos descritos en el algoritmo. Se hace con valores bajos y tiene como fin detectar errores.
- Codificación. Selección de un lenguaje y digitación del pseudocódigo haciendo uso de la sintaxis y estructura gramatical del lenguaje seleccionado.
- Compilación o interpretación del programa. El software elegido convierte las instrucciones escritas en el lenguaje a las universales comprendidas por el computador.
- Ejecución. El programa es ejecutado por la máquina para llegar a los resultados esperados.
- Depuración (debug). Operación de detectar, localizar y eliminar errores de mal funcionamiento del programa.
- Evaluación de resultados. Obtenidos los resultados se los evalúa para verificar si son correctos. Un programa puede arrojar resultados incorrectos aun cuando la ejecución es perfecta.

1.1.3 Representación de algoritmos

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Ello permitirá que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje.

Para la representación de un algoritmo, antes de ser convertido a lenguaje de programación, se utilizan algunos métodos de representación gráfica o numérica. Los métodos más conocidos son:

- a. **Lenguaje natural.**
- b. **Pseudocódigo.**
- c. **Diagramación libre (Diagramas de flujo).**
- d. **Diagramas Nassi-Shneiderman o Chapin.**

a) **Algoritmo con Lenguaje Natural**

● Algoritmo para leer las páginas de un libro:

1. Inicio.
2. Abrir el libro en la 1ª página.
3. Leer la página.
4. ¿Es la última que deseo leer?
Sí: Ve al paso 7.
No: Ve al paso 5
5. Pasar a la siguiente página.
6. Ve al paso 3.
7. Cerrar el libro.
8. Fin.

b) **Pseudocódigo**

- Mezcla de lenguaje de programación y de lenguaje natural.
- Representación narrativa de los pasos que debe seguir un algoritmo.
- Utiliza palabras, no gráficos.
- Ventajas:
 - Ocupa menos espacio.
 - Permite representar fácilmente operaciones repetitivas complejas.
 - Es muy fácil pasar del pseudocódigo al lenguaje de programación.
 - Quedan claros los niveles que tiene cada operación.

Ejemplo de pseudocódigo

- Algoritmo que lee 3 números, los suma e imprime su resultado.

PSEUDOCÓDIGO sumatorio

VARIABLES

eN1, eN2, eN3, eSuma: Entero

INICIO

ESCRIBE "Dame tres números:"

LEE eN1, eN2, eN3





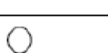

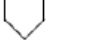
$eSuma = eN1 + eN2 + eN3$

ESCRIBE "El resultado de la suma es: ", eSuma

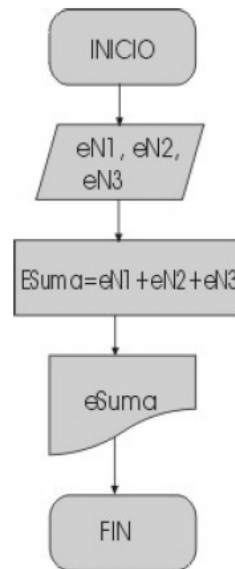
FIN

c) *Diagramas de Flujo*

- Es una forma de representar gráficamente un algoritmo.
- Cada paso se escribe dentro de un símbolo.
- Los pasos se conectan unos con otros mediante **líneas de flujo**.
- Son fáciles de diseñar, pero difíciles de actualizar.
- Los símbolos que utiliza están normalizados:

	Inicio/Final
	Entrada/ Salida
	Proceso
	Salida por impresora
	Conector dentro página
	Conector fuera página
	Salida por pantalla

- Diseñe un algoritmo que lea 3 números, los sume e imprima el resultado por impresora.



d) **Diagramas Estructurados (Chapin)**

- Como un diagrama de flujo en el que se omiten las flechas de unión, y las cajas son contiguas.
- Son fáciles de diseñar y difíciles de actualizar.
- Las acciones sucesiva se escriben en cajas sucesivas.

Inicio
Leer Nombre,Hrs,Precio
Calcular Salario = Hrs * Precio
Calcular Imp = Salario* 0.15
Calcular Neto = Salario + Imp
Escribir Nombre, Imp, SNeto
Fin

1.1.4 Componentes de un algoritmo

Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en el orden en el que han de ser ejecutadas. Los pasos en esta primera descripción de actividades deberán ser refinados añadiendo detalles a los mismos e incluso, algunos de ellos, pueden requerir un refinamiento adicional antes que podamos obtener un algoritmo claro preciso y completo. En un algoritmo se debe considerar tres partes:

- **Entrada:** Información dada al algoritmo
- **Proceso:** Operaciones o cálculos necesarios para encontrar la solución del problema
- **Salida:** Respuestas dadas por el algoritmo o resultados finales de los cálculos.

Como ejemplo imagínese que desea desarrollar un algoritmo que calcule la superficie de un rectángulo proporcionándole su base y altura. Lo primero que deberá hacer es plantearse y contestar a las siguientes preguntas:

Especificaciones de entrada:

- ¿Qué datos son de entrada?
- ¿Cuántos datos se introducirán?
- ¿Cuántos son datos de entradas válidos?

Especificaciones de salida:

- ¿Cuáles son los datos de salida?
- ¿Cuántos datos de salida se producirá?
- ¿Qué precisión tendrán los resultados?
- ¿Se debe imprimir una cabecera?

El algoritmo se podrá representar con los siguientes pasos:

Paso 1 Entrada de base y altura desde periférico de entrada por ejemplo teclado.

Paso 2 Cálculo de la superficie, multiplicando la base por la altura.

Paso 3 Salida por pantallas de base, altura y superficie.

Y utilizando Chapin quedaría:

sup \leftarrow 0
Leer(b) Leer(a)
sup \leftarrow b * a
Escribir("Base: " b)
Escribir("Altura: " a)
Escribir("Superficie: " sup)

1.2 CONSTANTES Y VARIABLES

IDENTIFICADOR

Como en el álgebra, a cada dato o elemento, ya sea constante o variable, se le bautiza con un nombre o identificador; el cual si se ha elegido adecuadamente, ayuda mucho a la persona que lea el programa.

CONSTANTES

Como su nombre lo indica, son datos que no varían durante la ejecución de un programa.

VARIABLES

Son datos que cambian o evolucionan durante la vida o ejecución de un programa.

1.3 SENTENCIA DE ASIGNACIÓN

Asigna el valor de la expresión que está a la derecha del signo $=$ a la variable que está a la izquierda. En el diagrama este signo puede ser reemplazado por \leftarrow ó $:=$.

Ejemplo: $NUM := 6 + R$ (siendo 6 una constante y R una variable, donde NUM tomará el valor de sumarle a 6 el contenido de la variable R).

1.4 OPERADORES ARITMETICOS

+ Suma
- Resta
* Multiplicación
/ División real
% Resto de la división entera

1.5 OPERADORES LÓGICOS y RELACIONALES

$=$ ó $==$ (Igual)
 $<>$ ó $!=$ (Distinto)
 $<$ (Menor)
 $>$ (Mayor)
 $<=$ ó $=<$ (Menor o igual)
 $>=$ ó $=>$ (Mayor o igual)
OR ó $||$ (es el **O** lógico)
AND ó $\&\&$ (es el **Y** lógico)
NOT ó $!=$ (es el **NO**)

EJEMPLO:

Realizar un algoritmo en diagrama de Chapin para resolver el siguiente problema:

Dados como datos el precio del kilowatt-hora, la lectura actual y la lectura anterior de un medidor; calcular el importe que una persona deberá abonar a la E.P.E.

Mostrar los datos y el resultado obtenido.

PRECIO, LECANT, LECACT	E
CONSUMO := LECACT - LECANT	
IMPORTE := CONSUMO * PRECIO	
PRECIO, LECANT, LECACT, IMPORTE	S

1.6 ESTRUCTURAS DE CONTROL

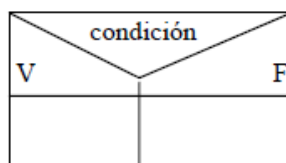
Hay tres estructuras básicas de control:

1. LA SECUENCIAL (vista en los ejemplos anteriores)
2. LA BIFURCACION O SELECCIÓN, que puede ser
 - 2.1 Simple ó
 - 2.2 Múltiple.
3. LA REPETICION O ITERACION, que puede ser:
 - 3.1 Con cantidad conocida de veces ó
 - 3.2 Con cantidad desconocida de veces.

1.6.2. LA BIFURCACION O SELECCIÓN

1.6.2.1 Bifurcación Simple:

Consiste en la selección entre dos caminos dependiendo de una condición. Esta sentencia es la sentencia IF (o SI en pseudocódigo), y gráficamente se expresa como sigue:

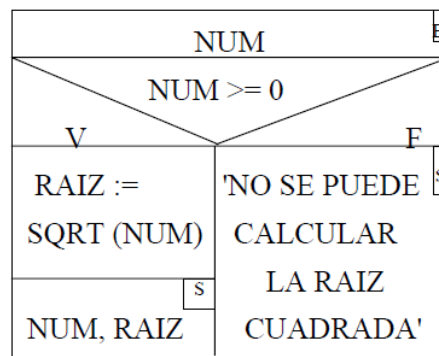


Pregunta o comparación

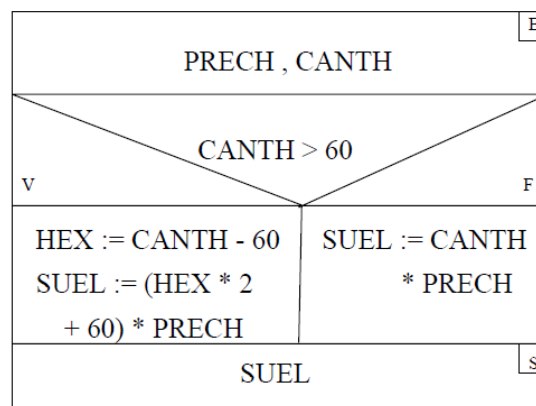
Decisión simple

EJEMPLOS:

- a) Dado un número real calcular, si es posible, su raíz cuadrada.

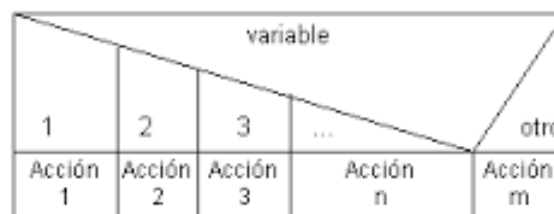


- b) A partir de los datos de Pago por hora y Cantidad de horas trabajadas calcular el Sueldo de un operario, sabiendo que si las horas trabajadas superan 60, las excedentes se pagan el doble.



1.6.2.2 LA BIFURCACION O SELECCIÓN Múltiple:

Si para optar por un camino a seguir no se depende de una sola condición, sino del valor que contenga un dato o expresión (el selector) podíamos utilizar la sentencia Select Case o **Switch**, que gráficamente se expresa como sigue:



EJEMPLO:

- a) Dado un número del 1 a 7 determinar el nombre del día de la semana que corresponde.

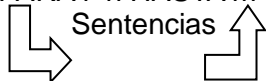
Leer (num)						
1	2	3	4	5	6	num
Mostrar('Lunes')	Mostrar('Martes')	Mostrar('Miercoles')	Mostrar('Jueves')	Mostrar('Viernes')	Mostrar('Sabado')	Mostrar('Domingo')

1.6.3 REPETICION O ITERACION:

1.6.3.1. CON CANTIDAD CONOCIDA DE VECES

Cuando una sentencia o un grupo de sentencias deben ejecutarse más de una vez utilizamos una estructura de repetición. Si sabemos qué cantidad de veces se van repetir, utilizamos la sentencia **PARA**; esta sentencia es conocida como la sentencia FOR, y se representa genéricamente como:

PARA $i=n$ HASTA m HACER:



A i la denominamos variable de control, n es el valor inicial y m el valor final.

La variable de control debe declararse como cualquier otra variable.

Donde dice **Sentencias** va una serie de pasos como cualquier algoritmo secuencial visto hasta ahora.

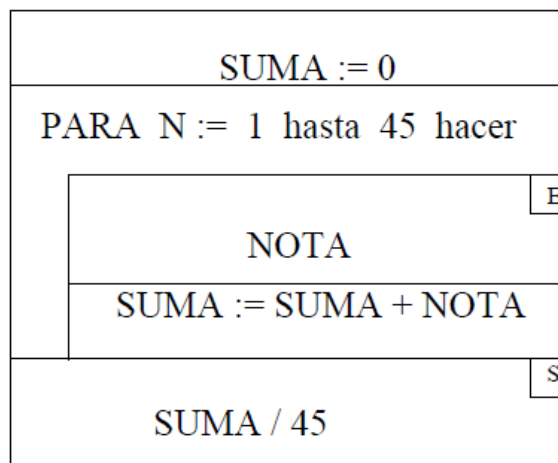
El bloque de sentencias se va a repetir tantas veces como valores haya entre n y m . En cada repetición, ciclo o iteración, la variable de control se incrementa automáticamente.

Si el valor de la variable de control es menor o igual que el valor final entonces se ejecutan una vez más las sentencias que constituyen el ciclo, se incrementa la variable de control y se vuelve a comparar.

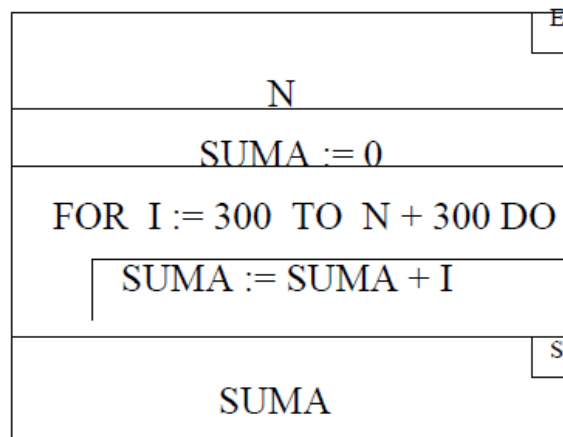
El proceso continúa hasta que la variable de control toma un valor mayor que el valor final, en cuyo caso termina el ciclo.

EJEMPLOS:

- Dadas las notas de un parcial de los 45 alumnos de un curso, se desea obtener la nota promedio del curso.



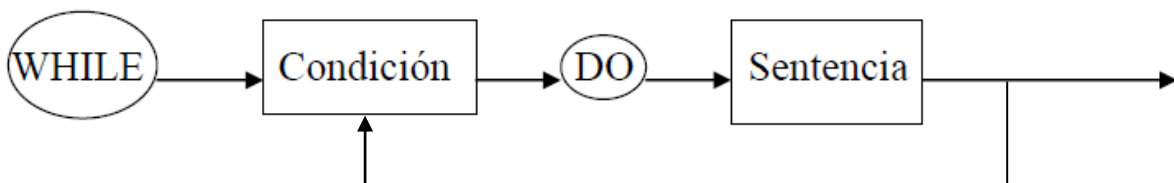
- b) Se desea obtener la suma de los N números naturales posteriores al número 300 inclusive.



1.6.3.2. CON CANTIDAD DESCONOCIDA DE VECES

Cuando una sentencia o un grupo de sentencias deben repetirse más de una vez, dependiendo de una condición, utilizamos la estructura MIENTRAS.

La sentencia Mientras (o sentencia WHILE DO), se expresa como sigue:



Donde dice Sentencia va una serie de pasos como cualquier algoritmo secuencial visto hasta ahora.

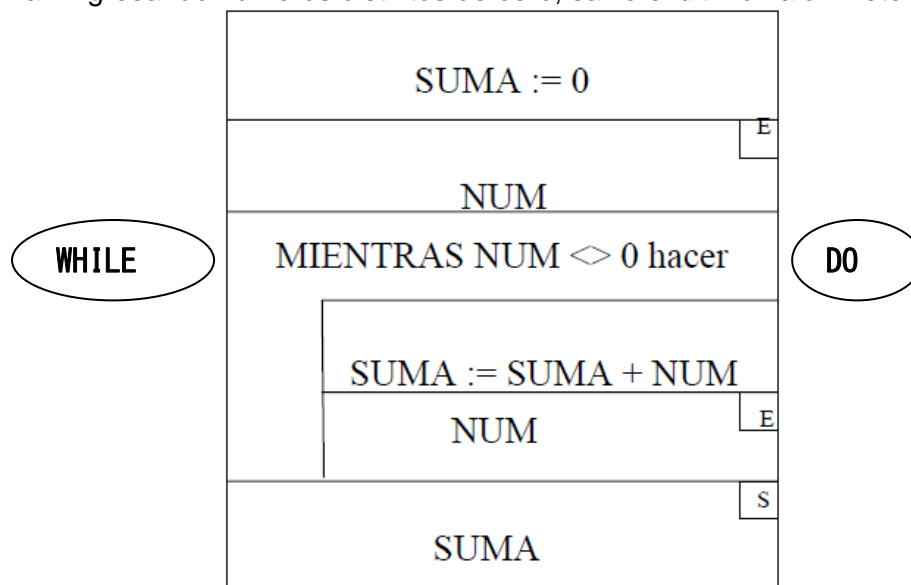
Mientras la Condición sea verdadera se ejecutan la o las sentencias del ciclo.

Cuando la condición sea falsa, el control pasa a la sentencia siguiente de la última que compone el ciclo.

Por lo tanto si la condición es falsa la primera vez, pueden no ejecutarse nunca las sentencias del ciclo de repetición.

EJEMPLOS:

- a) Se van ingresando números distintos de cero, salvo el último valor. Determinar su suma.

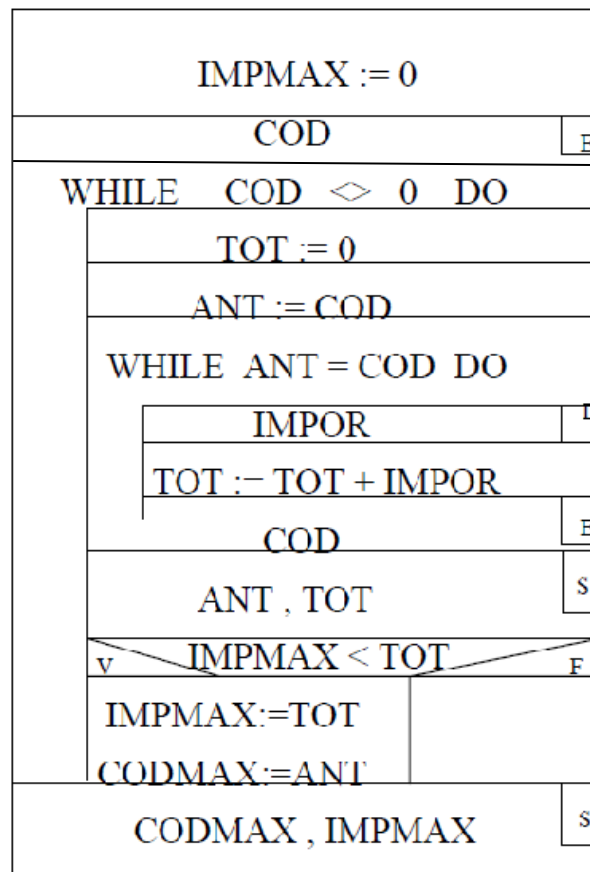


- b) Se desea saber el total de ventas de cada uno de los vendedores de una empresa. A tal fin se tienen como datos: el código de vendedor y el importe de cada una de las ventas; un vendedor puede haber realizado más de una venta. No se sabe la cantidad de vendedores que tiene la empresa ni la cantidad de ventas hechas por cada vendedor (un código de vendedor igual a cero es fin de datos).

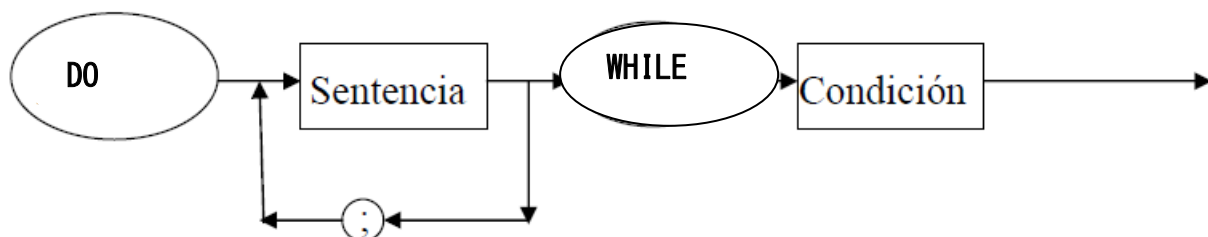
ESTOS DATOS ESTAN ORDENADOS POR CODIGO DE VENDEDOR.

Exhibir cada código de vendedor y su total correspondiente y al final, el código de vendedor con mayor importe vendido y dicho importe.

Resolverlo usando CORTE DE CONTROL.



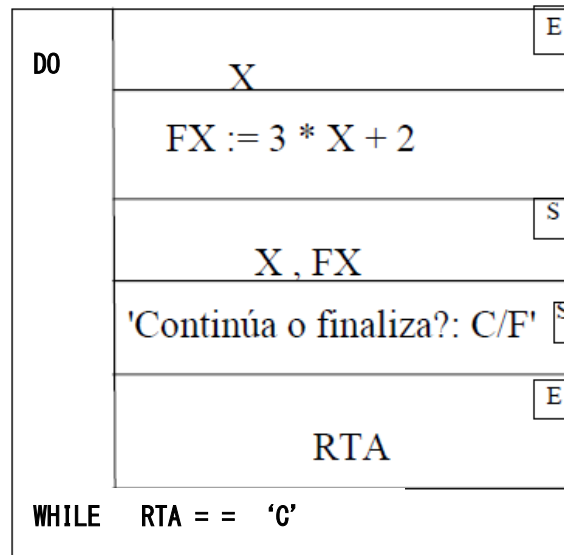
A diferencia del MIENTRAS, la sentencia HACER MIENTRAS (o DO WHILE) es la siguiente:



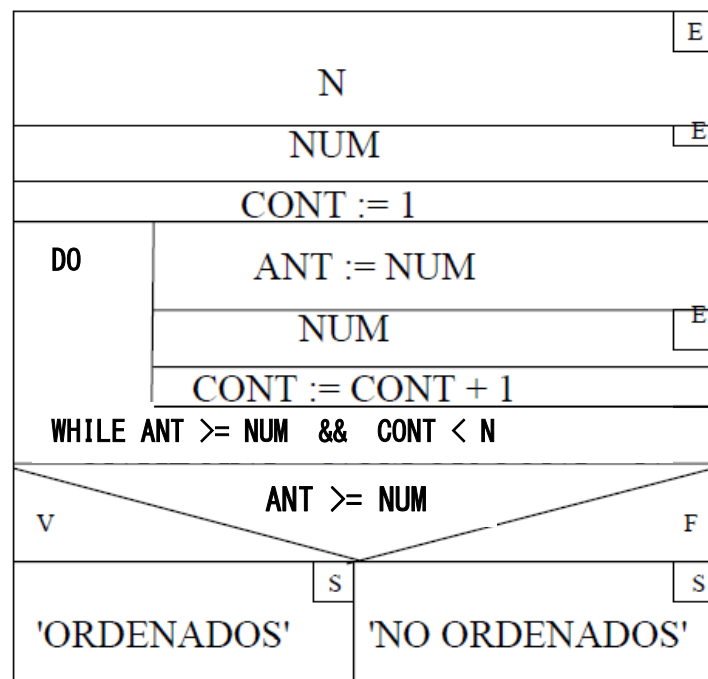
Repite la ejecución de la o las sentencias del ciclo hasta que la condición sea verdadera. Por lo tanto el ciclo **se ejecuta al menos una vez**, pues compara al final del mismo. Esta es la gran diferencia que tiene con la sentencia MIENTRAS (o WHILE).

EJEMPLOS:

- a) Evaluar y tabular la función $f(X) = 3X + 2$ para diferentes valores de X.



- b) Determinar si una lista de N números está ordenada de mayor a menor.



2. ARREGLOS

Hasta ahora hemos visto tipos de datos simples que pueden contener las variables:

- Números Enteros, por ej. $X = 123$
- Números Decimales, por ej. $TotalVta = 875423.75$
- Char, por ej. $Continua = 'S'$
- String, por ej. $Nombre = "Juan"$

Algunas veces es necesario almacenar y referenciar variables como un grupo.

Estas estructuras de datos nos permiten escribir programas para manipular datos más fácilmente.

Un arreglo (ARRAY) es un grupo de elementos a los que se les da un nombre común.

Se accede a cada elemento por su posición dentro del grupo.

Para algunos lenguajes, todos los elementos de un ARRAY deben ser del mismo tipo, para otros no.

2.1 ARRAYS UNIDIMENSIONALES (Vectores):

Un ARRAY unidimensional es un tipo de datos estructurados compuesto de un número fijo de componentes del mismo tipo, con cada componente directamente accesible mediante el índice.

La forma general de declaración es:

nombre = new Array(índice superior)

Donde *nombre* es el nombre de la variable (cualquier identificador válido); índice superior indican el rango del ARRAY.

Por ejemplo $NUM2 = new Array(20)$

Un elemento particular del ARRAY se representa por el nombre de la variable seguido de un índice encerrado entre corchetes, que indica la posición que ocupa ese elemento:

Nombre[índice]

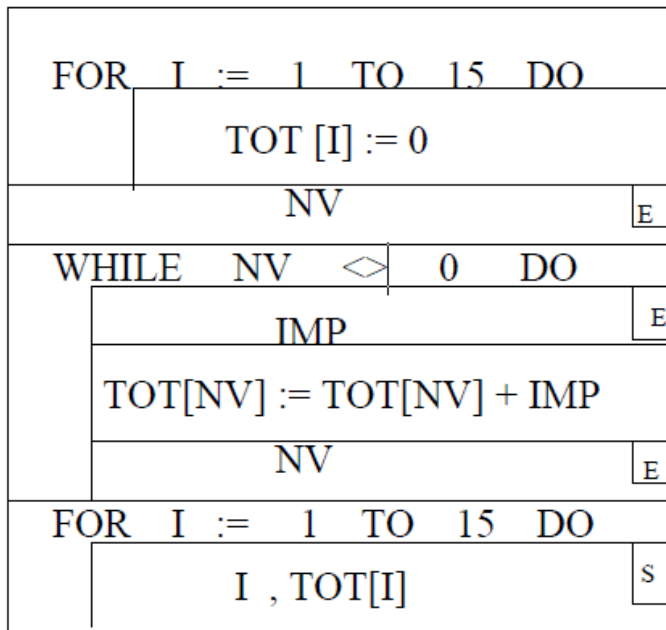
Por ejemplo $NUM[5]$ contiene un valor almacenado en la posición 5 del arreglo.

EJEMPLO:

Se tienen como datos NRO. DE VENDEDOR e IMPORTE de cada una de las ventas realizadas por cada uno de los vendedores de una empresa.

No se sabe cuántas ventas se han realizado por lo que un NRO. DE VENDEDOR igual a cero indica fin de datos. Los vendedores están numerados del 1 al 15.

Se desea obtener el total vendido por cada vendedor.



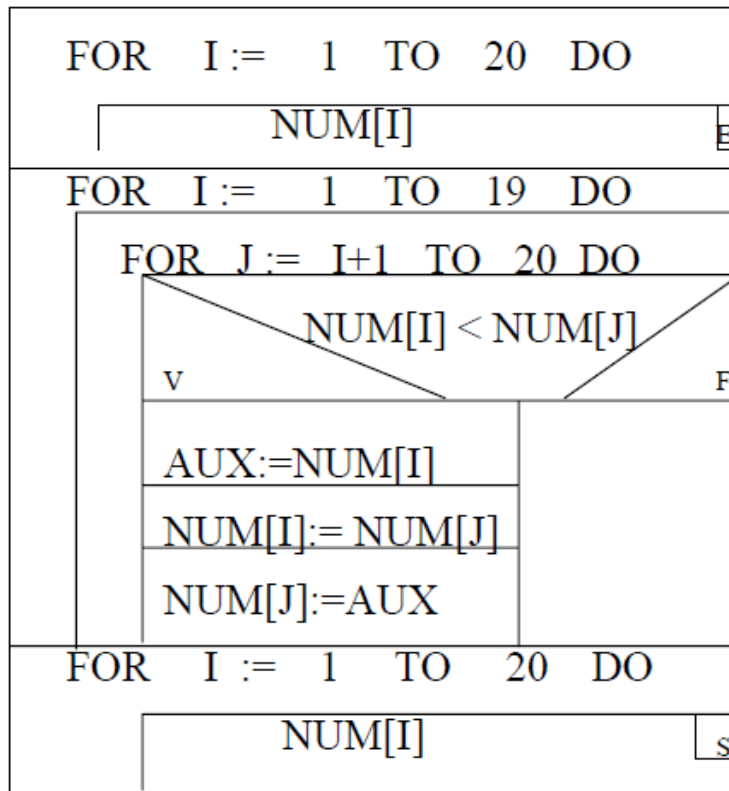
2.2 ORDENAMIENTO DE UN ARREGLO UNIDIMENSIONAL

Para ordenar los elementos de un arreglo, ya sea en forma creciente o decreciente, existen varios métodos; nosotros veremos el método conocido por Falso Burbuja, que sin ser el más rápido en algunos casos, es uno de los más fáciles de entender.

Este método consiste en comparar cada uno de los elementos del arreglo con todos los restantes y cambiarlos o no entre sí de acuerdo a la forma en que estemos ordenando el arreglo.

EJEMPLO:

Ingresar 20 números enteros en un arreglo y luego mostrarlos ordenados en forma decreciente.



2.3 BUSQUEDA DE UN VALOR EN UN ARREGLO ORDENADO

Si tenemos un arreglo ordenado, ya sea en forma creciente o decreciente y tenemos que buscar un valor dentro del mismo, la forma más rápida para hacerlo es por medio de la Búsqueda Dicotómica.

Para explicar este método, supongamos que los elementos del arreglo están ordenados en forma creciente procediéndose entonces, de la siguiente manera:

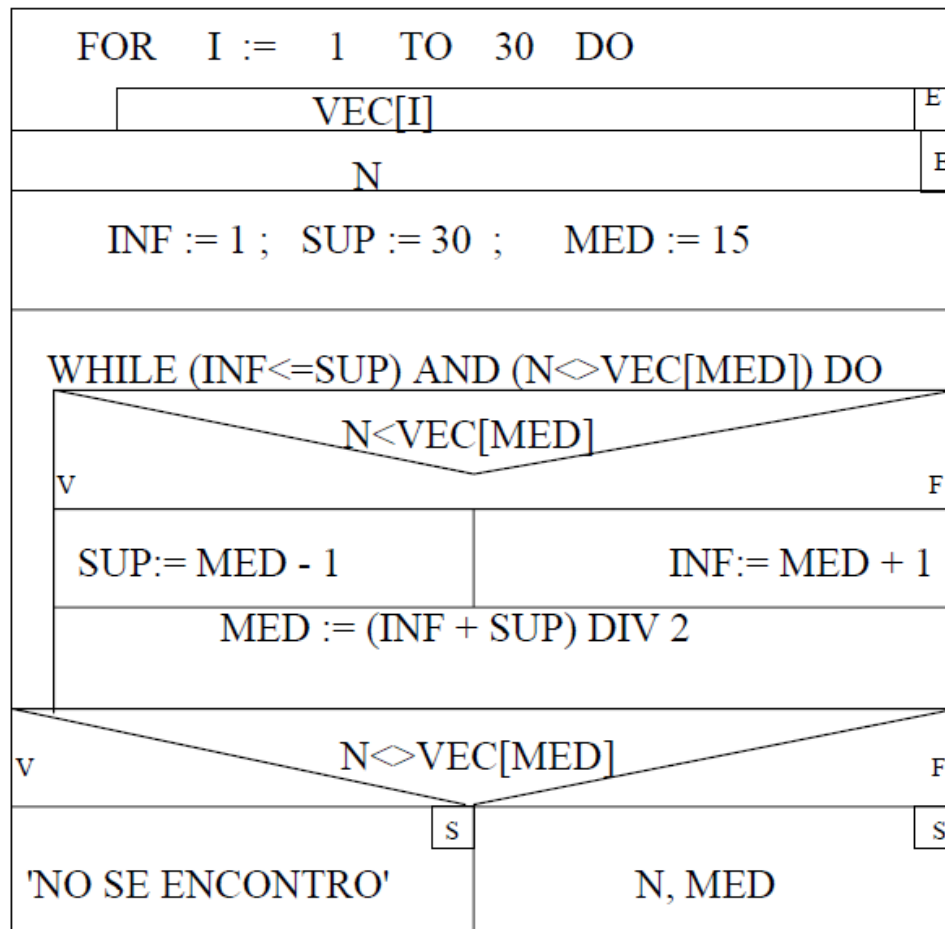
Primero se fijan los extremos del intervalo de búsqueda, que serán las posiciones que ocupan el primero y el último elemento del arreglo respectivamente.

Luego se obtiene la posición media de dicho intervalo y se compara el valor a buscar con el contenido del arreglo en esa posición media; si el valor es mayor que dicho elemento se continúa la búsqueda en la segunda mitad del arreglo; si por el contrario el valor es menor, se continúa en la primera mitad y se halla nuevamente la posición media del nuevo intervalo de búsqueda.

El proceso se repite hasta que se encuentra el valor a buscar, o bien hasta que no haya más intervalo de búsqueda, lo que significará que el valor buscado no se encuentra en el arreglo.

EJEMPLO:

Se ingresan 30 números enteros ordenados en forma creciente y un valor N. Se desea saber si el valor N coincide con algún elemento del arreglo; si es así, indicar la posición en que fue encontrado, sino exhibir cartel aclaratorio.



2.4 INTERCALACION DE ARRAYS ORDENADOS:

La intercalación de arrays es un proceso que consiste en tomar dos arrays ordenados, ya sea ambos en forma creciente o en forma decreciente, y obtener un array ordenado de igual manera que los dados.

Para explicar este método, supongamos tener dos arrays ordenados en forma creciente cuyas dimensiones son n y m respectivamente, se obtendrá un nuevo array de dimensión (n + m), también ordenado en forma creciente, procediéndose de la siguiente manera:

Se comparan los primeros elementos de cada array, se selecciona el menor y se coloca en la primera posición del nuevo array.

Luego se compara el elemento que quedó sin colocar, con el elemento que sigue del array cuyo elemento fue utilizado poniéndose el menor en la siguiente posición del nuevo array.

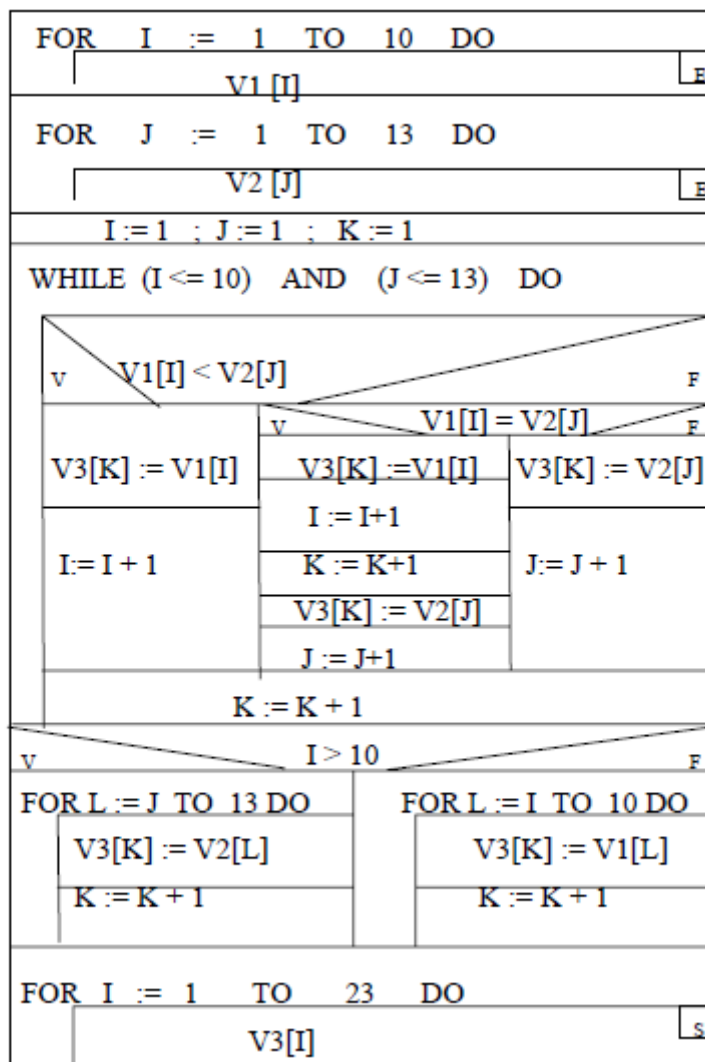
Si dos elementos comparados son iguales, se coloca uno y luego el otro en el nuevo array y se pasan a comparar los elementos siguientes de cada uno de los dos arrays dados.

El proceso continúa hasta que todos los elementos del primero o segundo array hayan sido utilizados, en cuyo caso los elementos restantes se agregan como están en el nuevo array.

EJEMPLO:

Ingresa 10 números enteros ordenados en forma creciente en un array; luego 13 números enteros, también ordenados en forma creciente en otro array.

Obtener por intercalación, un tercer array ordenado en forma creciente, y luego mostrarlo.



2.5 ARRAYS BIDIMENSIONALES (matrices)

Un array bidimensional es también un tipo de datos estructurados compuesto de un número fijo de componentes del mismo tipo, accediéndose a cada una de ellas mediante dos subíndices, el primero indica el número de fila y el segundo el número de columna en donde está dicho elemento.

La forma general de declaración es:

Nombre = new Array(índ sup índ sup)

En particular en JavaScript se declara un array dentro de otro, de la forma:

F = new Array(C1 = new Array(2) , C2 = new Array(2) , C3 = new Array(2));

En este caso declaramos una matriz de 3x3.

Un elemento determinado del ARRAY se representa por:

Nombre[índice fila , índice columna]

En particular en JavaScript accedemos a un elemento de la forma: *F[i][j]*

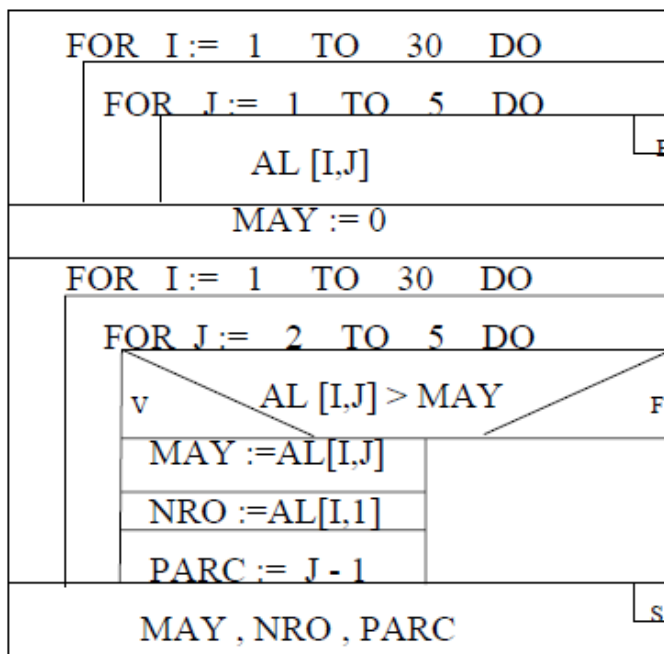
EJEMPLO:

Se tienen los siguientes datos de 30 alumnos de primer año:

NRO. ALUMNO (entero)

NOTA 1, NOTA 2, NOTA 3, NOTA 4 (enteras)

Se desea cargar los datos en un arreglo bidimensional y luego exhibir el nro. de alumno que tuvo la mejor nota de todas y en qué parcial la obtuvo.



2.6 ORDENAMIENTO DE UN ARRAY BIDIMENSIONAL

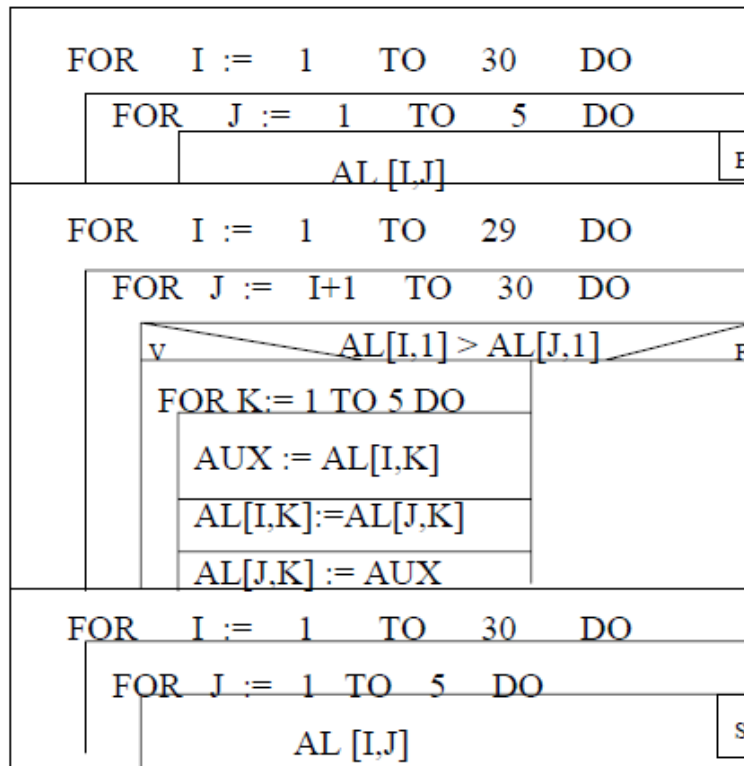
Tenemos que tener en cuenta que los datos que se cargan en un arreglo bidimensional responden, por fila o por columna, a una persona, a una agencia, a una comisión, etc.; por lo tanto, lo mismo que en una matriz matemática, no podemos cambiar un elemento por otro de lugar; lo que sí podemos hacer es cambiar una fila o una columna por otra. Luego, un

ordenamiento en un arreglo bidimensional tiene sentido, si se pide que se ordene el mismo de manera tal que los elementos de una fila o columna queden ordenados en forma creciente o decreciente.

EJEMPLO:

Se tienen los mismos datos del ejercicio anterior.

Se desea un listado de los mismos ordenados en forma creciente por NRO. DE ALUMNO.



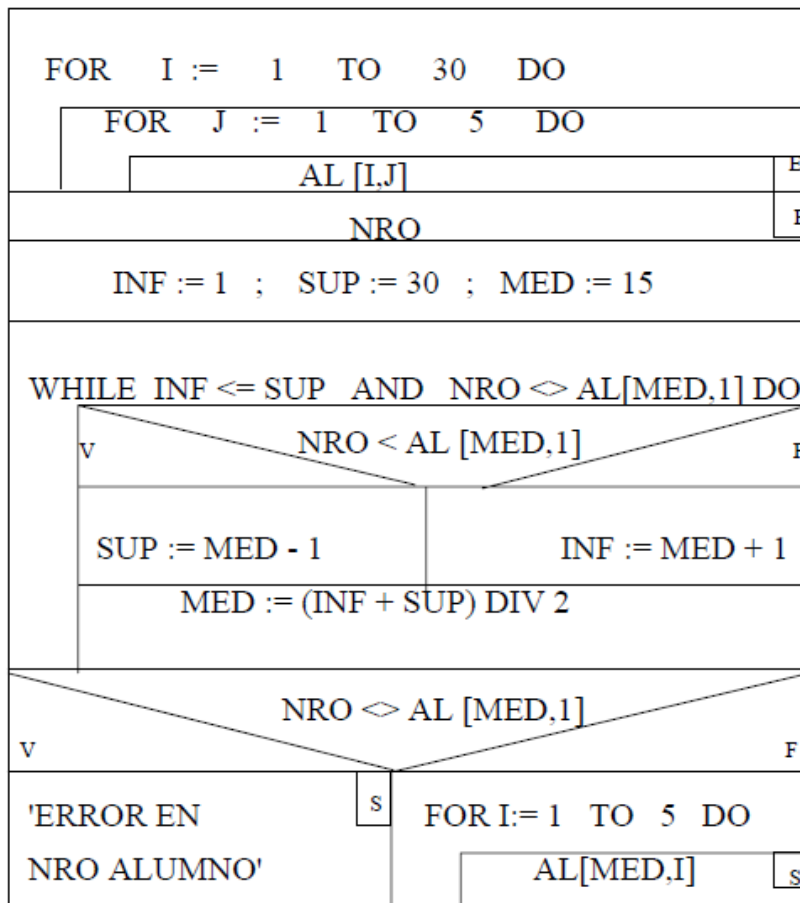
2.7 BUSQUEDA EN UN ARREGLO BIDIMENSIONAL

Si los elementos de un arreglo bidimensional están ordenados por una columna o una fila, ya sea en forma creciente o decreciente, se puede buscar un valor dentro de dicha fila o columna por medio de la búsqueda dicotómica.

EJEMPLO:

Se tiene los datos de los 30 alumnos del ejercicio anterior ya ordenados en forma creciente por NRO DE ALUMNO.

Se desea ingresar un NRO DE ALUMNO y buscarlo por medio de la búsqueda dicotómica dentro del arreglo. Si se encuentra, dar el nro. de alumno junto con las notas de los 4 parciales; sino exhibir cartel aclaratorio.



3. FUNCIONES

3.1 INTRODUCCIÓN

Es frecuente en programación que un grupo de sentencias deba repetirse varias veces con distintos datos, o sea que debamos escribirlas varias veces. Los lenguajes de programación permiten escribirlas una sola vez bajo la forma de funciones y usarlas las veces que sea necesario.

Además, si un grupo de sentencias realiza una tarea específica, puede estar justificado el aislarlas formando una función, aunque se las use una sola vez.

3.2 DEFINICIÓN

Una función es un grupo de sentencias dentro de un programa que forman un bloque (un subprograma) que realiza un número determinado de operaciones sobre un conjunto de argumentos dado y puede devolver un valor, o realizar una serie de pasos y volver al flujo del programa que llamó a la función.

Cada vez que se llama a la función, se transfiere el control al bloque de sentencias definidas por esa función.

Después que las sentencias han sido ejecutadas, el control vuelve a la sentencia en que fue llamada la función.

La invocación de una función es de la forma:

nombre (argumento1, argumento2,...)

donde *nombre*, que es el nombre de la función, es un identificador válido y es donde vuelve el resultado; y cada argumento puede ser cualquier variable válida, constante o expresión.

Esta invocación debe ser asignada a una variable, formar parte de una expresión asignada a una variable, puede estar en un MOSTRAR o en un SI.

Una definición de función tiene la forma:

FUNCTION *nombre (declaración de parámetros)*

```
{  
    declaración de identificadores locales  
    sentencias ejecutables  
}
```

donde *nombre* es el nombre de la función; *declaración de parámetros* contiene los parámetros (cada uno de los cuales debe ser un identificador válido) de la función.

El orden de la lista de parámetros es el orden de correspondencia de dichos parámetros con la lista de argumentos de la llamada. Por lo tanto en número de parámetros y de argumentos debe ser el mismo, y el tipo de los que se correspondan debe coincidir.

Si una función no necesita pasar parámetros, entonces se omiten las listas de parámetros y de argumentos.

En la declaración de identificadores locales van las declaraciones de las variables que no son parámetros, usadas por la función.

Las sentencias propiamente dichas de la función (cuerpo de la función) van entre { }, como ocurre con el programa principal.

Las definiciones de función siempre tienen lugar después de la sección de declaración de variables, pero antes del cuerpo del programa en el cual es invocada dicha función.

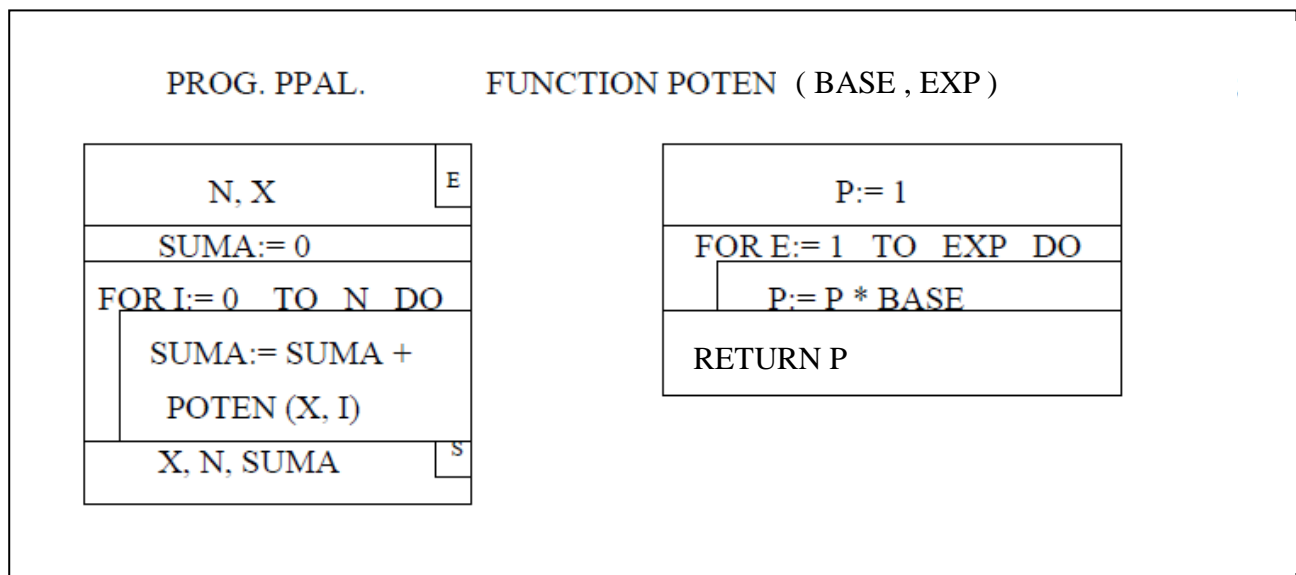
EJEMPLO:

Escribir un programa que calcule la expresión: $\sum_{i=0}^n x^i$

para cualquier par de valores n y x.

Para evaluar cada uno de los términos de la sumatoria, crear y utilizar una función llamada POTEN, que tenga como parámetros la base x y el exponente i.

Exhibir: x, n y el resultado de la sumatoria.



Las variables declaradas en un programa que contenga funciones, son válidas (pueden utilizarse) en cualquier parte del programa, también dentro de las funciones; a estas variables se las denomina **variables globales**.

Si durante la ejecución de una función se usa y se altera el valor de una variable global, al terminar la función, el valor de la variable corresponde al último valor y no al original que tenía antes de que se ejecutara dicha función.

3.3 CORRESPONDENCIA ARGUMENTO-PARAMETRO

Como ya hemos visto, cada vez que se llama a una función, se establece una correspondencia entre cada argumento y su parámetro.

Esta llamada puede ser: **por valor o por referencia**.

LLAMADA POR VALOR:

Es la asignación del valor del argumento a su parámetro. El parámetro es entonces una variable independiente, de nueva creación que recibe el valor del argumento al comienzo de la ejecución de la función.

Puesto que parámetro y argumento son variables independientes, cualquier cambio que se produzca en el parámetro durante la ejecución de la función no tiene efecto en el valor del argumento.

Por lo tanto cuando se utiliza la llamada por valor, es imposible la devolución de valores al punto de llamada por medio de los parámetros; pues al terminar la ejecución de la función, la variable parámetro se destruye y el valor que contenía se pierde.

LLAMADA POR REFERENCIA:

En el caso de que se requiera que el valor de una variable sea modificado por la función invocada, debe hacerse el paso de parámetro por referencia, por medio del cual el subprograma invocado tiene acceso a la dirección en que se guarda el valor a modificar.

No implica la creación de una posición aparte de memoria para el parámetro, sino más bien produce el paso de la dirección de memoria donde se almacena el valor del argumento.

El parámetro, en efecto, viene a ser simplemente otro nombre (o el mismo) para la misma dirección ya creada para el valor del argumento.

Lenguajes como C o Pascal y sus derivados manejan pasaje por referencia. En nuestro caso, **en JavaScript sólo utilizamos pasaje por valor**.

3.4 RECURSIVIDAD

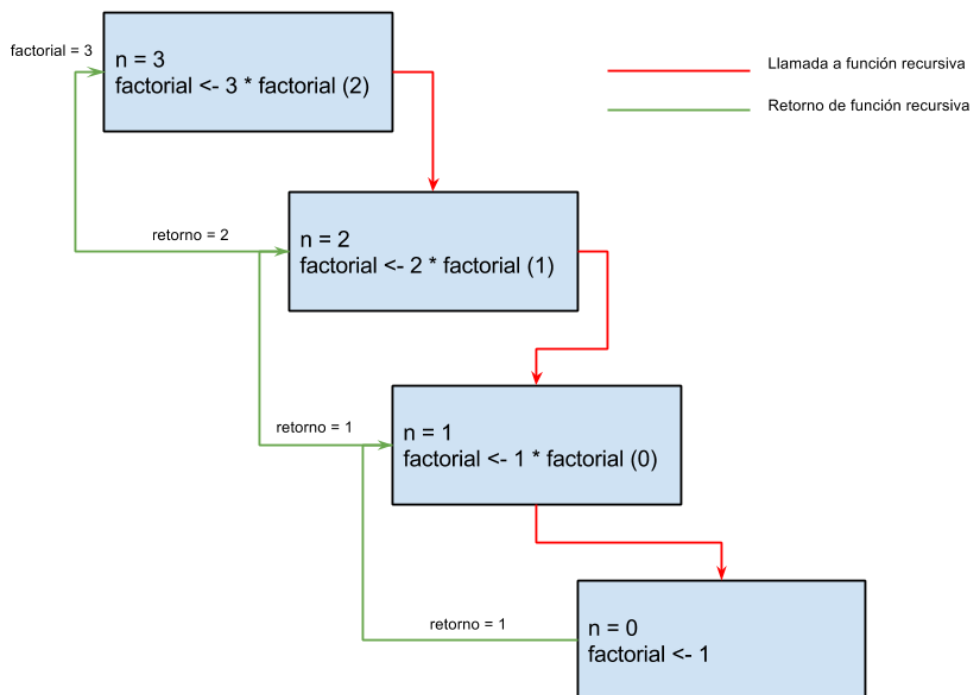
A una función le es permitido no sólo invocar a otra función, sino también invocarse a sí misma. Una invocación de éste tipo se dice que es *recursiva*.

La *función recursiva* más utilizada como ejemplo es la que calcula el factorial de un número entero no negativo, partiendo de las siguientes definiciones:

factorial (0) = 1

factorial (n) = n*factorial(n-1), para n>0.

Por ejemplo, si n = 3, la función realiza los siguientes pasos:



- 1°. $\text{factorial}(3) = 3 * \text{factorial}(2)$ Se invoca a si misma y crea una tercera variable cuyo nombre es n y su valor es igual a 2.
- 2°. $\text{factorial}(2) = 2 * \text{factorial}(1)$. Se invoca a si misma y crea una cuarta variable cuyo nombre es n y su valor es igual a 1.
- 3°. $\text{factorial}(1) = 1 * \text{factorial}(0)$ Se invoca a si misma y crea una quinta variable cuyo nombre es n y su valor es igual a 0.
- 4°. Como $\text{factorial}(0) = 1$, con éste valor se regresa a completar la invocación: $\text{factorial}(1) = 1 * 1$, por lo que $\text{factorial}(1) := 1$
- 5°. Con $\text{factorial}(1) = 1$, se regresa a completar: $\text{factorial}(2) := 2 * 1$, por lo que $\text{factorial}(2) := 2$
- 6°. Con $\text{factorial}(2) = 2$, se regresa a completar : $\text{factorial}(3) := 3 * 2$, por lo que $\text{factorial}(3) = 6$ y éste será el valor que la función factorial devolverá al módulo que la haya invocado con un valor de parámetro igual a 3.

La función queda de la siguiente manera:

```
factorial(n)  
{  
    int resp=0;  
    if(n<=1) return 1;  
    resp = factorial(n-1)*n;  
    return resp;  
}
```

3.5 FUNCIONES PREDEFINIDAS

La mayoría de los lenguajes de programación cuentan con un set de funciones ya desarrolladas y disponible para el programador, donde sólo hay que invocarlas y pasarles los parámetros adecuados para obtener su valor.

A nivel de algoritmos, vamos a trabajar con una serie de funciones que suponemos ya programadas y disponibles, en especial, las referidas a la manipulación de texto y matemáticas. Son extraídas del lenguaje JavaScript. Ellas son:

Nombre de la Función	Descripción	Ejemplo	Resultado
asin, acos, atan	Funciones trigonométricas inversas	asin(1)	1.57
exp, log	Exponenciación y logaritmo, base E	log(E)	1
ceil	Devuelve el entero más pequeño mayor o igual al argumento	ceil(-2.7)	-2
floor	Devuelve el entero más grande menor o igual al argumento	floor(-2.7)	-3
round	Devuelve el entero más cercano o igual al argumento	round(-2.7)	-3
min, max	Devuelve el menor (o mayor) de sus dos argumentos	min(2, 4)	2
pow	Exponenciación, siendo el primer argumento la base y el segundo el exponente	pow(2, 3)	8
sqrt	Raíz cuadrada	sqrt(25)	5
random	Genera un valor aleatorio comprendido entre 0 y 1.	random()	Ej. 0.7345
toFixed(digitos)	Devuelve el número original con tantos decimales como los indicados por el parámetro digitos y realiza los redondeos necesarios.	nro = 4564.34567 nro.toFixed(2)	4564.35
length	Retorna la cantidad de caracteres de un String.	cadena = "hijo" cadena.length	4
charAt(pos)	Retorna el carácter del índice especificado. Comienzan a numerarse de la posición cero.	nombre = "Juan" nombre.charAt(0);	'J'

substring (posini, posfin)	Retorna un string extraído de otro, desde el carácter 'posini' hasta el 'posfin'-1 (cuidado que comienza en cero).	cadena1 = "programar" cadena3=cadena1.substring(2, 5);	cadena3 contendrá "ogr"
toUpperCase()	Convierte todos los caracteres del string que invoca el método a mayúsculas.	cad=" minúscula" cad=cad.toUpperCase();	cad=" MINUSCULA"
toLowerCase()	Convierte todos los caracteres del string que invoca el método a minúsculas.	cad=" MAYUSCULA" cad=cad.toLowerCase();	cad=" mayuscula"
split(separador)	Convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado. Con esta función se pueden extraer fácilmente las letras que forman una palabra.	mensaje = "Hola Mundo, soy una cadena de texto!"; palabras = mensaje.split(" "); palabra = "Hola"; letras = palabra.split("");	palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"] letras = ["H", "o", "l", "a"]