Introducción a la Programación Estructurada

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.

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.

Grupo de instrucciones que 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, ingles 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.

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.

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 aún cuando la ejecución es perfecta.

Algoritmos cualitativos y algoritmos cuantitativos

Un algoritmo es cualitativo cuando en sus pasos o instrucciones no están involucrados cálculos numéricos.

Los algoritmos cuantitativos involucran cálculos numéricos. Por ejemplo, solución de una ecuación de segundo grado, solución de un factorial.

Técnicas de diagramación

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.

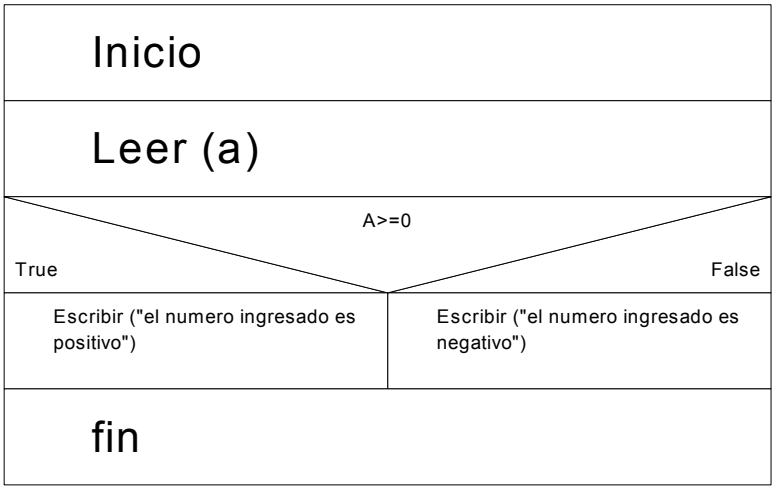
Para la representación de un algoritmo, antes de ser convertido a lenguaje de programación, se utilizan algunos métodos de representación gráfica o numérica. Los métodos más conocidos son:

- Diagramación libre (Diagramas de flujo)
- Diagramas Nassi-Shneiderman o Chapin
- Pseudocódigo
- Lenguaje español
- Fórmulas

Diagrama Nassi-Schneiderman

El diagrama N-S de Nassi-Schneiderman (conocido también como Chapin) es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se escriben en cajas sucesivas y como en los diagramas de flujo se pueden escribir diferentes acciones en una caja.

Ejemplo de diagrama de Nassi-Schneiderman (N-S)



Componentes de un algoritmo

Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en le orden en el que han de ser ejecutadas. Los pasos en esta primera descripción de actividades deberán ser refinados añadiendo detalles a los mismo 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
¿Cuantos 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 en el primer diseño 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
Pase 3 Salida por pantallas de base, altura y superficie.

sup ← 0
Leer(b) Leer(a)
sup ← b * a
Escribir(“Base: ” b)
Escribir(“Altura: ” a)
Escribir(“Superficie: ” sup)

Diagrama de Chapin

La representación en este diagrama se realiza a través de los siguientes componentes:

- Declaración de variables, constantes e inicialización.
- Ingreso de valores
- Proceso de valores con el objetivo de obtener resultados en el que se puede combinar diferentes componentes (Estructuras de Asignación, Estructuras Secuenciales, Selectivas o de Decisión, Repetitivas)
- Salida de resultados obtenidos

La primer componente consiste en una caja en la cual se van a declarar todas las variables y/o constantes que se utilizarán en el resto del algoritmo a las cuales se le asigna un valor inicial.

v1 ← 0, v2 ←0 , c1 ← 0

El ingreso de valores se representa escribiendo dentro del bloque la palabra leer y entre paréntesis el valor que se ingresa.

Leer(A)

Dentro del proceso tenemos las siguientes componentes:

Estructura de Asignación

Las operaciones de asignación son el modo de darles valores a una variable. La operación de asignación se representa con el símbolo u operador ←.

El formato general de una operación de asignación es:

Nombre de la variable ← expresión

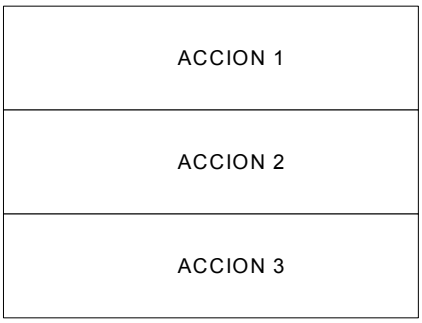
Ejemplos:
A←5
total←3+6
N←0
N←N+1
M←8<5
Y←'CALLE'

Los tipos de asignación pueden ser:

Asignación simple.- cuando a una variable le corresponde nada más que un dato
Asignación aritmética- cuando a una variable le corresponde el resultado de una expresión aritmética.
Asignación lógica.- cuando a una variable le corresponde el resultado de una expresión lógica

Estructura Secuencial

Se caracterizan porque una acción se ejecuta detrás de otra. El flujo del programa coincide con el orden físico en el que se ha ido poniendo las instrucciones.



Ejemplos.

1- Declarar dos constantes e inicializarlas con los valores 10 y 20 respectivamente, sumarlas y mostrar el resultado.

```
c1 <-- 10, c2 <-- 20, s <-- 0
s <-- c1 + c2
Escribir (s)
```

2- Idem anterior pero con los valores a sumar dinámicos.

```
s <-- 0
Leer(n1, n2)
s <-- n1 + n2
Escribir (s)
```

3- Dado el importe de una factura calcular el valor correspondiente al IVA.

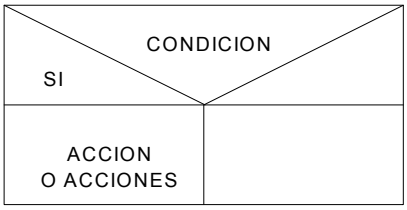
```
iva <-- 0
Leer(imp)
iva <-- imp * 21 / 100
Escribir ("El IVA es: " iva)
```

Estructuras Selectivas o de Decisión

Para la aplicación se debe tener en cuenta las expresiones lógicas cuyo valor es verdadero o falso, se denomina también expresiones booleanas.

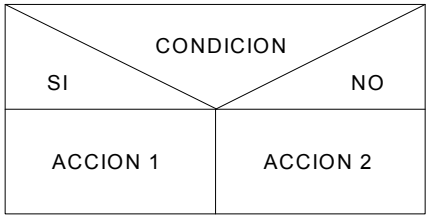
En conclusión, la estructuras selectivas o decisión o comparación o pregunta, se ejecutan unas acciones u otras según se cumpla o no una determinada condición; pueden ser: simples, dobles, o múltiples.

Simples.- Se evalúa la condición y si esta da como resultado verdad se ejecuta una determinada acción o grupo de acciones, en caso contrario se salta dicho grupo de acciones.



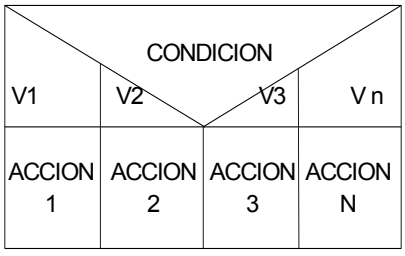
SI <condición> **entonces**
<Acción o acciones>
fin_si

Dobles.- Cuando el resultado de evaluar la condición es verdad se ejecuta una determinada acción o grupo de acciones y si el resultado es falso otra acción o grupo de acciones diferentes.



SI <condición> **entonces**
<Acción 1>
SI_NO
<Acción 2>
fin_si

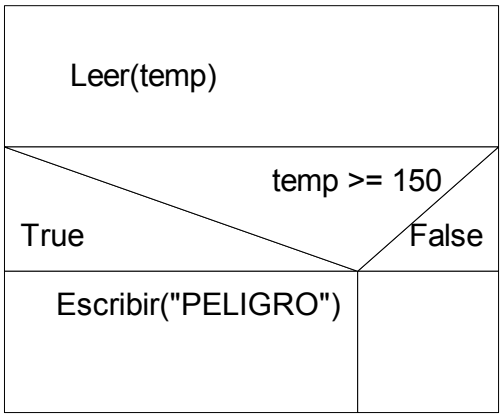
Múltiples.- Se ejecutan unas acciones u otras según el resultado que se obtenga al evaluar una expresión.



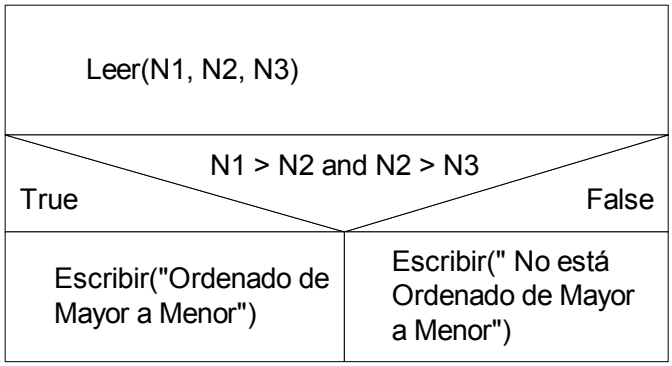
Según_sea <expresion> **hacer**
    <lista1>:<Acción 1>
    <lista2>:<Acción 2>
    .....
SI_NO
    <Acción n>
fin_segun

Ejemplos.

1- La temperatura crítica de una caldera es 150 grados centígrados. Determinar e informar mediante el mensaje PELIGRO si la temperatura medida en un momento dado es mayor o igual que la crítica.



2- Determinar si un conjunto de tres valores numéricos dados están ordenados de mayor a menor.



3- Ingresar un número del 1 al 7 y devolver el día de la semana correspondiente al mismo.

Leer(N1)						
N1						
1	2	3	4	5	6	7
Escribir("D")	Escribir("L")	Escribir("M")	Escribir("M")	Escribir("J")	Escribir("V")	Escribir("S")

Estructuras Repetitivas

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces o denominadas bucles, y se llama iteración al hecho de repetir la ejecución de una secuencia de acciones.

Tales opciones repetitivas se denominan bucles o lazos. La acción (o acciones) que se repiten en un bucle se denomina iteración. Las dos principales preguntas a realizarse en el diseño de un bucle son:

¿Qué contiene el bucle?
¿Cuántas veces se debe repetir?

Cuando se utiliza un bucle para sumar una lista de números, se necesita saber cuantos números se han de sumar. Para ello necesitaremos conocer algún medio para detener el bucle. En el ejemplo siguiente usaremos la técnica de solicitar al usuario el número que se desea, por ejemplo, **N**. Existen dos procedimientos para contar el número de iteraciones usar la variable TOTAL que se inicializa a la cantidad de números que se desea y a continuación se decrementa en uno cada vez que el bucle se repite (este procedimiento añade una acción más al cuerpo del bucle: (TOTAL←TOTAL-1) o bien inicializar la variable TOTAL e 0 o en 1, e ir incrementando en uno a cada iteración hasta llegar al número deseado.

Estructura Repetitiva For (Para)

En muchas ocasiones se conoce de antemano el número de veces que se desea ejecutar las acciones de un bucle. En estos casos en los que el número de iteraciones es fijo se deberá usar la estructura **FOR** (para)

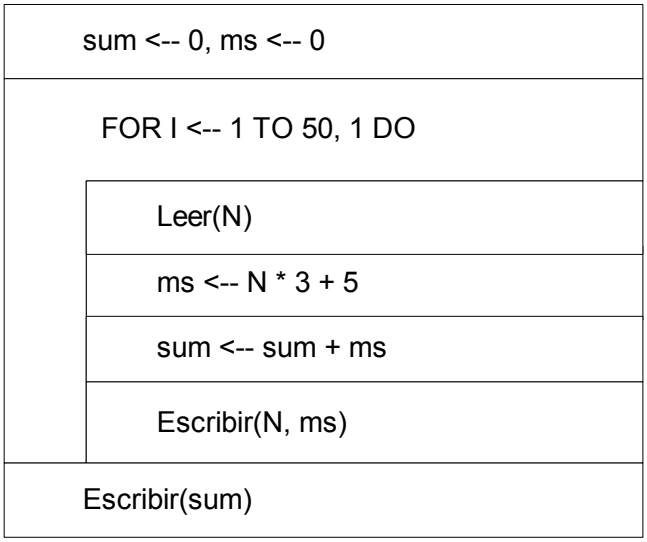
La estructura FOR ejecuta las acciones del cuerpo de un bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

```
Para Variable=val. Inicial, Val. Final, Pasos
    ACCIONES A REPETIR
Siguiente (fin_para)
```

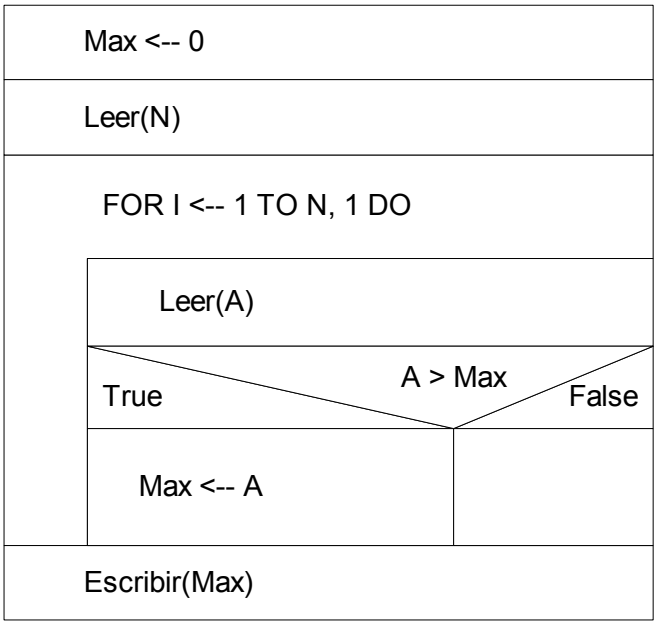
PARA V = VI HASTA VF PASOS VP
<Acción 1>
.....
<Acción N>
SIGUIENTE V
NOTA:
V- Variable
VI- Valor Inicial
VF- Valor Final
VP- Valor de secuencia de incremento o decremento

Ejemplos.

1- Leer sucesivamente 50 valores numéricos. A cada valor multiplicarlo por tres y sumarle 5. Informar el resultado de dicha expresión junto al número que lo origina. Al final exhibir el valor acumulado de los 50 valores calculados.

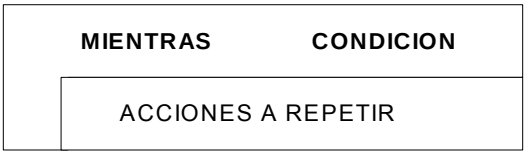


2- Hallar e informar el máximo de un conjunto de N números (N >= 2)



Estructura Repetitiva While (Mientras)

La estructura repetitiva **WHILE** (mientras o hacer mientras) es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta la instrucción WHILE, lo primero que se verifica es la condición (una expresión booleana). Si se evalúa **falsa**, el programa finaliza el bucle y sigue la instrucción a continuación en el cuerpo del algoritmo, si la expresión booleana es **verdadera**, entonces se ejecuta el cuerpo del bucle, y de nuevo se evalúa la expresión booleana, este proceso se repite una y otra vez **mientras** el resultado de la expresión booleana (condición) sea **verdadera.**



MIENTRAS condición **HACER**
<Acción 1>
<Acción 2>
.....
<Acción N>
FIN_MIENTRAS

Ejemplos

1- Se dispone de un conjunto de números distintos de cero salvo el último valor. Determinar e informar la cantidad de números que lo forman.

```
Acum <-- 0
Leer(N)
WHILE N <> 0 DO
    Acum <-- Acum + 1
    Leer(N)
Escribir(Acum)
```

2- Se tiene un conjunto de números formados de la siguiente manera: primero todos los positivos, luego un valor nulo, a continuación todos los negativos y finalmente otro valor nulo. Calcular y exhibir el promedio de los valores positivos y el promedio de los negativos.

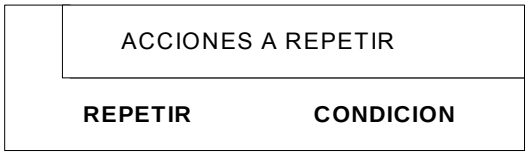
```
FOR I <-- 1 TO 2, 1 DO
    sum <-- 0, cont <-- 0, prom <-- 0
    Leer(N)
    WHILE N <> 0 DO
        sum <-- sum + N
        cont <-- cont + 1
        Leer(N)
    prom <-- sum / cont
    Escribir(prom)
```

Estructura Repetitiva Repeat (Repetir)

Existen muchas situaciones en las que se desea que un bucle se ejecute por lo menos una vez antes de comprobar la condición de repetición. En la estructura mientras si el valor de la expresión booleana es inicialmente falsa, el cuerpo del bucle no se ejecutará; por ello se necesitan otros tipos de estructuras repetitivas.

La estructura repeat (repetir) se ejecuta hasta que se cumpla la una condición determinada que se comprueba al final del bucle.

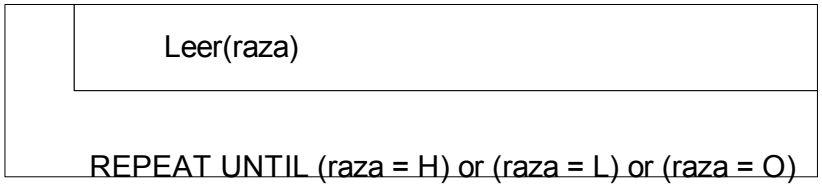
Este bucle se **repite** siempre y cuando el valor de la condición booleana sea **falsa**, lo opuesto de la sentencia WHILE.



REPETIR
<Acción 1>
<Acción 2>
.....
<Acción N>
HASTA_QUE condición

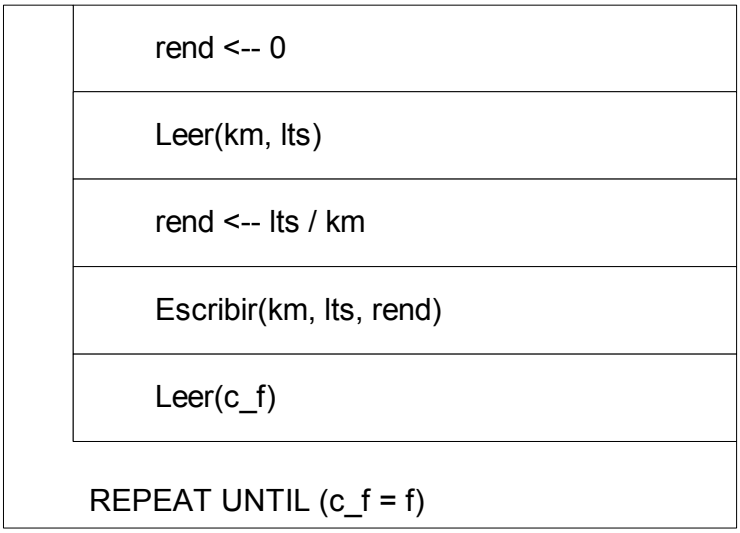
Ejemplos

1- Validar que una raza ingresada sea H, L u O.

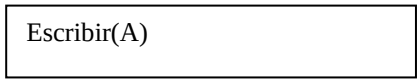


2- Se leen pares de valores. El primero es la cantidad de kilómetros recorridos y el segundo el consumo de nafta. Para cada par calcular el rendimiento (litros / kilómetros). Emitir un listado con título y valor.

Indicar si se introducirá un nuevo par de datos o se dará por finalizado el proceso.



La salida se representa escribiendo dentro del bloque la palabra escribir y entre paréntesis lo que se quiere mostrar (o sea el resultad obtenido)



Funciones y Procedimientos

Si un grupo de tareas dentro de un programa realizan una tarea específica, pueden transformarse en un nuevo programa al cual se llama desde el anterior para realizar dicha labor. Y no solo es eso, a este nuevo programa se lo puede llamar desde otros programas, para evitar escribir repetidas veces las mismas tareas. Este nuevo programa, al cual se accede llamándolo desde otro se lo denomina Subprograma.

Hay dos tipos de subprogramas: Funciones Y Procedimientos.

Funciones

Una función es un grupo de tareas dentro de un programa que forman un número determinado de operaciones sobre un conjunto de argumentos dados y devuelve **UN SOLO VALOR.**

Cada vez que es llamada una función, se transfiere el control al broque de operaciones definidas por esta función.

Después de que las operaciones han sido ejecutadas, el control vuelve al programa donde fue llamada la función.

La invocación de una función es de la forma:

nombre(argumnto1, argumento2,...)

donde nombre es el nombre de la función y donde vuelve el valor y cada argumento puede ser cualquier variable válida, constante o expresión.

El orden de la lista de argumentos en la llamada a la función es el mismo que el orden que tienen los argumentos en la definición de la función.

Si una función no necesita de argumentos, estos se omiten de la declaración.

Ejemplo.

Escribir un programa que calcule la expresión $\sum x^i$, sumatoria de N elementos. Para evaluar cada uno de los términos de la sumatoria, crear y utilizar una función llamada Poten, que tenga como argumentos la base x y el exponencial i. Exhibir: x, n y el resultado de la sumatoria.

```
suma <-- 0
Leer(N, X)
FOR I <-- 1 TO N, 1 DO
    suma <-- suma + Poten(X, I)
Escribir(X, I, suma)
```

FUNCTION Poten(Base, Exp)

```
P <-- 1
FOR E <-- 1 TO Exp, 1 DO
    P <-- P * Base
Poten <-- P
```

Procedimientos

Los procedimientos son subprogramas similares a las funciones pero con dos diferencia importantes.

La llamada a un procedimientos es igual a la de la función:

nombre(argumnto1, argumento2,...)

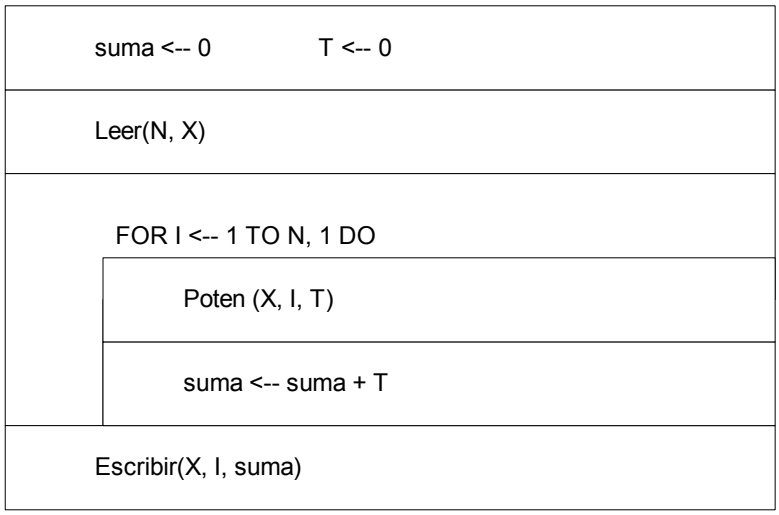
La primer diferencia es que la llamada debe estar sola, es decir, no puede estar formando parte de expresiones, ni asignada a una variable, ni en un while, ni en un if.

La segunda es que con el procedimiento no se devuelve un solo valor al punto de llamada, sino que se puede devolver cualquier número de valores por medio de los argumentos.

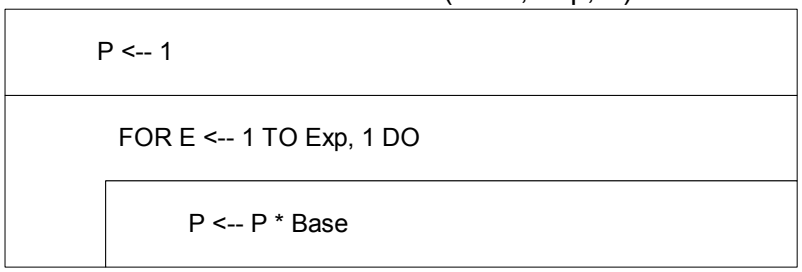
La devolución de valores se produce a través de los argumentos.

Ejemplo

1- Realizar el ejemplo anterior pero en lugar de la función llamar a un procedimiento de nombre Poten.



PROCEDURE Poten(Base, Exp, P)



Llamadas por valor y por referencia

Cada vez que se llama a un subprograma, se establece la correspondencia entre los argumentos de la llamada con los declarados en la definición del mismo.

Esta llamada puede ser Por Valor o Por Referencia.

Llamada por Valor

Se realiza una copia del valor del argumento de la llamada al argumento del subprograma, es decir, son independientes. Por esto es que cualquier cambio en el argumento que se produce en el subprograma no tiene efecto en el programa que se realiza la llamada.

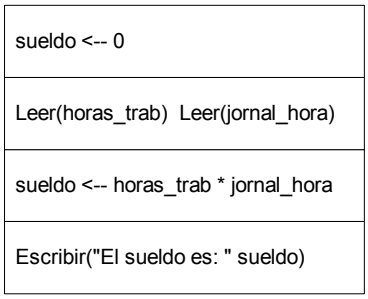
Llamada por Referencia

Al revés que las llamadas por valor el argumento de la llamada es el mismo que el del subprograma, por lo tanto cualquier modificación producida en los argumentos se verá reflejada cuando se desee usar esos mismos argumentos en el programa principal.

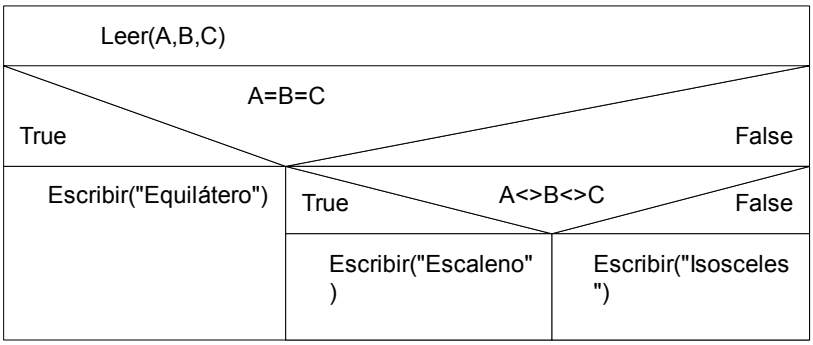
En este tipo de llamada, los argumentos van precedidos por la palabra VAR.

Ejercicios

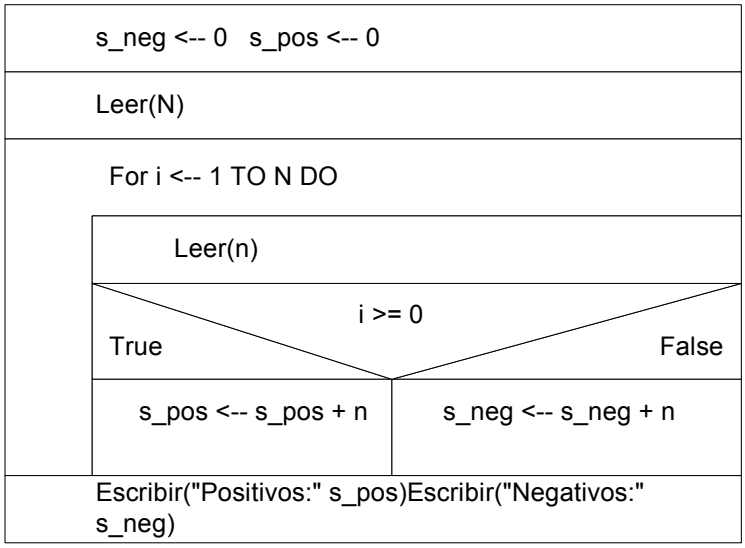
1- Calcular el sueldo de un operario conociendo la cantidad de horas que trabajó en el mes y el jornal horario.



2- Se leen los tres lados de un triángulo. Determinar e informar si el mismo es equilátero (3 lados iguales), isósceles (2 lados iguales) o escaleno (3 lados distintos).



3- Ingresar un conjunto de N valores (cada uno puede ser positivo o negativo). Obtener y exhibir la suma de unos y otros.


s_neg <-- 0 s_pos <-- 0
Leer(N)
For i <-- 1 TO N DO
Leer(n)
i >= 0
True
False
s_pos <-- s_pos + n
s_neg <-- s_neg + n
Escribir("Positivos:" s_pos)Escribir("Negativos:"
s_neg)

Indice