Elementos de Programación

UNIDAD 5. ITERACION

INDICE

UNIDAD 5 - Iteración

Objetivos: *Aplicar en la solución de los problemas las estructuras iterativas, en sus variantes. Realizar programas con algoritmos de búsqueda de máximos y mínimos. Comprender los conceptos de variables acumuladoras y contadoras. Utilizar las estructuras de iteración para validar los datos de entrada.*

1. Iteración o repetición

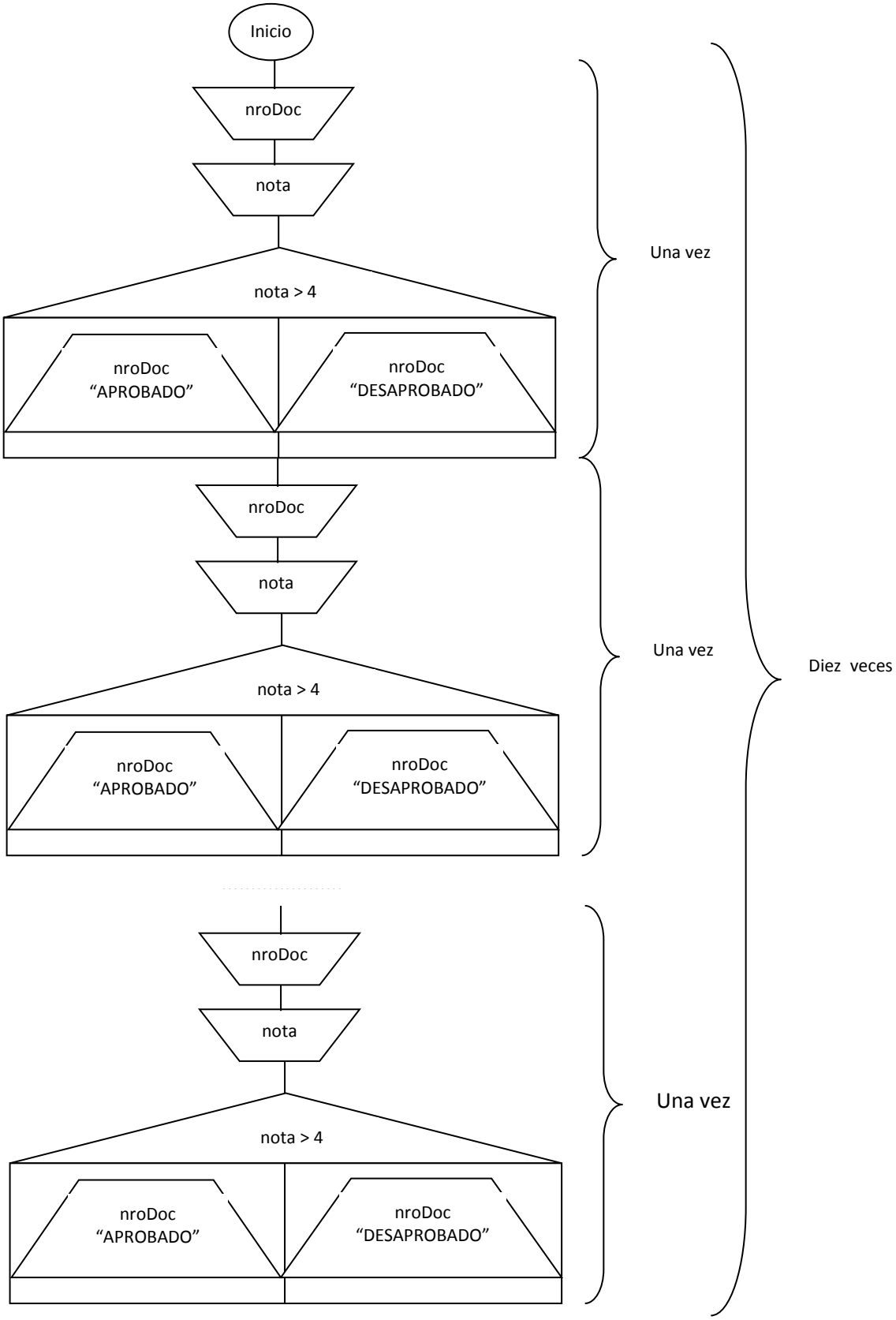
En los programas vistos hasta ahora cada instrucción se ejecutaba una sola vez, en el orden que aparece en el programa. Pero a veces es necesario repetir la ejecución de un grupo de instrucciones, para lo cual se van a utilizar las estructuras de repetición o iteración.

Existen dos tipos de ciclos de repetición:

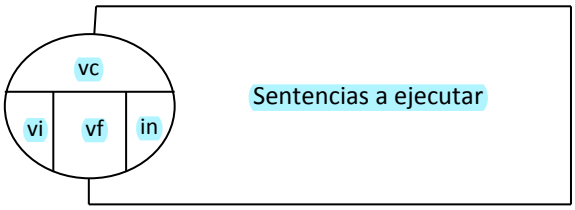
a) ITERACIÓN DEFINIDA: Cuando se conoce de antemano la cantidad "exacta" de veces que se debe repetir ese proceso o grupo de sentencias.
b) ITERACIÓN CONDICIONADA: Cuando NO se conoce la cantidad de iteraciones a efectuar, es decir, que la repetición depende del cumplimiento de cierta condición.

2. Estructuras de iteración definida

Se desarrolla el siguiente ejemplo: Ingresar número de documento y nota de los 10 alumnos de un curso, informar por cada uno el número de documento y la leyenda "APROBADO" o "DESAPROBADO" si la nota es mayor a 4 o no.


Inicio
nroDoc
nota
nota > 4
nroDoc
“APROBADO”
nroDoc
“DESAPROBADO”
Una vez
nroDoc
nota
nota > 4
nroDoc
“APROBADO”
nroDoc
“DESAPROBADO”
Una vez
nroDoc
nota
nota > 4
nroDoc
“APROBADO”
nroDoc
“DESAPROBADO”
Una vez
Diez veces

Analizando el diagrama se observa que un grupo de instrucciones se repiten 10 veces. Para eliminar las repeticiones se debe escribir las instrucciones repetidas una sola vez y efectuar un mecanismo

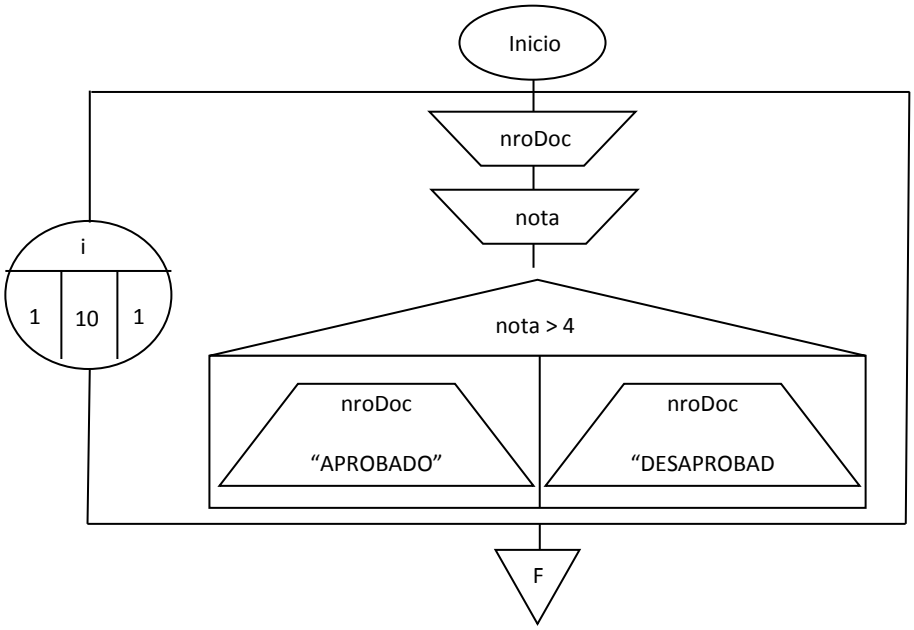


para realizar la repetición 10 veces. Para ello, se debe agregar en el diagrama el ciclo de repetición definida:

Donde:

- **vc**: es la variable de control, ya que con ella se controla el ciclo.
- **vi**: es el valor inicial que toma la variable de control.
- **vf**: es el valor final que debe tomar la variable.
- **in**: incremento que se le aplica a la variable cada vez que se ejecuta el ciclo.

Funcionamiento: Se asigna a la variable de control el valor inicial, se compara con el valor final, si es menor o igual que éste, se ejecutan las sentencias dentro del ciclo, luego incrementa a la variable con el valor del incremento, si sigue siendo menor o igual al valor final realiza otra ejecución y así hasta que llegue a superar el valor final.



Se aplica esta estructura al problema anterior:

El ciclo de iteración definida no siempre necesariamente debe iniciar en 1 ni ser creciente, por ejemplo, para repetir 10 veces un proceso también se puede:

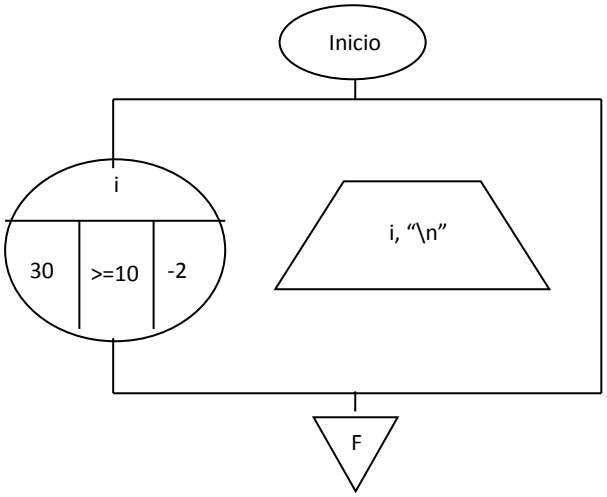
- Poner como valor inicial el 10
- Poner como valor final la condición > 0
- Poner -1 como el valor de incremento, generando de esta forma un decremento

Observe entonces que en realidad el valor final no es un número sino una condición, que se denomina condición de permanencia dentro del ciclo. Si esa condición se cumple entonces el ciclo

se repite. Si no se escribe el operador de comparación se toma por defecto el <= (menor o igual) en cualquier otro caso se debe escribir el operador de comparación.

Tampoco el incremento o decremento debe ser siempre 1 o -1 sino que puede ser cualquier otro valor que modifique la variable de control.

Ejemplo: Realizar un programa para mostrar los números pares entre 30 y 10 en forma descendente.



3. Contadores y acumuladores

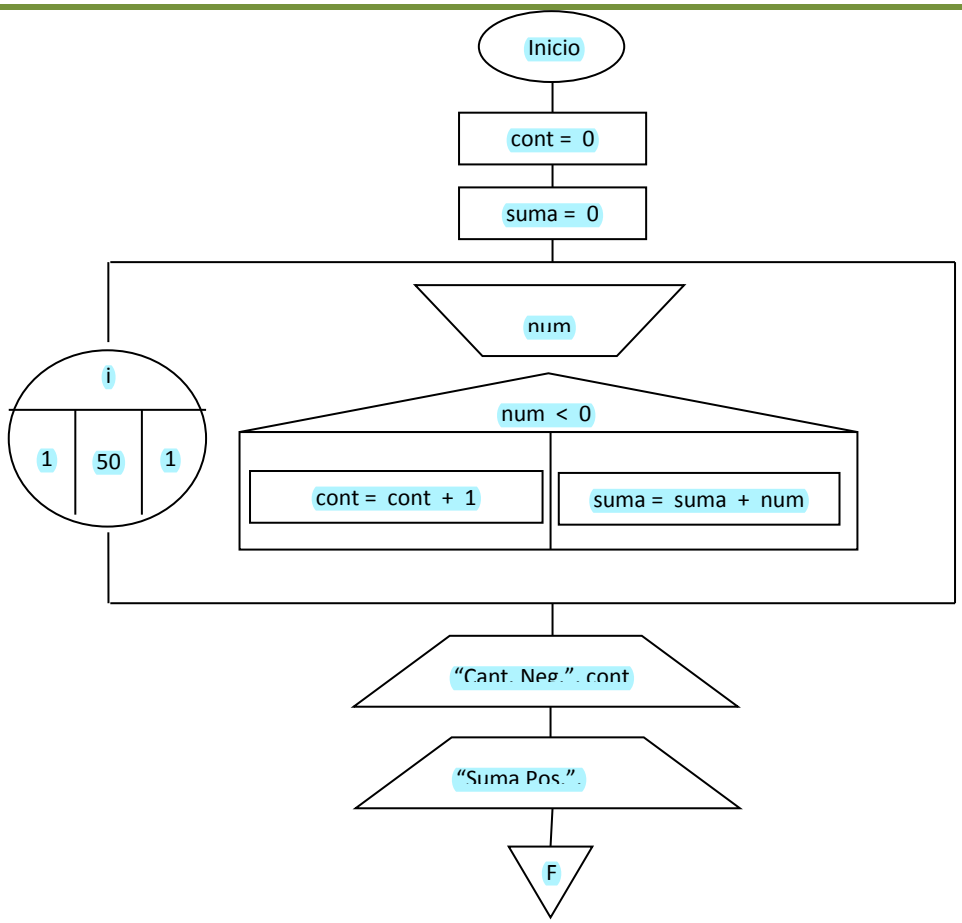
Un contador es una variable en la memoria que se incrementará en una unidad cada vez que se ejecute el proceso. El contador se utiliza para llevar la cuenta de determinadas acciones que se pueden solicitar durante la resolución de un problema. Hay que realizar la inicialización del contador o contadores. La inicialización consiste en poner el valor inicial de la variable que representa al contador (generalmente con el valor 0).

Un acumulador es una variable que suma sobre sí misma un conjunto de valores, para de esta manera tener la suma de todos ellos en una sola variable. De igual forma, se pueden efectuar decrementos en un acumulador.

La diferencia entre un contador y un acumulador es que mientras el primero va aumentando de uno en uno, el acumulador va aumentando o decrementando en una cantidad variable.

Ejemplo: Ingresar 50 números, informar la cantidad de números negativos y la sumatoria de los positivos.

Se deben ingresar los números en un ciclo de repetición definido. Por cada número se tiene que consultar si es menor que cero o no. Si es menor que cero se cuenta utilizando un contador (incrementando el contador en uno), si no se suma en un acumulador (sumarle al acumulador ese número).



4. Lenguaje c: estructuras de iteración definida

El ciclo de repetición definida se codifica en lenguaje C con el siguiente formato:

```
for (expresión de inicialización; expresión de verificación; expresión de incremento)
{
        sentencia1;
        sentencia2;
        ……..
}
```

Las llaves son opcionales de haber una sola sentencia.

Ejemplo: Se escribe en lenguaje C el Ejemplo anterior que ingresa 50 números, cuenta y acumula:

```
#include <stdio.h>
int main( )
{
        int cont, suma, i, num;
        cont = 0;
        suma = 0;
        for ( i = 1; i <= 50; i++)
        {
              printf("\n Ingrese un numero: ");
              scanf("%d", &num);
              if (num < 0)
                    cont = cont + 1;
              else
                    suma = suma + num;
        }
```

```
    printf("La cantidad de numeros negativos es %d \n", cont);
    printf("La sumatoria de los numeros positivos es %d \n",
suma);
    return 0;
}
```

Para incrementar el contador, en lugar de cont = cont + 1 se podría usar el operador de autoincremento cont++. En el uso del acumulador, se podría usar el operador de acumulación suma += num.

OBSERVACIONES IMPORTANTES:

- La prueba de la condición de final se efectúa siempre al inicio del bucle.
- El valor de la variable que actúa como índice, en el lenguaje C, puede ser modificado dentro del ciclo, pero se recomienda NO hacerlo ya que se modifica el comportamiento de la estructura de repetición definida.
- Finalizado el for, el valor del índice queda con el valor de la última ejecución realizada más el valor del incremento, o sea el valor con el cual no pudo ejecutar otra pasada.
- La expresión del incremento puede ser del tipo in = in + x, puede contener variables que no figuran como parámetros del for.
- Tener la precaución de "no" colocar un ; (punto y coma) luego de los paréntesis del for, lo cual hace ignorar el bucle, generando un ciclo vacío.
- Pueden omitirse una o todas las expresiones del for, pero NO los ";"
    a) Si se omite la primera expresión, NO SE ASIGNA VALOR INICIAL dentro del ciclo, pero puede asignarse antes:

    Ejemplo:

```
j = 1;
suma = 0;
for ( ; j <=10 ; j++ )
      suma += j;    // Suma los números del 1 al 10.
```

    b) Si se omite la segunda, se genera un ciclo INDEFINIDO, porque al no estar la expresión 2, se establece que la prueba siempre será verdadera y por lo tanto nunca saldrá del ciclo.

    Ejemplo: `for (j=1; ; j++)`

    c) Si se omite la tercera, se puede realizar el incremento dentro del ciclo for. NO SE RECOMIENDA realizar esta operación ya que es poco clara. Siempre el mejor utilizar todas las expresiones del for.

    Ejemplo

```
j  = 1;
suma = 0;
for (; j <=10 ;)
{
      suma += j; // Suma los números del 1 al 10.
      j++;
}
```

5. Máximos y mínimos

Es un **problema** común querer encontrar el mayor o el menor valor de una determinada serie de valores. Si se quiere encontrar el mayor valor, se debe pensar igual a un trabajo manual con una pila de hojas donde cada una tiene un valor numérico. Tomo el primero y lo retengo en la mano izq. (variable **mayor**), tomo al segundo con la mano derecha (variable **dato**) y lo comparo con el de la izquierda, si es más grande descarto el de la mano izquierda y paso el de la derecha a la izquierda; se procede igual con los restantes valores, quedando finalmente el mayor en la mano izquierda, o sea, en la variable mayor.

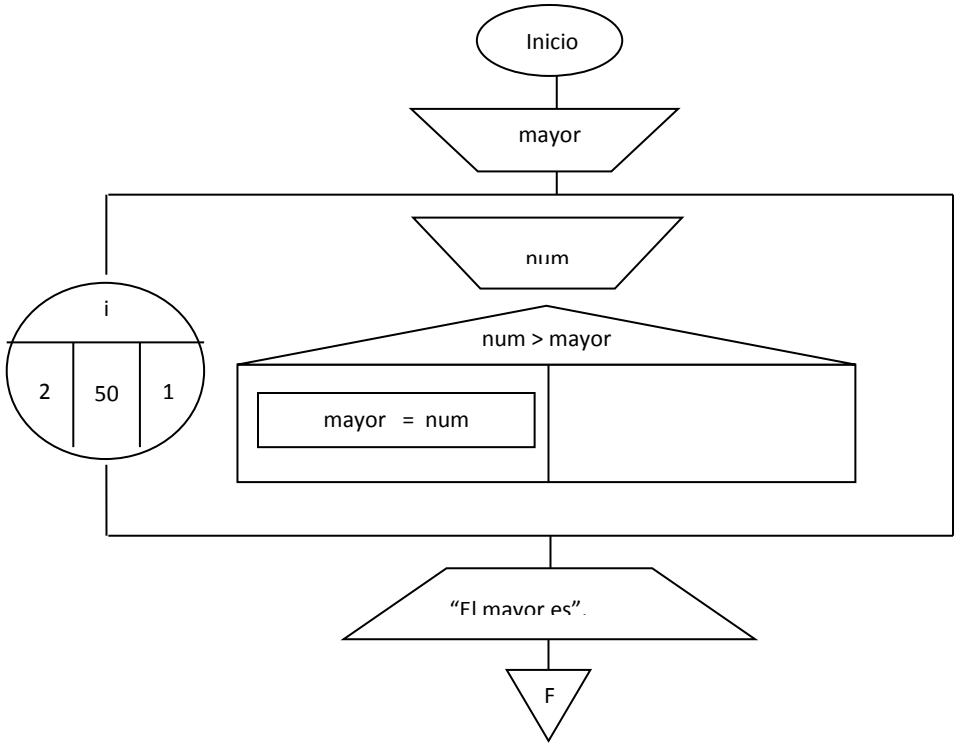
Es decir, que para calcular un máximo de un conjunto de valores se toma el primer valor como referencia, siendo este momentáneamente el más grande. Luego al ingresar otro número se compara con el que se tiene de referencia. Si el nuevo número es mayor al de referencia entonces se reemplaza por el recién ingresado, de esta forma siempre quedará como valor de referencia el más grande y se comparará con cada uno del número ingresados. Si se desea buscar el valor mínimo el procedimiento es exactamente igual, pero se compara si el número recién ingresado es menor al de referencia en cuyo caso se lo reemplaza.

Ejemplo: Ingresar 50 valores numéricos y determinar e informar el mayor.

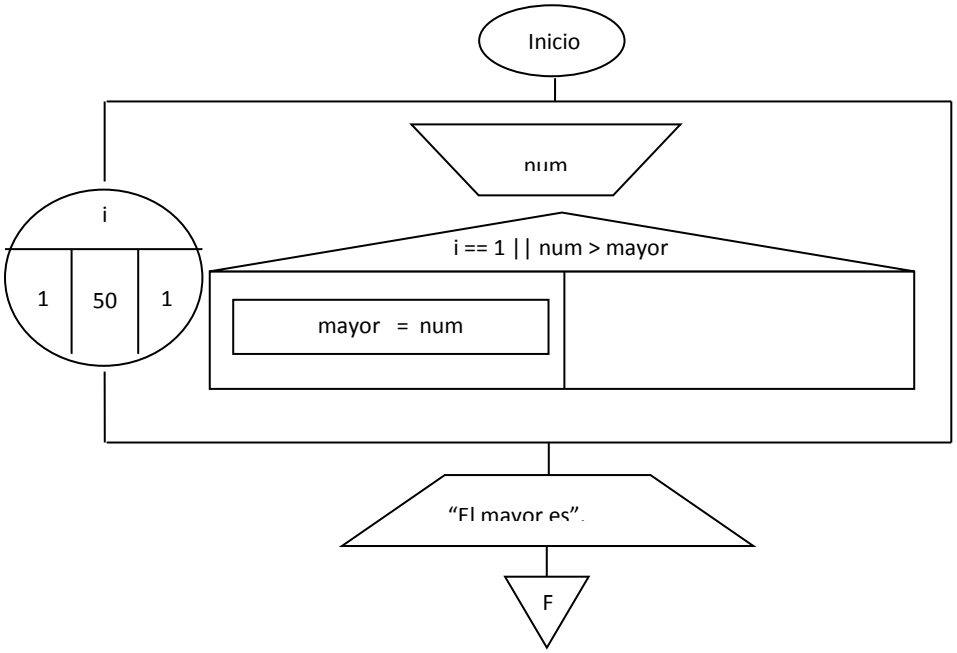
Se puede realizar de 3 formas:

a) Leer el primero afuera del ciclo
b) Una pregunta para separar el primero del resto.
c) Usar una señal para separar el primero del resto.

a) Leer el primero afuera de ciclo y lo almacenamos en la variable mayor.

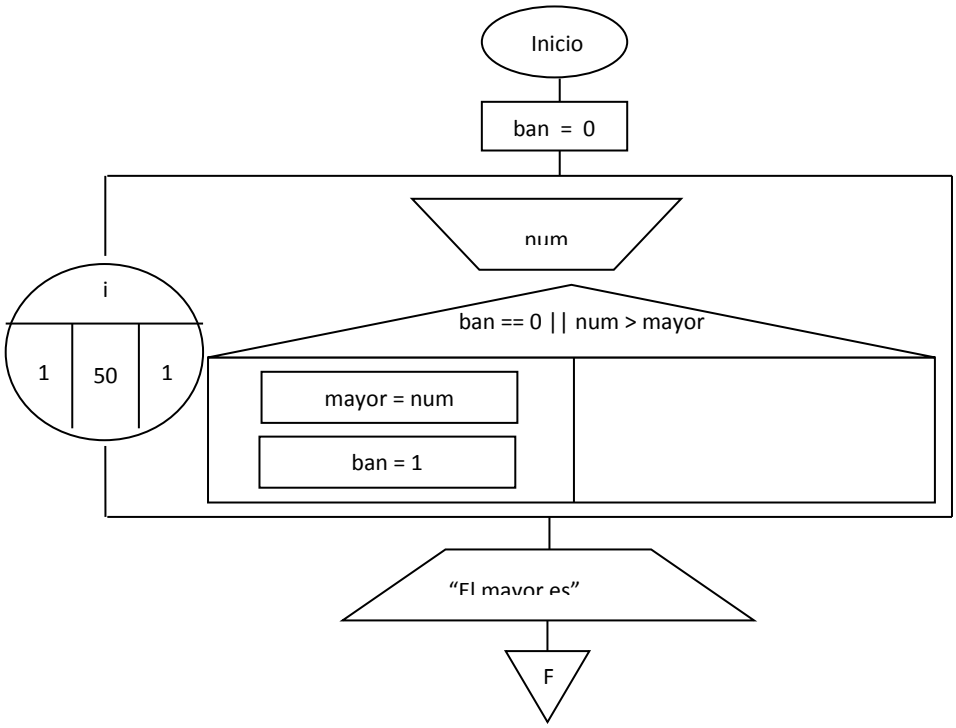


b) Una pregunta para separar el primero del resto, para darle valor inicial a la variable mayor.



En este ejemplo se realiza un solo if con dos condiciones, se pregunta si es la primera vez que se ingresa al ciclo, condición que se da cuando i vale 1. Luego esa condición será siempre falsa y por lo tanto se evalúa si el número recién ingresado es más grande que el guardado en la variable mayor. Sin embargo, la primera vez que se ingresa la variable mayor aún no tiene valor asignado, pero como el lenguaje C no necesita evaluar todas las condiciones, la primera vez que se ingresa al ciclo la primera condición es verdadera y al ser un or con una condición que sea verdadera es suficiente y por lo tanto no se evalúa la segunda condición.

c) Una pregunta para separar el primero del resto, para darle valor inicial a la variable mayor, utilizando una señal.



Una bandera o señal cumple con la función de tener el conocimiento de que un evento haya sucedido o no, es una variable a la cual se le asigna solo dos valores, uno se le asigna al inicio del proceso y el otro, al suceder el evento buscado.

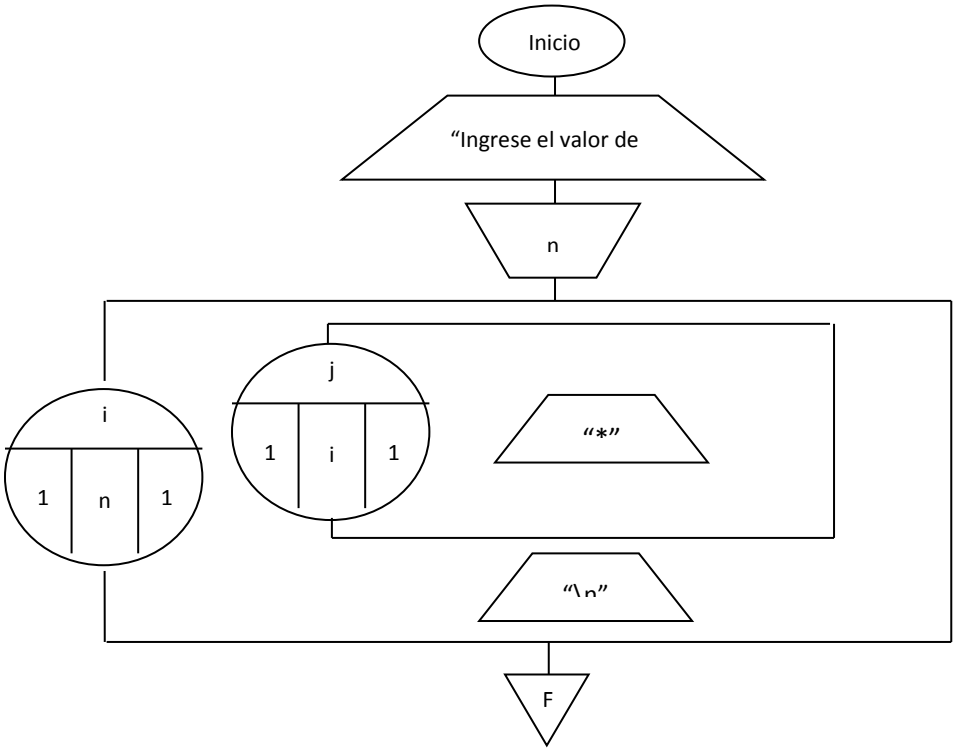
6. Estructuras de iteración definida anidadas

Las estructuras de control iterativas pueden anidarse, es decir, incluir a un ciclo completo dentro de otro. Las estructuras interior y exterior no necesitan ser del mismo tipo, pero es "esencial" que la estructura interior esté "totalmente" incluida en la exterior, no pueden "solaparse". Estamos analizando "for anidados", destacando que pueden ser 2 o más y que las variables utilizadas como control en los ciclos deben ser distintas.

Ejemplo 5: Hacer un algoritmo que imprima un triángulo rectángulo de * con base y altura de n cantidad de *. El valor de n debe ser ingresado por teclado.

Por ejemplo, si n es 4, debe imprimir:

```
*
**
***
****
```



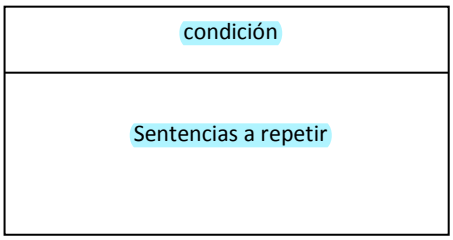
En código C:

```c
#include <stdio.h>

int main()
{
   int n, i, j;
   printf("Ingrese el valor de n \n");
   scanf("%d",&n);
   for(i=1; i<=n; i++)
   {
         for(j=1; j<=i; j++)
         {
               printf("*");
         }
         printf("\n");
   }
   return 0;
}
```

7. Estructuras de iteración condicionada

El ciclo definido no da respuesta a los casos en que desconocemos la cantidad de datos a procesar. Existen otras estructuras iterativas que permiten una mayor flexibilidad en la resolución de los problemas, sin tener que conocer previamente la cantidad de datos. Esta estructura requiere que pueda establecerse previamente una condición para poder finalizar, que, en general, es sencilla de definir.

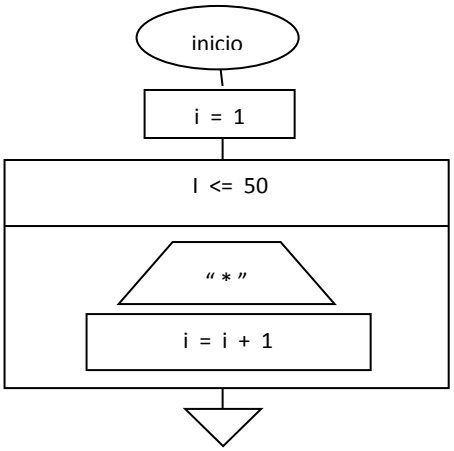


En esta estructura, primero se evalúa la condición, si es verdadera se ejecutan las sentencias que están dentro del ciclo, si es falso se pasa a la sentencia que continua, o sea se abandona el ciclo. Se debe tener precaución en no quedar atrapado dentro del ciclo, para lo cual dentro de las sentencias a repetir se debe incluir alguna que altere la condición de iteración.
Su codificación en lenguaje C es:

```
while (condición)
{
   Sentencia 1;
   Sentencia 2;
}
```

Las llaves son opcionales de haber una sola sentencia.

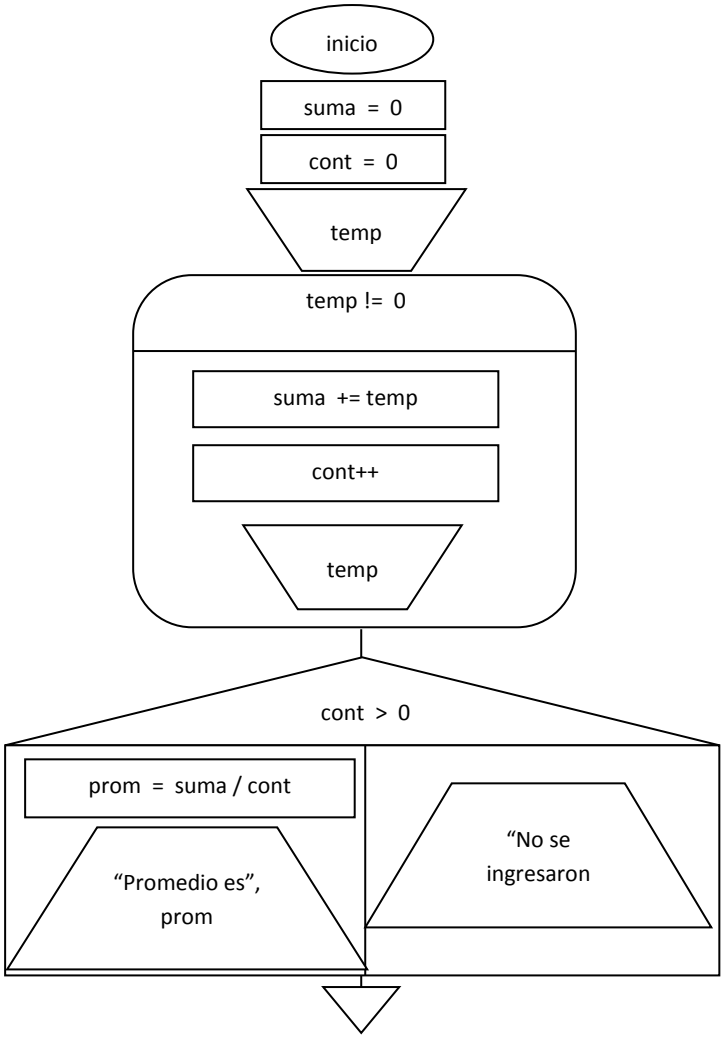
Ejemplo: Confeccionar un programa para imprimir una línea con 50 asteriscos.




```
#include <stdio.h>
int main()
{
   int i = 1;
   while(i <= 50)
   {
       printf("*");
       i++;
   }
   return 0;
}
```

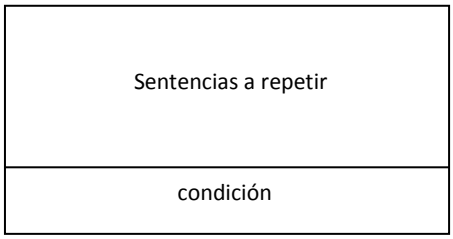
Podríamos resolverlo con una estructura de iteración definida, pero también podemos utilizar una estructura de iteración condicionada como se muestra en el ejemplo.

Ejemplo: Confeccionar un programa para ingresar diversos valores de temperatura hasta que aparezca uno igual a cero. Calcular e informar el promedio de los valores ingresados, sin considerar el cero.




```
#include <stdio.h>
int main()
{
   int cont;
   float temp, suma, prom;
   suma = 0;
   cont = 0;
   printf("Ingrese una temperatura (0 fin): ");
   scanf("%f",&temp);
   while(temp != 0)
   {
        suma += temp;
        cont ++;
        printf("Ingrese una temperatura (0 fin):");
        scanf("%f",&temp);
   }
   if(cont > 0)
   {
        prom = suma/cont;
        printf("El promedio de las temperaturas es: %.2f",prom);
   }
   else
       printf("No se ingresaron temperaturas (la primera fue cero)");
   return 0;
}
```

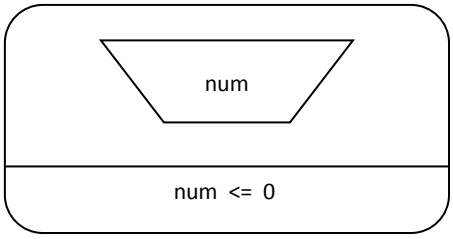
Existe otra estructura de iteración condicionada, similar en gran medida al "while" denominada do while.




```
do
{
    sentencia1;
    sentencia2;
        ……
} while (condición);
```

Se van a ejecutar las sentencias SIEMPRE UNA VEZ y todas las otras veces que sean necesarias mientras la condición sea "verdadera".
Es útil cuando es necesario ejecutar una serie de sentencias por lo menos una vez. Es un ciclo (1 – n), a diferencia con el ciclo while que es (0 - n). Una aplicación puede ser para controlar el ingreso de los datos.

Ejemplo: Solicitar el ingreso de un valor que debe ser positivo, reiterar la solicitud mientras no se ingrese un valor positivo.




```
do
{
    printf("Ingrese un entero positivo para
    continuar: ");
    scanf("%d",&num);
} while(num <= 0);
```

Comparación entre while y do/while

a) En ambas estructuras la repetición se produce cuando la condición es verdadera.
b) En el while la condición se evalúa "antes" de ejecutar el bucle. Ciclo 0 –N por lo tanto las variables de la condición deben tener valor asignado antes del ciclo.
c) En el do/while la condición se evalúa "después" de ejecutar una vez el cuerpo del bucle. Ciclo 1 – N. Por lo tanto, las variables de la condición pueden asignarse y modificarse directamente dentro del ciclo.
d) Las variables que figuren en la condición deben modificar su valor dentro del ciclo sino una vez dentro nunca se saldrá del mismo.