Fundamentos de Programación – Programación Estructurada en C:
5.- ESTRUCTURAS DE CONTROL DE FLUJO

Maider Huarte Arrayago

Fundamentos de Programación – Programación Estructurada en C:

5.- ESTRUCTURAS DE CONTROL DE FLUJO

5.- ESTRUCTURAS DE CONTROL DE FLUJO

La ejecución de un programa en C, se hace, por norma general, ejecutando las instrucciones de forma secuencial (una instrucción detrás de otra, siguiendo el orden en el que están escritas). Sin embargo, a veces hace falta ejecutar un bloque de secuencias u otro, dependiendo del valor de una condición, o repetir la ejecución de un bloque de secuencias varias veces. Para ello, la sintaxis de C proporciona una serie de estructuras de control de flujo, que se agrupan en dos categorías, Sentencias Selectivas (Bifurcaciones) y Sentencias Repetitivas (Bucles).

5.1- SENTENCIAS SELECTIVAS

Las Sentencias Selectivas, permiten la ejecución de un bloque de sentencias u otro, dependiendo de la evaluación de una condición.

Para indicar la condición a evaluar en cada caso, se pueden usar los Operadores Relacionales presentados en el tema anterior (**4.2.5.- Operadores Relacionales**), que sirven para evaluar condiciones. El resultado de la evaluación de una expresión con Operador Relacional dominante, tiene un valor 1, cuando la condición es verdadera y un valor 0, cuando es falsa.

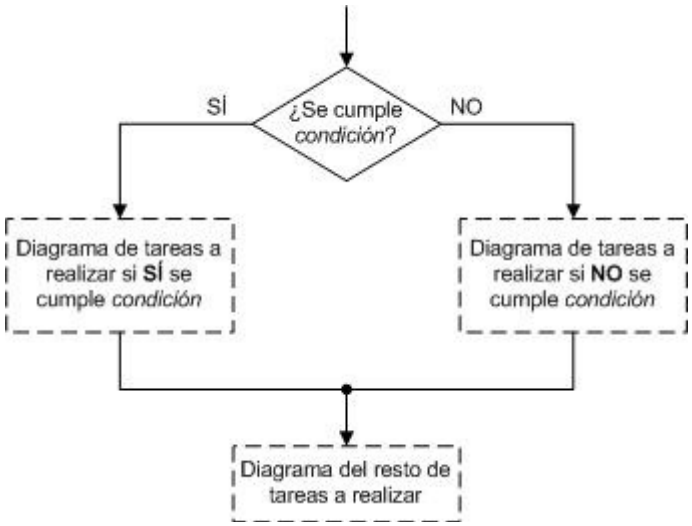
Sin embargo, pueden aparecer como condiciones, expresiones que no tengan un Operador Relacional como operador dominante. En esos casos, se calcula el valor numérico de la expresión. Si ese valor es 0, se considera que la condición es falsa, y en cualquier otro caso, verdadera. Hay que decir que esta segunda forma de expresar condiciones, aunque sintácticamente correcta, no se recomienda usarla.

Ejemplo:

```
a>b //es una expresión de condición válida, porque utiliza un Operador Relacional
v //Se evaluará su valor numérico
```

5.1.1.- Sentencia *if*

A la hora de resolver un problema mediante un programa escrito en un lenguaje estructural, puede ocurrir que dependiendo de si se cumple o no una condición, las tareas a realizar sean diferentes. El diagrama de flujo del programa, en ese punto de decisión, tendría una representación similar a esta:



Así, si hubiese tareas diferentes a realizar en ambos casos, habrá diagramas diferentes en cada ramal. También podría darse el caso en el que sólo haya que realizar tareas si la condición se cumple, o al revés; en ese caso, sólo habrá diagrama en el ramal correspondiente, mientras que en el otro, la flecha se conectará directamente con el *resto de tareas a realizar*.

En C, la forma de indicar este tipo de funcionamiento es mediante el uso de la sentencia condicional *if*.

Sintaxis:

```
if(<condición>)
    <sentencias del bloque if>
[else
    <sentencias del bloque else>
]
```

En tiempo de ejecución, cuando se ejecuta una sentencia *if*, lo primero que se hace es evaluar la condición indicada por la misma. Si el valor obtenido de esa evaluación es 0, entonces la condición es falsa (NO se cumple), si no, verdadera (SÍ se cumple).

Cuando la condición evaluada es verdadera, lo siguiente que se ejecuta son las *sentencias del bloque if*. Si sólo hay una sentencia, no hace falta delimitar el bloque con llaves ({ y }), porque en realidad no es un bloque. Si no, sí. En el diagrama de flujo correspondiente, se habrán representado como las *tareas a realizar si la condición SÍ se cumple*.

Cuando la condición evaluada es falsa, el funcionamiento es diferente. Según lo indicado en la sintaxis, hay una línea *else* con un bloque de sentencias asociado que es opcional.

- Si existe una línea *else*, se ejecuta su bloque de sentencias asociado.

- Si no existe ninguna línea *else*, se considera que la ejecución de la sentencia *if* ha terminado, y se sigue con la siguiente instrucción después de la sentencia *if*.

En el diagrama de flujo correspondiente, se habrán representado como las *tareas a realizar si la condición NO se cumple*. Si no hubiese ninguna tarea a realizar en ese caso, esa parte del diagrama no existiría y se enlazaría la flecha del NO directamente con la parte del *resto de tareas a realizar.*

Ejemplo:

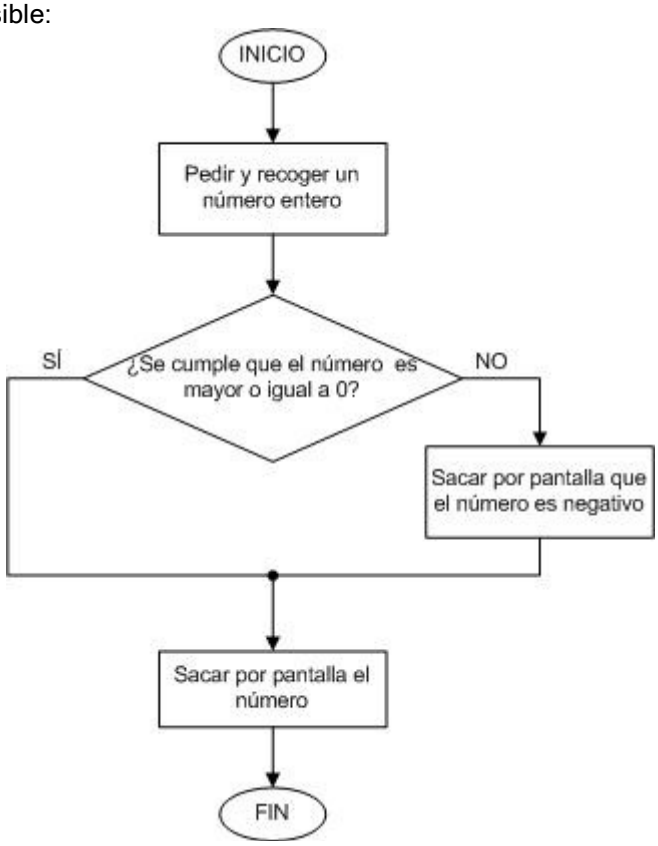
```
#include <stdio.h>

int main()
{
    int a;

    printf("Introduzca un valor entero para a:");
    scanf("%d",&a);

    if(a>=0)
        printf("\na es un número positivo");
    else
        printf("\na es un número negativo");

    printf("\na: %d\n\n",a);

    return 0;
}

/* Resultados de la ejecución:

Caso 1
Introduzca un valor entero para a:2

a es un n·mero positivo
a: 2

Caso 2
Introduzca un valor entero para a:-7

a es un n·mero negativo
a: -7
*/
```

Según la sintaxis, puede haber bloque *if* sin bloque *else*, pero nunca un bloque *else* sin ningún bloque *if* (la *condición* está en el *if*...). Si en el diagrama de flujo fuese el ramal del NO el único con tareas a realizar, como a la hora de escribirlo en C no se puede poner *else* sin *if*, habría que poner un bloque *if* con la condición contraria a la indicada en el diagrama y poner las tareas a realizar como sentencias de ese bloque *if*.

Ejemplo:

/*Pedir al usuario un nº entero, decir que es un número negativo cuando este no sea ≥ 0 y por último, visualizarlo.

Un diagrama posible:



*/

```
#include <stdio.h>

int main()
{
    int a;

    printf("Introduzca un valor entero para a:");
    scanf("%d",&a);

    if(a<0)// otra forma de expresar la condicón : !(a>=0)
        printf("\na es un número negativo");

    printf("\na: %d\n\n",a);

    return 0;
}
```

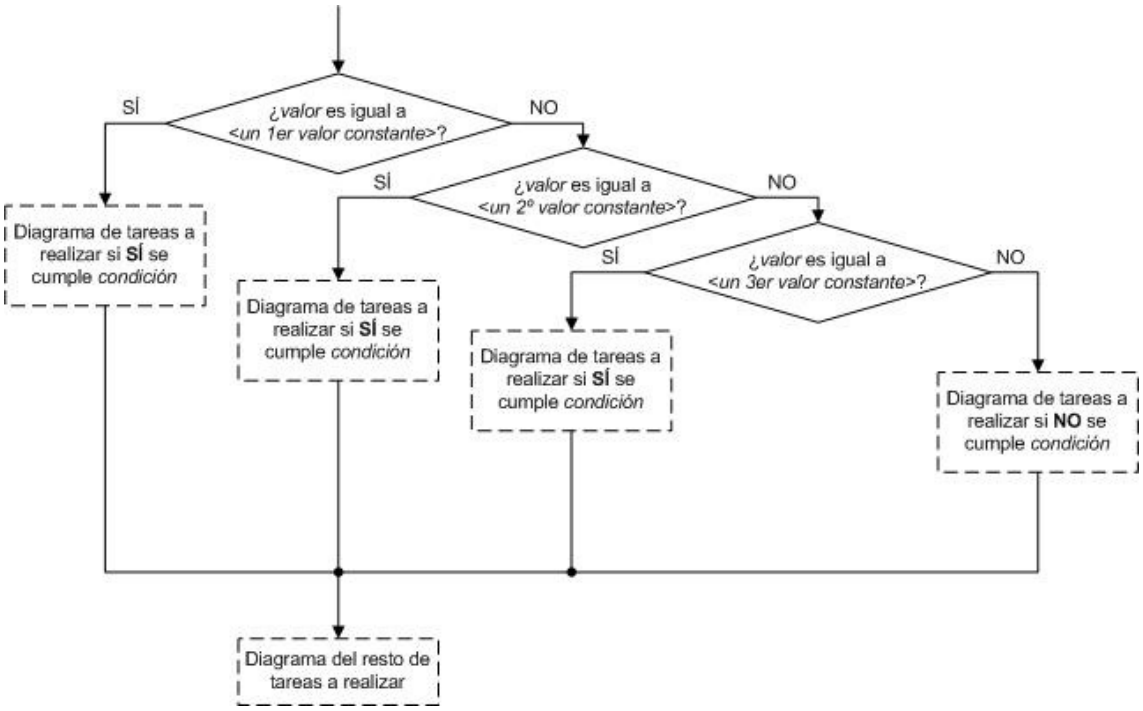
Dentro de un bloque de sentencias asociado a un *if*, y también dentro de un bloque de sentencias asociado a un *else*, puede haber cualquier tipo de sentencias, incluso, también sentencias *if*. En esos casos, se dice que hay un anidamiento de sentencias if.

Ejemplo:

```
if(a>b)
{
    if(b>c)
    {
        printf("b es mayor que c y menor que a");
        if(b!=0)
            printf("b es diferente de 0");
    }
    else
    {
        printf("b no es mayor que c, pero es menor que a");
    }
}
else
{
    if(b==a)
        printf("b es igual que a");
    else
        printf("b no es ni menor ni igual que a, por lo que es mayor que a");
}
```

5.1.2.- Sentencia *switch*

En otros problemas a resolver mediante un programa en un lenguaje estructural, puede ocurrir que según una serie de posibles valores (números enteros) alcanzados, las tareas a realizar sean diferentes en cada caso. El diagrama de flujo de un programa, con tres casos posibles diferentes, tendría una representación similar a esta:



Así, si hubiese tareas diferentes a realizar en cada caso, habrá diagramas diferentes en cada ramal SÍ en el que se compara el caso correspondiente. En cada caso, el ramal NO lleva a la evaluación del siguiente.

En C, la forma de indicar este tipo de funcionamiento podría ser mediante sentencias *if-else* anidadas, pero existe otra sentencia condicional más fácil de usar: la sentencia condicional *switch*.

Sintaxis:

```
switch(<expresión entera>)
{
    case <valor constante entero 1>:
        [<sentencias constante 1>
        break;]

    case <valor constante entero 2>:
        [<sentencias constante 2>
        break;]

    ...

    case <valor constante entero n>:
        <sentencias constante n>
        break;

    [default:
        <sentencias default>
    ]
}
```

En una sentencia *switch*, lo primero que se hace es evaluar la expresión. El valor de esa expresión entera, como su propio nombre indica, ha de ser un valor numérico entero, con lo cuál, serán válidas expresiones que den como resultado un valor de un tipo de dato de número entero (tipos *unsigned short*, *short*, *unsigned int* o *int*) o un carácter (tipo *char*), que se admite ya que los códigos ASCII son números enteros. Así, el valor resultado del cálculo de la expresión se va comprobando si es igual con las **constantes enteras** (números enteros o constantes carácter) indicadas en cada *case* del bloque del *switch*, siguiendo el orden en el que aparecen. Si hubiese un *case* cuya constante coincide con el valor de la expresión del *switch*, se ejecutan las sentencias correspondientes.

Si dentro del *switch*, hubiese un *default*, las sentencias asociadas al mismo sólo se ejecutarían si ninguna de las constantes de los *case* anteriores coincidiese con el valor de la expresión (sería lo indicado en el ramal NO de la última condición). *default* y sus sentencias asociadas son opcionales, por lo que si no apareciesen, cuando la expresión no coincide con ninguna constante de los *case*, no se ejecuta nada del bloque *switch*, y se sigue con la siguiente instrucción después del mismo.

Ejemplo:

```
#include <stdio.h>

int main()
{
    int mes;

    printf("\nIntroduzca el número de un mes: ");
    scanf("%d",&mes);

    switch(mes)
    {
        case 1:
            printf("\nEl mes es Enero");
            break;

        case 2:
            printf("\nEl mes es Febrero");
            break;

        case 3:
            printf("\nEl mes es Marzo");
            break;

        case 4:
            printf("\nEl mes es Abril");
            break;

        case 5:
            printf("\nEl mes es Mayo");
            break;

        case 6:
            printf("\nEl mes es Junio");
            break;

        case 7:
            printf("\nEn mes es Julio");
            break;

        case 8:
            printf("\nEl mes es Agosto");
            break;

        case 9:
            printf("\nEl mes es Septiembre");
            break;

        case 10:
            printf("\nEl mes es Octubre");
            break;

        case 11:
            printf("\nEl mes es Noviembre");
            break;

        case 12:
```

```
            printf("\nEl mes es Diciembre");
            break;

        default:
            printf("\nEl valor introducido no corresponde a ningún mes");
    }
    printf("\nFin del programa\n\n");

    return 0;
}

/* Resultados de la ejecución:

Caso 1

Introduzca el n·mero de un mes: 4

En mes es Abril
Fin del programa

Caso 2

Introduzca el n·mero de un mes: 21

El valor introducido no corresponde a ning·n mes
Fin del programa
*/
```

Según la sintaxis, las sentencias asociadas a todos los *case* menos el último, son opcionales. Lo más claro, sería escribir asociado a cada *case* su bloque de sentencias propio, pero puede pasar, que para constantes diferentes, las sentencias que queramos que se ejecuten sean las mismas. En ese caso, para evitar el escribir el mismo conjunto de sentencias para cada *case*, lo que se hace es poner todos esos *case* seguidos sin sus *break;* asociados, y escribir las sentencias sólo en el último. Por eso, el último *case* siempre ha de tener un bloque de sentencias, porque si no, el *switch* no serviría de nada.

Ejemplo:

```
#include <stdio.h>

int main()
{
    int mes;

    printf("\nIntroduzca el número de un mes: ");
    scanf("%d",&mes);

    switch(mes)
    {
        case 10:
        case 11:
        case 12:
        case 1:
            printf("\nEs un mes del primer cuatrimestre");
            break;

        case 2:
```

```
            case 3:
            case 4:
            case 5:
                printf("\nEs un mes del segundo cuatrimestre");
                break;

            case 6:
            case 9:
                printf("\nEs un mes de exámenes");
                break;

            case 7:
            case 8:
                printf("\nEs un mes de vacaciones");
                break;

            default:
                printf("\nEl valor introducido no corresponde a ningún mes");
        }
        printf("\nFin del programa\n\n");

        return 0;
}

/* Resultados de la ejecución:

Introduzca el n·mero de un mes: 3

Es un mes del segundo cuatrimestre
Fin del programa

*/
```

La instrucción *break;* sirve para dejar de ejecutar las sentencias de un bloque y salir de él, siguiendo la ejecución en la siguiente instrucción después del bloque. Por eso, es necesario poner una sentencia *break;* después de cada conjunto de sentencias asociado a un *case*, ya que si no, se seguiría ejecutando la siguiente instrucción dentro del bloque *switch* (línea terminada en ;, aunque no fuese del *case* correspondiente).

Si en el ejemplo anterior no hubiésemos puesto ningún *break;*, el compilador no daría ningún error. El funcionamiento del programa, sin embargo, sería erróneo, ya que después de encontrar el *case* correspondiente, ejecutaría todas las instrucciones (terminadas en *;*) que encontrara, hasta el final del bloque *switch*. Así, se ve la necesidad de usar la instrucción *break;* al final de cada grupo de sentencias asociadas a los *case* y al *default*.

```
#include <stdio.h>

int main()
{
    int mes;

    printf("\nIntroduzca el número de un mes: ");
    scanf("%d",&mes);
```

```
    switch(mes)
    {
        case 10:
        case 11:
        case 12:
        case 1:
            printf("\nEs un mes del primer cuatrimestre");

        case 2:
        case 3:
        case 4:
        case 5:
            printf("\nEs un mes del segundo cuatrimestre");

        case 6:
        case 9:
            printf("\nEs un mes de exámenes");

        case 7:
        case 8:
            printf("\nEs un mes de vacaciones");

        default:
            printf("\nEl valor introducido no corresponde a ningún mes");
    }
    printf("\nFin del programa\n\n");

    return 0;
}

/* Resultados de la ejecución:

Introduzca el n·mero de un mes: 4

Es un mes del segundo cuatrimestre
Es un mes de exβmenes
Es un mes de vacaciones
El valor introducido no corresponde a ning·n mes
Fin del programa

*/
```

Aunque el compilador permita el uso de la sentencia *break;* en otros casos, en esta asignatura, sólo se admitirá su uso en bloques *switch*.

Así, puede decirse que el funcionamiento de una sentencia *switch*, es el de varias sentencias *if* anidadas, en las que se comprueba una condición de igualdad de un valor con una serie de **constantes enteras**. En el caso de que se tenga un valor que haya de ser comparado con variables o constantes no enteras o haya que aplicarle condiciones que no son de igualdad, es necesario usar sentencias *if* anidadas, ya que *switch* no sirve para ello.

Los *switch*, se pueden anidar, ya que asociado a cada *case* o al *default*, se pueden poner todo tipo de sentencias.

5.2.- SENTENCIAS REPETITIVAS

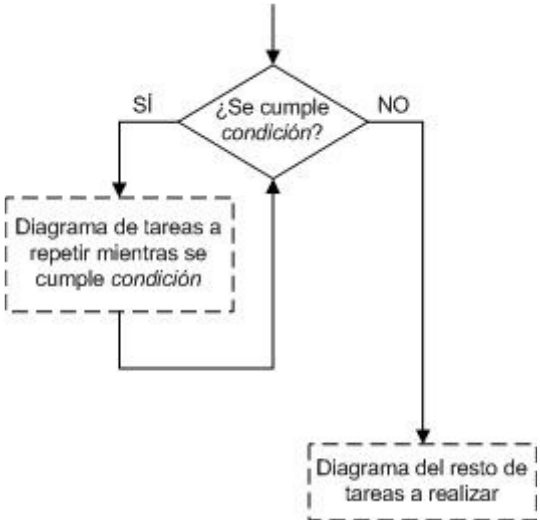
Las Sentencias Repetitivas, permiten repetir la ejecución de un bloque de sentencias mientras la evaluación de una condición dé un resultado verdadero.

Tal y como pasa con las Sentencias Selectivas, la ejecución de las Sentencias Repetitivas pasa por la evaluación de una condición. Esas condiciones, pueden ser expresiones construidas con Operadores Relaciones o no. Si se diese el último caso, se calcularía el valor numérico de la expresión, considerando sólo el valor 0 como falso.

La evaluación de la condición en las Sentencias Repetitivas, ha de cambiar con cada una de las repeticiones de ejecución. Hay que tener cuidado a la hora de poner esas condiciones, porque si su evaluación no cambia en cada una de las repeticiones, el programa podría entrar en un *bucle infinito*, y estar repitiendo continuamente el mismo bloque de sentencias, sin poder hacer nada más. Los bucles infinitos se consideran erróneos (a no ser que se indique su necesidad en la especificación del problema), ya que suponen que habría que terminar el programa desde fuera del mismo.

5.2.1.- Sentencia *while*

La resolución de un problema mediante un programa estructurado podría requerir la ejecución repetida de una serie de tareas, mientras una condición se cumpla. El diagrama de flujo en ese punto sería:



En C, una forma simple de indicar este tipo de funcionamiento es mediante la sentencia *while*.

Sintaxis:

while(<condición>**)**
    <sentencias del bloque *while*>

En una sentencia *while*, se evalúa la <condición>, y si es verdadera, se ejecuta el bloque de sentencias asociado a *while*. Cuando se termina la ejecución del bloque de sentencias, comienza otra vez el proceso, volviendo a evaluarse la <condición>. En cuanto el resultado de la evaluación sea falso, se termina la ejecución de la sentencia *while* y se sigue ejecutando la siguiente instrucción después de la misma.

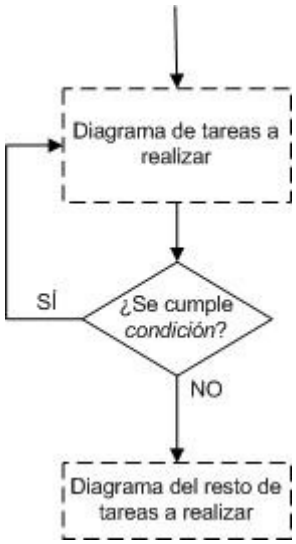
Como en el caso de *if*, si sólo hay una sentencia a repetir, no hace falta meterla en un bloque (poner las llaves).

Ejemplo:

```
#include <stdio.h>

int main()
{
    int n;

    printf("\nEscribo los números del 1 al 10, uno en cada línea: ");

    n=1;
    while(n<=10)
    {
        printf("\n%d",n);
        n++;
    }

    return 0;
}
```

5.2.1.- Sentencia *do-while*

Otras veces, se puede necesitar ejecutar por lo menos una vez una serie de tareas, y después repetirlas mientras se cumpla una cierta condición. Es decir:



En C, la forma de indicar este tipo de funcionamiento es mediante la sentencia *do-while*.

Sintaxis:

```
do
    <sentencias del bloque do>
while(<condición>);
```

En una sentencia *do-while*, se empieza ejecutando el bloque de sentencias asociado a *do*. Una vez ejecutado, se evalúa la <condición>, y si es verdadera, se repite la ejecución del bloque *do*. Así, al contrario que con la sentencia *while*, la evaluación de la <condición> se hace al finalizar cada ejecución del bloque de sentencias; la primera ejecución del bloque de sentencias *do*, se hace **una primera vez** sin haber evaluado la <condición>, es decir, **siempre**.

Con la sentencia *do-while*, el bloque de sentencias asociado al *do*, se ejecuta al menos una vez (con *while*, como lo primero que se evalúa es la <condición>, podría darse el caso de que el bloque de sentencias asociado no se ejecutara nunca).

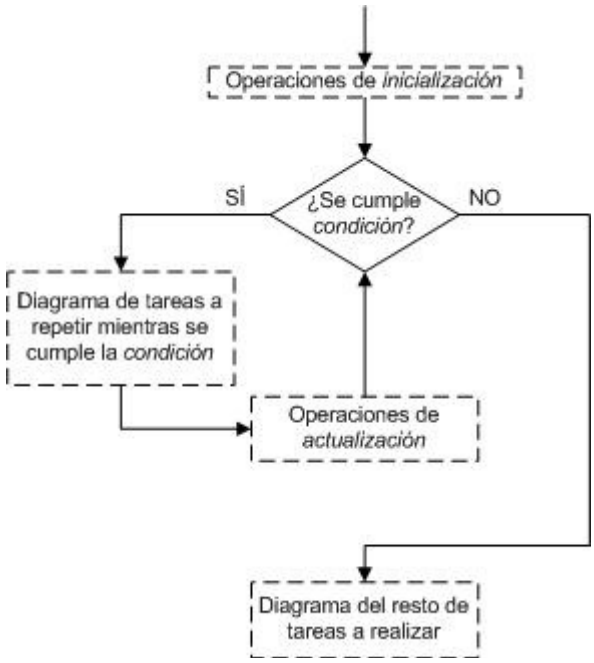
Como en el caso de *if*, si sólo hay una sentencia a repetir, no hace falta meterla en un bloque (poner las llaves). Esta estructura es muy útil para realizar el control de los datos introducidos por el usuario a un programa.

Ejemplo:

```
#include <stdio.h>

int main()
{
    int n,i=1;

    do
    {
        printf("\nIntroduzca un número del rango [1,100]: ");
        scanf("%d",&n);
    }
    while(n<1||n>100);

    printf("\nEscribo los números del 1 al %d, uno en cada línea: ",n);
    while(i<=n)
    {
        printf("\n%d",i);
        i++;
    }

    return 0;
}
```

5.2.3.- Sentencia *for*

Al resolver un problema, se puede necesitar preparar una posible repetición de tareas (*inicialización*) y repetir esa serie de tareas mientras se cumpla una condición, ejecutando una última serie de operaciones antes de cada evaluación de la condición (*actualización*). Es decir:



En C, la forma simple de indicar este tipo de funcionamiento es mediante la sentencia *for*.

Sintaxis:

for([<inicialización>]**;**[<condición>]**;**[<actualización]**)**
    <sentencias del bloque *for*>

Tal y como se ve en la sintaxis, asociado a la sentencia *for*, hay un paréntesis que puede llegar a contener 3 elementos, separados entre sí mediante **;**.

- *inicialización*: se trata de una instrucción (o conjunto de instrucciones separadas por ,), que se ejecutan sólo cuando se empieza a ejecutar la sentencia *for*.
- *condición*: se trata de la condición que se evalúa antes de empezar a ejecutar las sentencias del bloque *for*. Si el resultado de la evaluación es verdadero, se repetirá un vez más la ejecución de las sentencias del bloque *for*. Si no, la ejecución de la sentencia *for* se dará por terminada, y se seguirá con la siguiente instrucción.
- *actualización*: se trata de una instrucción (o conjunto de instrucciones separadas por ,) que se ejecuta en cuanto se acaba la ejecución de las sentencias del bloque *for*, y antes de evaluar otra vez la condición para saber si hay que repetir la ejecución.

inicialización, *condición* y *actualización* son elementos opcionales. Si no se quiere que se ejecute nada especial antes de la primera ejecución de las sentencias del bloque *for*, no se indica nada en *inicialización*. Si tampoco se quiere que se ejecute nada especial cada vez que se termine una repetición de ejecución de las sentencias del bloque *for*, tampoco se pone *actualización*.

Aunque la sintaxis indique que *condición* es opcional, hay que saber que el no escribir nada en ese elemento, supone una condición verdadera, con lo que la repetición del bucle *for* no terminaría nunca (bucle infinito).

Ejemplo:

```
#include <stdio.h>

int main()
{
    int cont,num;

    printf("\nEsta es la tabla del 7:");
    for(cont=1;cont<=10;cont++)
    {
        num=7*cont;
        printf("\n%dx7=%d",cont,num);
    }

    return 0;
}
```

Según lo explicado sobre el funcionamiento de *for*, se entiende que es equivalente a un *while* de la siguiente forma:

```
for(<inicialización>;<condición>;<actualización>)
{
    <sentencias>
}
```

⇔

```
<inicialización>
while(<condición>)
{
    <sentencias>
    <actualización>
}
```

Así, el ejemplo anterior se podía haber escrito de otra forma.

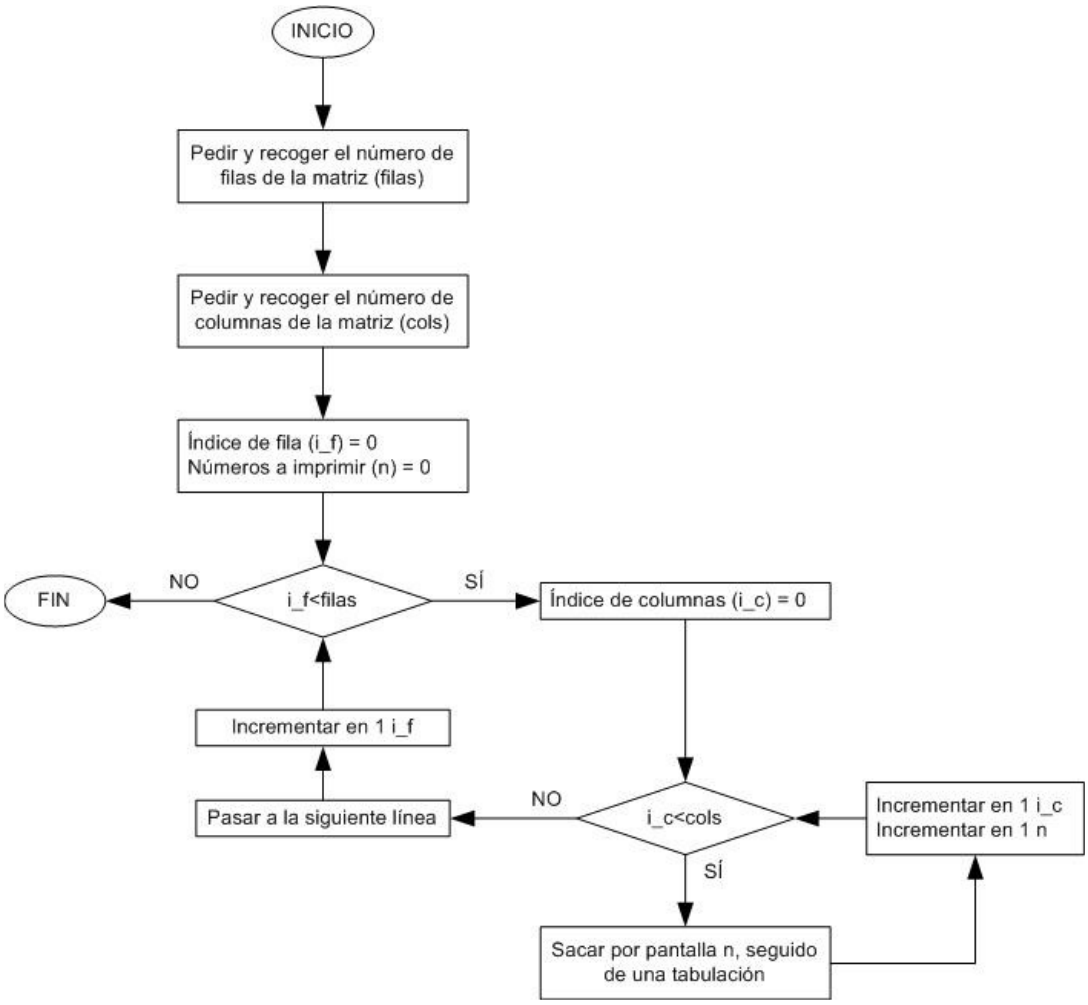
Todas las Sentencias Repetitivas, *while*, *do-while* y *for*, al igual que las Sentencias Selectivas, se pueden anidar, ya que no hay restricciones en cuanto a las sentencias de los bloques asociados a cada una. Como en los demás casos, si sólo hay una sentencia a repetir, no hace falta meterla en un bloque (poner las llaves).

Ejemplo:

```
#include <stdio.h>

int main()
{
    int n=0;
    int filas,cols,i_f,i_c;

    printf("\nIntroduzca el número de filas: ");
    scanf("%d",&filas);

    printf("\nIntroduzca el número de columnas: ");
    scanf("%d",&cols);

    printf("\n\n");

    for(i_f=0;i_f<filas;i_f++)
    {
        for(i_c=0;i_c<cols;i_c++,n++)
            printf("%d\t",n);
        printf("\n");
    }

    printf("\n");

    return 0;
}

/* Resultados de la ejecución:
```

```
Introduzca el n-mero de filas: 3

Introduzca el n-mero de columnas: 4


0       1       2       3
4       5       6       7
8       9       10      11
```

Diagrama de flujo:



*/