

INSTITUTO RAÚL SCALABRINI ORTIZ

Análisis de Sistemas

Algoritmos

Segunda Parte

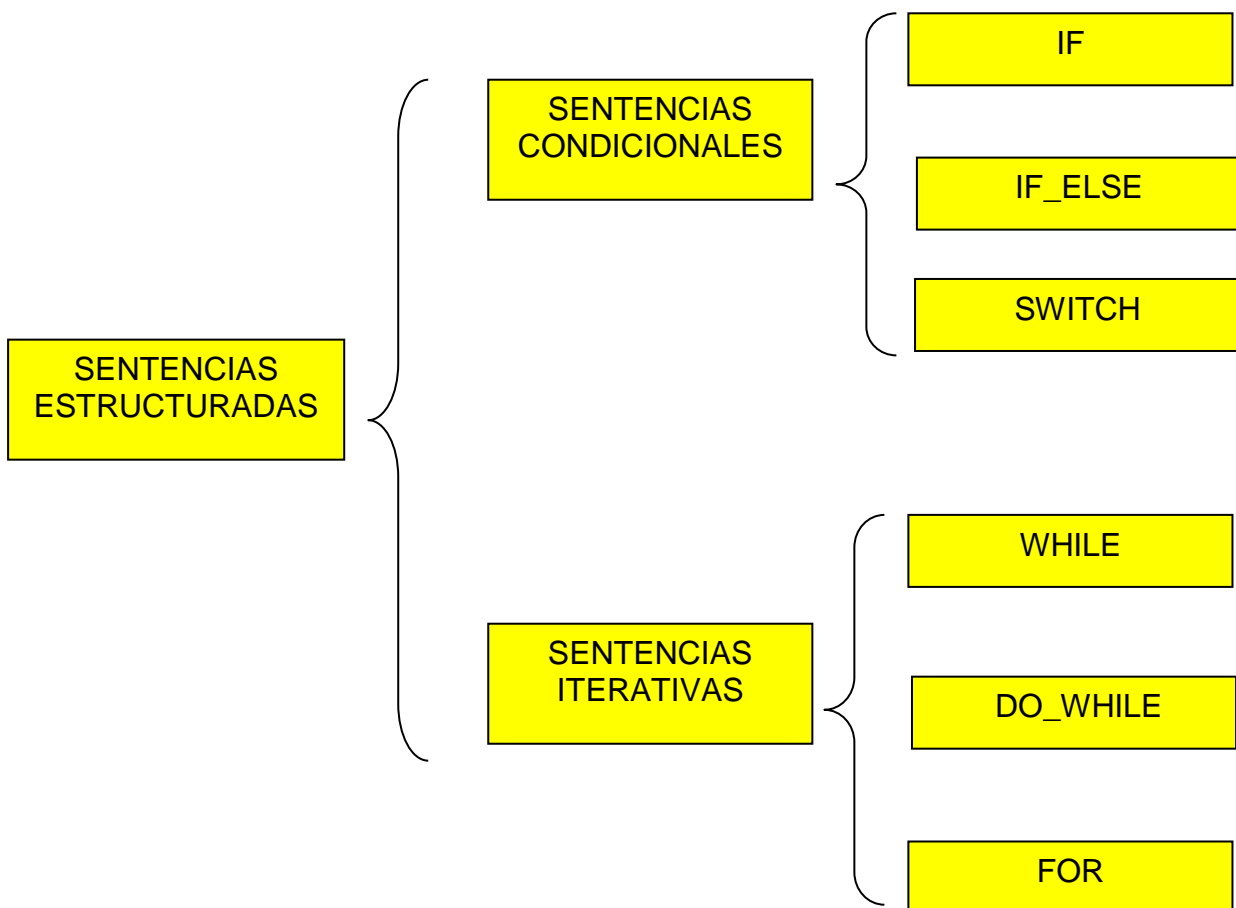
Profesor: Roberto Adrián San Jorge

SENTENCIAS COMPUESTAS O ESTRUCTURADAS

La **sentencia compuesta** especifica que puede tener otras sentencias en ella y sus sentencias componentes deben ser ejecutadas en el mismo orden en que aparecen escritas.

Los símbolos “{” y “}” **actúan como delimitadores**. Note que el bloque principal del programa tiene la forma de una sentencia compuesta.

El C++ usa el punto y coma para *terminar* las sentencias, pero, el **delimitador** (punto y coma) **NO ES** parte de la sentencia.



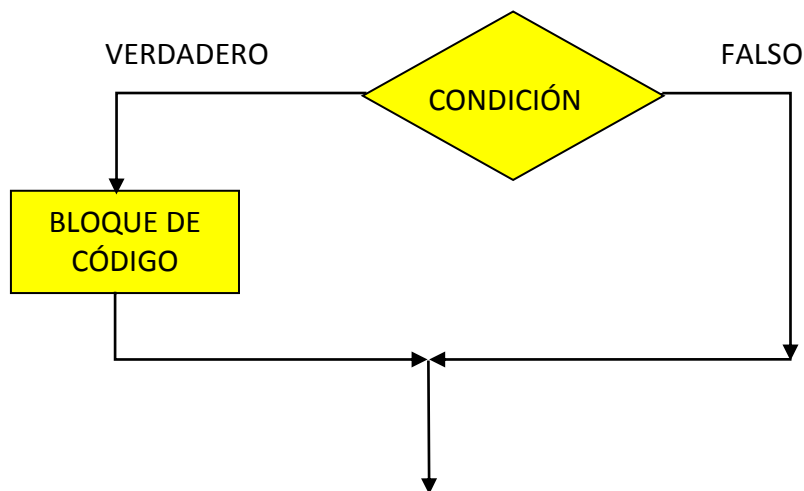
SENTENCIAS CONDICIONALES

Una **sentencia condicional** selecciona un *único Bloque de Código*¹ de entre sus componentes para ser ejecutado. La **sentencia if** especifica que un enunciado debe ejecutarse **sólo si una cierta condición (una expresión Booleana) es verdadera**. Si es falsa, entonces o no se ejecuta enunciado alguno, o se ejecutan los que sigan al símbolo **else**.

IF

Nos permite ejecutar un bloque de código sólo si se cumple una condición.
La forma básica que tiene esta instrucción es la siguiente:

if (<condición>)
 <bloque de código>



Ejemplo: Para poder visualizar más concretamente esta instrucción, podemos utilizar el siguiente ejemplo,

```
if (p==3)
    z=8;
```

¹ Un Bloque de Código puede estar formado por varias sentencias, en cuyo caso van siempre entre llaves, o por una única sentencia, escrita entre llaves o no.

La **condición** es: **p==3**. La condición será verdadera si la variable p vale 3.

La **acción** es: **z=8**. La acción que se ejecutará es asignarle 8 a la variable z.

Por lo tanto, lo que aquí dice es que si el valor de la variable p es igual a 3 entonces asigna el valor 8 a la variable z. Debemos con estas instrucciones comenzar a trabajar escuchando lo que nuestro interlocutor quiere cada vez que nos solicita una solución de un problema utilizando un programa. En este caso él nos dirá “si el valor de p es 3 entonces quiero que se le asigne el valor 8 a la variable z”. Esto es lo que nosotros expresamos coloquialmente cada vez que queremos hacer algo. De esta forma es más que importante que comencemos a saber escuchar qué es lo que nuestros interlocutores desean que realice un programa para utilizar específicamente la instrucción que corresponde. Esto es quizás el punto clave sobre el que todo programador debe poner especial énfasis. La tarea de comprensión de la inquietud de nuestro interlocutor será lo que garantice el correcto funcionamiento de nuestro programa.

Debemos comprender cuál es nuestro problema para encontrarle una solución y luego programar. La programación debe ser la última tarea en nuestro esquema de trabajo.

Programas ejemplo:

```
#include<stdio.h>
main()
{int numero;
printf("Ingrese un valor entero: ");
scanf("%d",&numero);
fflush(stdin);
if(numero>0)
    printf("\n El valor ingresado es positivo.");
printf("\n Adiós."); //ESTA LÍNEA YA ESTÁ FUERA DE LA SENTENCIA IF
getchar();
return 0;}
```

```
#include<stdio.h>
main()
{int numero;
printf("Ingrese un valor entero: ");
scanf("%d",&numero);
fflush(stdin);
if((numero>=0)&&(numero<=9))
    printf("\n El valor ingresado tiene un sólo dígito.");
printf("\n Adiós.");
getchar();
return 0;}
```

IF-ELSE

Nos permite ejecutar un bloque de código si se cumple una condición, y sino se ejecutará otro bloque de código.

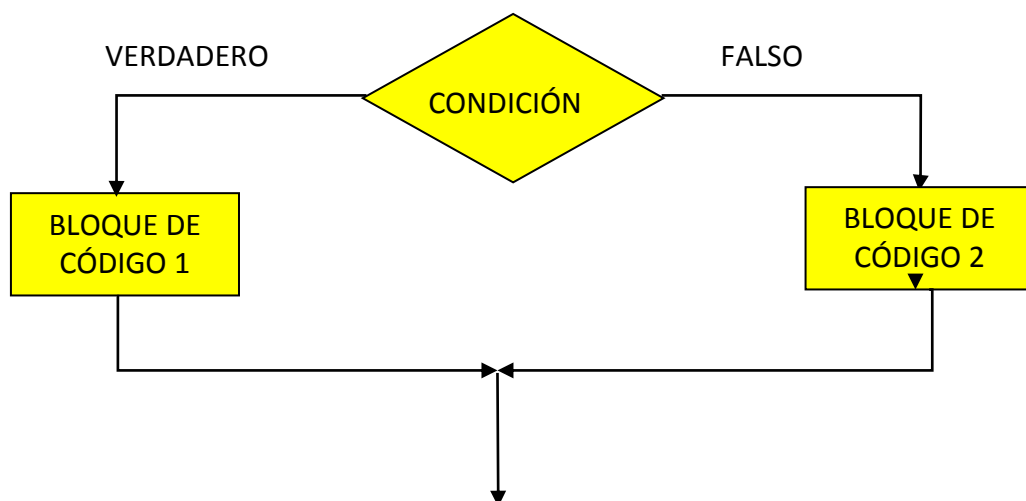
La forma básica que tiene esta instrucción es la siguiente:

if (<condición>)

 <bloque de código1>

else

 <bloque de código2>



Si la condición después del “if” es verdadera, las acciones que están después de la condición se ejecutarán. Si en cambio la condición después del “if” es falsa, las acciones que están después del “else” se ejecutarán.

Ejemplo: Si utilizamos la siguiente instrucción,

```
if (p==3)
    z=8;
else
    z=11;
```

La **condición** es: $p=3$. La condición será verdadera si la variable p vale 3.

La **acción** es: $z=8$. La acción que se ejecutará es asignarle 8 a la variable z **si la condición $p=3$ es verdadera**.

La **acción** es: $z=11$. La acción que se ejecutará es asignarle 11 a la variable z **si la condición $p=3$ es falsa**.

Por lo tanto, resumiendo y volviendo al lenguaje coloquial, tenemos que **“si p es igual a 3 entonces z quedará cargado con el valor 8; si en cambio p tiene cualquier otro valor, z quedará cargado con el valor 11”**. Esto es lo que dice nuestra instrucción “**if_else**” precedente.

Obviamente, es posible colocar más de una acción a cada condición. En el caso que esto suceda, deberán utilizarse los delimitadores “{” y “}” para indicar las acciones que tiene comprendidas esa condición.

Ejemplo: “Si p es igual a 3 quiero asignarle el valor 8 a la variable z y el valor 1 a la variable m , en cambio si p es distinto de 3 quiero asignarle el valor 11 a z y 2 a m ”. Nótese que este enunciado debe escribirse de la siguiente manera siguiendo las reglas que hemos ido definiendo en la presente unidad:

```
if (p==3)
    {z=8;
    m=1;}
else
    {z=11;
    m=2;}
```

Veamos que las acciones están comprendidas entre los “{” y “}” correspondientes a la rama del “**if**” y a la rama del “**else**”. En cada una hay dos acciones. Al final de cada acción de las que se encuentran en la primera posición tenemos un delimitador que indica que la instrucción terminó y que luego viene otra instrucción.

Programas ejemplo:

```
#include<stdio.h>
main()
{int numero;
printf("Ingrese un valor entero: ");
scanf("%d",&numero);
fflush(stdin);
if (numero>0)
    printf("\n El valor ingresado es positivo.");
else
    printf("\n El valor ingresado no es positivo.");
printf("\n Adiós.");
getchar();
return 0;}
```

```
#include<stdio.h>
main()
{int clave;
printf("Ingrese la contraseña: ");
scanf("%d",&clave);
fflush(stdin);
if (clave==1234)
    {printf("\n La contraseña es correcta.");
    printf("\n Puede ingresar.");}
else
    printf("\n La contraseña es incorrecta.");
printf("\n Adiós.");
getchar();
return 0;}
```

SWITCH

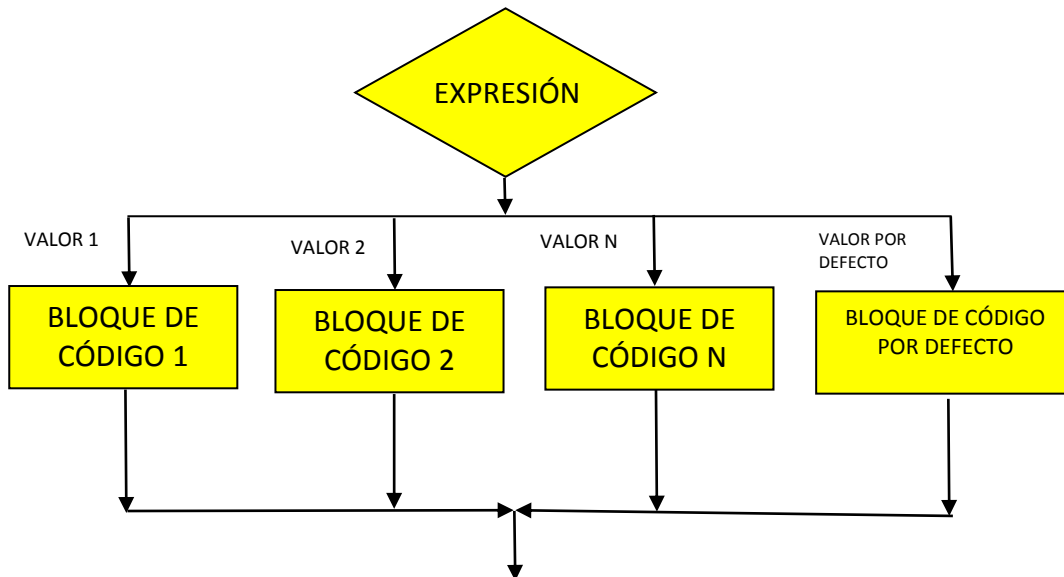
Nos permite según el valor de una expresión (identificador o valor concreto), ejecutar un bloque de código, u otro, ..., u otro, u otro por defecto, de una manera muy clara.

La sentencia “**switch**” consiste en una *expresión* (el **selector**), y en una *lista de enunciados* (**rótulos case**), cada uno de ellos rotulados por una constante del mismo tipo que el selector. El tipo de dato del selector debe ser escalar, **excluyendo el tipo**

real. El enunciado “switch” selecciona como siguiente sentencia a ejecutarse aquella cuyo rótulo case es igual al valor actual del selector; si tal rótulo no existe, ejecuta el rótulo de default, y si éste no existe, su efecto queda indefinido. Una vez completada la ejecución de la sentencia seleccionada, se transfiere el control al final del enunciado “switch”. La forma es:

switch (<expresión>)

```
{  
  case <valor 1> : <bloque de código 1>; break;  
  case <valor 2> : <bloque de código 2>; break;  
  ...  
  case <valor n> : <bloque de código n>; break;  
  default      : <bloque de código por defecto>;  
}
```



Por ejemplo

```
#include<stdio.h>
main()
{
    int valor;
    char vocal;
    printf("Ingrese un valor del 1 al 5: ");
    scanf("%d",&valor); fflush(stdin);
    switch(valor)
    {
        case 1: printf("\n Ingresó un uno."); break;
        case 2: printf("\n Ingresó un dos."); break;
        case 3: printf("\n Ingresó un tres."); break;
        case 4: printf("\n Ingresó un cuatro."); break;
        case 5: printf("\n Ingresó un cinco."); break;
        default: printf("\n Ingresó un valor incorrecto.");
    }

    printf("Ingrese una vocal: ");
    scanf("%c",&vocal); fflush(stdin);
    switch(valor)
    {
        case 'a': printf("\n Ingresó una a."); break;
        case 'e': printf("\n Ingresó una e."); break;
        case 'i': printf("\n Ingresó una i."); break;
        case 'o': printf("\n Ingresó una o."); break;
        case 'u': printf("\n Ingresó una u."); break;
        default: printf("\n No ingresó una vocal.");
    }

    printf("Pulse enter para salir...");
    getchar();
    return 0;
}
```

Funcionamiento del switch

Consiste en comparar el valor de la expresión con los especificados en cada rótulo *case*.

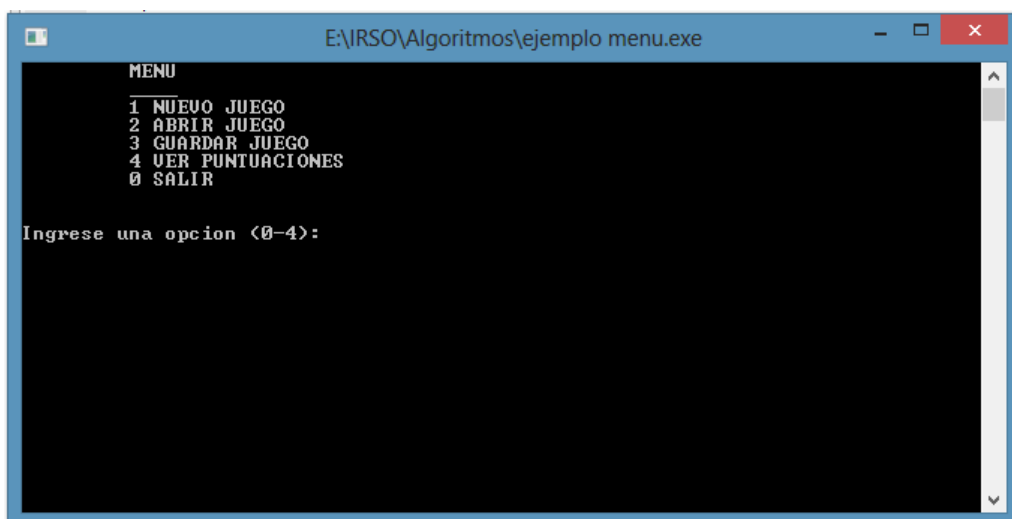
En caso en que el valor sea igual a uno de ellos, se ejecuta el bloque de código situado después de los dos puntos, tras lo cual se ejecuta la instrucción *break*, que hace finalizar la sentencia *switch*, continuando el flujo de datos con el código que haya tras la sentencia.

Si el valor que almacena la expresión no coincide con ninguno de los rótulos *case*, entonces se ejecuta el bloque de código por defecto, que es el situado a continuación de la palabra *default*, y tras el cual la sentencia *switch* finaliza. La cláusula *default* es opcional, por lo que si no queremos que se ejecute ningún código, en ese caso podemos no escribir la cláusula *default*.

No olvidar:

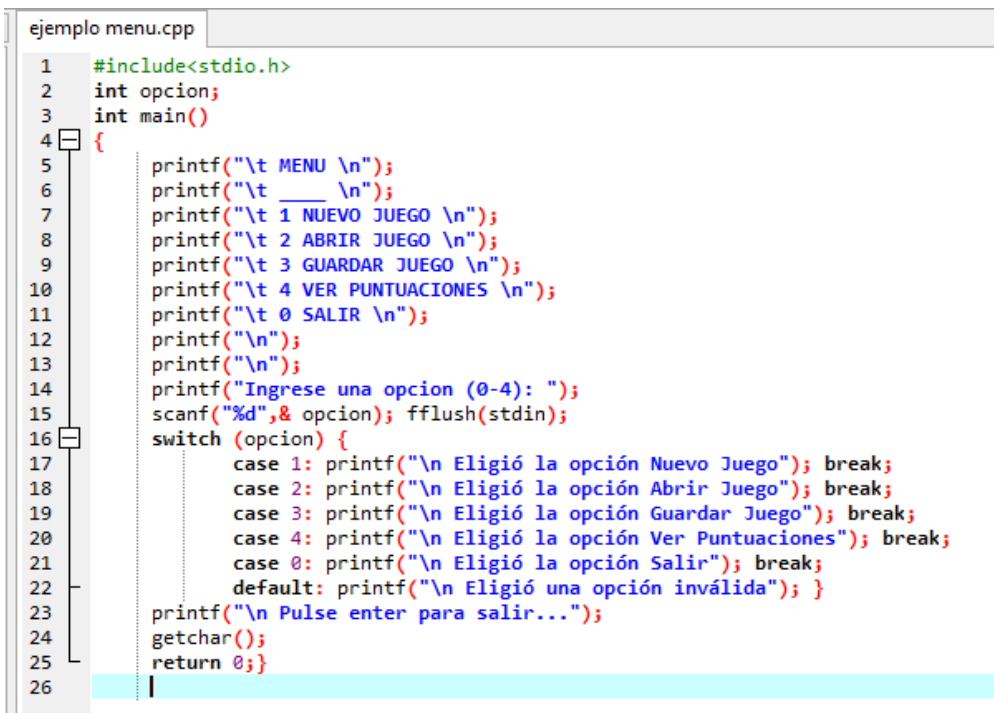
Cuando se ejecuta una instrucción *break* el flujo del programa sale inmediatamente de la sentencia *switch*.

La sentencia switch suele usarse habitualmente para gestionar menús en los que se muestran distintas opciones de las que disponen, se pide al usuario que elija una y se ejecuta el código correspondiente a esa opción



```
MENU
1 NUEVO JUEGO
2 ABRIR JUEGO
3 GUARDAR JUEGO
4 VER PUNTUACIONES
0 SALIR

Ingrese una opcion <0-4>:
```



```
ejemplo menu.cpp
1  #include<stdio.h>
2  int opcion;
3  int main()
4  {
5      printf("\t MENU \n");
6      printf("\t ____ \n");
7      printf("\t 1 NUEVO JUEGO \n");
8      printf("\t 2 ABRIR JUEGO \n");
9      printf("\t 3 GUARDAR JUEGO \n");
10     printf("\t 4 VER PUNTUACIONES \n");
11     printf("\t 0 SALIR \n");
12     printf("\n");
13     printf("\n");
14     printf("Ingrese una opcion (0-4): ");
15     scanf("%d",&opcion); fflush(stdin);
16     switch (opcion) {
17         case 1: printf("\n Eligió la opción Nuevo Juego"); break;
18         case 2: printf("\n Eligió la opción Abrir Juego"); break;
19         case 3: printf("\n Eligió la opción Guardar Juego"); break;
20         case 4: printf("\n Eligió la opción Ver Puntuaciones"); break;
21         case 0: printf("\n Eligió la opción Salir"); break;
22         default: printf("\n Eligió una opción inválida"); }
23     printf("\n Pulse enter para salir...");
24     getchar();
25     return 0; }
26
```

SENTENCIAS ITERATIVAS O DE REPETICIÓN

Las **sentencias de repetición** especifican que ciertas sentencias deben ejecutarse en forma repetitiva. Si el número de repeticiones se conoce de antemano (antes que las repeticiones comiencen), la sentencia **“for”** resulta la construcción más apropiada para expresar la situación; de otro modo se recomienda el empleo de las sentencias **“while”** o **“do_while”**.

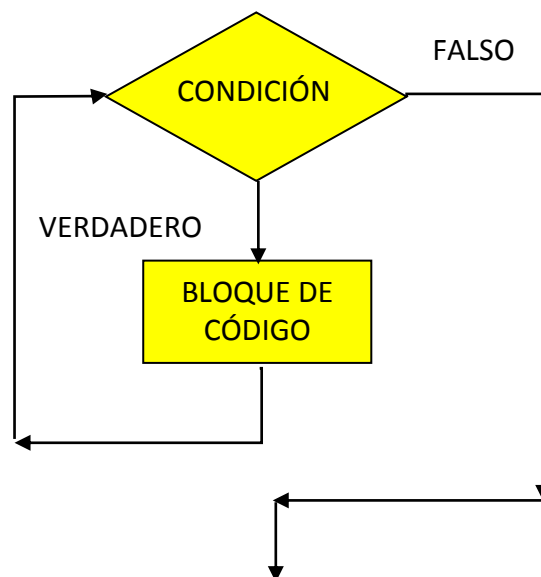
En muchas ocasiones queremos que se repita un bloque de código mientras se den una serie de circunstancias. A cada repetición se le llama **iteración**.

WHILE

Nos permite ejecutar un bloque de código mientras se cumpla una condición.

La instrucción **“while”** tiene la siguiente forma:

```
while (<condición>)  
    <bloque de código>;
```



Esta estructura tiene una condición que **controla la ejecución repetida de las acciones**.

En la misma, **las acciones se ejecutan mientras que la condición sea verdadera**.

En el caso que la condición es falsa, la ejecución del ciclo finaliza.

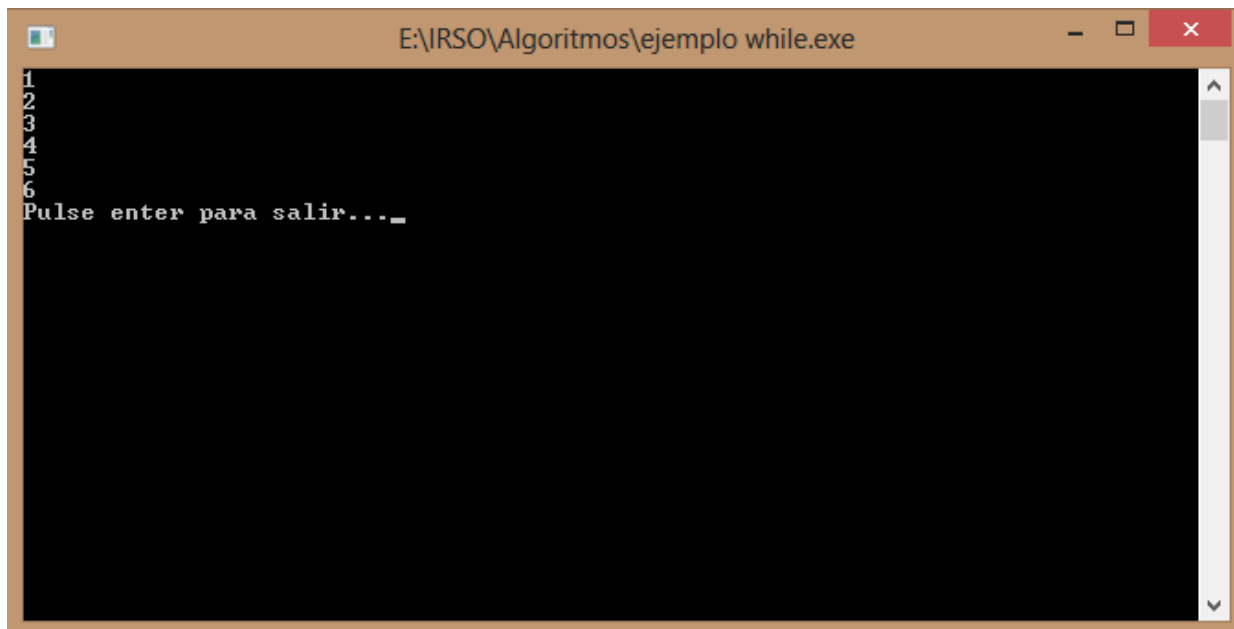
La condición es evaluada **siempre antes de que la acción se ejecute** de modo que, si la condición al inicio del ciclo es falsa, **la acción no será ejecutada nunca**.

Desde el punto de vista coloquial, la instrucción “**while**” puede ser pensada como:
“**mientras se cumpla la condición ejecuto las acciones**”.

La instrucción *while* se utiliza cuando se deba ejecutar un bloque de código:

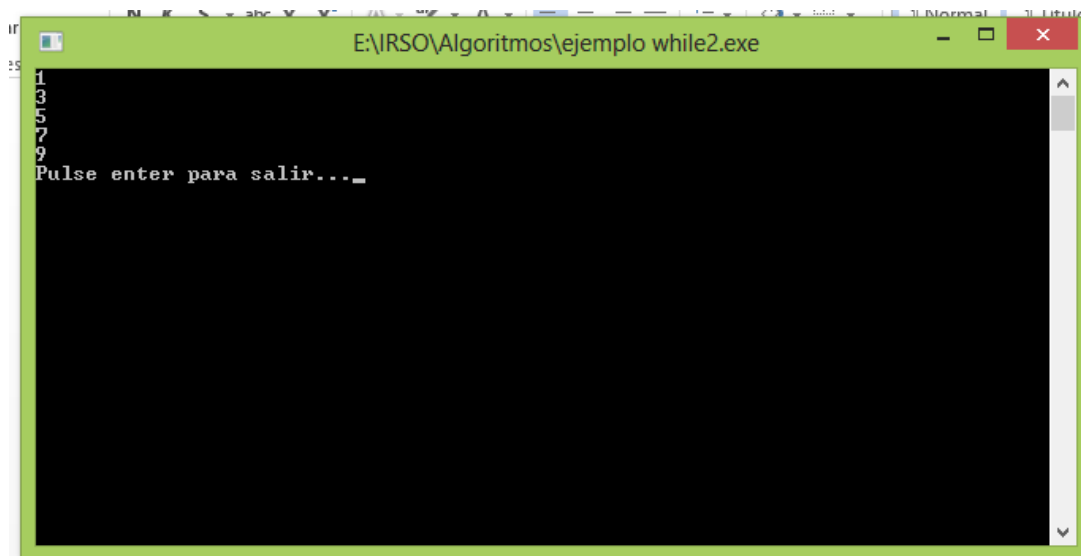
- Quizás ninguna vez.
- Quizás sólo una vez.
- Quizás varias veces.

```
ejemplo while.cpp
1 // Mostrar los enteros positivos del 1 al 6
2 #include<stdio.h>
3 int num;
4 int main()
5 {num=1;
6 while(num<=6)
7 {printf("%d \n",num);
8 num++; }
9 printf("Pulse enter para salir...");
10 getchar();
11 return 0;}
12
```



ejemplo while2.cpp

```
1 // Mostrar Los impares del 1 al 10
2 #include<stdio.h>
3 int num;
4 int main()
5 {
6     num=1;
7     while(num<=10)
8     {
9         printf("%d \n", num);
10        num=num+2;
11    }
12    printf("Pulse enter para salir...");
13    getchar();
14    return 0;
15 }
```



[*] ejemplo while3.cpp

```
1 /* pedir al usuario una clave numérica, si no es igual
2    a la especificada en el programa, se la vuelve a pedir.
3    El programa no finalizará hasta que se acierte con la
4    contraseña, que es 1234 */
5 #include<stdio.h>
6 const int CLAVE=1234;
7 int num;
8 int main()
9 {
10     printf("Ingrese la contraseña: ");
11     scanf("%d",& num); fflush(stdin);
12     while(num!=CLAVE)
13     {
14         printf("\n La contraseña es incorrecta");
15         printf("\n Ingrese la contraseña: ");
16         scanf("%d",& num); fflush(stdin);
17     }
18     printf("\n La contraseña es correcta");
19     printf("\n Pulse enter para salir...");
20     getchar();
21     return 0;
22 }
```

DO WHILE

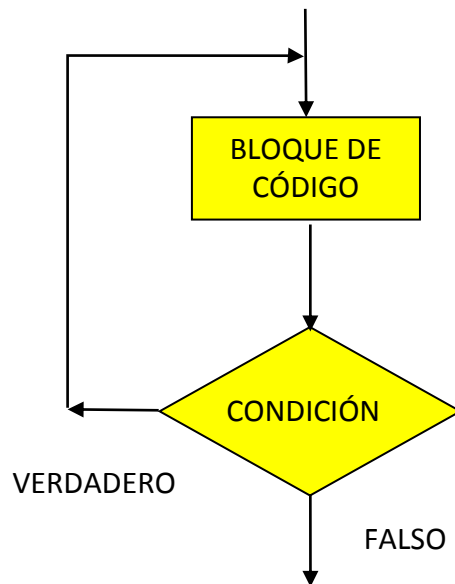
Nos permite ejecutar un bloque de código mientras se cumpla la condición, pero el funcionamiento y su sintaxis difieren al while.

La instrucción “**do while**” tiene la siguiente forma:

do

<bloque de código>

while (<condición>);



Esta estructura tiene una condición que **controla la ejecución de las acciones mientras se cumpla la condición**. Esto es, las acciones se ejecutarán **mientras la condición sea verdadera**. Cuando la condición sea falsa, el ciclo finalizará.

La condición es evaluada **siempre después de que la acción se ejecute** de modo que **las acciones se ejecutarán al menos una vez**.

Desde el punto de vista coloquial, la instrucción “**do while**” puede ser pensada como: **“ejecutar las acciones mientras se cumpla la condición”**.

La instrucción *do while* se utiliza cuando se deba ejecutar un bloque de código:

- Al menos una vez.
- Quizás varias veces.

ejemplo do while.cpp

```
1  #include<stdio.h>
2  int opcion;
3  int main()
4  {
5      do
6      {
7          printf("\t \n MENU \n");
8          printf("\t _____ \n");
9          printf("\t 1 NUEVO JUEGO \n");
10         printf("\t 2 ABRIR JUEGO \n");
11         printf("\t 3 GUARDAR JUEGO \n");
12         printf("\t 4 VER PUNTUACIONES \n");
13         printf("\t 0 SALIR \n");
14         printf("\n");
15         printf("\n");
16         printf("Ingrese una opcion (0-4): ");
17         scanf("%d",&opcion); fflush(stdin);
18         switch (opcion) {
19             case 1: printf("\n Eligió la opción Nuevo Juego"); break;
20             case 2: printf("\n Eligió la opción Abrir Juego"); break;
21             case 3: printf("\n Eligió la opción Guardar Juego"); break;
22             case 4: printf("\n Eligió la opción Ver Puntuaciones"); break;
23             case 0: printf("\n Eligió la opción Salir"); break;
24             default: printf("\n Eligió una opción inválida"); } }
25         while(opcion != 0);
26         printf("\n Pulse enter para salir...");
27         getchar();
28         return 0;
29 }
```

[*] ejemplo do while 2.cpp

```
1  /*CALCULAR Y MOSTRAR EN PANTALLA EL DOBLE DE UN NUMERO
2  INGRESADO POR EL USUARIO, REPITIENDO EL ALGORITMO
3  MIENTRAS EL USUARIO LO DESEE */
4  #include<stdio.h>
5  int val, doble; char letra;
6  main()
7  {
8      do
9      {
10         printf("\n Ingrese un valor: ");
11         scanf("%d",&val); fflush(stdin);
12         doble=val*2;
13         printf("\n El doble de %d es %d.",val,doble);
14         printf("\n Desea continuar? (s/n): ");
15         scanf("%c",&letra); fflush(stdin);
16     }
17     while((letra=='s')||(letra=='S'));
18     return 0;
19 }
```

FOR

Nos permite repetir un bloque de código, pero su sintaxis es bastante distinta a la del while y do while.

La forma que tiene la instrucción “**for**” es la siguiente:

for (<inicialización>;<condición>;<incremento>)
 <bloque de código>;

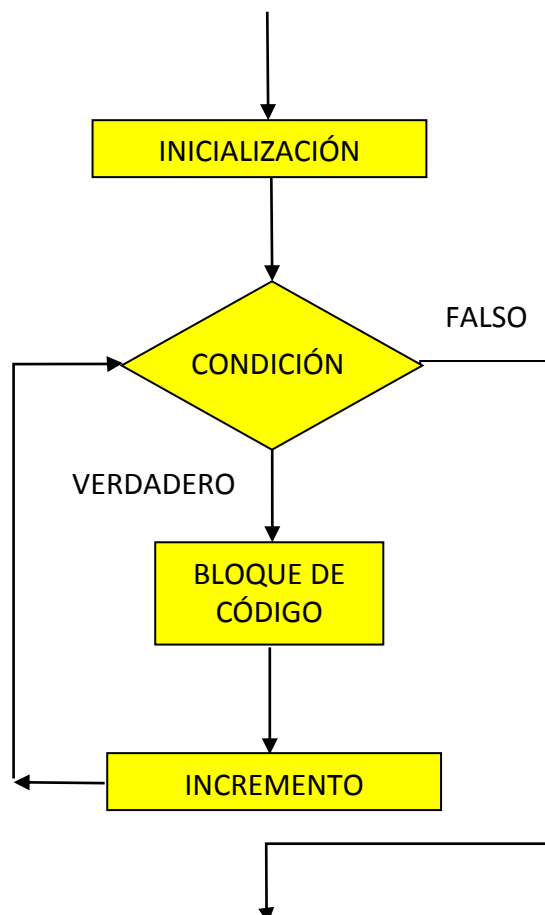
Inicialización: Es la asignación del valor inicial a la variable de control.

Condición: Es una expresión lógica que involucra a la variable de control y determina si se sigue repitiendo el bloque de código.

Incremento: Es una expresión que incrementa el valor de la variable de control.

La instrucción “**for**” hace que el bloque de código sea ejecutado una vez para cada valor en el rango valor inicial al valor final.

La **variable de control** y los **valores inicial** y **final** deben ser de un tipo ordinal.

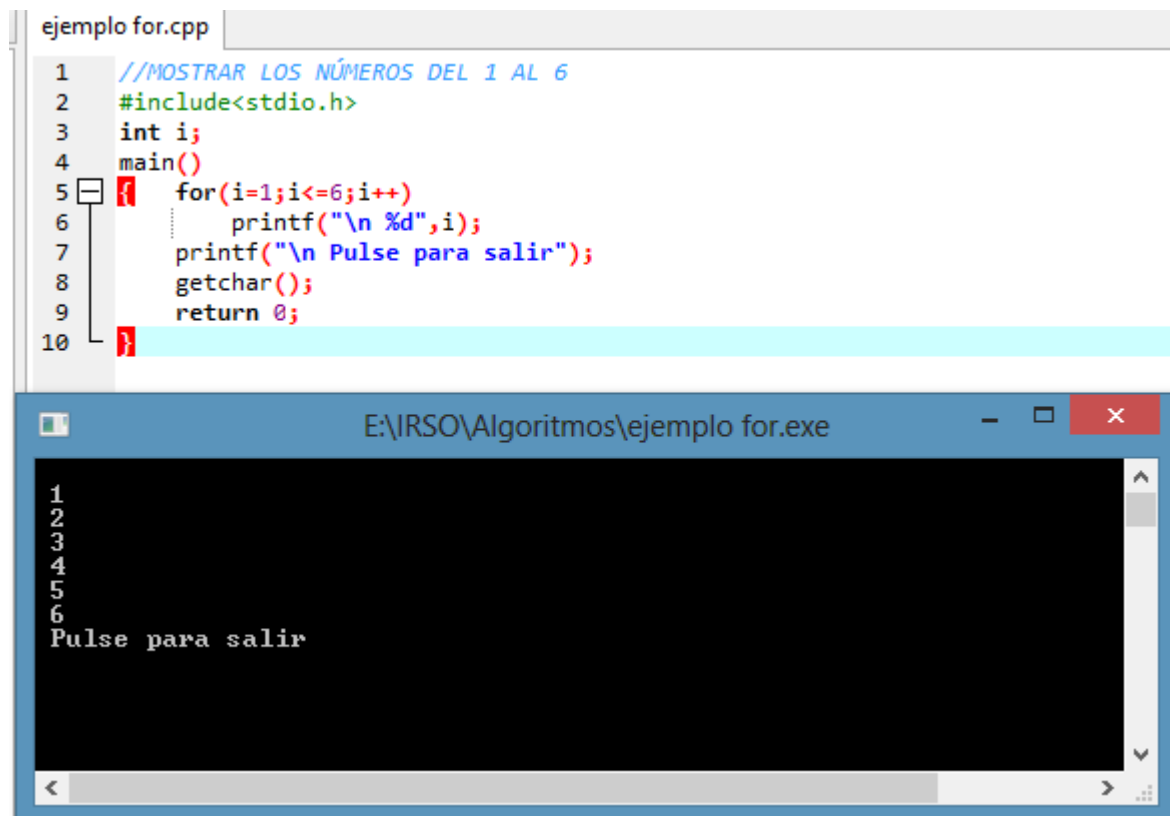


El funcionamiento del For se puede definir en una serie de pasos:

1. Ejecución de la inicialización.
2. Evaluación de la condición: si es verdadero se continúa con el paso 3, sino se salta al paso 6.
3. Ejecución del bloque de código.
4. Ejecución del incremento.
5. Salto al paso 2.
6. Fin de la sentencia For.

La repetición del bloque de código de la sentencia For no es "hasta que se cumpla la condición", sino que es "**mientras se cumpla la condición**".

Ejemplo:



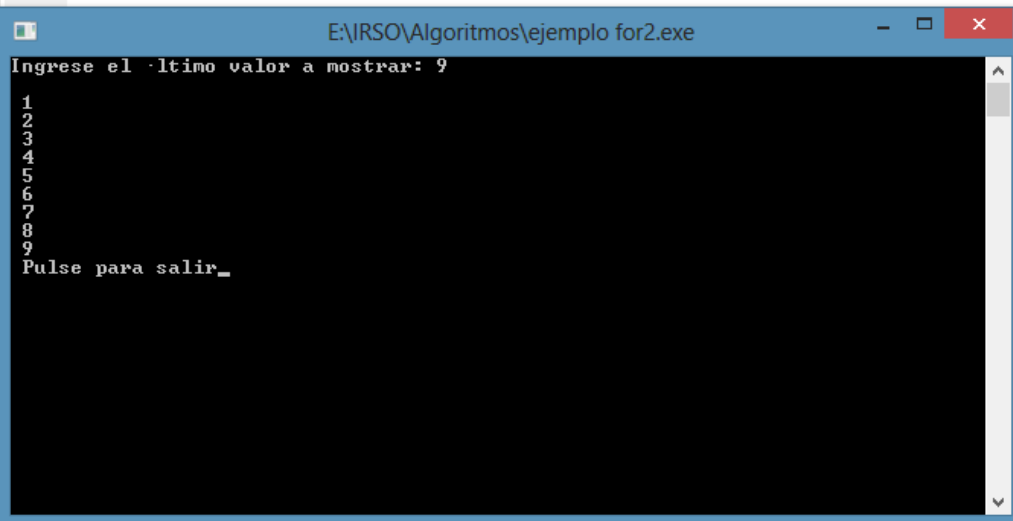
The image shows a code editor window titled 'ejemplo for.cpp' and a terminal window titled 'E:\IRSO\Algoritmos\ejemplo for.exe'. The code in the editor is as follows:

```
1 //MOSTRAR LOS NÚMEROS DEL 1 AL 6
2 #include<stdio.h>
3 int i;
4 main()
5 {
6     for(i=1;i<=6;i++)
7         printf("\n %d",i);
8     printf("\n Pulse para salir");
9     getchar();
10    return 0;
}
```

The terminal window shows the output of the program: the numbers 1 through 6 on separate lines, followed by the text 'Pulse para salir'.

ejemplo for2.cpp

```
1 //MOSTRAR LOS NÚMEROS DEL 1 A AQUEL QUE INDIQUE EL USUARIO
2 #include<stdio.h>
3 int i, ultimo;
4 main()
5 { printf("Ingrese el último valor a mostrar: ");
6   scanf("%d",& ultimo); fflush(stdin);
7   for(i=1;i<=ultimo;i++)
8     printf("\n %d",i);
9   printf("\n Pulse para salir");
10  getchar();
11  return 0;
12 }
```



ejemplo for3.cpp

```
1 //PEDIR UN VALOR AL USUARIO Y MOSTRAR SU TABLA DE MULTIPLICAR
2 #include<stdio.h>
3 int i, val, r;
4 main()
5 { printf("Ingrese un valor: ");
6   scanf("%d",& val); fflush(stdin);
7   printf("\n La tabla del %d es: ",val);
8   printf("\n");
9   for(i=1;i<=10;i++)
10  { r=i*val;
11    printf("\n \t %d x %2d = %4d",val, i, r);}
12  printf("\n \n Pulse para salir");
13  getchar();
14  return 0;
15 }
16
```

