Estructuras Básicas de Algoritmos y de Datos II

José Otero

[1]Departamento de informática
Universidad de Oviedo

9 de octubre de 2008

La mayor parte de los cálculos realizados mediante un programa requieren:

- Realizar una secuencia de acciones.
- Realizar unas acciones u otras en función de determinados indicadores.
- Repetir determinada secuencia de acciones:
  - Un cierto número de veces.
  - Mientras se cumpla determinada condición.

En C++ existen las composiciones:

- **Secuencial**
- **Condicional**
- **Iterativa**

NOTA: En el contexto de la escritura de un programa, la secuenciación de las acciones es tal que las que se escriben antes se ejecutan antes también: es decir de arriba a abajo y de izquierda a derecha.

Para que el procesador ejecute una varias acciones de forma consecutiva:

- Se escriben los nombres de las acciones
  - Una a continuación de otra.
  - En líneas separadas.
  - Si la especificación de la acción requiere varias líneas, en varias líneas, por legibilidad.
    - Excepto palabras reservadas, constantes de tipo carácter o cadenas de caracteres.

```
cin»x;
y=x*10;
cout«y;
```

Este fragmento de código pide un valor por el teclado, se multiplica por 10 ese valor, se almacena en otra variable y se muestra el valor por la pantalla.

- **Composición secuencial** es una lista de acciones o composiciones de acciones que se ejecutarán una tras otra.
- **Bloque** es una lista de definiciones de variables,de definiciones de constantes y de composiciones secuenciales encerrada entre dos llaves (la primera abierta y la segunda cerrada). Por ejemplo:

```
{
int a=1,b;
a++;
b=a*a+1;
}
```

Un programa sencillo en C++ consiste en un bloque, precedido de ciertas definiciones y otros elementos que se explicarán más adelante.
Ejemplo: conversión de grados centígrados a Farenheit.

```
#include<iostream>
using namespace std;
int main()
{
const float factor=1.8, hielo_F=32;
float grados_C,grados_F;
cin>>grados_C;
grados_F=grados_C*factor+hielo_F;
cout<<grados_F;
}
```

- Necesario para poder usar `cin` y `cout`
```
#include<iostream>
using namespace std;
```
- Se definen las constantes y variables
```
const float factor=1.8, hielo_F=32;
float grados_C,grados_F;
```
- El procesador solicita al usuario el valor de la temperatura en grados centígrados y almacena su valor en la variable `grados_C`
```
cin>>grados_C;
```
- Se realiza la conversión y el resultado se guarda en la variable `grados_F`
```
grados_F=grados_C*factor+hielo_F;
```
- Se muestra en el monitor el valor de `grados_F`
```
cout<<grados_F;
```

La composición de acciones no es conmutativa, las mismas acciones escritas en un orden distinto pueden producir un resultado distinto. Ejemplo:

```
#include<iostream>
using namespace std;
int main()
{
int a;
a=1;
a=a+1;
a=a*2;
cout<<a;
}
```

Muestra 4

```
#include<iostream>
using namespace std;
int main()
{
int a;
a=1;
a=a*2;
a=a+1;
cout<<a;
}
```

Muestra 3

- En C++ pueden declararse variables dentro de cualquier bloque.
- Las variables declaradas dentro de un bloque sólo pueden usarse dentro de ese bloque.
- Se destruyen al abandonar ese bloque.
- Si dentro de un bloque se declara una variable de mismo nombre que otra declarada *antes* y *fuera* de ese bloque, la segunda declaración ensombrece a la primera.
- Recuérdese esto cuando se usen las composiciones que se verán a continuación.

- Permite ejecutar un bloque u otro en función del valor de una expresión.
- Existen tres tipos de composiciones condicionales:
  - Alternativa simple
  - Alternativa doble
  - Alternativa múltiple.

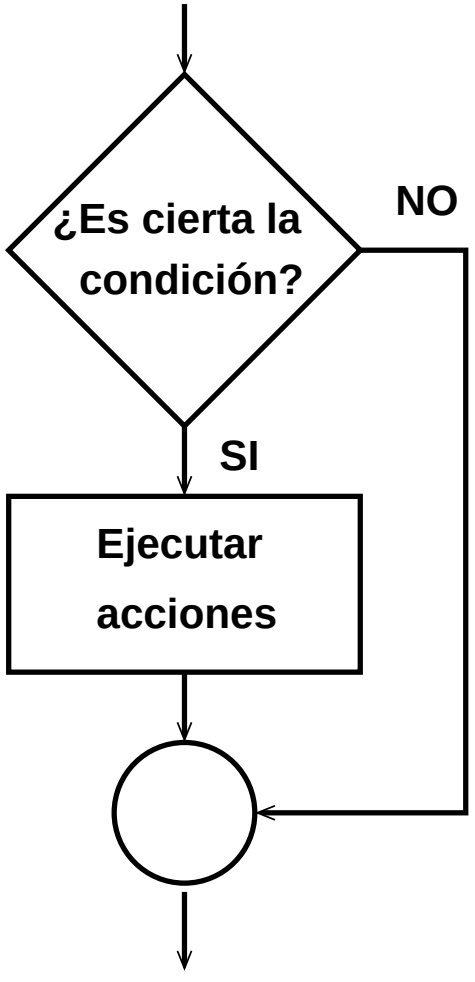
- **Alternativa simple:**
  - La expresión es de tipo booleano o se convierte a booleano.
  - Si la condición es cierta se ejecuta la acción o bloque que va a continuación.
  - De lo contrario el programa continúa después de la acción o bloque.

La sintaxis es como sigue:

```
if(condicion) accion
```

o bien:

```
if(condición) bloque
```

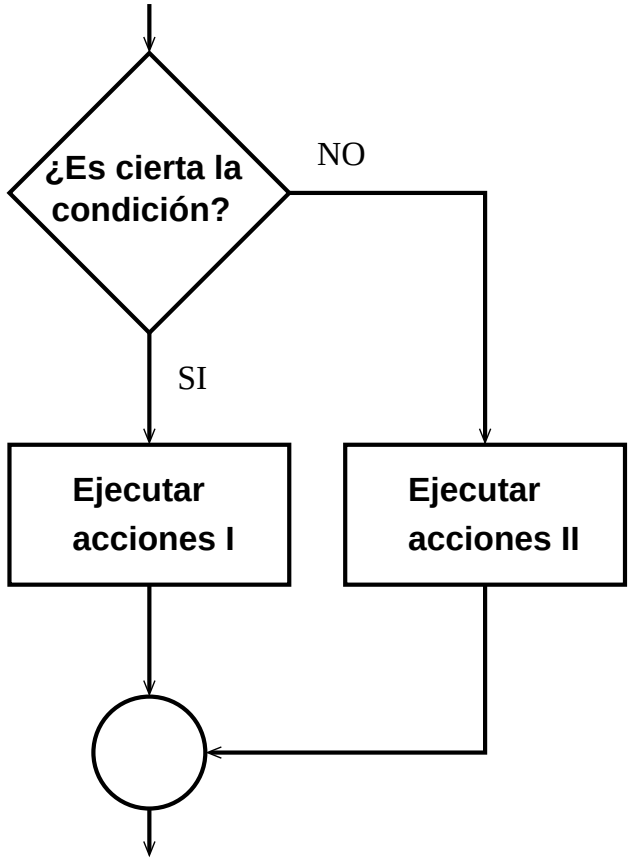


Ejemplo: ordenar dos números de mayor a menor.

- Si están desordenados hay que intercambiarlos.
- Si están ordenados no hay que hacer nada.

```
#include<iostream>
using namespace std;
int main()
{
int a,b,tmp;
cin>>a>>b;
//ordenar dos numeros
//de mayor a menor
if (a<b)
   {
   tmp=a;
   a=b;
   b=tmp;
   }
cout<<a<<' '<<b;
}
```

- **Alternativa doble:**
  - La expresión es de tipo booleano,
  - Si la condición es cierta se ejecuta el bloque o acción que va a continuación
  - En caso contrario se ejecuta el bloque o acción que va a continuación de la palabra `else`.



Sintaxis:

```
if(condicion) accion|bloque
else accion|bloque
```

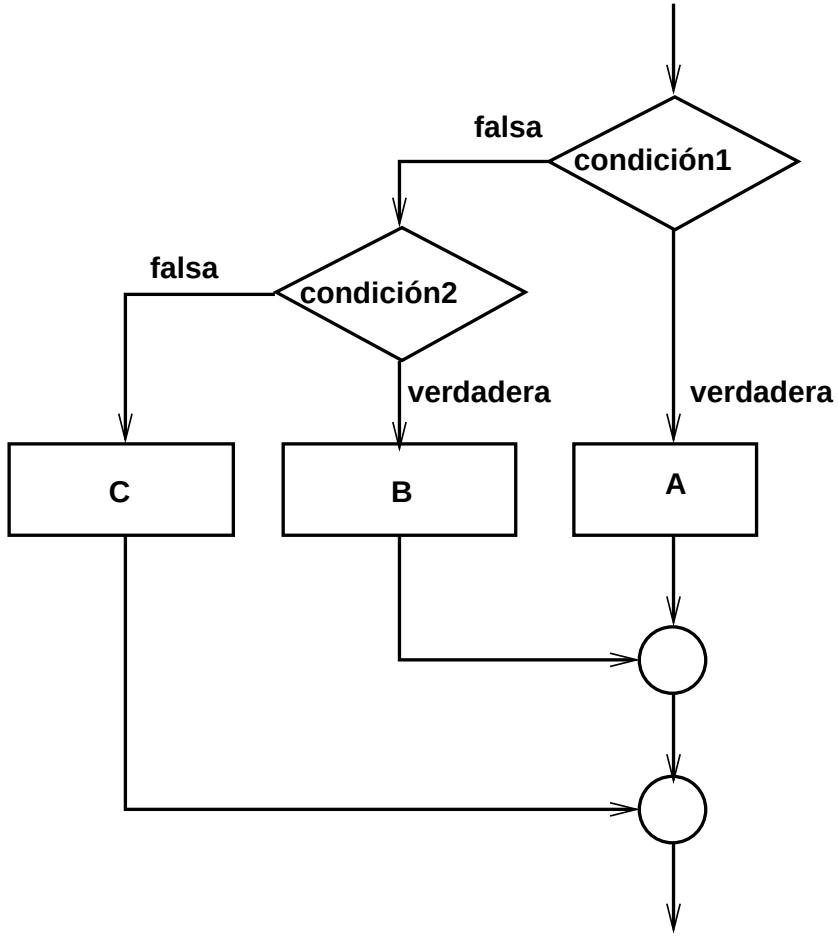
NOTA: El signo '|' significa que puede utilizarse cualquiera de las posibilidades.

Ejemplo: mostrar el mayor de dos números.

```
#include<iostream>
using namespace std;
int main()
{
int a,b,mayor;
cout<<"Introduce dos numeros:";
cin>>a>>b;
if (a>b)
   mayor=a;
else
   mayor=b;
cout<<"El mayor es:"<<mayor<<endl;
}
```

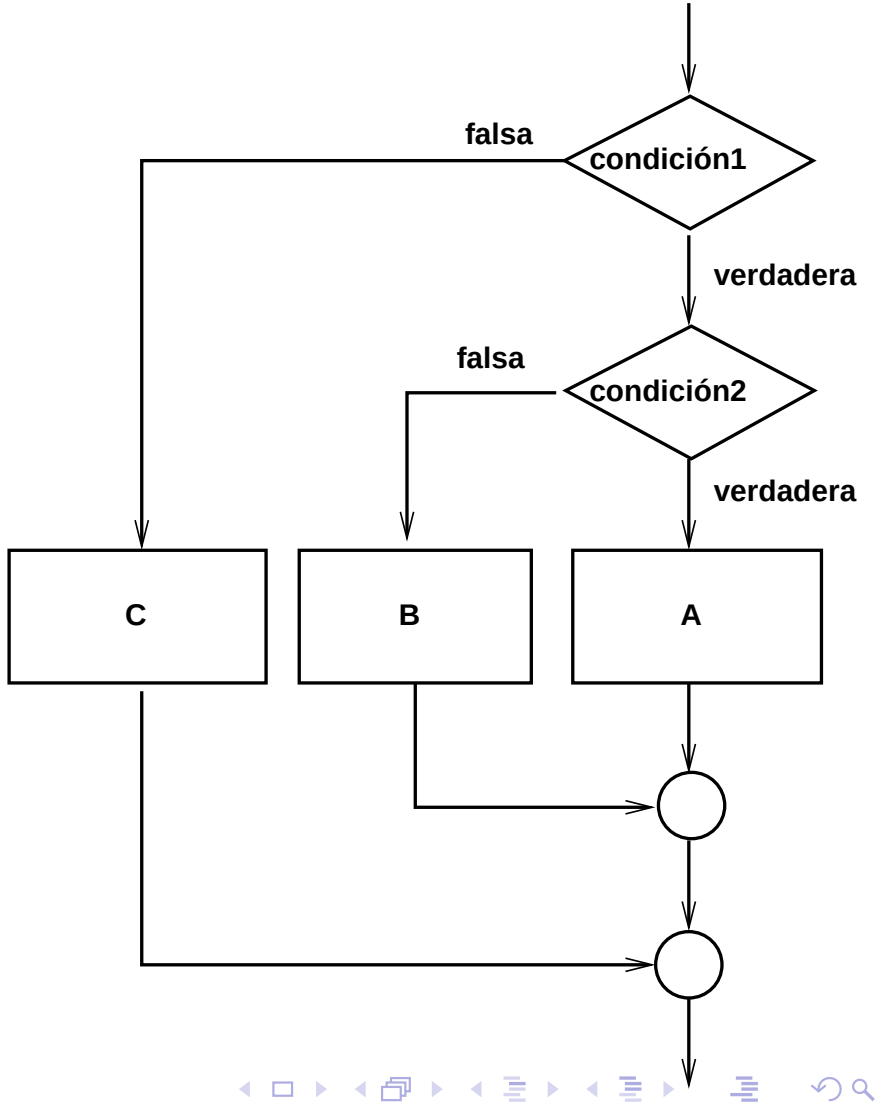
Entre las sentencias después de `if` o `else` puede haber más `if` e `if-else`. Por el lado del `else`:

```
if (condicion1)
   {
   //A...
   }
else
   if (condicion2)
      {
      //B...
      }
   else
      {
      //C...
      }
```



Entre las sentencias después de `if` o `else` pueden haber más `if` e `if-else`. Por el lado del `if`:

```
if (condicion1)
   {
   if (condicion2)
      {
      //A...
      }
   else
      {
      //B...
      }
   }
else
   //C...
```



Entre las sentencias después de `if` o `else` pueden haber más `if` e `if-else`. Por los dos lados:

```
if (condicion1)
   {
   if (condicion2)
      {
      ...
      }
   else
      {
      ...
      }
   }
```

```
else
   {
   if (condicion3)
      {
      ...
      }
   else
      {
      ...
      }
   }
```

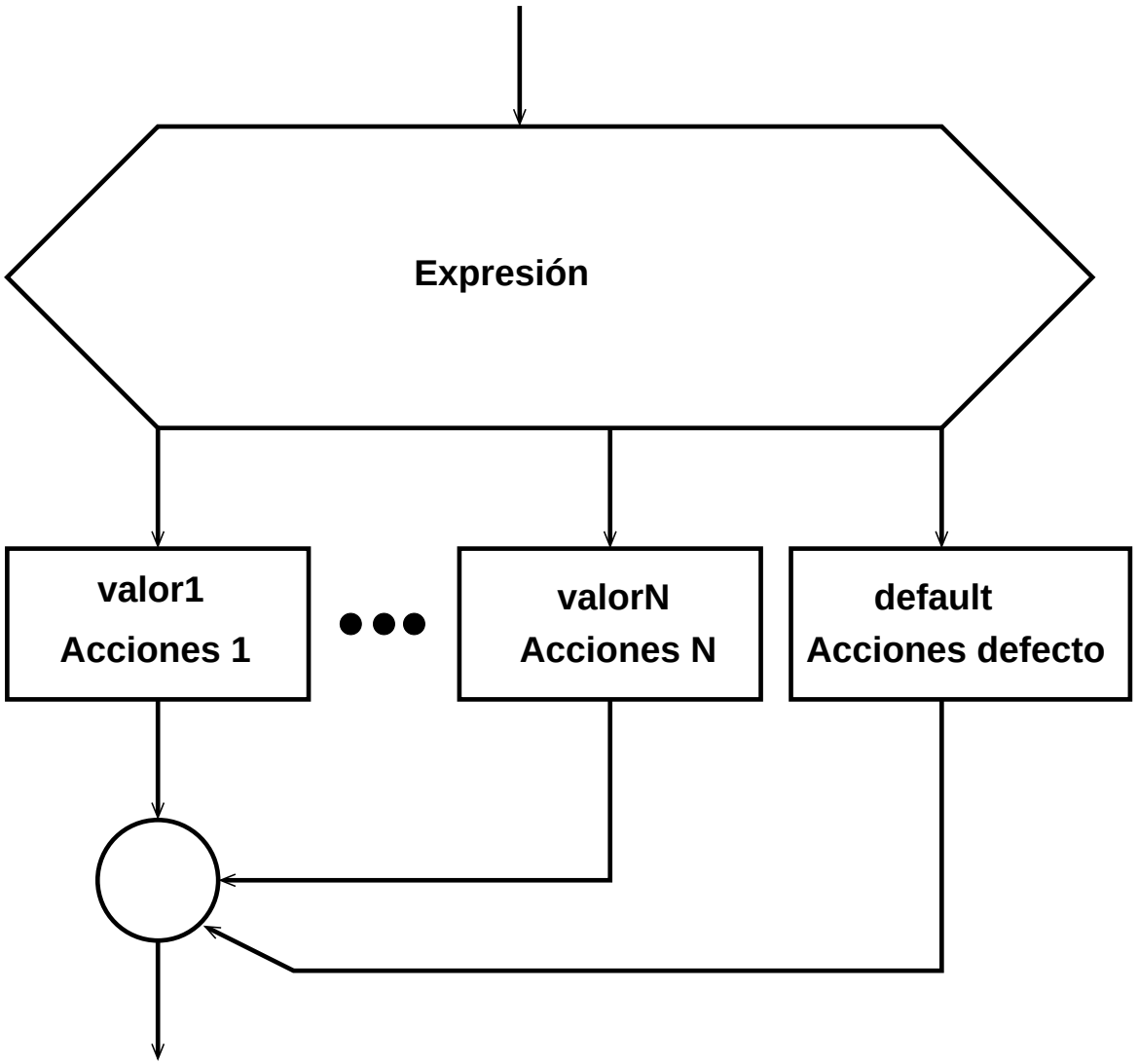
En caso de ambigüedad, cada `else` se asocia con el `if` mas cercano en sentido *ascendente* en el código:

```
if (...) //mas lejano
   ...
if (...) //mas cercano
   ...
else     //se asocia con el mas cercano
   ...
```

- Alternativa múltiple:
  - Similar a una composición secuencial de composiciones alternativas simples.
  - La condición se reduciría a la comparación a igualdad con una constante de tipo entero, enumerado o carácter.
  - En lugar de existir una condición, se tiene una expresión de tipo entero.
  - A continuación figuran una serie de composiciones secuenciales etiquetadas con constantes de tipo entero, enumerado o carácter.
  - De forma *implícita* se compara el valor de la expresión con cada constante.
  - Se ejecuta la composición secuencial etiquetada la constante igual al valor de la expresión.
  - Si no coincide con ninguna, se ejecuta la composición secuencial identificada con la palabra `default` (opcional).

La sintaxis de la alternativa múltiple en C++ es:

```
switch (expresion)
{
case valor1:
  composición secuencial 1;
  break;
case valor2:
  composición secuencial 2;
  break;
...
default:
  composición secuencial por defecto;
}
```


Expresión
valor1
Acciones 1
valorN
Acciones N
default
Acciones defecto

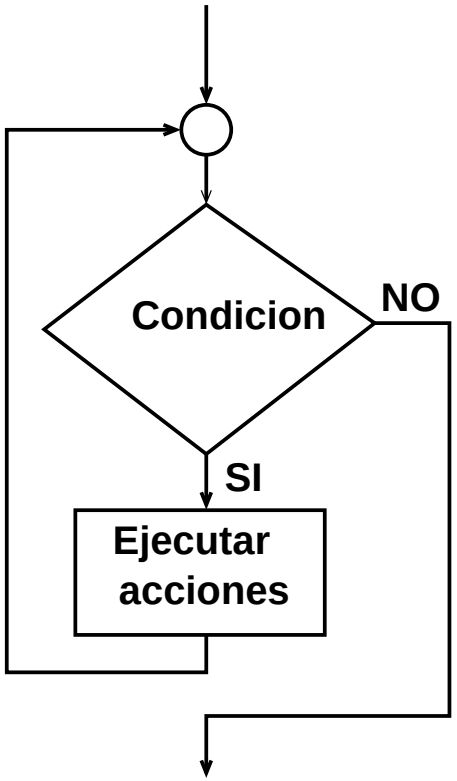
Cadena `if else` equivalente

```
if (expresion==valor1)
   {
   composicion secuencial 1
   }
else
   if (expresion==valor2)
      {
      composicion secuencial 2
      }
   ...
   else
      {
      composicion secuencial por defecto
      }
```

```
#include<iostream>
using namespace std;
int main()
{
    float a,b,res;
    bool ok=true;
    char op;
    printf("\nIntroduce  num op num:");
    cin>>a>>op>>b;
    switch(op)
        {
        case '+':res=a+b;
                 break;
        case '-':res=a-b;
                 break;
        case '*':res=a*b;
                 break;
        case '/':res=a/b;
                break;
        default: ok=false;
        }
   if(ok)
     cout<<res<<endl;
   else
     cout<<"error"<<endl;
}
```

- La composición iterativa ejecuta una acción o bloque un cierto número de veces:
  - **Mientras** el valor de una expresión sea `true`.
- Hay tres composiciones iterativas:
  - La composición `while`.
  - La composición `do while`.
  - La composición `for`.

- Composición `while`:
- Repite una acción o bloque mientras una condición sea cierta.
- **En primer lugar** se evalúa la condición.
- Si es cierta:
  - Se ejecuta la acción o bloque que va a continuación.
  - Se vuelve a evaluar la condición.
- Si no es cierta:
  - Se continúa después del bloque o acción que abarca el `while`



La sintaxis de `while` es:

```
while (condicion)
    accion|bloque
```

IMPORTANTE:

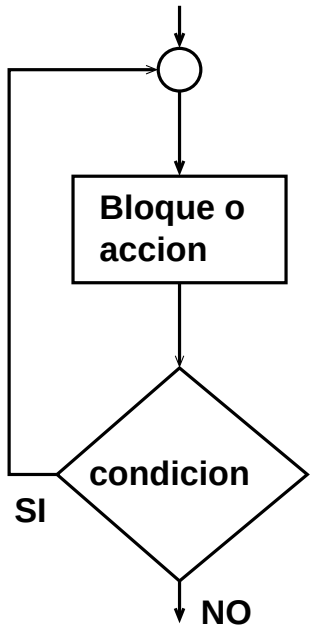
- La primera vez que se evalúe `condicion` tienen que tener valor definido todas las variables que intervengan en ella. De momento, tener valor definido es:
  - Que se haya pedido por el teclado.
  - Que haya estado a la izquierda de una asignación.
  - Esto tiene que hacerse *antes* del bucle.
- Si la primera vez que se evalúa `condicion` es falsa, no se realiza ninguna iteración.
- De entre las sentencias que se repiten, algunas de ellas tienen que modificar alguna variable de las que conforman `condicion`, haciéndola falsa en algún momento.
  - De lo contrario el bucle no termina.

Ejemplo: suma de las cifras de un número

```
#include<iostream>
using namespace std;
int main()
{
    //declaracion de variables, inicializar suma
    int cifra,n,suma=0;
    cout<<"Introduce un entero:";
    cin>>n;
    //mientras n!=0 quedan cifras por extraer
    while(n!=0)
        {
            //se extrae la cifra
            cifra=n%10;
            //se suma la cifra
            suma=suma+cifra;
            //se elimina la cifra del numero
            n=n/10;
        }
    cout<<endl<<"Suma de las cifras:"<<suma;
}
```

Funcionamiento del programa:
`int cifra,n,suma=0;` suma se inicializa a cero
Se muestra:
`Introduce un entero:`
Supongamos que el usuario introduce `143`
`n!=0` es `true` porque n vale `143` se entra en el bucle
`cifra=n%10;` `cifra` vale `3`
`suma=suma+cifra;` `suma` vale `3`
`n=n/10;` n vale `14`
`n!=0` es `true` porque n vale `14` se entra en el bucle
`cifra=n%10;` `cifra` vale `4`
`suma=suma+cifra;` `suma` vale `7`
`n=n/10;` n vale `1`
`n!=0` es `true` porque n vale `1` se entra en el bucle
`cifra=n%10;` `cifra` vale `1`
`suma=suma+cifra;` `suma` vale `8`
`n=n/10;` n vale `0`
`n!=0` es `false` porque n vale `0` termina el bucle
Se muestra `8` por la pantalla

- Composición `do while`:
- Repite una acción o bloque mientras una condición sea cierta.
- **En primer lugar** se ejecuta la acción o bloque que va a continuación del `do`
- **Después** se evalúa la condición.
- Si es cierta:
  - Se ejecuta la acción o bloque que va a continuación del `do`.
  - Se vuelve a evaluar la condición.
- Si no es cierta:
  - Se continúa después del bloque o acción que abarca el `do while`



Sintaxis:

```
do
   accion|bloque
while (condicion);
```

IMPORTANTE:

- La primera vez que se evalúe `condicion` tienen que tener valor definido todas las variables que intervengan en ella. De momento, estar definida es:
  - Que se haya pedido por el teclado.
  - Que haya estado a la izquierda de una asignación.
  - Esto *puede* hacerse durante la primera iteración.
- Como primero se ejecutan las sentencias, *al menos* se realiza una iteración.
- De entre las sentencias que se repiten, algunas de ellas tienen que modificar alguna variable de las que conforman `condicion`, haciéndola falsa en algún momento.
  - De lo contrario el bucle no termina.

Ejemplo: Cálculo de la raíz cuadrada de un número *a* mediante la sucesión $x\_i = \frac{x_{i-1} + \frac{a}{x_{i-1}}}{2}$

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
    float x_i,x_i_1=.5,a,dif,precision;
    cout<<"Introduce un num y precision:";
    cin>>a>>precision;
    do{
        //siguiente termino
        x_i=.5*(x_i_1+a/x_i_1);
        //diferencia entre dos consecutivos
        dif=fabs(x_i_1-x_i);
        //actualizacion del anterior
        x_i_1=x_i;
        //mientras la dif  mayor que precision
    }while(dif>precision);
cout<<"Raiz cuadrada="<<x_i<<endl;
}
```

Funcionamiento del programa:
`float x_i,x_i_1=.5,a,dif,precision;` `x_i_1` vale `0.5` Se muestra:
`Introduce un num y precision:`
Supongamos que el usuario/a teclea:
`4 0.1`
`x_i=.5*(x_i_1+a/x_i_1);` `x_i` vale `4.25`
`dif=fabs(x_i_1-x_i);` `dif` vale `3.75`
`x_i_1=x_i;` `x_i_1` vale `4.25`
`dif>precision` es `true` el bucle sigue
`x_i=.5*(x_i_1+a/x_i_1);` `x_i` vale `2.59559`
`dif=fabs(x_i_1-x_i);` `dif` vale `1.65441`
`x_i_1=x_i;` `x_i_1` vale `2.59559`
`dif>precision` es `true` el bucle sigue
`x_i=.5*(x_i_1+a/x_i_1);` `x_i` vale `2.06833`
`dif=fabs(x_i_1-x_i);` `dif` vale `0.527256`
`x_i_1=x_i;` `x_i_1` vale `2.06833`
`dif>precision` es `true` el bucle sigue
`x_i=.5*(x_i_1+a/x_i_1);` `x_i` vale `2.00113`
`dif=fabs(x_i_1-x_i);` `dif` vale `0.0672038`
`x_i_1=x_i;` `x_i_1` vale `2.00113`
`dif>precision` es `false` el bucle termina
Se muestra:
`Raiz cuadrada=2.00113`

- Composición `for`
- Es una forma compacta de escribir el bucle `while` para ciertos usos, por ejemplo para realizar un número fijo de iteraciones.

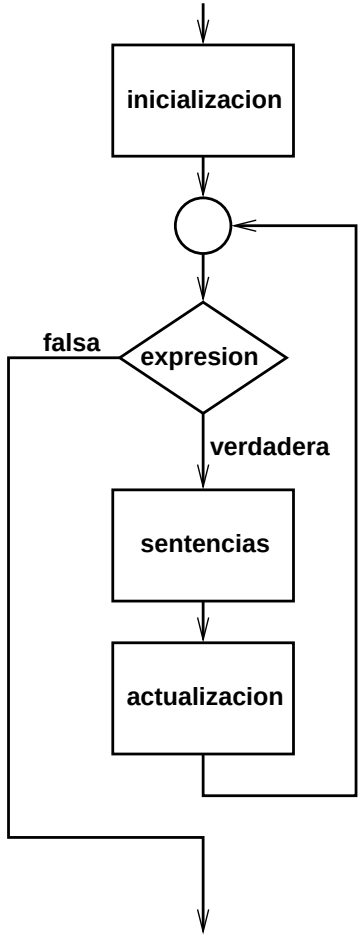
```
for(inicializacion;expresion;actualizacion)
   accion;

for(inicializacion;expresion;actualizacion)
   {
   //bloque
   ...
   }
```

- La `inicializacion` se hace una sola vez.
  - Consiste en al menos una inicialización de las variables que intervienen en `expresion`.
- Después se evalúa `expresion`, si es cierta, se ejecutan las sentencias.
- Finalmente se realiza la `actualizacion`.
  - Consiste en al menos una asignación en las que se cambia el valor de alguna variable de las que intervienen en `expresion`.

Equivale a un `while` con la siguiente estructura.

```
inicializacion;
while(expresion)
    {
    sentencias;
    actualizacion;
    }
```



- Si hay varias inicializaciones se separan por comas.
- Todas las variables de `expresion` deben de tener valor definido la primera vez que se evalúe.
- Si hay varias actualizaciones se separan por comas.
  - Algunas deben hacer que `expresion` se haga falsa. De lo contrario el bucle no termina.
- En este curso casi siempre será suficiente una inicialización y una actualización.

Ejemplo: cálculo del factorial

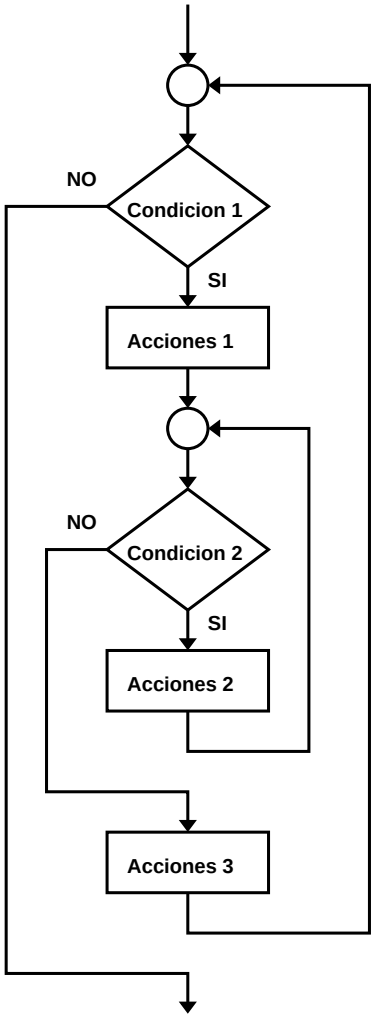
```
#include<iostream>
using namespace std;
int main()
{
  int n,fact=1,i;
  cout<<"Numero?";
  cin>>n;
  for (i=2;i<=n;i++)
    fact=fact*i;
  cout<<fact<<endl;
}
```

```
#include<iostream>
using namespace std;
int main()
{
  int n,fact=1,i;
  cout<<"Numero?";
  cin>>n;
  i=2;
  while(i<=n)
    {
    fact=fact*i;
    i++;
    }
  cout<<fact<<endl;
}
```

`int n,fact=1,i;` `fact` vale 1
Se pide n por el teclado, supongamos que se teclea 6
`Introduce un numero:6`
`i=2;` `i` vale 2
`i<=n` es `true` se entra en el bucle
`fact=fact*i;` `fact` vale 2
`i++;` `i` vale 3
`i<=n` es `true` el bucle sigue
`fact=fact*i;` `fact` vale 6
`i++;` `i` vale 4
`i<=n` es `true` el bucle sigue
`fact=fact*i;` `fact` vale 24
`i++;` `i` vale 5
`i<=n` es `true` el bucle sigue
`fact=fact*i;` `fact` vale 120
`i++;` `i` vale 6
`i<=n` es `true` el bucle sigue
`fact=fact*i;` `fact` vale 720
`i++;` `i` vale 7
`i<=n` es `false` el bucle termina
Se muestra:
`720`

De entre las acciones que se repiten en un bucle algunas pueden ser otros bucles.

```
while(condicion 1)
    {
    acciones 1
    while (condicion 2)
        {
        acciones 2
        }
    acciones 3
    }
```



Ejemplo: mostrar la suma de las cifras de cada número entre 10 y 30, ambos inclusive.

```
#include<iostream>
using namespace std;
int main()
{
  int cifra,n,num,suma;
  for (num=10;num<=30;num++)
    {
      n=num;
      suma=0;
      while(n!=0)
        {
          cifra=n%10;
          suma=suma+cifra;
          n=n/10;
        }
      cout<<endl<<"Suma cifras "<<
      num<<'='<<suma;
    }
}
```

```
Suma cifras 10=1
Suma cifras 11=2
Suma cifras 12=3
Suma cifras 13=4
Suma cifras 14=5
Suma cifras 15=6
Suma cifras 16=7
Suma cifras 17=8
Suma cifras 18=9
Suma cifras 19=10
Suma cifras 20=2
Suma cifras 21=3
Suma cifras 22=4
Suma cifras 23=5
Suma cifras 24=6
Suma cifras 25=7
Suma cifras 26=8
Suma cifras 27=9
Suma cifras 28=10
Suma cifras 29=11
Suma cifras 30=3
```

- La ejecución de `break` hace que el bucle en el que está contenido termine, independientemente del valor de al condición.
  - Sólo se puede usar en composiciones iterativas o en alternativa múltiple **NO** en `if` o `if else`
  - Si existen bucles anidados sólo se termina el bucle en el que esté `break` **NO** el/los que lo puedan contener.
- La ejecución de `continue` hace que se salte desde esa instrucción hasta la evaluación de la condición del bucle.
  - Sólo puede utilizarse en composiciones iterativas.