

# INSTRUCCIONES DE CONTROL

## CONTENIDOS

- 1. Introducción a las sentencias de control.**
- 2. Instrucciones o sentencias condicionales: IF, IF-ELSE, SWITCH.**
- 3. Instrucciones o sentencias repetitivas o iterativas:  
WHILE, DO-WHILE, FOR.**
- 4. Sentencias BREAK y CONTINUE.**

# INSTRUCCIONES DE CONTROL

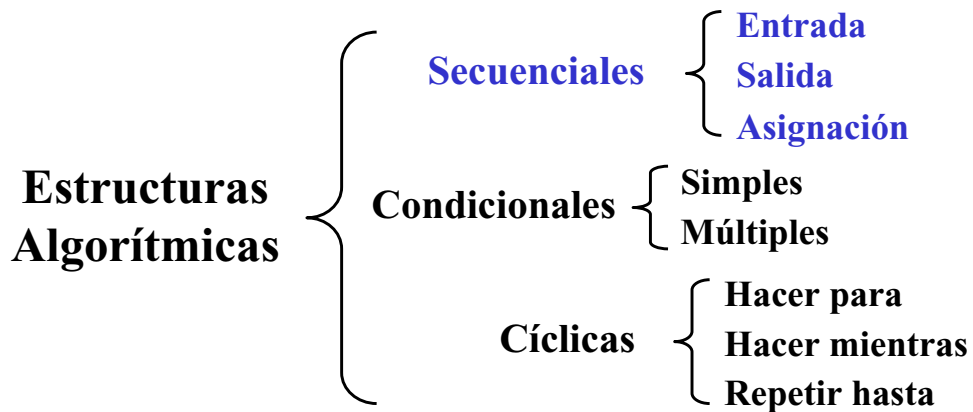
## 1

### **Introducción a las sentencias de control**

# INSTRUCCIONES DE CONTROL

## Introducción

Hasta ahora solo hemos visto la codificación en el lenguaje C++ de las estructuras secuenciales, lo que nos proporciona programas lineales, es decir, comienzan por la primera instrucción y acaban por la última, ejecutándose todas una sola vez.



3

# INSTRUCCIONES DE CONTROL

Esta forma de programación sólo me permite resolver problemas sencillos.

- ➔ Para resolver problemas más complejos, nos puede interesar que dependiendo de los valores de los datos, se ejecuten unas instrucciones u otras. Las **instrucciones condicionales** nos van a permitir representar éste tipo de comportamiento.

**Sentencias IF y SWITCH**

- ➔ En otro casos, nos encontraremos con la necesidad de repetir una instrucción ó instrucciones un número determinado de veces. En éstos casos utilizaremos **instrucciones de control iterativas ó repetitivas** (ciclos).

**Sentencias WHILE, DO-WHILE y FOR**

4

## 2

### Instrucciones o sentencias condicionales IF, IF-ELSE, SWITCH

5

#### Instrucciones Condicionales

##### Instrucción condicional simple: IF

Se corresponde con la estructura algorítmica

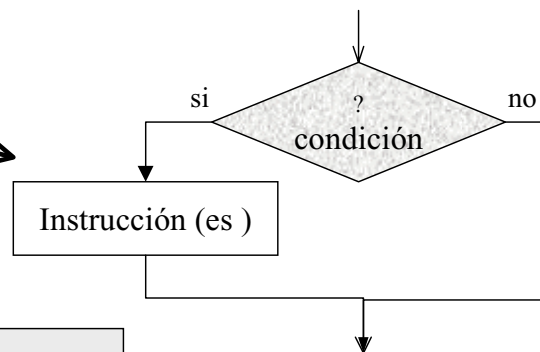
El formato general de una sentencia **if** es la siguiente:

```
if (condición)
{
    instrucción 1;
    ...
    instrucción n;
}
```

Atención !!!

```
if (condición)
    instrucción;
```

Si se cumple la condición, entonces se ejecuta la *instrucción* ó el *bloque de instrucciones*; en caso contrario, no se ejecutan.



6

# Instrucciones Condicionales

## Instrucción condicional simple: IF

Ejemplos:

```
#include <iostream.h>
int main()
{
    int a, x = 0, y;
    cin >> a;

    if (a==0)
        x = 1;

    cout << x;
    return 0;
}
```

```
#include <iostream.h>
int main()
{
    ...

    if (cantidad > 100)
    {
        descuento = 0.2;
        precio = n*descuento;
    }

    ...
    return 0;
}
```

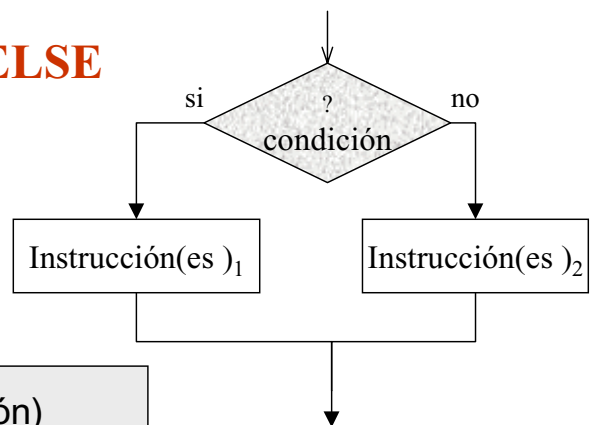
7

# Instrucciones Condicionales

## Instrucción condicional doble : IF-ELSE

Estructura algorítmica asociada:

Formato general de la  
sentencia **if-else**



```
if (condición)
{
    varias instrucciones 1;
}
else
{
    varias instrucciones 2;
}
```

```
if (condición)
    instrucción 1;
else
    instrucción 2;
```

Si se cumple la condición, se ejecutan las instrucciones del primer bloque;

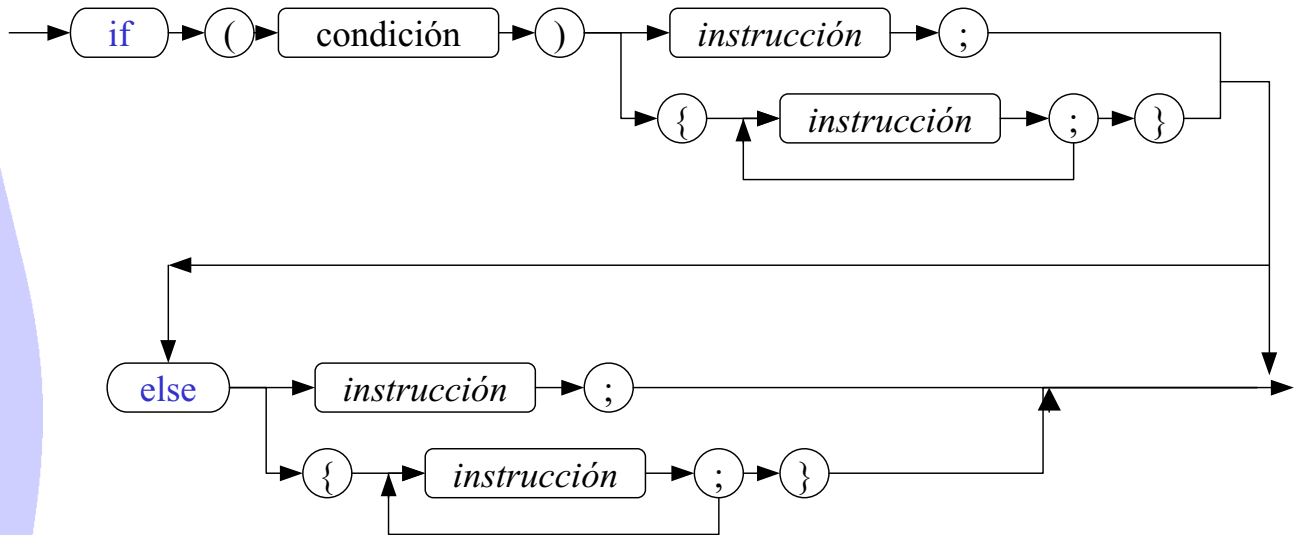
si no, se ejecutan las instrucciones del segundo bloque.

8

# Instrucciones Condicionales

## Instrucción condicional IF, IF-ELSE

Diagrama sintáctico:



9

# Instrucciones Condicionales

## Instrucción condicional doble : IF-ELSE

Ejemplos:

```
#include <iostream.h>
int main()
{
    int a, x;
    cin >> a;

    if (a==0)
        x = 1;
    else
        x= 0;

    cout << x;
    return 0;
}
```

```
#include <iostream.h>
int main()
{
    ...

    if (cantidad > 100)
    {
        descuento = 0.2;
        precio = n*descuento;
    }
    else
        precio = n;

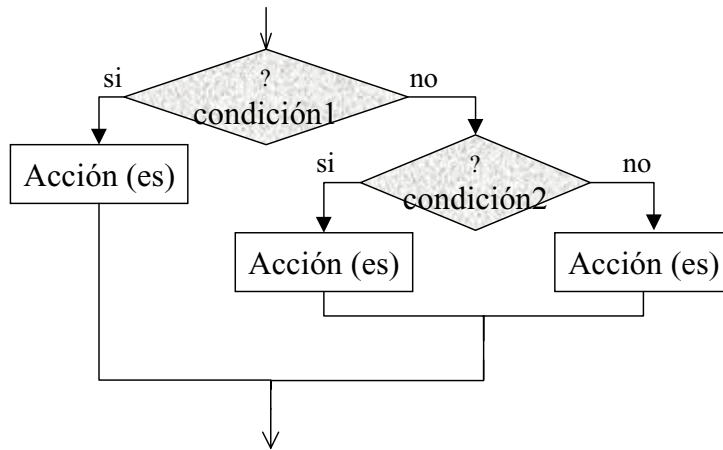
    ...
    return 0;
}
```

10

# Instrucciones Condicionales

## Instrucción condicional doble : IF-ELSE anidadas

Hemos visto dos tipos de instrucciones condicionales, con una o dos alternativas. Podemos utilizar las instrucciones IF-ELSE anidadas, es decir, que alguna de las ramas sea a su vez otra instrucción IF-ELSE. Así podemos implementar decisiones que implican más de dos alternativas.



11

# Instrucciones Condicionales

## Instrucción condicional doble : IF-ELSE anidadas

```
if (condición1)
    instrucción 1;
else
    if (condición2)
        instrucción 2;
    else
        instrucción 3;
```

La escritura en sangría facilita la comprensión del código así como el chequeo del buen funcionamiento.

La sintaxis de instrucciones IF-ELSE anidadas

```
if (condición1)
    instrucción 1;
else
    if (condición2)
        instrucción 2;
    else
        if (condición3)
            instrucción 3;
        else
            instrucción 4;
```

12

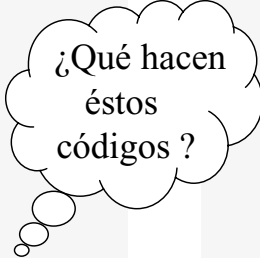
# Instrucciones Condicionales

```
#include <iostream.h>
int main()
{
    int a, b, c, max;
    cin >> a >> b >> c;

    if (a > b)
        if (a > c)
            cout << a;
        else
            cout << c;
    else
        if (b > c)
            cout << b;
        else
            cout << c;

    return 0;
}
```

A



```
#include <iostream.h>
int main()
{
    int a, b, c, max;
    cin >> a >> b >> c;

    if (a > b)
        if (a > c)
            cout << a;
        else
            cout << c;
    else
        if (b > c)
            cout << b;
        else
            cout << c;
            cout << "fin del programa";
    return 0;
}
```

B

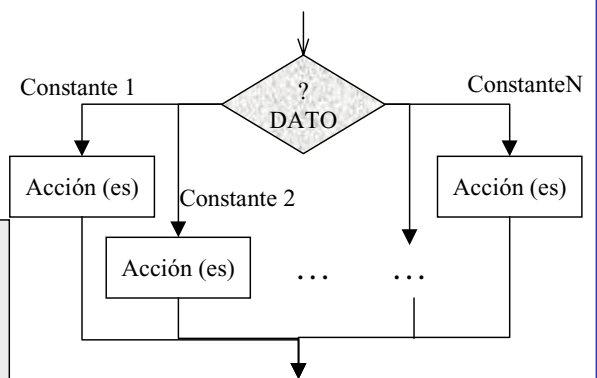
# Instrucciones Condicionales

## Instrucción condicional múltiple : SWITCH

Estructura algorítmica asociada

Formato general de la  
sentencia **witch**

```
switch (selector)
{
    case constante1:
        instrucción1 ó bloque de instrucciones
        break;
    case constante2:
        instrucción2 ó bloque de instrucciones
        break;
    default:
        instrucción2 ó bloque de instrucciones
}
```



Permiten comparar una 'variable' con distintos valores posibles, ejecutando para cada caso una serie de instrucciones específicas.

# Instrucciones Condicionales

## Instrucción condicional múltiple : **SWITCH**

Formato general de la  
sentencia **witch**

```
switch (selector)
{
    case constante1:
        instrucción1 ó bloque de instrucciones
        break;
    case constante2:
        instrucción2 ó bloque de instrucciones
        break;
    default:
        instrucción3 ó bloque de instrucciones
}
```

El valor de *selector* debe ser un número entero. Puede ser una variable, una expresión ó una llamada a una función.

Cada caso comienza con un **case** y termina con un **break**

¿Qué ocurre si se me olvida algún **break** ?

15

# Instrucciones Condicionales

## Instrucción condicional múltiple : **SWITCH**

```
#include <iostream.h>
int main ()
{
    cin>> num;
    switch (num)
    {
        case 1:
            cout << "Ha introducido el nº 1\n";
        case 2:
            cout << "Ha introducido el nº 2\n";
            break;
        default:
            cout << "Ha introducido otro nº";
    }
    return 0;
}
```

### Ejemplo

Si al ejecutar el programa introducimos un 2, obtenemos el mensaje:

**'Ha introducido el nº 2'**

Si al ejecutar el programa introducimos un 1, obtenemos el mensaje:

**'Ha introducido el nº 1'**  
**'Ha introducido el nº 2'**

16

## 3

### **Instrucciones o sentencias repetitivas o iterativas WHILE, DO-WHILE, FOR**

## Instrucciones Iterativas o repetitivas

### Instrucciones de control repetitivas

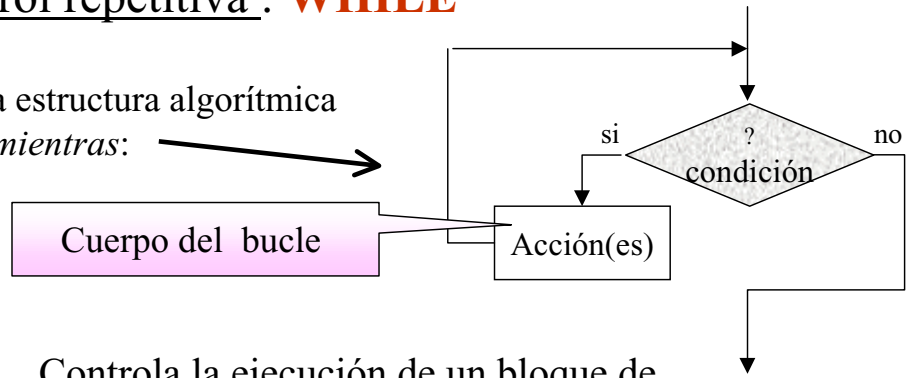
Son aquellas que controlan la repetición de un conjunto de instrucciones denominado bloque o *cuerpo del bucle*, mediante la evaluación de una condición o mediante un contador.

**Sentencias WHILE, DO-WHILE y FOR**

# Instrucciones Iterativas o repetitivas

## Instrucción de control repetitiva : **WHILE**

Se corresponde con la estructura algorítmica *hacer\_mientras*:



Formato general de la sentencia **while**

```
while ( condición )  
{  
    instrucción 1;  
    ...  
    instrucción n;  
}
```

Controla la ejecución de un bloque de **instrucciones** de tal forma que éstas **se ejecutan mientras se cumpla la condición**, que será evaluada siempre **antes** de cada repetición.

```
while ( condición )  
    instrucción;
```

Cada repetición del cuerpo del bucle se denomina *iteración*

# Instrucciones Iterativas o repetitivas

## Instrucción de control repetitiva : **WHILE**

Ejemplo:

```
#include <iostream.h>  
int main()  
{  
    ...  
    contador = 0;  
    while (contador < 100)  
    {  
        cout << "Hola";  
        contador ++;  
    }  
    ...  
    return 0;  
}
```

La condición será evaluada siempre antes de cada iteración.

El cuerpo del bucle while se repite mientras la condición sea cierta.

El cuerpo de un bucle **while** se ejecutará cero o más veces.

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : WHILE

#### Inicialización

Se realiza antes de la instrucción **while**

```
...
contador = 0;
while (contador < 100)
{
    cout << "Hola";
    contador ++;
}
...
```

cuerpo

#### Actualización

Se realiza dentro del cuerpo del bucle durante cada iteración

La variable que representa la condición del bucle se denomina *variable de control del bucle*.

Dicha variable debe ser:

- inicializada
- comprobada
- actualizada

#### Comprobación

Se comprueba el valor de la variable antes de comenzar la repetición

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : WHILE

```
int main()
{
    const bool continuar = true;
    bool a = true;
    int n;
    while (a ==continuar)
    {
        cin >> n;
        if (n<0)
        {
            cout << "No se admiten negativos";
            a = false;
        }
        else
            cout << "Muy bien, otro más: " ;
    }
    return 0;
}
```

Inicialización

Comprobación

Es importante comprobar que en algún momento, la condición del bucle se hace falsa, de forma que no obtengamos bucles infinitos.

Actualización

## Instrucciones Iterativas o repetitivas

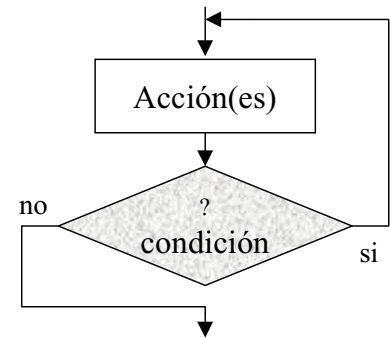
### Instrucción de control repetitiva : DO-WHILE

Se corresponde con la estructura algorítmica

Formato general de la  
sentencia **do-while**

```
do
{
    instrucción 1;
    ...
    instrucción n;
} while ( condición );
```

```
do
    instrucción;
while ( condición );
```



Se utiliza para ejecutar un bloque de instrucciones al menos una vez. El cuerpo del bucle **se repite mientras se verifica la condición**. Dicha condición será evaluada **después** de cada repetición.

23

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : DO-WHILE

Ejemplo:

```
#include <iostream.h>
int main()
{
    char n;
    do
    {
        cout << "introducir número: ";
        cin >> n;
        if ((isdigit(n)) == false)
            cout << "Solo se admiten números";
    } while ((isdigit(n)) != false);
    return 0;
}
```

El cuerpo de un bucle **do-while** se ejecutará una o más veces.

El cuerpo del bucle **do-while** se repite mientras la condición sea cierta.

La condición será evaluada siempre después de cada iteración.

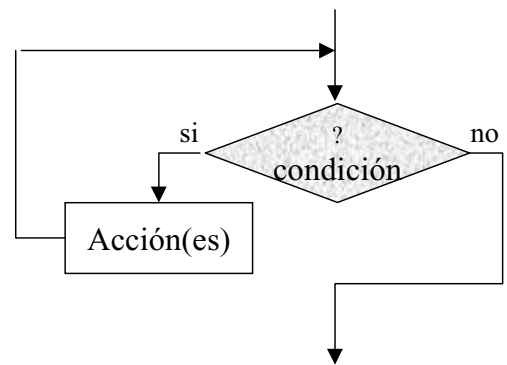
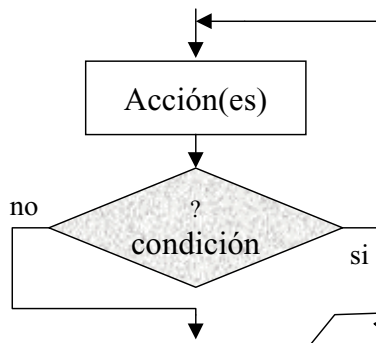
24

## Instrucciones Iterativas o repetitivas

### Diferencias entre las instrucciones

### **WHILE** y **DO-WHILE**

En **while** primero se tiene que cumplir la condición, y luego se ejecuta el bloque de instrucciones.



En **do-while** primero se ejecuta el bloque de instrucciones y luego se comprueba la condición.

## Instrucciones Iterativas o repetitivas

### Traducción de **WHILE** a **DO-WHILE**

```
while ( condición )  
{  
  instrucción 1;  
  ...  
  instrucción n;  
}
```

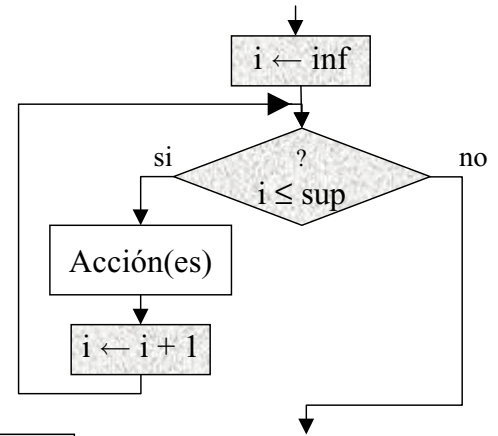
```
if (condición)  
do  
{  
  instrucción 1;  
  ...  
  instrucción n;  
} while ( condición );
```

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

Se utiliza para ejecutar un bloque de instrucciones un número fijo de veces que se conoce de antemano.

Se corresponde con la estructura algorítmica *hacer\_para*:



```
for ( inicialización ; condición ; actualización )  
{  
    instrucción 1;  
    ...  
    instrucción n;  
}
```

Formato general de la sentencia **for**

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

La instrucción **for** se vale de una variable de control del ciclo. El ciclo se repite desde el límite inferior, hasta que la variable de control llegue la límite superior.

```
for ( inicialización; condición; actualización )  
{  
    instrucción 1;  
    ...  
    instrucción n;  
}
```

Cabecera del **for**

Una diferencia con el bucle **while** es que en el **for**, las operaciones de control del bucle (inicialización, comprobación y actualización ) se realizan en un solo sitio: **la cabecera del for**.

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

En la cabecera del **for** nos podemos encontrar con 3 partes o secciones:

- **Parte de inicialización:** Inicializa la variable de control del bucle. Se pueden utilizar una o varias variables.
- **Parte de iteración:** Expresión lógica. El cuerpo del bucle se repite mientras la expresión sea verdadera.
- **Parte de incremento:** Incrementa o decrementa el valor de la variable o variables de control.

```
int main()
{
    ....
    for (int i = 0; i<10; i++)
    {
        cout << "Número: " << i ;
        cout << << endl;
    }
    ....
    return 0;
}
```

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

Ejemplo:

La variable de control es  $n$

```
int main()
{
    ....
    for (int n = 1; n<=10; n=n+2)
    {
        cout << "Número: " << n << "\t" << n*n ;
        cout << << endl;
    }
    ....
    return 0;
}
```

Es de tipo entero, su valor inicial es 1

Su valor final es 10

Se incrementa de 2 en 2.

Pasada	Valor de $n$
1	1
2	3
3	5
4	7
5	9

En la primera iteración ó pasada, el valor de  $n$  es 1, en la segunda pasada  $n$  vale 3. La última pasada se realiza cuando  $n$  vale 9.

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

Se puede inicializar más de una variable, se puede poner más de una condición ó más de un incremento.

```
int main()
{
    ....
    for ( int i = 0, j = 100; i<j ; i++, j--)
    {
        int k;
        k = i+2*j ;
        cout << k << endl;
    }
    ....
    return 0;
}
```

Se declaran 2 variables de control: *i* y *j*

Se inicializan a 0 y 100 respectivamente

El cuerpo se ejecutará mientras *i* sea menor que *j*

Se incrementa la variable *i* y se decrementa la variable *j*

31

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

```
int main()
{
    ....
    for ( int i = 0, j = 5; i+j <100 ; i++, j =j+5 )
    {
        cout << i <<"\t" <<j << (i+j) ;
        cout << endl;
    }
    ....
    return 0;
}
```

Valor de las variables de control en cada pasada

valor de <i>i</i>	valor de <i>j</i>	<i>i+j</i>
0	5	5
1	10	11
2	15	17
3	20	23
4	25	29
5	30	35
6	35	41
7	40	47
8	45	53
9	50	59
10	55	65
11	60	71
12	65	77
13	70	83
14	75	89
15	80	95

Se realizan un total de 16 pasadas.

Se ejecuta el cuerpo de la instrucción **for** mientras se cumpla la condición.

32

## Instrucciones Iterativas o repetitivas

### Instrucción de control repetitiva : FOR

Se puede omitir cualquiera de las partes de la cabecera del **for** (inicialización, condición o incremento).

```
int main()
{
    ....
    contador = 1;
    for ( ; contador < 10 ; )
    {
        cout << contador ;
        contador++ ;
    }
    ....
    return 0;
}
```

Se comporta igual que un while

```
int main()
{
    ....
    for ( ; ; )
    {
        ...
    }
    ....
    return 0;
}
```

Bucle infinito

## INSTRUCCIONES DE CONTROL

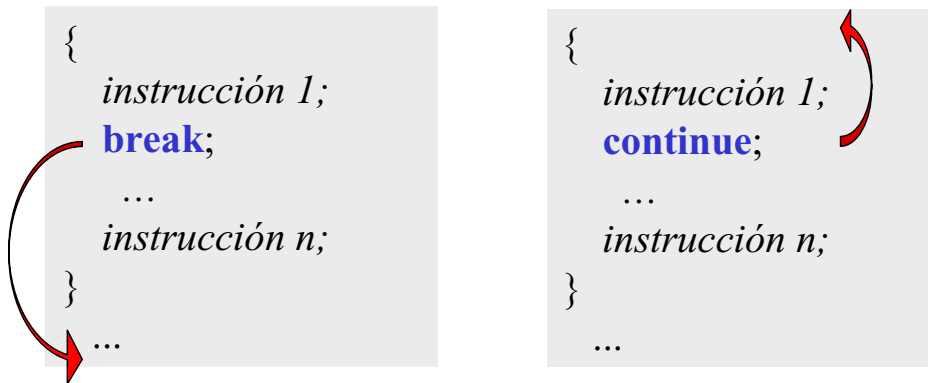
### 4

### Sentencias **BREAK** y **CONTINUE**

## Instrucciones Iterativas o repetitivas

### Instrucciones *break* y *continue*

La instrucción **break**, además de terminar los case de una instrucción **switch**, sirve para forzar la terminación inmediata de un bucle.



La instrucción **continue**, interrumpe la ejecución del cuerpo del bucle y fuerza una nueva iteración.

## Instrucciones Iterativas o repetitivas

### Instrucciones *break* y *continue*

```
int main()
{
    ....
    for ( int t = 0; t <100 ; t++ )
    {
        cout << t;
        if (t==10)
            break;    // salir del for
    }
    ....
    return 0;
}
```

A red arrow starts from the 'break;' statement and points downwards, indicating the loop's termination.

### Ejemplos

```
int main()
{
    ....
    for ( int t = 0; t <100 ; t++ )
    {
        if (t%10==0)
            continue;
        cout << t;
    }
    ....
    return 0;
}
```

A red arrow starts from the 'continue;' statement and points upwards, indicating the start of a new iteration.

# Instrucciones Iterativas o repetitivas

## Instrucciones *break* y *continue*

Ejemplo

```
int main()
{
    ....
    do
    {
        cin >> x;
        if ( x < 0 )
            continue;
        cout << x;
    }while (x !=100);
    ....
    return 0;
}
```

Ignora lo que sigue y vuelve a comprobar la condición.