

Escuela de Ingeniería en Informática e Ingeniería en Sistemas
Cátedra: Programación I y Lab. Programación I, Programación II

Ing. Yelenia Boada



Operadores relacionales (condiciones simples)

Expresiones lógicas (*booleanas*)

Operadores relacionales

Permiten hacer comparaciones (*condiciones*):

Condición ::= Expresión Operador_relacional Expresión

Las expresiones deben arrojar resultados del mismo tipo (comparables).

El resultado es de tipo **bool** (**true** o **false**).

<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual que
!=	distinto de

Operadores (prioridad)	Asociatividad
++ -- (postfijos) Llamadas a funciones	Izda. a dcha.
++ -- (prefijos) - (cambio de signo)	Dcha. a izda.
* / %	Izda. a dcha.
+ -	Izda. a dcha.
< <= > >=	Izda. a dcha.
== !=	Izda. a dcha.
= += -= *= /= %=	Dcha. a izda.

Expresiones lógicas (*booleanas*)

Operadores relacionales

```
bool resultado;  
int a = 2, b = 3, c = 4;  
resultado = a < 5;           // 2 < 5 ? true  
resultado = a * b + c >= 12; // 10 >= 12 ? false  
resultado = a * (b + c) >= 12; // 14 >= 12 ? true  
resultado = a != b;          // 2 != 3 ? true  
resultado = a * b > c + 5;    // 6 > 9 ? false  
resultado = a + b == c + 1;   // 5 == 5 ? true
```

Los operadores aditivos y multiplicativos tienen mayor prioridad.



Error común de programación:

Confundir el operador de igualdad (==) con el operador de asignación (=).

Fundamentos de la programación

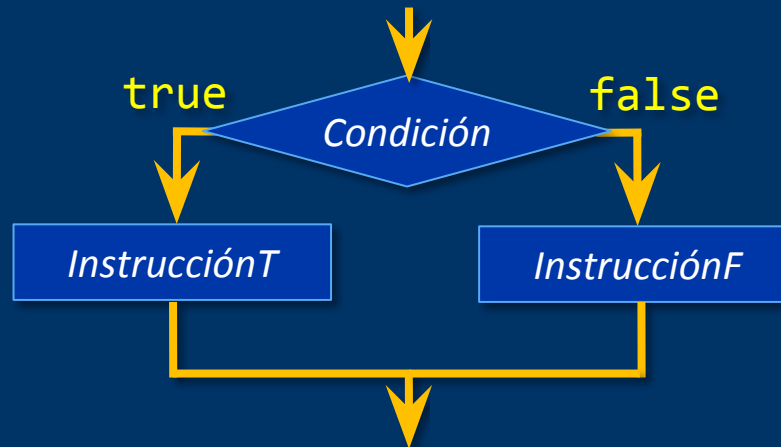
Toma de decisiones (if)

Hacer esto... o hacer esto otro...

Selección

Si la condición es cierta, hacer esto... ; si no, hacer esto otro...

Bifurcación condicional:



```
if (condición)
  •→ instrucciónT
else
  •→ instrucciónF
```

La instrucción if

Selección

seleccion.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int op1 = 13, op2 = 4;
    int opcion;
    cout << "1 - Sumar" << endl;
    cout << "2 - Restar" << endl;
    cout << "Opcion: ";
    cin >> opcion;
    if (opcion == 1)
        cout << op1 + op2 << endl;
    else
        cout << op1 - op2 << endl;

    return 0;
}
```

```
D:\FP\Tema2>seleccion
1 - Sumar
2 - Restar
Opcion: 1
17
```

```
D:\FP\Tema2>seleccion
1 - Sumar
2 - Restar
Opcion: 2
9
```

Fundamentos de la programación

Bloques de código

Bloques de código

¿Hay que hacer más de una cosa?

Si en alguna de las dos ramas del `if` es necesario poner varias instrucciones, se necesita crear un bloque de código:

```
{  
  Tab 0 | instrucción1  
  2 ó 3 → | instrucción2  
  esp. | ...  
        | instrucciónN  
}
```



Cada instrucción simple irá terminada en ;

Por ejemplo: `int num, total = 0;`

```
cin >> num;
```

```
if (num > 0)
```

```
{  
    cout << "Positivo";  
    total += num;  
}
```

```
cout << endl;
```

Cada bloque crea un nuevo ámbito en el que las declaraciones son locales.

Bloques de código

Posición de las llaves: cuestión de estilo

```
if (num > 0)    if (num > 0) {
{      cout << "Positivo";
    cout << "Positivo";    total += num;
    total += num;    }
}    cout << endl;
cout << endl;
```

No te dejes engañar

```
if (num > 0)    if (num > 0)
    cout << "Positivo";    cout << "Positivo";
    total += num;    total += num;
```

Aunque la sangría pueda dar a entender, a simple vista, que las dos instrucciones se ejecutan como destino del `if`, la segunda instrucción se ejecuta sea `num` mayor que cero o no: *¡en cualquier caso!*

Bloques de código

bloques.cpp

Ejemplo

```
#include <iostream>
using namespace std;

int main()
{
    int op1 = 13, op2 = 4, resultado;
    int opcion;
    char operador;
    cout << "1 - Sumar" << endl;
    cout << "2 - Restar" << endl;
    cout << "Opcion: ";
    cin >> opcion;
    if (opcion == 1) {
        operador = '+';
        resultado = op1 + op2;
    }
    else {
        operador = '-';
        resultado = op1 - op2;
    }
    cout << op1 << operador << op2 << " = " << resultado << endl;
    return 0;
}
```

```
D:\FP\Tema2>bloques
1 - Sumar
2 - Restar
Opcion: 1
13+4 = 17

D:\FP\Tema2>bloques
1 - Sumar
2 - Restar
Opcion: 2
13-4 = 9
```

Fundamentos de la programación

Bucles (while)

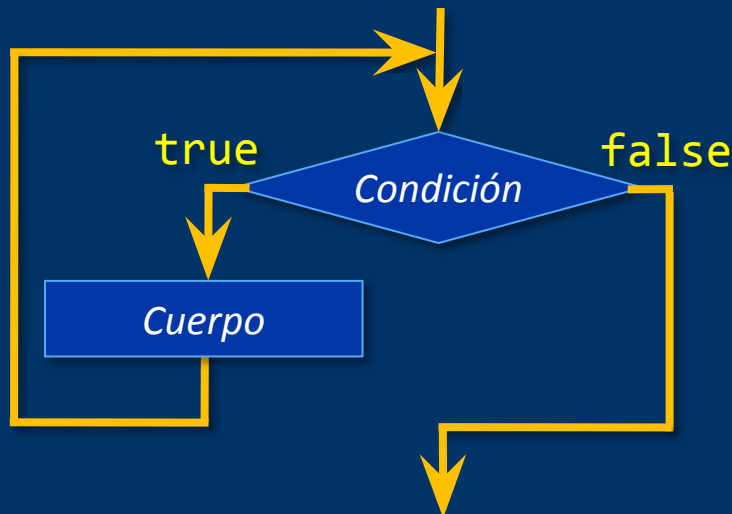
Repetir esto...

Iteración

Mientras que la condición sea cierta, repetir esto...



Repetición condicional:



`while (condición)`
`→ cuerpo`

Mientras la *condición* sea **true** se ejecuta el *cuerpo*.

Si la *condición* es **false** al empezar, no se ejecuta el *cuerpo* ninguna vez.

El *cuerpo* del bucle es una instrucción (simple o compuesta –bloque –).

La instrucción while

Iteración

serie.cpp

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int i = 1, n = 0, suma = 0;
    while (n <= 0) { // Sólo n positivo
        cout << "Cuantos numeros quieres sumar? ";
        cin >> n;
    }
    while (i <= n) {
        suma += i;
        i++;
    }
    cout << "Sumatorio de i (1 a " << n << ") = " << suma << endl;

    return 0;
}
```

$$\sum_{i=1}^n i$$

```
D:\FP\Tema2>serie
Cuantos numeros quieres sumar? -3
Cuantos numeros quieres sumar? 0
Cuantos numeros quieres sumar? 5
Sumatorio de i (1 a 5) = 15
```

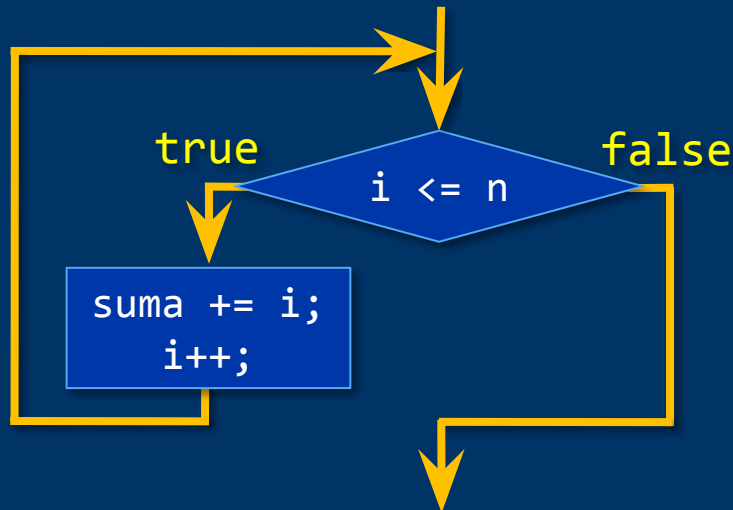
La instrucción while

serie.cpp

Iteración

```
while (i <= n) {  
    suma += i;  
    i++;  
}
```

$$\sum_{i=1}^n i$$



n	5
i	6
suma	15

Sumatorio de i (1 a 5) = 15

Fundamentos de la programación

Entrada/salida por consola

Entrada/salida por consola (teclado/pantalla)

Flujos de texto

C++ no tiene facilidades de E/S en el propio lenguaje.

La E/S se realiza a través de flujos (*streams*).

Los flujos conectan la ejecución del programa con los dispositivos de entrada/salida.

Los flujos de texto son secuencias de caracteres.

Entrada por teclado:

Se colocan los caracteres en el flujo de entrada `cin` (de tipo `istream`).

Salida por pantalla:

Se colocan los caracteres en el flujo de salida `cout` (de tipo `ostream`).



Entrada/salida por consola

Flujos

`cin` y `cout` son objetos de datos definidos en la biblioteca `iostream`. La E/S se realiza por medio de sus operadores.

Operadores de entrada/salida:

- >> *Extractor*: Operador binario para expresiones de entrada de datos. El operando izquierdo es un flujo de entrada (`cin`) y el operando derecho una variable. El resultado de la operación es el propio flujo de entrada.



- << *Insertor*: Operador binario para expresiones de salida de datos. El operando izquierdo es un flujo de salida (`cout`) y el operando derecho una expresión. El resultado de la operación es el propio flujo de salida.



La asociatividad de ambos operadores es de izquierda a derecha.

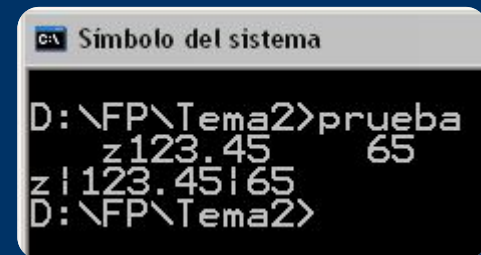
Entrada por teclado

Expresiones de entrada



- **char**: Se saltan los *espacios en blanco** y se asigna el carácter a la variable.
- **int**: Se saltan los *espacios en blanco** que haya. Se leen los dígitos seguidos que haya y se transforma la secuencia de dígitos en un valor que se asigna.
- **float/double**: Se saltan los *espacios en blanco** que haya. Se leen los dígitos seguidos que haya; si hay un punto se leen los dígitos que le sigan; se transforma la secuencia en un valor que se asigna.
- **bool**: Si lo leído es **1**, la variable toma el valor **true**; con cualquier otra entrada toma el valor **false**. No se suelen leer este tipo de variables.

```
int i;  
char c;  
double d;  
cin >> c >> d >> i;  
cout << c << "|" << d << "|" << i;
```

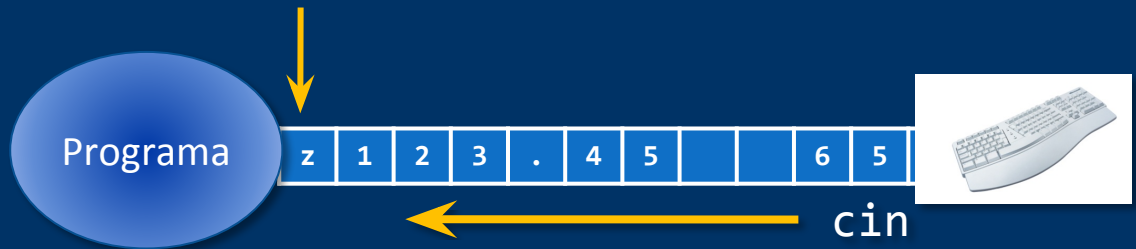


* *Espacios en blanco*: espacios, tabuladores o saltos de línea.

Entrada por teclado

Flujos de entrada

```
int i;  
char c;  
double d;  
cin >> c >> d >> i;  
  
cin >> d >> i;  
  
cin >> i;
```



Se lee el carácter 'z' y se asigna a la variable c; resultado: cin

Se leen los caracteres 123.45, se convierten al valor 123.45 (válido para double) y se asigna a la variable d; resultado: cin

Se ignoran los espacios, se leen los caracteres 65, se convierten al valor 65 (válido para int) y se asigna a la variable i

C++ siempre intentará leer los datos sin provocar errores de ejecución, pero si las secuencias de caracteres de la entrada no son correctas, los valores asignados a las variables serán impredecibles.

Entrada por teclado

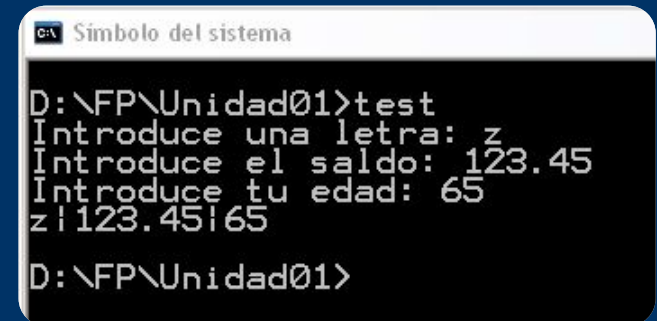
Facilitar la entrada

Programas amigables con el usuario

Indicar al usuario para cada dato qué se espera que introduzca:

```
int edad;  
char letra;  
double saldo;  
cout << "Introduce una letra: ";  
cin >> letra;  
cout << "Introduce el saldo: ";  
cin >> saldo;  
cout << "Introduce tu edad: ";  
cin >> edad;  
cout << letra << "|" << saldo << "|" << edad;
```

Tras escribir cada dato el usuario pulsará la tecla Intro.



```
D:\FP\Unidad01>test  
Introduce una letra: z  
Introduce el saldo: 123.45  
Introduce tu edad: 65  
z|123.45|65  
D:\FP\Unidad01>
```



Buenos hábitos de programación:
Ser amigable con el usuario 😊

Entrada por teclado

Ejecución de funciones con el operador punto (.)

Las variables de algunos tipos (como `istream` u `ostream`) admiten que se llamen sobre ellas sus funciones con la notación del operador punto:

variable.función(argumentos)

Por ejemplo:

```
char c;  
cin.get(c);  
// Llama a la función get() sobre la variable cin
```

La función `get()` lee el siguiente carácter sin saltar espacios en blanco. Los caracteres de espacio en blanco también se asignan a la variable.

Entrada por teclado

Lectura de cadenas de caracteres (string)

La lectura de cadenas de tipo **string** con `cin >>` termina cuando encuentra el primer espacio en blanco.

El resto de los caracteres tecleados quedan pendientes para la siguiente lectura. Si queremos descartarlos usamos `cin.sync()`.

```
string nombre, apellidos;
cout << "Nombre: ";
cin >> nombre;
cout << "Apellidos: ";
cin >> apellidos;
cout << "Nombre completo: "
    << nombre << " "
    << apellidos << endl;
```

```
Nombre: Luis Antonio
Apellidos: Nombre completo: Luis Antonio
apellidos toma la cadena "Antonio"
```

```
string nombre, apellidos;
cout << "Nombre: ";
cin >> nombre;
cin.sync();
cout << "Apellidos: ";
cin >> apellidos;
cout << "Nombre completo: "
    << nombre << " "
```

```
Nombre: Luis Antonio
Apellidos: Hernández Yáñez
Nombre completo: Luis Hernández
```

¿Cómo leer varias palabras?

Entrada por teclado

Lectura de cadenas de caracteres (**string**)

string.cpp

Para leer incluyendo espacios en blanco se usa la función `getline()`:

```
getline(cin, cadena)
```

Se guardan en la *cadena* todos los caracteres leídos hasta el final de la línea (Intro).

```
string nombre, apellidos;  
cout << "Nombre: ";  
getline(cin, nombre);  
cout << "Apellidos: ";  
getline(cin, apellidos);  
cout << "Nombre completo: "  
    << nombre << " "  
    << apellidos << endl;
```

```
D:\FP\Tema2>string  
Nombre: Luis Antonio  
Apellidos: Hernández Yáñez  
Nombre completo: Luis Antonio Hernández Yáñez
```

¡Se leen bien los caracteres castellanos!

Salida por pantalla

Expresiones de salida



Se calcula el resultado de la expresión. Se convierte (si es necesario) a su representación en caracteres y se envían los caracteres al flujo de salida (pantalla).

```
int meses = 7;
```

```
cout << "Total: " << 123.45 << endl << " Meses: " << meses;
```

```
Total: 123.45
Meses: 7
```

La biblioteca `iostream` define la constante `endl` como un salto de línea.

Salida por pantalla

Flujos de salida



T o t a l : 1 2 3 . 4 5 ↵ M e s e s : 7

cout ←

Programa

```
int meses = 7;
cout << "Total: " << 123.45 << endl << " Meses: " << meses;
    cout << 123.45 << endl << " Meses: " << meses;
        cout << endl << " Meses: " << meses;
            cout << " Meses: " << meses;
                cout << meses;
```

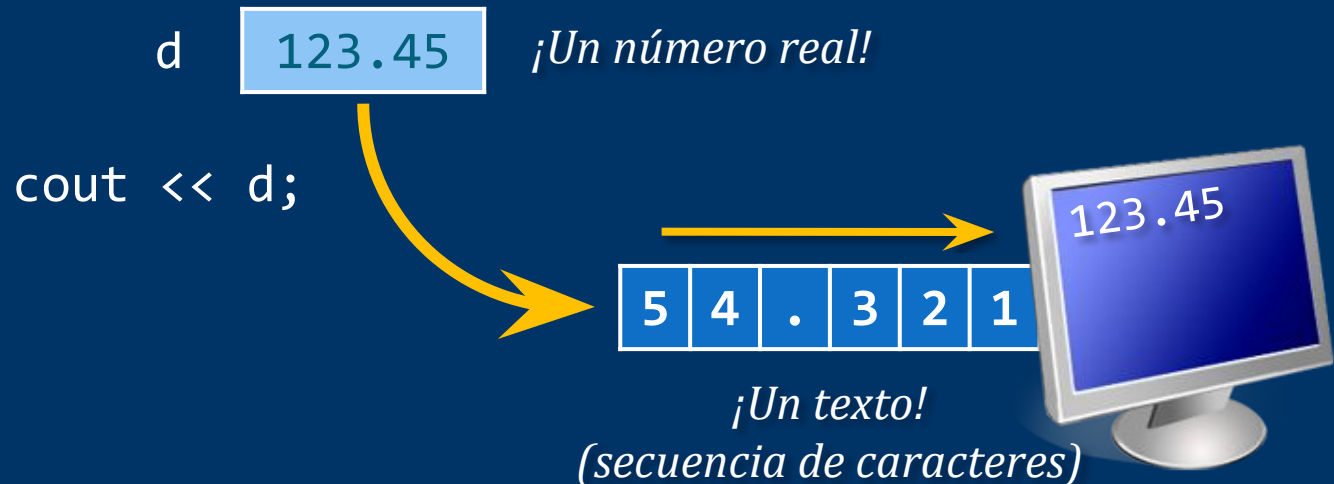
```
Total: 123.45
Meses: 7
```

Salida por pantalla

Datos y flujos de texto

No es lo mismo el valor `double` `123.45` que su representación textual, que es la secuencia de caracteres: `'1'` `'2'` `'3'` `'.'` `'4'` `'5'`

```
double d = 123.45;
```



Salida por pantalla

Formato de la salida

Las bibliotecas `iostream` e `iomanip` tienen definidas constantes que permiten, cuando se envían a `cout`, establecer *opciones de formato* que afecten a la salida que se realice a continuación.

Biblioteca	Identificador	Propósito
iostream	<code>showpoint</code> / <u><code>noshowpoint</code></u>	Mostrar o no el punto decimal para reales sin parte decimal (añadiendo 0 a la derecha).
	<code>fixed</code> / <code>scientific</code>	Notación de punto fijo / científica (reales).
	<code>boolalpha</code>	Valores <code>bool</code> como <code>true</code> / <code>false</code> .
	<code>left</code> / <u><code>right</code></u>	Ajustar a la izquierda/derecha.
iomanip	<code>setw(anchura)*</code>	Nº de caracteres (anchura) para el valor.
	<code>setprecision(p)</code>	Precisión: Nº de dígitos (antes y después del .). Con <code>fixed</code> o <code>scientific</code> , nº de decimales.

*`setw()` sólo afecta al siguiente dato que se escriba, mientras que los otros afectan a todos.
(Subrayadas las opciones predeterminadas.)

Salida por pantalla

Formato de la salida (ejemplo)

```
bool fin = false;
cout << fin << "->" << boolalpha << fin << endl;
double d = 123.45;
char c = 'x';
int i = 62;
cout << d << c << i << endl;
cout << "|" << setw(8) << d << "|" << endl;
cout << "|" << left << setw(8) << d << "|" << endl;
cout << "|" << setw(4) << c << "|" << endl;
cout << "|" << right << setw(5) << i << "|" << endl;
double e = 96;
cout << e << " - " << showpoint << e << endl;
cout << scientific << d << fixed << endl;
cout << setprecision(8) << d << endl;
```

0->>false

123.45x62

| 123.45 |

|123.45 |

|x |

| 62 |

96 - 96.0000

1.234500e+002

123.45000000

Se han añadido saltos de línea
adicionales.

Referencias bibliográficas



- ✓ *Programming. Principles and Practice Using C++*
B. Stroustrup. Pearson Education, 2009
- ✓ *C++: An Introduction to Computing* (2ª edición)
J. Adams, S. Leestma, L. Nyhoff. Prentice Hall, 1998
- ✓ *El lenguaje de programación C++* (Edición especial)
B. Stroustrup. Addison-Wesley, 2002