Fundamentos de la programación

2 Tipos e instrucciones I

Doble grado ADE + Ingeniería Informática
Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores

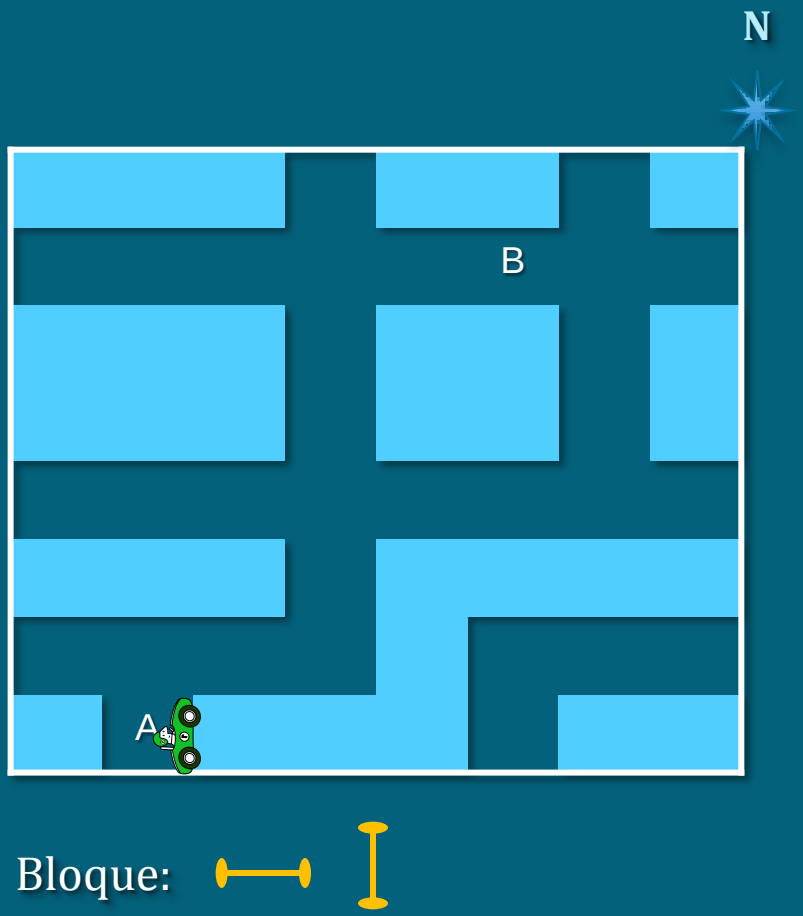
Material de la Prof.ª Mercedes Gómez Albarrán
Versión revisada y ampliada del material del Prof. Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Programación

Una computadora de un coche

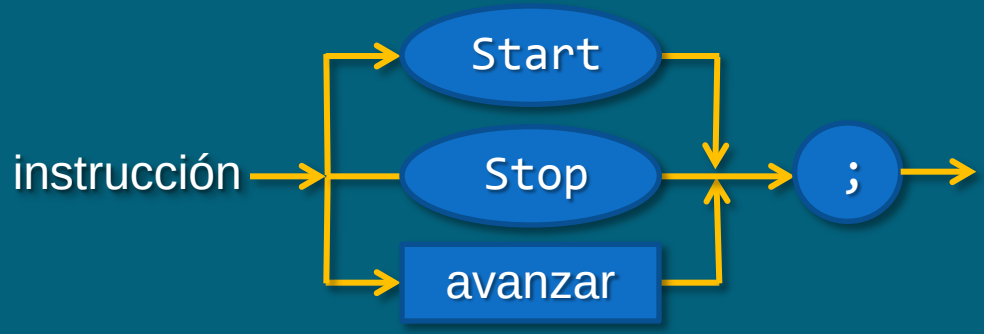
Estando el coche en la posición A, conseguir llegar al Cine Tívoli (B).

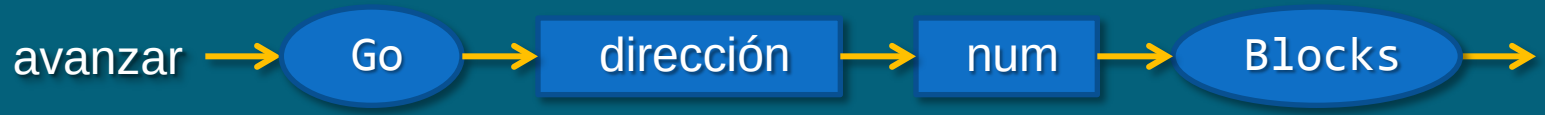
¿Cuáles son los pasos que hay que seguir para llegar al destino?

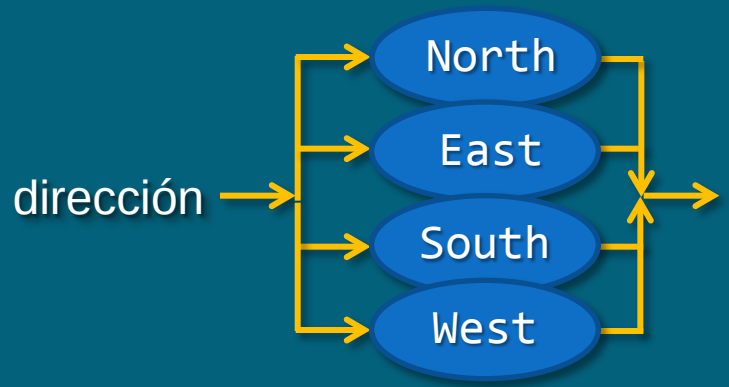


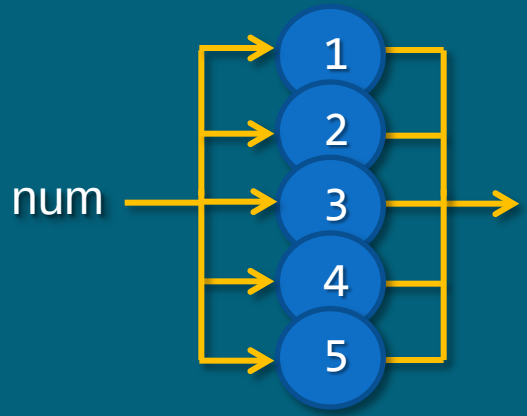
Programación

¿Qué entiende la computadora del coche? Sintaxis del lenguaje de programación



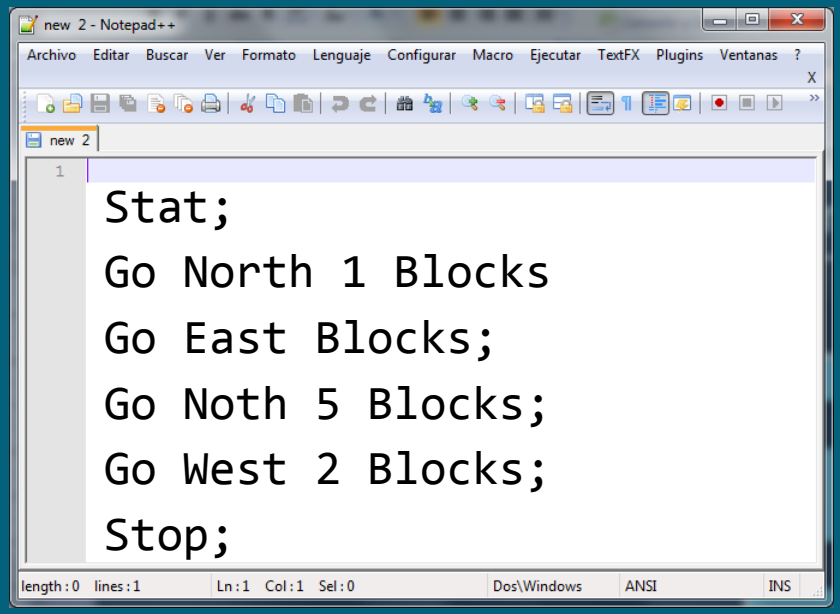






Programación

La edición



Programación

La compilación

```
Stat;
----^ Unknown word.
Go North 1 Blocks
-----------------^ ; missing.
Go East Blocks;
--------^ Number missing.
Go Noth 5 Blocks;
-------^ Unknown word.
Go West 2 Blocks;
Stop;
There are errors. Impossible to run the program.
```

Errores de sintaxis

Programación

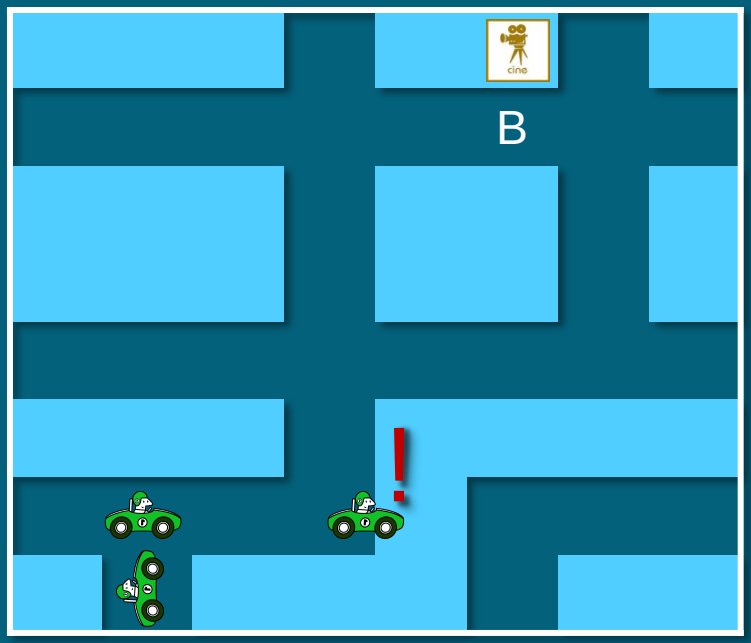
La depuración

Editamos el código para arreglar los errores de sintaxis.

```
Stat;
Go North 1 Blocks
Go East Blocks;
Go Noth 5 Blocks;
Go West 2 Blocks;
Stop;
```

```
Start;
Go North 1 Blocks;
Go East 3 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```


La ejecución




```
Start;
Go North 1 Blocks;
Go East 3 Blocks;
```

¡No se puede seguir ejecutando el programa!

Error de ejecución:

¡Una instrucción no se puede ejecutar!

La depuración

Editamos el código para arreglar el error de ejecución.

```
Start;
Go North 1 Blocks;
Go East 3 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```

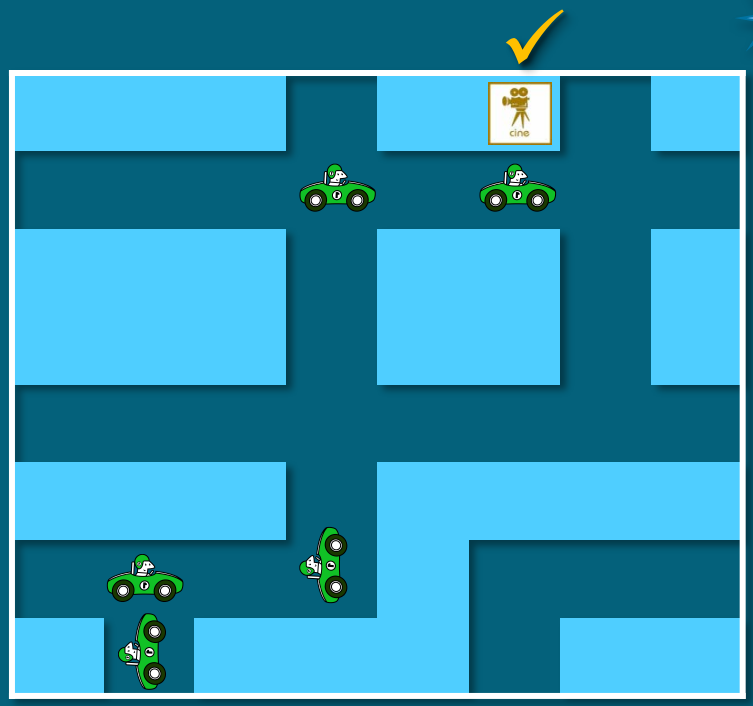
Con la segunda versión se detecta que la 5ª instrucción dirige al coche en la dirección opuesta

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```

Programación

La ejecución

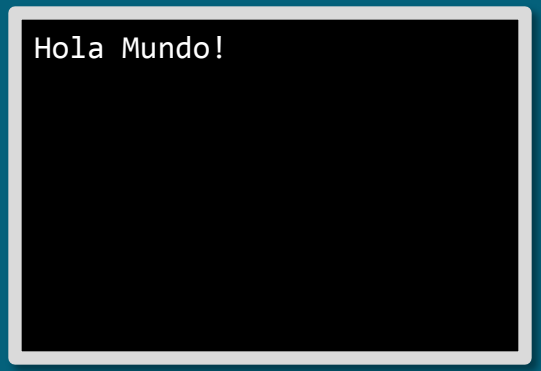
```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```



¡Conseguido!

Programas en C++

Hola Mundo en C++




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

int main()
{
    cout << "Hola Mundo!" << endl;
    // endl produce un salto de línea.
    return 0;
}
```


Programas en C++

Un programa en C++ se compone de

- Uno o más archivos con declaraciones y funciones.
- Una sola función llamada `main()` donde empieza la ejecución.

La forma más básica de un programa en C++ cuenta con un solo archivo:

```
Declaraciones globales

int main()
{
   Secuencia de declaraciones e instrucciones

   return 0;
}
```

Programas en C++

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Hola Mundo!" << endl;
   // endl produce un salto de línea.
   return 0;
}
```

Las bibliotecas de funciones

- Conjunto de funciones de utilidad
- Pareja de archivos fuente: definición (*cabecera*) e implementación
- Para utilizar algo de una biblioteca estándar en un módulo de programa hay que colocar una directiva `#include` con el nombre del archivo de cabecera entre ángulos:

```
#include <cabecera>
```

- Espacios de nombres: Si la biblioteca tiene definidos espacios de nombres, diremos cuál usaremos

```
using namespace nombre;
```

Programas en C++

Uso de espaciado

Los elementos del lenguaje se pueden separar por uno o más *elementos de espaciado* (espacios, tabuladores y saltos de línea). Deben usarse para mejorar la legibilidad.

Uso de comentarios para aclarar aspectos del código

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

int main()
{
   cout << "Hola Mundo!" << endl;      // endl produce un salto de línea.
   return 0;
}
```

¿Cuál se lee mejor?

```
#include <iostream> using namespace std;
int main(){cout << "Hola Mundo!" <<
endl; // endl produce un salto de línea.
return 0;}
```

Programas en C++

C++ es case-sensitive

- Distingue entre mayúsculas y minúsculas
- Palabras reservadas (o clave): en minúsculas

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

int main()
{
 cout << "Hola Mundo!" << endl;
 // endl produce un salto de línea.
 return 0;
}
```

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

int MAIN()
{
 COUT << "Hola Mundo!" << endl;
 // endl produce un salto de línea.
 return 0;
}
```

Programas en C++

Ejemplo 1: Programa que visualiza por pantalla el área de un triángulo a partir de su base y su altura solicitadas por pantalla e introducidas por teclado.

¿Entrada?

¿Salida?

¿Algoritmo?

1. Mostrar en la pantalla el texto que pida la base del triángulo.
2. Leer del teclado el valor para la base del triángulo.
3. Mostrar en la pantalla el texto que pida la altura del triángulo.
4. Leer del teclado el valor para la altura del triángulo.
5. Calcular el área del triángulo.
6. Mostrar el área del triángulo.

Programas en C++

```
...
int main()
{
   Declaraciones

   Algoritmo
   traducido
   a código
   en C++

   return 0;
}
```

1. Mostrar en la pantalla el texto que pida la base del triángulo.
2. Leer del teclado el valor para la base del triángulo.
3. Mostrar en la pantalla el texto que pida la altura del triángulo.
4. Leer del teclado el valor para la altura del triángulo.
5. Calcular el área del triángulo.
6. Mostrar el área del triángulo.

Programas en C++

```
...
int main()
{
  double base, altura, area;                          // 1. Declarar...
  cout << "Introduzca la base del triángulo: ";        // 2. Mostrar ...
  cin >> base;                                        // 3. Leer del...
  cout << "Introduzca la altura del triángulo: ";     // 4. Mostrar ...
  cin >> altura;                                      // 5. Leer del...
  area = base * altura / 2;                           // 6. Calcular...
  cout << "El área de un triángulo de base " << base << " y altura "
       << altura << " es: " << area << endl;          // 7. Mostrar...
  return 0;
}
```

Las instrucciones terminan en ;

Programas en C++

Inclusión de bibliotecas

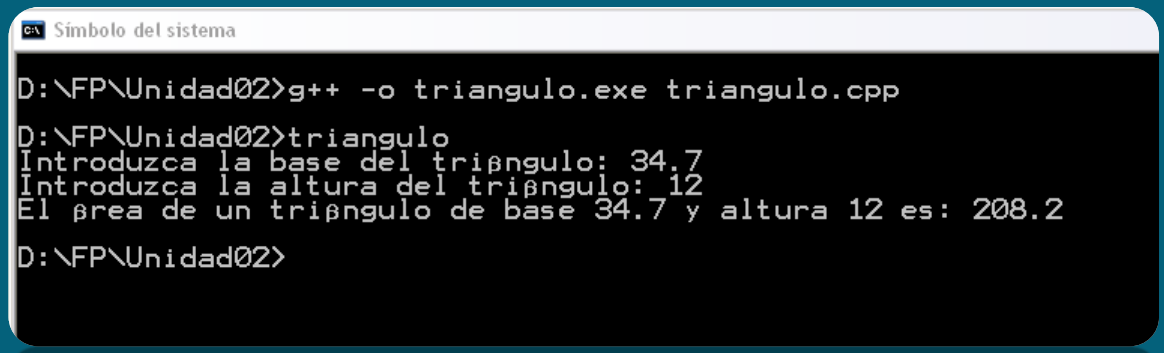
triangulo.cpp

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

int main()
{
  double base, altura, area;
  cout << "Introduzca la base del triángulo: ";
  cin >> base;
  cout << "Introduzca la altura del triángulo: ";
  cin >> altura;
  area = base * altura / 2;
  cout << "El área de un triángulo de base " << base << " y altura "
       << altura << " es: " << area << endl;
  return 0;
}
```

`cin` y `cout` están declaradas en `iostream`, dentro del espacio de nombres `std`.

Programas en C++

Ejecución



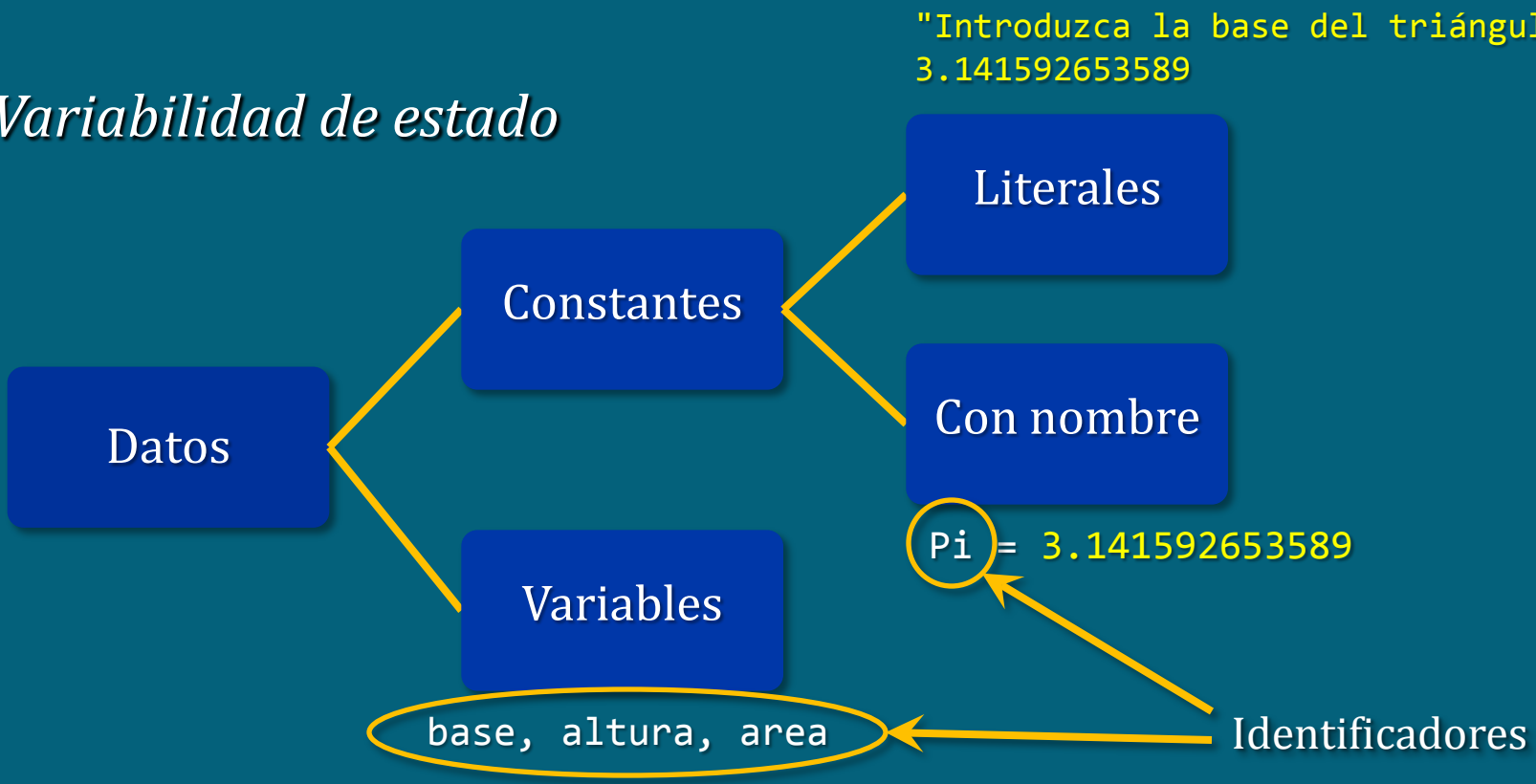

```
#include <iostream>
using namespace std;

int main()
{
  double base, altura, area;
  cout << "Introduzca la base del triángulo: ";
  cin >> base;
  cout << "Introduzca la altura del triángulo: ";
  cin >> altura;
  area = base * altura / 2;
  cout << "El área de un triángulo de base " << base << " y altura "
       << altura << " es: " << area << endl;
  return 0;
}
```

¿¿¿triβngulo???
Problemas con los juegos de caracteres

Los datos de un programa

Tipo

Variabilidad de estado



Identificadores

- Un identificador es el nombre de un dato (o de una función) concreto
- Se debe usar nombres descriptivos (autoexplicativos)
- Los identificadores no pueden coincidir con las palabras reservadas del lenguaje
- No pueden repetirse identificadores en un mismo ámbito (bloque)
- Recuerda… C++ es *case-sensitive*

Sintaxis de los identificadores

Identificador ::= Encabezado {Resto}

Encabezado ::= Letra | “_”

Resto ::= Letra | Digito | “_”

Letra ::= “a” | “b”| ... | “z” | “A” | “B”| ... | “Z”

Digito ::= “0” | “1” | “2” | ... | “9”

¡OJO, no se admiten ni eñes ni vocales acentuadas!

Identificadores

Palabras reservadas del lenguaje C++

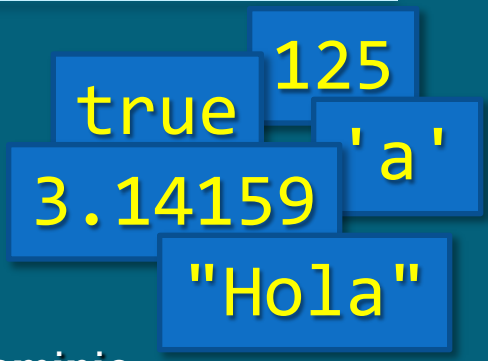
```
asm  auto  bool  break  case  catch  char  class  const
const_cast  continue  default  delete  do  double
dynamic_cast  else  enum  explicit  extern  false
float  for  friend  goto  if  inline  int  long
mutable  namespace  new  operator  private  protected
public  register  reinterpret_cast  return  short
signed  sizeof  static  static_cast  struct  switch
template  this  throw  true  try  typedef  typeid
typename  union  unsigned  using  virtual  void
volatile  while  …
```


Identificadores

¿Qué identificadores son válidos y cuáles no?

```
balance                 interesAnual

_base_imponible         años

EDAD12                  salario_1_mes

__edad                  cálculoNómina

valor%100               AlgunValor

100caracteres           valor?

_12_meses               ____valor
```

Tipos de datos



Cada dato del programa es de un tipo concreto.

Cada tipo establece:

- El conjunto (intervalo) de valores válidos, esto es, el dominio
- El conjunto de operaciones que se pueden realizar sobre los valores del tipo (esto es, los operadores que se pueden aplicar en las expresiones)

Tipos de datos básicos: cuáles son y dominios

- `int`: para números enteros (sin parte decimal).
  `1363`, `-12`, `49`
- `float`: para números reales (con parte –punto– decimal).
  `12.45`, `-3.1932`, `1.16E+12`
- `double`: para números reales; mayores intervalo y precisión.
- `char`: para caracteres.
  `'a'`, `'{'`, `'\t'`
- `bool`: para valores lógicos (verdadero/falso).
  `true`, `false`
- `void`: *nada*, ausencia de tipo, ausencia de dato.

El tipo int

Números enteros

Bytes de memoria: 4*

(*) Depende de la máquina.
4 bytes es lo más habitual.

Sintaxis:

V_entero ::= Decimal | Octal | Hexadecimal

Decimal ::= [“+” | “-”] DigitoPositivo {Digito}

Octal ::= [“+” | “-”] “0” {DigitoO}

Hexadecimal ::= [“+” | “-”] “0” “x” {DigitoH}

Digito ::= “0” | “1” | ... | “9”

DigitoPositivo ::= “1” | ... | “9”

DigitoO ::= “0” | “1” | ... | “7”

DigitoH ::= “0” | “1” | ... | “9” | “A” | “B” | ... | “F”

Ejemplos de valores literales: `1,+10, 363, 1363, -12, 010 , 0x1A`

El tipo int

Números en notación octal (base 8: dígitos entre 0 y 7):

`-010` = -8 en notación decimal

$10 = 1 \times 8^1 + 0 \times 8^0 = 1 \times 8 + 0$

`0423` = 275 en notación decimal

$423 = 4 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = 4 \times 64 + 2 \times 8 + 3 = 256 + 16 + 3$

Números en notación hexadecimal (base 16):

Dígitos posibles: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

`0x1F` = 31 en notación decimal

$1F = 1 \times 16^1 + F \times 16^0 = 1 \times 16 + 15$

`0xAD` = 173 en notación decimal

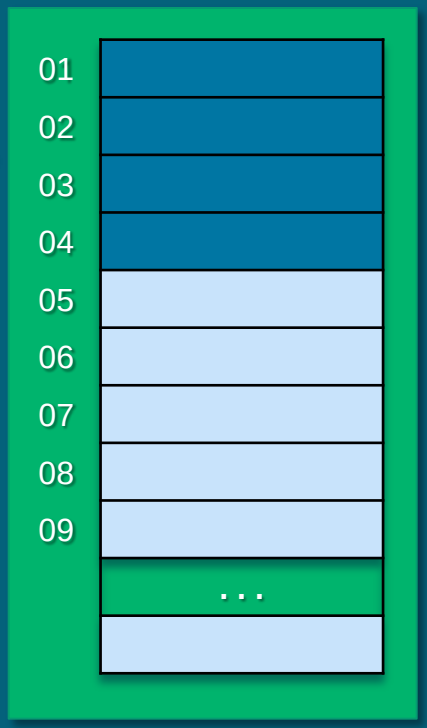
$AD = A \times 16^1 + D \times 16^0 = 10 \times 16 + 13 = 160 + 13$

El tipo float

Números reales (con decimales)

Bytes de memoria: 4*

(*) Depende de la máquina.
4 bytes es lo más habitual.



Sintaxis:

V_real ::= [Signo] (SecDigitos Exp |
SecDigitos "." {SecDigitos} [Exp] |
"." SecDigitos [Exp])
Exp ::= ("E" | "e") [Signo] SecDigitos
SecDigitos ::= Digito {Digito}
Signo ::= "+" | "-"

Ejemplos de valores literales:

- Notación normal: `134.45`, `-1.1764`
- Notación científica: `1.4E2`, `-5.23e-10`

El tipo double

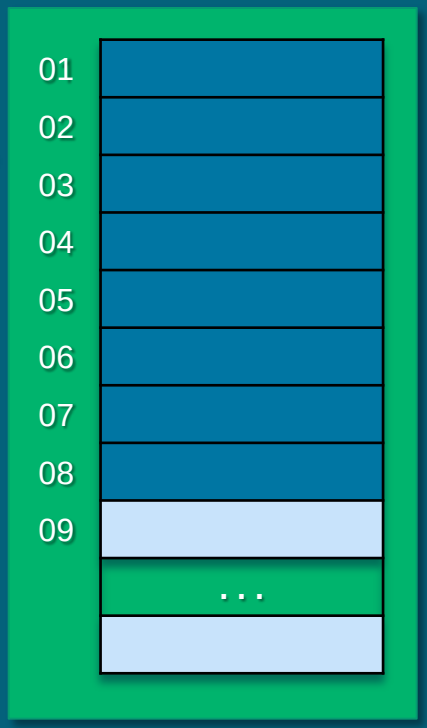
Números reales (con decimales)

Intervalo de valores: `+/- 2.23e-308 .. 1.79e+308`

Bytes de memoria: 8*

Misma sintaxis que float

(*) Depende de la máquina.
8 bytes es lo más habitual.



El tipo char

Caracteres

Intervalo de valores: Juego de caracteres (ASCII)

Bytes de memoria: 1

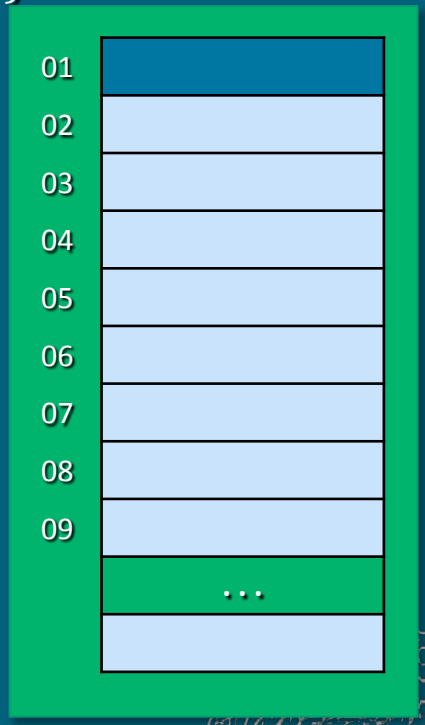
Ejemplos de valores literales:

- Caracteres visibles (dígitos, letras, símbolos de puntuación)

  `'a'`, `'8'`, `'.'`

- No visibles (caracteres de control)

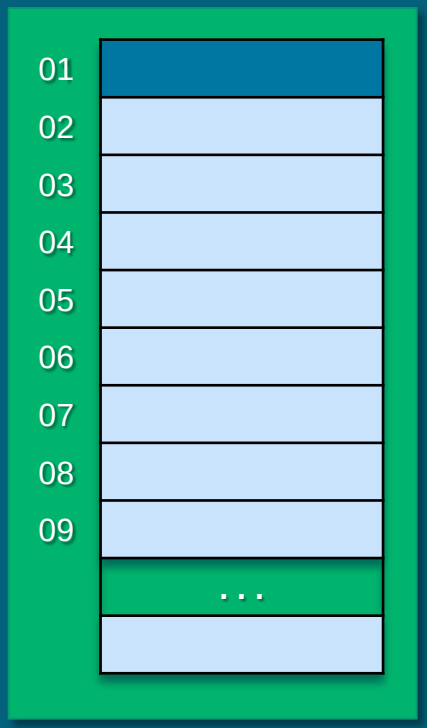
  `'\t'` es el tabulador, `'\n'` es el salto de línea, …



El tipo bool

Valores lógicos

- Sólo dos valores posibles: `true`, `false`
- Bytes de memoria: 1



Variables

Una variable es un dato de estado mutable

Sintaxis de la declaración de variables

DeclaracionVariables ::= DecVar “;” {DecVar “;”}

DecVar ::= Tipo Identificador {“,” Identificador}

Tipo ::= “int” | “float” | “double” | “char” | “bool” | DefinidoPorUsuario

Ejemplos de declaraciones de variables

```
double base, altura, area;
```

```
char caracter1;
double beneficio, perdida;
char caracter2;
```

Una variable sólo puede ser utilizada en el ámbito (bloque) en el que se declara.

Variables

¿Qué ocurre en memoria cuando se declara una variable?

```
int inicio;
double balance;
```

¿Y si se inicializa al declararla?

```
int inicio = 0;
double balance = 10.0;
```

¡Nunca debe usarse una variable que no ha recibido antes un valor!

Constantes

Una constante es un dato de estado inmutable (su valor no cambia a lo largo de la ejecución del programa)

Sintaxis de declaración de constantes

DeclaracionConstantes ::= “const” DecCons “;” {“const” DecCons “;”}

DecCons ::= Tipo Identificador “= ” Valor {“,” Identificador “= ” Valor}

Tipo ::= “int” | “float” | “double” | “char” | “bool” | DefinidoPorUsuario

Valor ::= V_entero | V_float | V_double | V_char | V_bool |

…

```
const int Meses = 12;
const double Pi = 3.14, RATIO = 2.179 * Pi;
```

Constantes

¿Qué ocurre en memoria cuando se declara una constante?

```
const int Meses = 12;
const double Pi = 3.14;
```

¿Por qué usar constantes con nombre?

- Aumentan la legibilidad del código

```
cambioPoblacion = (
```


Tipos de datos: operadores

Operaciones sobre valores de los tipos

Tipos de datos numéricos (`int`, `float` y `double`):

- Operadores aritméticos
- Operadores relacionales

Tipo de datos `bool`:

- Operadores lógicos

Tipo de datos `char`:

- `++` `--`
- Operadores relacionales

Operadores aritméticos

Operadores para tipos de datos numéricos

Operador	Operandos	Posición	`int`	`float` / `double`
-	1 (monario)	Prefijo	Cambio de signo	
+	2 (binario)	Infijo	Suma	
-	2 (binario)	Infijo	Resta	
*	2 (binario)	Infijo	Producto	
/	2 (binario)	Infijo	División entera	División real
%	2 (binario)	Infijo	Módulo	No aplicable
++	1 (monario)	Prefijo / postfijo	Incremento	
--	1 (monario)	Prefijo / postfijo	Decremento	

Operadores binarios: *Operando_izquierdo* *Operador* *Operando_derecho* `a + b`

Operadores monarios:

Prefijo: *Operador* *Operando* `-a`

Postfijo: *Operando* *Operador* `a++`

Expresiones aritméticas

Combinaciones válidas de operandos y operadores aritméticos

Operadores monarios

- El cambio de signo se puede aplicar a una variable, una constante o una expresión entre paréntesis:

```
-saldo      -RATIO      -(3 * a - b)
```

- Los operadores de incremento y decremento sólo se aplican a variables. Incrementan y decrementan el operando en una unidad.

```
interes++      --meses      ++j
```

Forma prefija: Se incrementa/decrementa antes de usar el valor del operando

```
cout << ++i;
```

Forma posfija: Se incrementa/decrementa después de usar el valor del operando

```
cout << i--;
```

Programación con buen estilo:
Evitar usar los operadores ++ y -- en expresiones.

Expresiones aritméticas

Combinaciones válidas de operandos y operadores aritméticos

Operadores binarios

- Los operandos pueden ser literales, constantes, variables o expresiones (entre paréntesis o sin paréntesis).
- / : ¿división entera o división real?

  Si ambos operandos son enteros, / hace un división entera:

```
int i = 23, j = 2;
 i / j evalúa a 11
```

  Si alguno de los operandos es real, / hace una división real:

```
int i = 23;
double j = 2;
 i / j evalúa a 11.5
```

- Se pueden concatenar.

```
2 + 3      a * RATIO      -a + b
a + b * c – d      (a % b) * c / (d – e)
```


Expresiones aritméticas

Evaluación de expresiones

¿Cómo evaluamos una expresión como `3 + 5 * 2 / 2 - 1`?

¿En qué orden se aplican los distintos operadores?

- ¿De izquierda a derecha?
- ¿De derecha a izquierda?
- ¿Unos operadores antes que otros?

Resolución de ambigüedades → cada operador tiene una precedencia y una asociatividad

Es posible cambiar las reglas de precedencia y asociatividad mediante el uso de paréntesis

- Siempre se evalúan en primer lugar las subexpresiones parentizadas.

Expresiones aritméticas

Evaluación de expresiones

PRECEDENCIA

Un operador ⊗ precede a (o tiene prioridad sobre) otro ⊕ si, ante dos interpretaciones posibles –una en la que se aplica primero ⊗ y otra en la que se aplica primero ⊕–, se elige siempre aquélla que supone aplicar primero ⊗

ASOCIATIVIDAD (orden a igual precedencia)

Si el operador infijo binario ⊗ asocia a izquierdas, a ⊗ b ⊗ c se interpreta como (a ⊗ b) ⊗ c

Si el operador infijo binario ⊗ asocia a derechas, a ⊗ b ⊗ c se interpreta como a ⊗ (b ⊗ c)

Expresiones aritméticas

Evaluación de expresiones

Resumiendo:

- Las subexpresiones entre paréntesis se evalúan primero. Si hay anidamiento se evalúa de *dentro hacia fuera*
- Dentro de una expresión, los operadores se evalúan según su precedencia
- A igual precedencia los operadores que estén en una misma expresión evalúan según su asociatividad

Expresiones aritméticas

Precedencia y asociatividad de los operadores

Precedencia	Operadores	Asociatividad
Mayor prioridad	`++ --` (postfijos)	Izquierda a derecha
↑	`++ --` (prefijos) `-` (cambio de signo)	Derecha a izquierda
	`* / %`	Izquierda a derecha
Menor prioridad	`+ -`	Izquierda a derecha

`3 + 5 * 2 / 2 - 1` → `3 + 10 / 2 - 1` → `3 + 5 - 1` → `8 - 1` → `7`

¿Cómo se evaluaría `(3 + 5) * 2 / (2 - 1)` ?

Funciones aritméticas

Algunas funciones matemáticas de la biblioteca `cmath`

`abs(x)`	Valor absoluto de x
`pow(x, y)`	x elevado a y
`sqrt(x)`	Raíz cuadrada de x
`ceil(x)`	Menor entero que es mayor o igual que x
`floor(x)`	Mayor entero que es menor o igual que x
`exp(x)`	e^x
`log(x)`	Ln x (logaritmo natural de x)
`log10(x)`	Logaritmo en base 10 de x
`sin(x)`	Seno de x
`cos(x)`	Coseno de x
`tan(x)`	Tangente de x
`round(x)`	Redondeo al entero más próximo
`trunc(x)`	Pérdida de la parte decimal (entero)

Las llamadas a funciones tienen la mayor prioridad.

Para poder utilizarlas :

```
#include <cmath>
```

Llamada (invocación) de una función :

`nombre(argumentos)`

Los argumentos irán separados por comas.

```
3 * pow(x, 2)
5 + 2 * sqrt(x)
```

Operadores relacionales

Permiten comparar dos operandos numéricos

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.

```
int j = 2;
```

`2 < j` evalúa a `false`

Operadores para caracteres

Los únicos operadores definidos para el tipo `char` son el incremento/decremento (código siguiente/anterior) y los operadores relacionales.

`<`	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.

Funciones para caracteres

Funciones para caracteres (biblioteca `cctype`)

- `isalnum(c)` `true` si `c` es una letra o un dígito; si no, `false`.
- `isalpha(c)` `true` si `c` es una letra; si no, `false`.
- `isdigit(c)` `true` si `c` es un dígito; si no, `false`.
- `islower(c)` `true` si `c` es una letra minúscula; si no, `false`.
- `isupper(c)` `true` si `c` es una letra mayúscula; si no, `false`.
- `toupper(c)` devuelve la mayúscula de `c`.
- `tolower(c)` devuelve la minúscula de `c`.

…

Operadores lógicos

Se aplican sobre operandos de tipo `bool`

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

!	NO (negación)	Monario
&&	Y	Binario
||	O	Binario

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.
`&&`	Izda. a dcha.
`||`	Izda. a dcha.

Operadores lógicos

Tablas de verdad

!	
true	false
false	true

NO

&&	true	false
true	true	false
false	false	false

Y

||	true	false
true	true	true
false	true	false

O

Expresiones lógicas (o booleanas)

Pueden resultar de usar operadores relacionales

```
int a = 2, b = 3, c = 4;

a < 5                 // 2 < 5 → true
a * b + c >= 12       // 10 >= 12 → false
a * (b + c) >= 12     // 14 >= 12 → true
a != b                // 2 != 3 → true
a * b > c + 5         // 6 > 9 → false
a + b == c + 1        // 5 == 5 → true
```

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.
&&	Izda. a dcha.
||	Izda. a dcha.

Expresiones lógicas (o booleanas)

Pueden resultar de usar operadores lógicos

```
bool cond1 = false, cond2 = true, cond3 = true;
int a = 2;

cond1 && cond2 // false && true → false
cond1 || cond2 // false || true → true
cond1 && (cond2 || cond3) // false && (true || true) → false
a < 5 && cond2 // true && true → true
(a < 5) && cond2 // true && true → true
```

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.
&&	Izda. a dcha.
||	Izda. a dcha.

Expresiones lógicas (o booleanas)

Evaluación perezosa (shortcut boolean evaluation)

- Si el operando izquierdo de un `||` es `true`, seguro que el resultado es `true`, por lo que no hace falta evaluar el operando derecho:

```
true || X ≡ true
```

- Si el operando izquierdo de un `&&` es `false`, seguro que el resultado es `false`, por lo que no hace falta evaluar el operando derecho:

```
false && X ≡ false
```


Expresiones lógicas (o booleanas)

Ejercicio 1: Escribe expresiones booleanas para expresar las siguientes condiciones en un programa

- Hombre casado
- Mujer soltera, divorciada o viuda

Ejercicio 2: Escribe expresiones booleanas para expresar la siguiente condición en un programa

- Un valor entero está en el intervalo [0, 100]

La asignación

- Instrucción básica de la programación imperativa para cambiar el estado de un programa (cambiar el estado de sus variables)
- Permite dar valor a una variable → operación destructiva
- Sintaxis

Asignacion ::= Variable "=" ValorAsignado ";"

ValorAsignado ::= Constante | Variable | Expresion

Error común de programación:

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

La asignación

El operador =

- Es el operador para asignaciones
- Tiene la mínima prioridad y asocia de derecha a izquierda

```
int i, j = 2;
i = j * 5;
i = 23 + i;

int a, b, c;
a = b = c = 5;
```

¿Qué ocurre en memoria?

La asignación

¿Qué ocurre con estas instrucciones?

```
int a, b, c;
const int cte = 0;


5 = a;
cte = 5;
a + 23 = 5;
b = "abc";
c = 23  5;
```


Volviendo a nuestro programa en C++

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

int main()
{
  double base, altura, area;
  cout << "Introduzca la base del triángulo: ";
  cin >> base;
  cout << "Introduzca la altura del triángulo: ";
  cin >> altura;
  area = base * altura / 2;
  cout << "El área de un triángulo de base " << base << " y altura "
       << altura << " es: " << area << endl;
  return 0;
}
```

Entrada / salida por consola (teclado / pantalla)

Flujos de texto

```
#include <iostream>
```

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 (teclado / pantalla)

Los operadores de E/S son binarios y asocian de izquierda a derecha

Operador de `cin`

`>>`

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.

Operador de `cout`

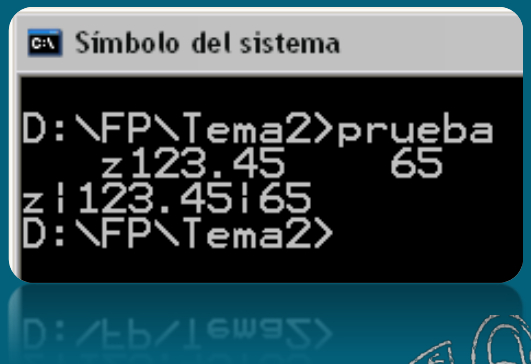
`<<`

El operando izquierdo es un flujo de salida (`cout`) y el operando derecho una constante, una variable o una expresión.
El resultado de la operación es el propio flujo de salida.

Entrada por consola (teclado)

- `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`: **No se suelen leer este tipo de variables.**

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



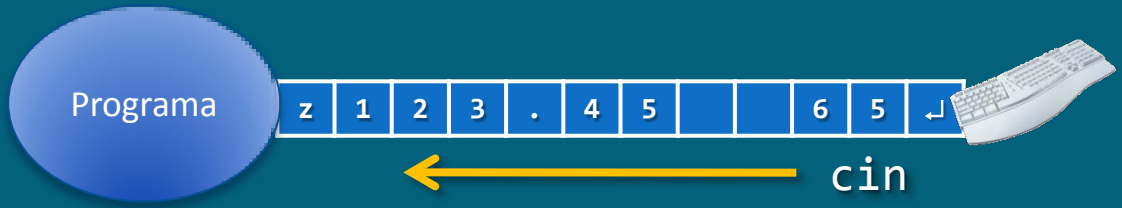
* *Espacios en blanco*: espacios, tabuladores o saltos de línea.

Entrada por consola (teclado)

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

```
   cin   >> d >> i;
```

```
      cin    >> i;
```



Entrada por consola (teclado)

Invocación de funciones con el operador punto (.)

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

`variable.función(argumentos)`

Por ejemplo:

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

La función `get()` lee el siguiente carácter sin saltar espacios en blanco.

Entrada por consola (teclado)

```
char c1,c2,c3;

cin.get(c1);
cin.get(c2);
cin.get(c3);
```

versus

```
char c1,c2,c3;

cin >> c1 >> c2 >> c3;
```

AB<intro>

CD<intro>

Entrada por consola (teclado)

Lectura de cadenas de caracteres (`string`)

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

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 << " "
     << apellidos << endl;
```

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

Entrada por consola (teclado)

Lectura de cadenas de caracteres (`string`)

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

¿Cómo leer varias palabras?

Para leer incluyendo espacios en blanco se usa:

`getline(cin, cadena)`

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

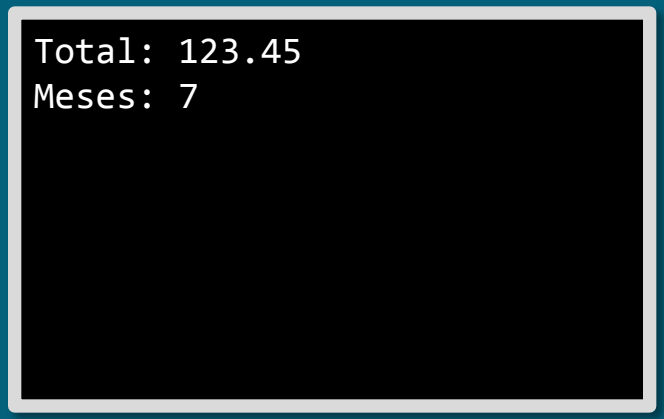
```
string nombre, apellidos;
cout << "Nombre: ";
getline(cin, nombre);
cout << "Apellidos: ";
getline(cin, apellidos);
cout << "Nombre completo: "
     << nombre << " "
     << apellidos << endl;
```

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

¡Se leen bien los caracteres castellanos!

Salida por consola (pantalla)

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

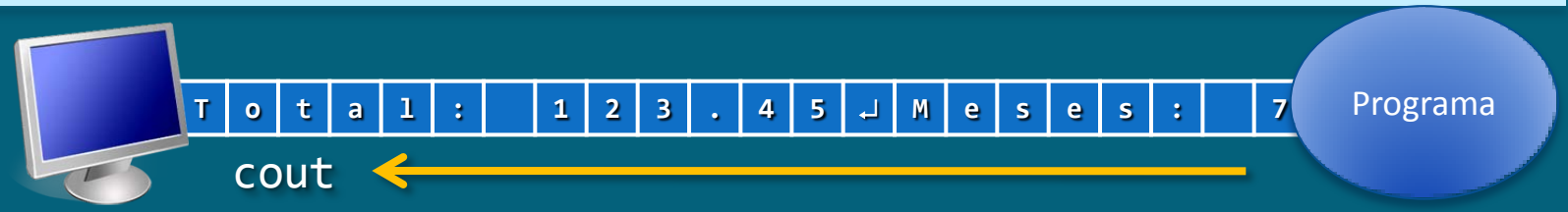


La biblioteca `iostream` define la constante `endl` como un salto de línea.

Las comillas tipográficas (apertura/cierre) “…” te darán problemas al compilar.

Asegúrate de utilizar comillas rectas: "…“

Salida por consola (pantalla)




```
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;
```


Salida por consola (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`	`showpoint` / `noshowpoint`	Mostrar o no el punto decimal para reales sin parte decimal (añadiendo 0 a la derecha).
	`fixed` / `scientific`	Notación de punto fijo / científica (reales).
	`boolalpha`	Valores `bool` como `true` / `false`.
	`left` / `right`	Ajustar a la izquierda/derecha.
`iomanip`	`setw(`*anchura*`)`*	Nº de caracteres (anchura) para el valor.
	`setprecision(`*p*`)`	Precisión: Nº de decimales.

*`setw()` sólo afecta al siguiente dato que se escriba, mientras que los otros afectan a todas las salidas siguientes.

(Subrayadas las opciones predeterminadas.)

Salida por consola (pantalla)

Formato de la salida (ejemplo)

```
bool fin = false;  double d = 123.45;
char c = 'x';  int i = 62;
double e = 96;

cout << fin << "->" << boolalpha << fin << endl;

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;

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
```


Volviendo a nuestro programa en C++

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

int main()
{
  double base, altura, area;
  cout << "Introduzca la base del triángulo: ";
  cin >> base;
  cout << "Introduzca la altura del triángulo: ";
  cin >> altura;
  area = base * altura / 2;
  cout << "El área de un triángulo de base " << base << " y altura "
       << altura << " es: " << area << endl;
  return 0;
}
```

Otro programa de ejemplo

$$f(x) = \frac{3x^2}{5} + \frac{6x}{7} - 3$$

formula.cpp

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

int main()
{
  double x, f;
  cout << "Introduce el valor de X: ";
  cin >> x;
  f = 3 * x * x / 5 + 6 * x / 7 - 3;
  cout << "f(x) = " << f << endl;
  return 0;
}
```

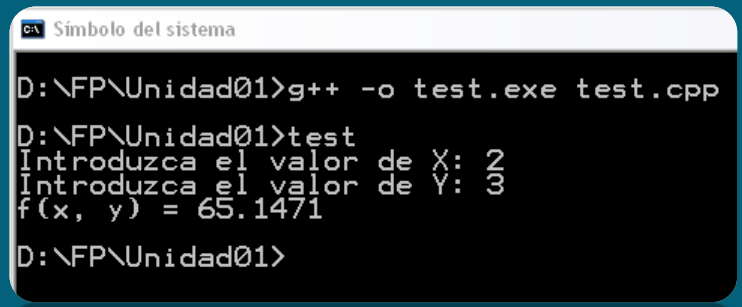
Y otro más

Un ejemplo

$$f(x,y) = 2x^5 + \frac{\sqrt{\frac{x^3}{y^2}}}{|x \times y|} - \cos(y)$$

funciones.cpp

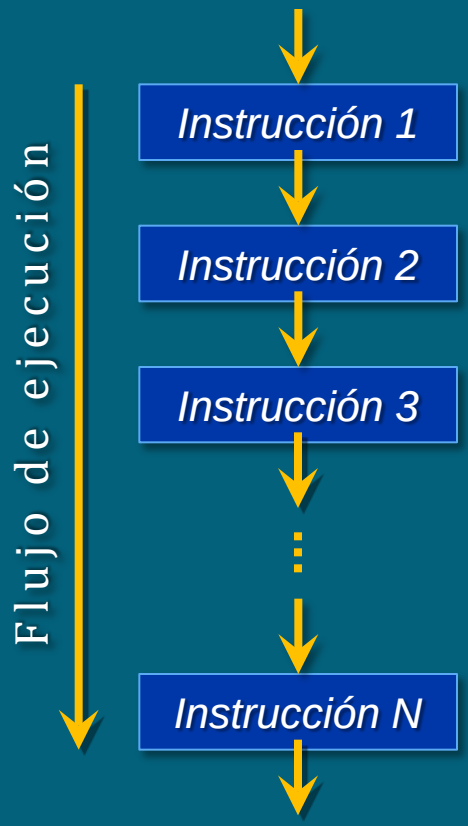
```
#include <iostream>
using namespace std;
#include <cmath>

int main()
{
  double x, y, f;
  cout << "Introduzca el valor de X: ";
  cin >> x;
  cout << "Introduzca el valor de Y: ";
  cin >> y;
  f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2)) / abs(x * y) - cos(y);
  cout << "f(x, y) = " << f << endl;
  return 0;
}
```



Programación con buen estilo:
Poner un espacio detrás de cada coma en las listas de argumentos.

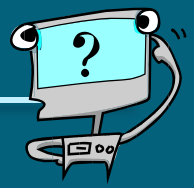
Flujo de ejecución

Ejecución secuencial




```
cout << "Introduzca la base del triángulo: ";

cin >> base;

cout << "Introduzca la altura del triángulo: ";

...

cout << "El área de un triángulo de base " << base
<< " y altura “ << altura << " es: " << area <<
endl;
```

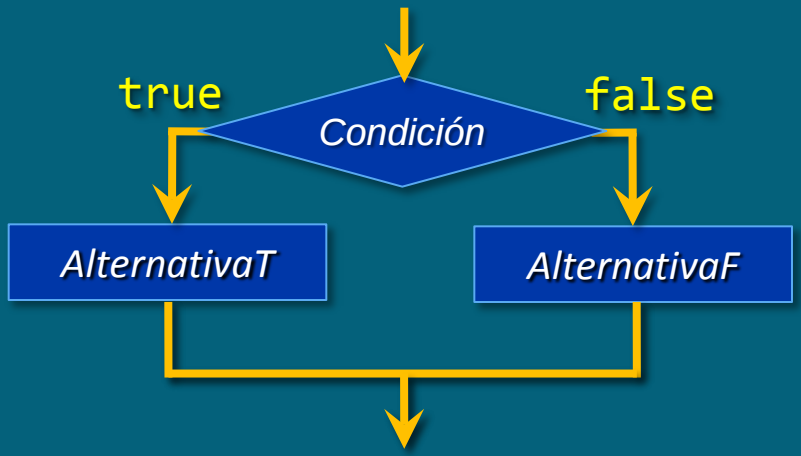
Flujo de ejecución



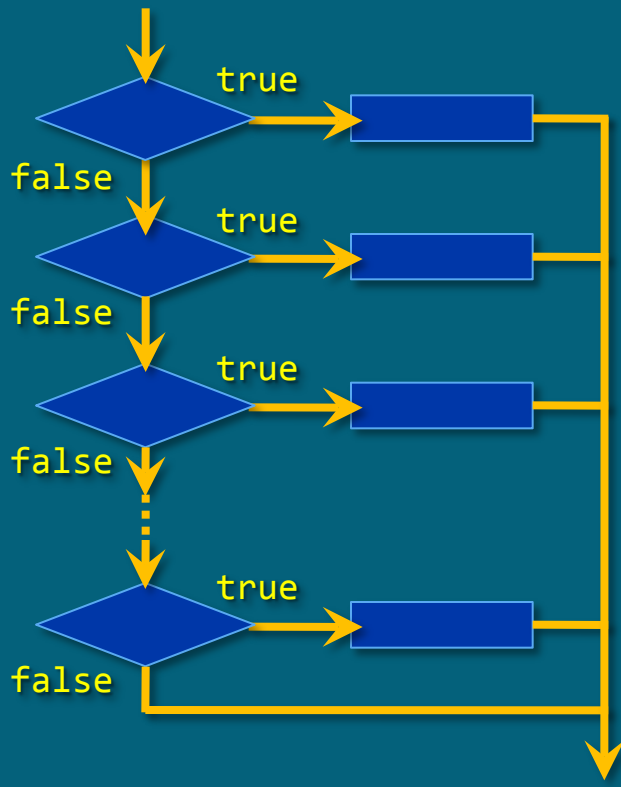
Selección

A menudo necesitamos elegir entre dos o más alternativas

2 caminos: Selección simple



> 2 caminos: Selección múltiple



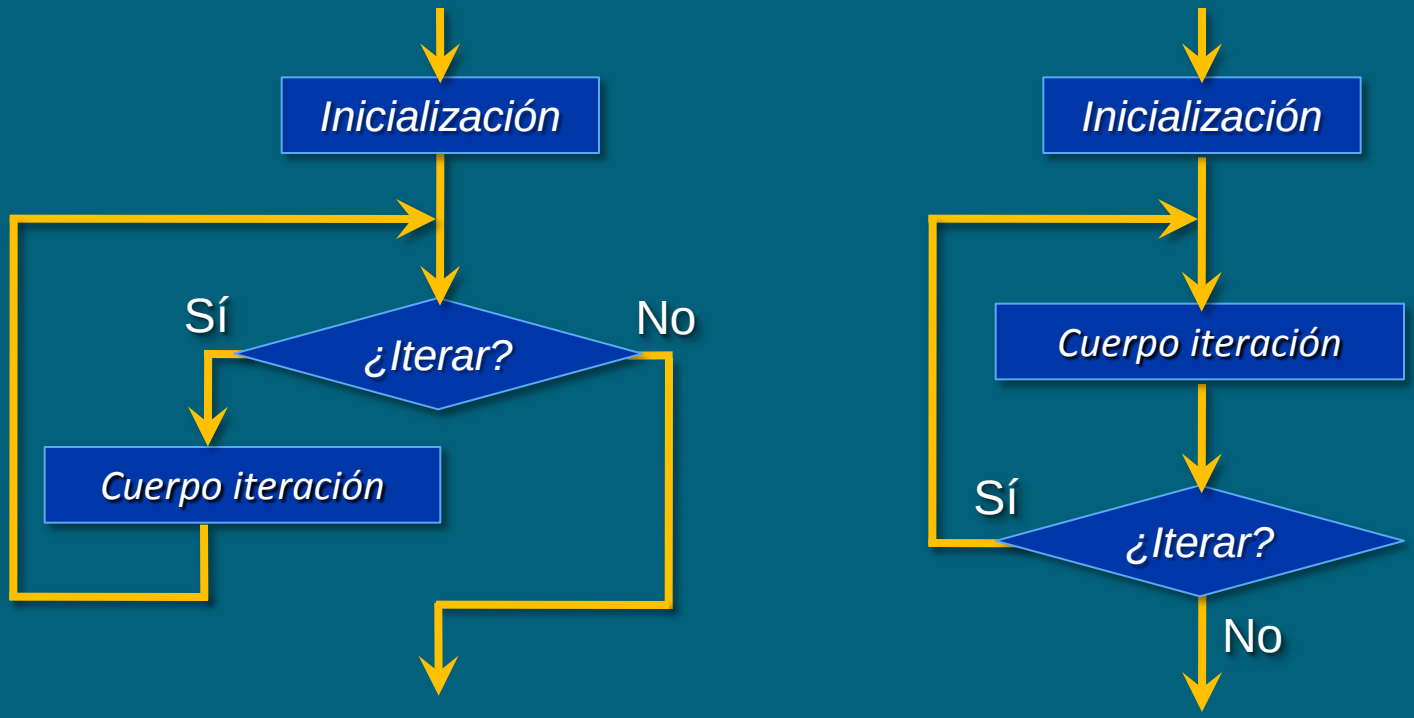
Diagramas de flujo

(no confundir con diagramas sintácticos)

Flujo de ejecución

Repetición o iteración

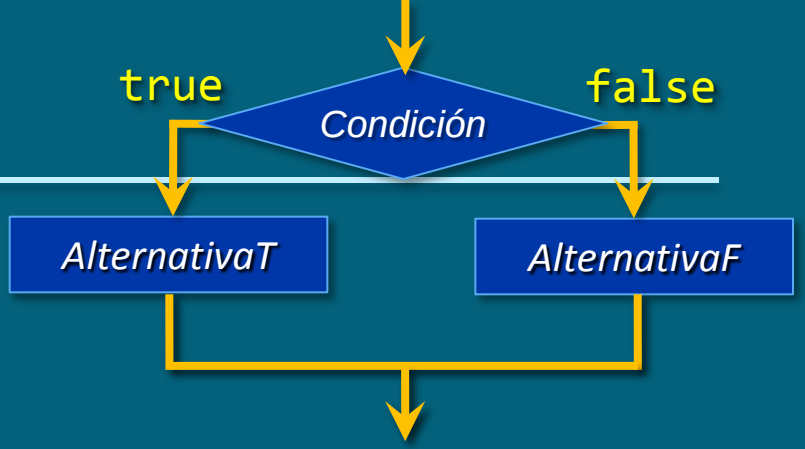
A menudo necesitamos repetir la ejecución de una o más instrucciones



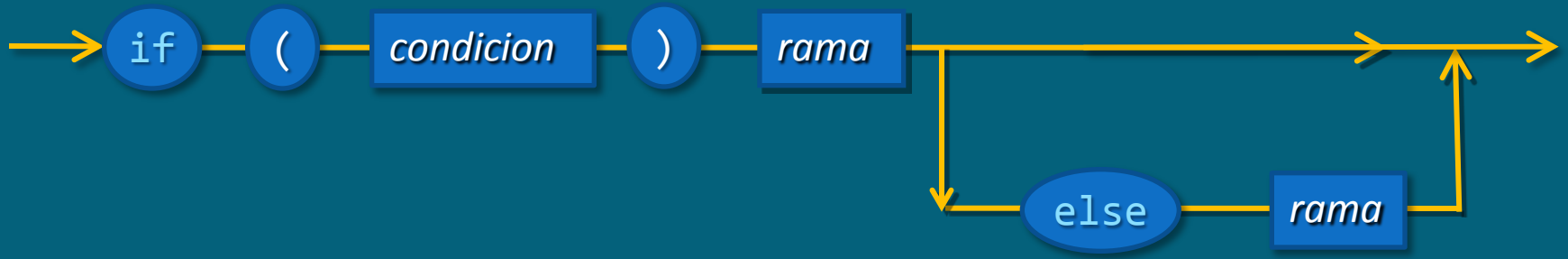
Diagramas de flujo

(no confundir con diagramas sintácticos)

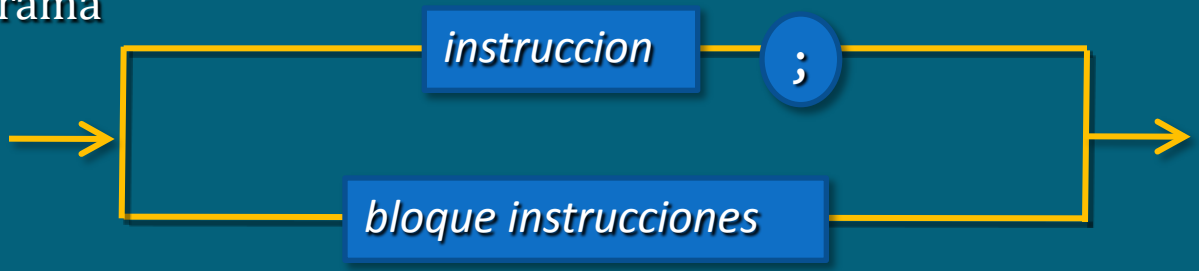
Selección: la instrucción `if`



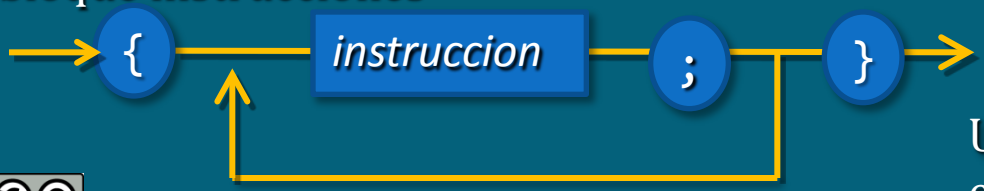
sentencia if



rama



bloque instrucciones



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

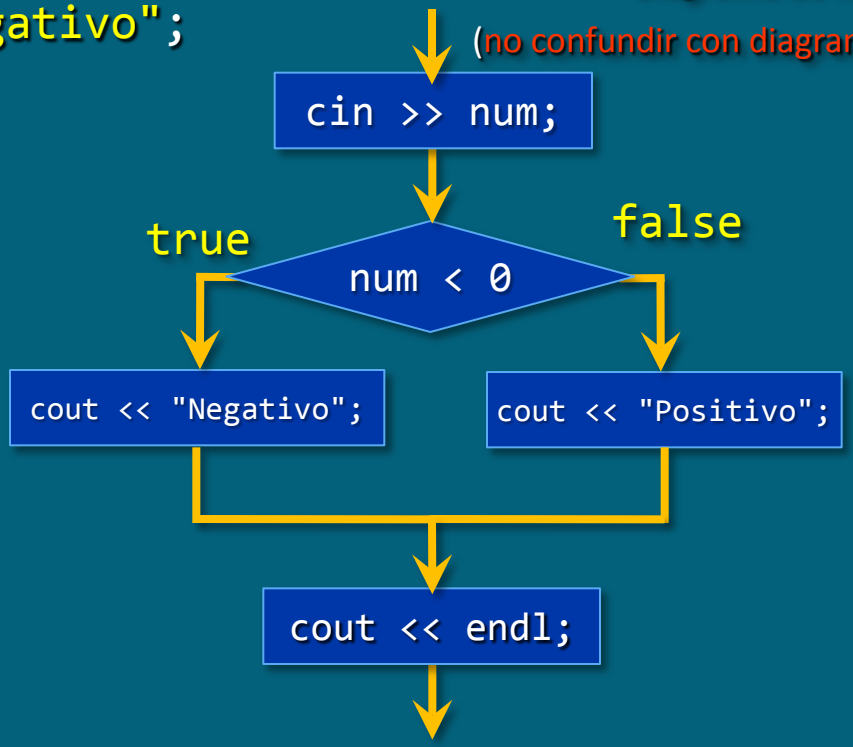
Selección: la instrucción `if`

signo.cpp

```
int num;

cin >> num;
if (num < 0) cout << "Negativo";
else cout << "Positivo";
cout << endl;
```

Diagrama de flujo

(no confundir con diagrama sintáctico)



Selección: la instrucción `if`

Ejemplo.- Mejorando el programa del área del triángulo.

Algoritmo1:

- Solicitar y leer los datos (base y altura)
- Si algún dato no es válido

  Mostrar mensaje de error

  sino
  - Calcular el área
  - Visualizar el área

Selección: la instrucción `if`

Ejemplo.- Mejorando el programa del área del triángulo.

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

int main()
{
  double base, altura, area;
  cout << "Introduzca la base: ";  cin >> base;
  cout << "Introduzca la altura: ";  cin >> altura;
  if ( ..............  ) cout << “Datos erroneos!";
  else {
     area = base * altura / 2;
     cout << "El área de un triángulo es: " << area << endl;
  }
  return 0;
}
```

Selección: la instrucción `if`

Ejemplo.- Mejorando el programa del área del triángulo.

Algoritmo2:

- Solicitar y leer los datos (base y altura)
- Si los datos son válidos
  - Calcular el área
  - Visualizar el área

  sino

  Mostrar mensaje de error

Selección: la instrucción `if`

Ejemplo.- Mejorando el programa del área del triángulo.

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

int main()
{
  double base, altura, area;
  cout << "Introduzca la base: ";  cin >> base;
  cout << "Introduzca la altura: ";  cin >> altura;
  if ( ..............  ) {
     area = base * altura / 2;
     cout << "El área de un triángulo es: " << area << endl;
  }
  else
    cout << "Datos erroneos!";
  return 0;
}
```

Selección: la instrucción `if`

Ejemplo.- División de dos números introducidos por teclado protegida frente a intento de división por 0

¿Algoritmo?

¿Programa?

Selección: la instrucción `if`

Asociación de cláusulas `else` en selección múltiple

Cada `else` se asocia al `if` anterior más cercano sin asociar (mismo ámbito).

```
if (condición1)
  if (condición2) ...
  else ...
else
  if (condición3) {
    if (condición4) {
      if (condición5) ...
      else ...
    }
  }
  else
  ...
```

Cuestión de estilo:

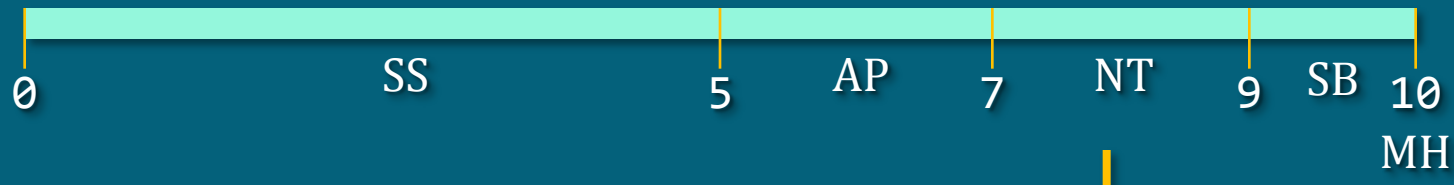
La sangría (*indentación*) ayuda a la persona que lee el código a asociar los `else` con sus `if`.

¡OJO! La sangría no afecta al compilador.

Selección: la instrucción `if`

nota.cpp

Ejemplo.- Nota simbólica equivalente a numérica




```
double nota;
cin >> nota;
if (nota == 10) cout << "MH";
else
  if (nota >= 9) cout << "SB";
  else
    if (nota >= 7) cout << "NT";
    else
      if (nota >= 5) cout << "AP";
      else cout << "SS";
```

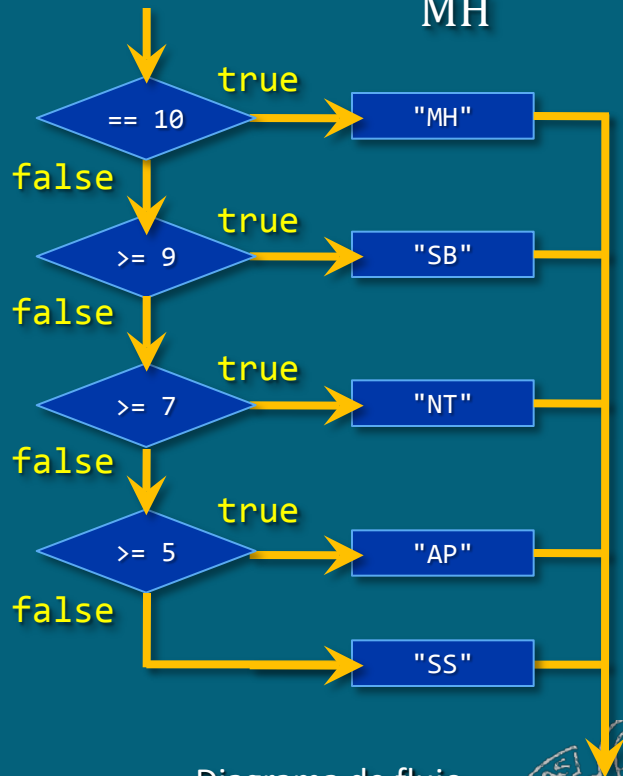


Diagrama de flujo

(no confundir con diagrama sintáctico)

Selección: la instrucción `if`

¡Cuidado con el orden de las condiciones!

```
double nota;
cin >> nota;
if (nota < 5) cout << "SS";
else if (nota < 7) cout << "AP";
else if (nota < 9) cout << "NT";
else if (nota < 10) cout << "SB";
else cout << "MH";
```

```
double nota;
cin >> nota;
if (nota >= 5) cout << "AP";
else if (nota >= 7) cout << "NT";
else if (nota >= 9) cout << "SB";
else if (nota == 10) cout << "MH";
else cout << "SS";
```

¿Son correctas estas soluciones?

Selección: la instrucción `if`

¡Extrema el control de la entrada de datos!

```
double nota;
cin >> nota;
if (nota == 10) cout << "MH";
else
  if (nota >= 9) cout << "SB";
  else
    if (nota >= 7) cout << "NT";
    else
      if (nota >= 5) cout << "AP";
      else cout << "SS";
```

¿Qué ocurre si se introduce un 11?

¿Qué ocurre si se introduce una nota negativa?

¿Cómo controlarías que el valor de la nota fuese correcto?

Selección: la instrucción `if`

Ejemplo.- Número de días del mes y año indicado por el usuario

`diasmes.cpp`

Algoritmo:

- Solicitar y leer los datos (mes y año)
- Si el mes es febrero ...

Para resolver este ejercicio ten en cuenta lo siguiente:

Según el calendario gregoriano un año es bisiesto si es divisible por 4, excepto el último de cada siglo (aquel divisible por 100), salvo que este último sea divisible por 400.

Por ejemplo, los años 1700, 1800, 1900 y 2100 no serán bisiestos pues son divisibles entre 100 y no entre 400. Sí que son bisiestos los años 1600, 2000 y 2400 pese a ser los últimos de siglo ya que son divisibles por 400.

Selección: la instrucción `if`

Ejemplo.- Número de días del mes y año indicado por el usuario

diasmes.cpp

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

int main()
{
  int mes, anio, dias;
  cout << "Número de mes: "; cin >> mes;
  cout << "Año: "; cin >> anio;
  if (mes == 2)
    if (anio % 4 == 0)
        if (anio % 100 == 0 && anio % 400 != 0) dias = 28;
        else dias = 29; // Año bisiesto
    else dias = 28;
  else
    if (mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8
                 || mes == 10 || mes == 12) dias = 31;
    else dias = 30;
  cout << "Días de ese mes: " << dias;
  return 0;
}
```


Selección: la instrucción `if`

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

int main()
{
  int mes, anio, dias;
  cout << "Número de mes: "; cin >> mes;
  cout << "Año: "; cin >> anio;
  if (mes == 2)
    if (anio % 4 == 0 && !(anio % 100 == 0 && anio % 400 != 0))
        dias = 29; // Año bisiesto
    else dias = 28;
  else
    if (mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8
                 || mes == 10 || mes == 12) dias = 31;
    else dias = 30;
  cout << "Días de ese mes: " << dias;
  return 0;
}
```

Otra solución con la condición mejorada

Selección: la instrucción `if`

Ejemplo.- Nivel de un valor

nivel.cpp

Si num = 5 entonces Muy alto
Si num = 4 entonces Alto
Si num = 3 entonces Medio
Si num = 2 entonces Bajo
Si num = 1 entonces Muy bajo

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

int main()
{
  int num;
  cout << "Introduce nivel: "; cin >> num;
  if (num == 5) cout << "Muy alto" << endl;
  else if (num == 4) cout << "Alto" << endl;
  else if (num == 3) cout << "Medio" << endl;
  else if (num == 2) cout << "Bajo" << endl;
  else if (num == 1) cout << "Muy bajo" << endl;
  else cout << "Valor no válido" << endl;
  return 0;
}
```


Selección: la instrucción `if`

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

int main()
{
  int num;
  cout << "Introduce nivel: "; cin >> num;
  if (num == 5) cout << "Muy alto" << endl;
  if (num == 4) cout << "Alto" << endl;
  if (num == 3) cout << "Medio" << endl;
  if (num == 2) cout << "Bajo" << endl;
  if (num == 1) cout << "Muy bajo" << endl;
  return 0;
}
```

¿Qué opinas de una solución como ésta?

Repetición

Tipos de bucles

	Comprobación pre-iteración	Comprobación post-iteración
Recorrido fijo	`for` `while`	`do-while`
Recorrido variable	`while`	`do-while`

Repetición: bucle `while`

Recorrido fijo + comprobación pre-iteración

Recorrido variable + comprobación pre-iteración

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

La *condición* se comprueba antes de cada pasada.

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

Repetición: bucle `while`

Ejemplo.- Bucle `while` usado para recorrido fijo

```
int i = 1;
while (i <= 100) {
  cout << i << endl;
  i++;
}
```

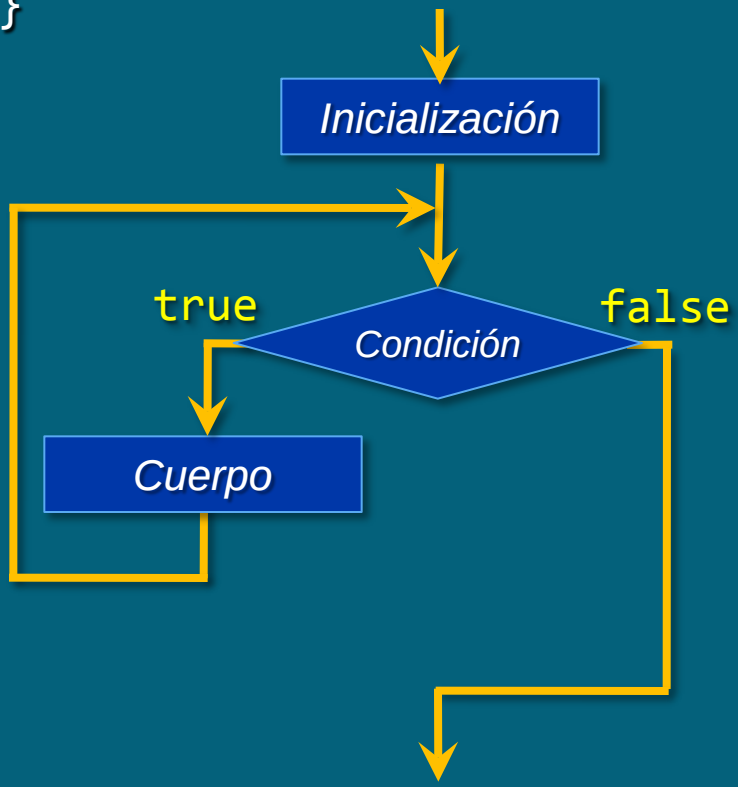
O bien:

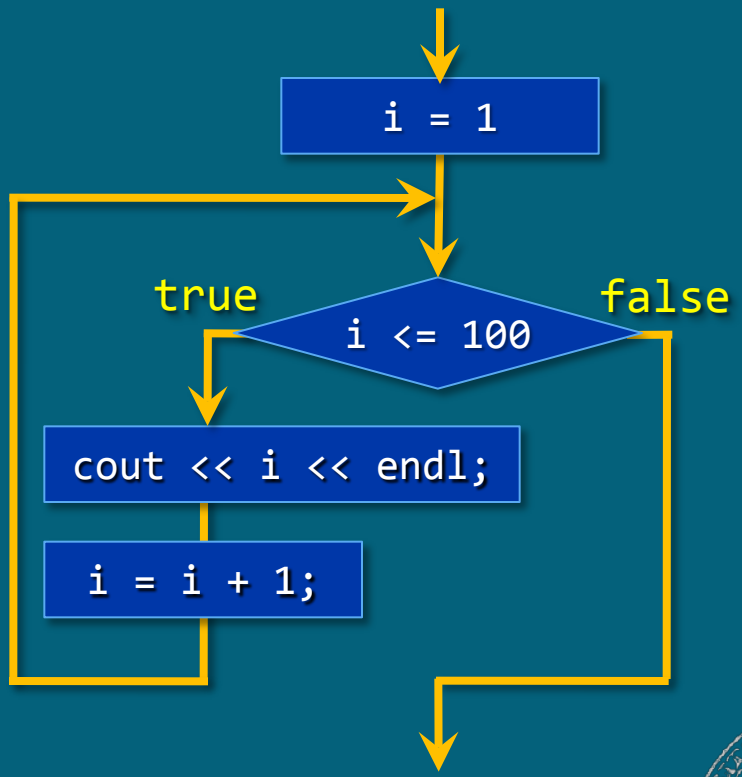
```
int i = 1;
while (i <= 100) {
  cout << i << endl;
  i = i + 1;
}
```

while1.cpp

Repetición: bucle `while`

```
int i = 1;
while (i <= 100) {
  cout << i << endl;
  i = i + 1;
}
```





Repetición: bucle `while`

Ejemplo.- Mejorando el programa del área del triángulo.

¿Se te ocurre cómo usar el bucle `while` para mejorar la versión inicial?

Repetición: bucle `while`

Ejemplo.- Mejorando el programa del área del triángulo mediante el uso del bucle `while`.

```
#include <iostream>
using namespace std;
int main()
{
  double base, altura, area;
  cout << "Introduzca la base del triangulo: ";  cin >> base;
  cout << "Introduzca la altura del triangulo: ";  cin >> altura;
  while ( altura <= 0 || base <= 0 ) {
     cout << "Introduzca la base del triangulo: ";  cin >> base;
     cout << "Introduzca la altura del triangulo: ";  cin >> altura;
  }
  area = base * altura / 2;
  cout << "El area de un triangulo es: " << area << endl;
  return 0;
}
```


Repetición: bucle `while`

Ejemplo.- Hallar el primer entero positivo cuyo cuadrado es mayor que 1.000.

¿Algoritmo?

¿Programa?

Repetición: bucle `while`

Ejemplo.- Calcular la suma de los enteros entre 1 y un número (entero positivo) dado por el usuario vía teclado.

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

int main()
{
  int num;

  cout << "Numero final: "; cin >> num;

  …

  return 0;
}
```

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

Repetición: bucle `while`

Ejemplo.- Suma y media de los números reales que introduzca el usuario por teclado. El fin de la introducción de datos se indica con un 0.

¿Algoritmo?

¿Programa?

Repetición: bucle `while`

Ejemplo.- Suma y media de los números reales que introduzca el usuario por teclado. El fin de la introducción de datos se indica con un 0.

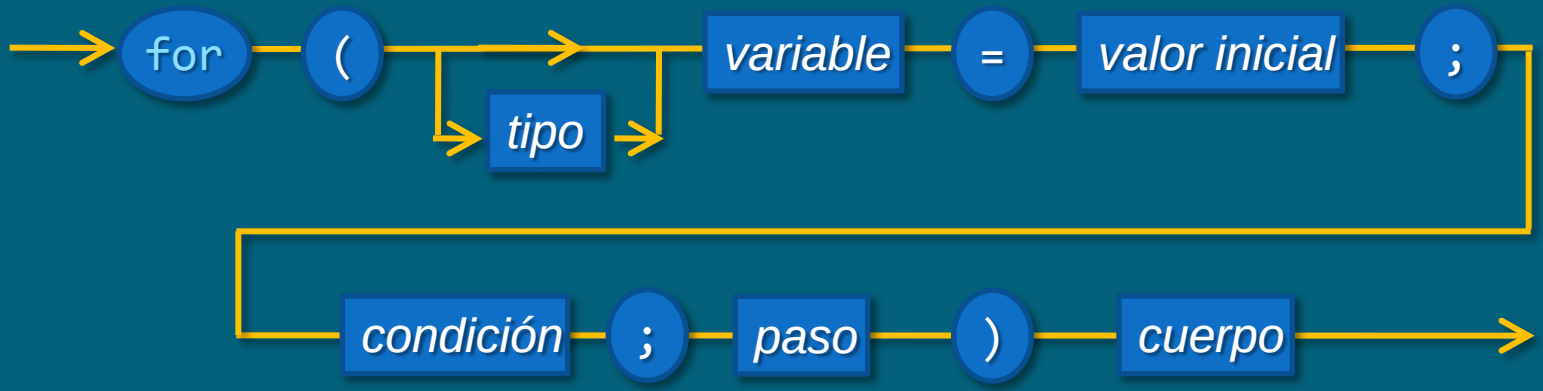
```
#include <iostream>
using namespace std;
int main()
{
  double d, suma = 0;
  int cont = 0;

  cout << "Introduce un numero (0 para acabar): ";
  cin >> d;
  while(d != 0) {
    suma = suma + d;
    cont++;
    cout << "Introduce un numero (0 para acabar): ";
    cin >> d;
  }
  cout << "Suma = " << suma << endl;
  cout << "Media = " << suma / cont << endl;

return 0;
}
```

¿Qué ocurre si la secuencia de números es vacía?

Repetición: bucle for

Recorrido fijo + comprobación pre-iteración

```
for (int i = 1; i <= 100; i++) ...      1, 2, 3, 4, 5, ..., 100
for (int i = 100; i >= 1; i--) ...      100, 99, 98, 97, ..., 1
```



La *condición* compara el valor de la variable contador con un valor final antes de cada pasada.

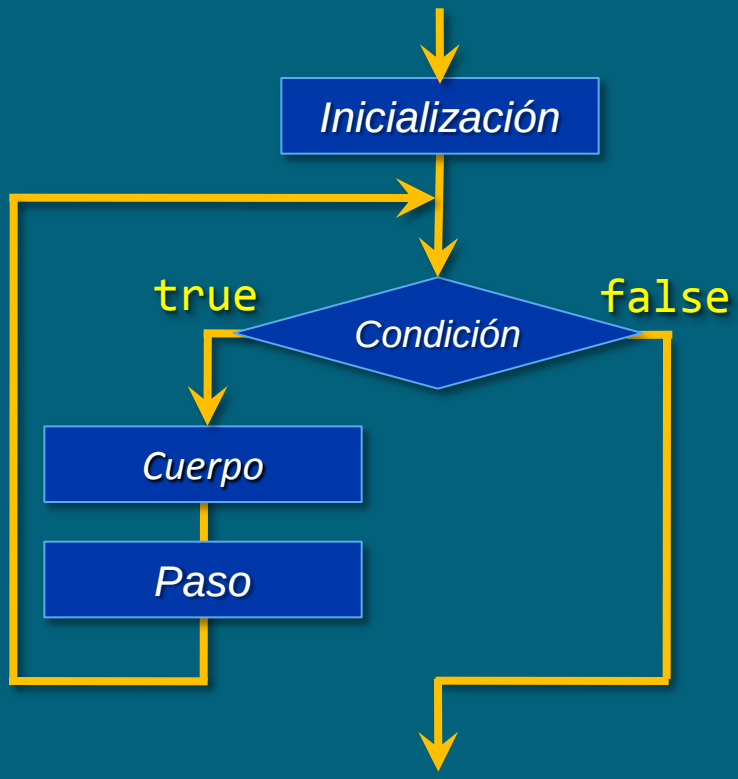
El *paso* modifica el valor de la variable contador (incremento/decremento/...) tras cada pasada

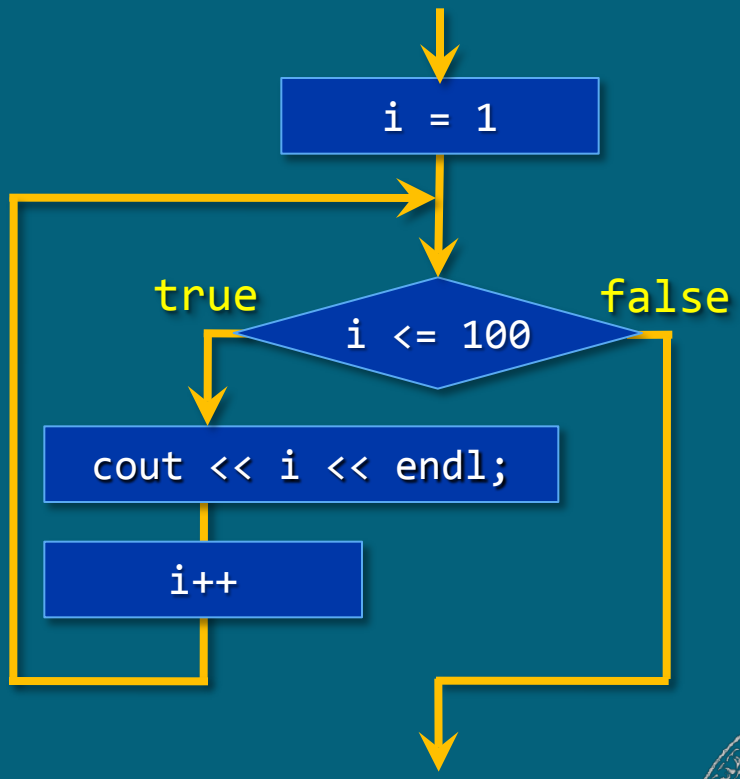
El *cuerpo* del bucle es una instrucción (simple o compuesta –bloque –).

Repetición: bucle for

for1.cpp

```
for (inicialización; condición; paso) cuerpo
for (int i = 1; i <= 100; i++) cout << i << endl;
```





Repetición: bucle for

El *paso* puede cambiar el valor de la variable en más de una unidad.

```
for (int i = 1; i <= 100; i=i+2) cout << i << endl;
```

```
1
3
5
7
...
99
```

for2.cpp

Importante:
El cuerpo del bucle no debe alterar el valor de la variable contador.

Repetición: bucle for

Garantía de terminación

Todo bucle bien diseñado debe terminar su ejecución.

```
for (int i = 1; i <= 100; i++) ...
```

```
for (int i = 1; i <= 100; i=i+2) ...
```

```
for (int i = 100; i >= 1; i--) ...
```

```
for (int i = 1; i <= 100; i--) ...
```

¿Alguno de estos bucles es infinito?

for3.cpp

Repetición: bucle for

Ejemplo.- Calcular la suma de los enteros entre 1 y un número (entero positivo) dado por el usuario vía teclado.

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

int main()
{
  int num;

  cout << "Numero final: "; cin >> num;

  …

  return 0;
}
```

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

Repetición: bucle for

¿Incremento/decremento prefijo o postfijo?

Es indiferente aplicar el operador ++/-- en forma prefija o postfija.

Estos dos bucles producen el mismo resultado:

```
for (int i = 1; i <= 100; i++) ...
```

```
for (int i = 1; i <= 100; ++i) ...
```


Repetición: bucle for

Ámbito de la variable contador

La variable contador se puede declarar en el propio bucle:

```
for (int i = 1; ...) cuerpo
```

La variable sólo se conoce en el cuerpo del bucle.

Su ámbito es el cuerpo del bucle.

O haber sido declarada antes del bucle:

```
int i;
for (i = 1; ...) cuerpo
```

La variable se conoce en el cuerpo del bucle y después del mismo.

Su ámbito es externo al bucle.

Repetición: bucle for

Anidamiento de bucles

Un bucle en el cuerpo de otro bucle.

```
for (int i = 1; i <= 100; i++)
  for (int j = 1; j <= 5; j++) cuerpo-bucle-anidado
```

i	j
1	1
1	2
1	3
1	4
1	5
2	1
2	2
2	3
2	4
2	5
3	1
...	

Repetición: bucle for

Ejemplo.- Tablas de multiplicación del 1 al 10.

tablas.cpp

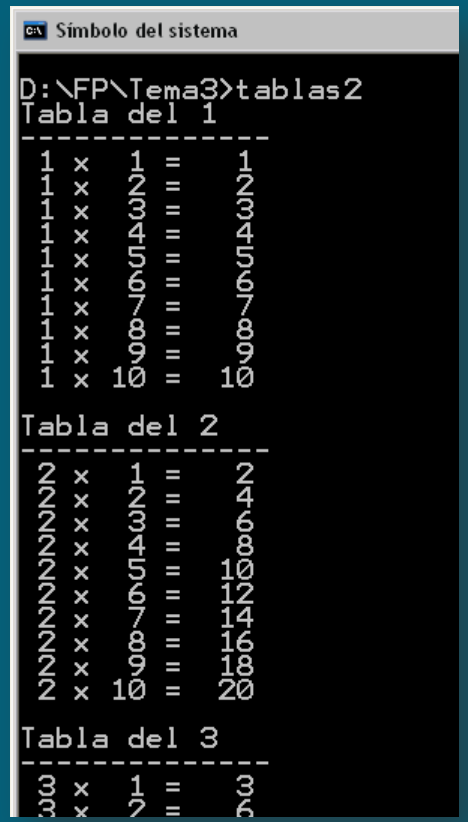
```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  for (int i = 1; i <= 10; i++)
    for (int j = 1; j <= 10; j++)
      cout << setw(2) << i << " x "
           << setw(2) << j << " = "
           << setw(3) << i * j << endl;

  return 0;
}
```

```
 1 x  1 =   1
 1 x  2 =   2
 1 x  3 =   3
 1 x  4 =   4
 ...
 1 x 10 =  10
 2 x  1 =   2
 2 x  2 =   4
 ...
10 x  7 =  70
10 x  8 =  80
10 x  9 =  90
10 x 10 = 100
```

Repetición: bucle for

Ejemplo.- Tablas de multiplicación (mejor presentadas). tablas2.cpp

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

int main()
{
  for (int i = 1; i <= 10; i++) {
    cout << "Tabla del " << i << endl;
    cout << "--------------" << endl;
    for (int j = 1; j <= 10; j++)
      cout << setw(2) << i << " x "
           << setw(2) << j << " = "
           << setw(3) << i * j << endl;
    cout << endl;
  }

  return 0;
}
```



Repetición: bucle for

Ejercicio.- Producto de números calculado en base a sumas.

El programa debe mostrar el producto de dos números enteros positivos introducidos por teclado, calculado en base a sumas.

Introducción a los subprogramas en C++

Forma general

```
Tipo Nombre(Parámetros) // Cabecera
{
  // Cuerpo
}
```

- *Tipo*: tipo del dato en el que se transforma la llamada al subprograma como resultado de su ejecución (tipo del resultado)
- *Parámetros* para intercambiar información con el exterior
- *Cuerpo*: secuencia de declaraciones locales e instrucciones.

Introducción a los subprogramas en C++

Tipos de subprogramas

- Procedimientos: subprogramas con Tipo `void`

  La llamada no se transforma en ningún dato.
  Llamada: instrucción independiente.

- Funciones: subprogramas con Tipo distinto de `void`

  La llamada se transforma en un dato del tipo indicado.
  Llamada: en cualquier punto donde pueda ir un valor del tipo del subprograma.

```
x = 12 * y + sqrt(20) - 3;
```

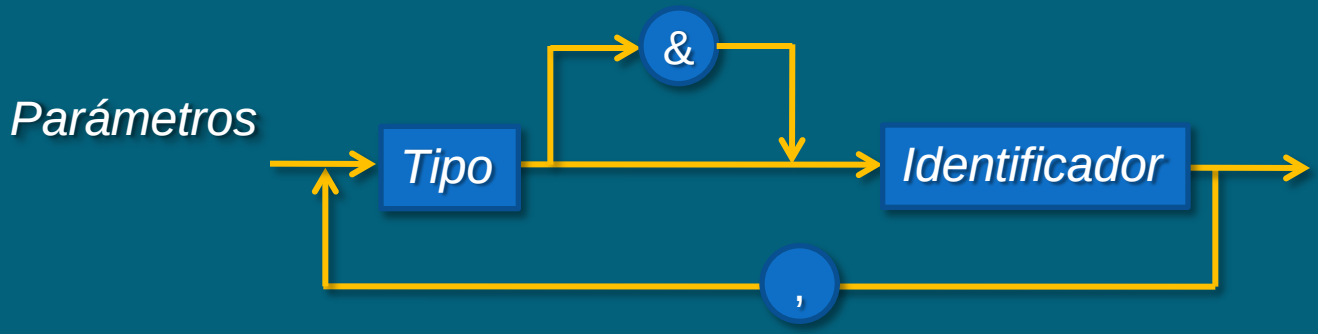

Introducción a los subprogramas en C++

Parámetros

Se distinguen dos tipos de parámetros:

- Parámetros por valor → datos de entrada
- Parámetros por referencia o por variable → datos de salida o E/S

Se declaran en la cabecera, a continuación del nombre de la función y entre paréntesis `Tipo Nombre(Parámetros)`



Introducción a los subprogramas en C++

Ejemplos de parámetros por valor y por referencia

```
void func1(int x)

void func2(int x, int y, int z, double& a)

bool func3(int x, int y, int& z, int& w)

int func4(float& a, int x)
```


Introducción a los subprogramas en C++

Llamada (o invocación) a subprograma

```
Nombre(Argumentos)
```

- Deben ponerse entre los paréntesis tantos argumentos como parámetros
- Debe respetarse el orden de declaración de los parámetros
- Cada argumento, del mismo tipo que el correspondiente parámetro
- Argumento para un parámetro por valor: una constante, una variable o una expresión
- Argumento para un parámetro por referencia/variable: sólo puede ser una variable

Introducción a los subprogramas en C++

Ejemplo:

```
int i;
double d;
void func(int x, double& a);


func(3, i, d);
func(i, d);
func(3 * i + 12, d);
func(i, 23);
func(d, i);
func(3.5, d);
func(i);
```

¿Qué llamadas son correctas?

Introducción a las funciones en C++

- Subprogramas con Tipo distinto de `void`
- La llamada (su ejecución) se transforma en un valor del tipo indicado

  La instrucción `return`

  - Devuelve el dato que se indique a continuación como resultado
  - Termina la ejecución de la función

  En una función puede haber más de una instrucción `return`
- Llamada: en cualquier punto donde pueda ir un valor del tipo del subprograma
- Las funciones suelen reservarse para cuando el subprograma devuelve un dato y sólo uno

Introducción a las funciones en C++

Ejemplo.- Función que calcula el área de un triángulo dadas su base y su altura


```
float calculaArea (float b, float a)  {
     return b * a / 2;
}

float calculaArea (float b, float a)  {
     float area;
     area = b * a / 2;
     return area;
}
```

¿Válidas ambas versiones?

Introducción a las funciones en C++

Ejemplo.- Función que calcula el área de un triángulo dadas su base y su altura

```
int main()
{
  float base, altura;
  cout << "Introduzca la base y la altura del triangulo: ";
  cin >> base;  cin >> altura;
  if ( altura > 0 && base > 0 )
    cout << "El area de un triangulo es: "
         << calculaArea(base, altura) << endl;
  else
    cout << "Datos erroneos!";
  return 0;
}
```


Introducción a las funciones en C++

Ejemplo.- Función que calcula el área de un triángulo dadas su base y su altura

```
int main()
{
  float base, altura, resultado;
  cout << "Introduzca la base y la altura del triangulo: ";
  cin >> base;   cin >> altura;
  if ( altura > 0 && base > 0 )  {
    resultado =  calculaArea(base, altura);
    cout << "El area de un triangulo es: " << resultado << endl;
  }
  else
    cout << "Datos erroneos!";
  return 0;
}
```

¿Es correcta esta segunda versión del programa?

Introducción a las funciones en C++

Ejemplo.- Programa que muestra por pantalla la suma de los enteros entre 1 y un número N (entero positivo) introducido por teclado

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

- *Función que solicita al usuario el número N, realizando control de datos*

- *Función que calcula la suma de los enteros entre 1 y un número N (entero positivo) que se le pasa como parámetro*

Programas con subprogramas en C++

¿Qué subprogramas hay en el programa?

¿En qué punto del archivo .cpp se colocan? ¿Antes de `main()`, después de `main()`?

El compilador necesita saber qué subprogramas se han declarado para saber si son correctas las llamadas a los mismos:

- ¿Existe el subprograma?
- ¿Concuerdan los argumentos con los parámetros?

Programas con subprogramas en C++

¿Antes de `main()`, después de `main()`?

```
#include <iostream>
using namespace std;
float calculaArea (float b, float a)  {
     return b * a / 2;
}
int main() {
  float base, altura, resultado;
  cout << "Introduzca la base y la altura del triangulo: ";
  cin >> base;   cin >> altura;
  if ( altura > 0 && base > 0 )  {
     resultado =  calculaArea(base, altura);
     cout << "El area de un triangulo es: " << resultado << endl;
  }
  else
   cout << "Datos erroneos!";
 return 0;
}
```

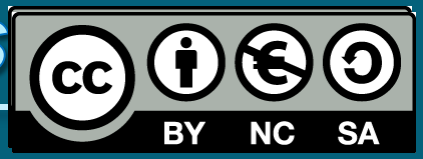

Programas con subprogramas en C++

¿Antes de `main()`, después de `main()`?

```
#include <iostream>
using namespace std;
float calculaArea (float, float);
int main() {
 float base, altura, resultado;
 cout << "Introduzca la base y la altura del triangulo: ";
 cin >> base;   cin >> altura;
 if ( altura > 0 && base > 0 )  {
    resultado =  calculaArea(base, altura);
    cout << "El area de un triangulo es: " << resultado << endl;
 }
 else
  cout << "Datos erroneos!";
 return 0;
}
float calculaArea (float b, float a)  {
       return b * a / 2;
}
```

Inclusión del prototipo

Acerca de *Creative Commons*



Licencia CC (Creative Commons)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

- Reconocimiento (*Attribution*):
  En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
- No comercial (*Non commercial*):
  La explotación de la obra queda limitada a usos no comerciales.
- Compartir igual (*Share alike*):
  La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

En http://es.creativecommons.org/ y http://creativecommons.org/ puedes saber más de Creative Commons.