Universidad LOYOLA

Departamento de Métodos Cuantitativos

# Tema 2. Conceptos básicos de C++

Alfonso Carlos Martínez Estudillo (acme@uloyola.es)

**Fundamentos de Informática I**

**1º de Grado en Ingeniería Informática y Tecnologías Virtuales**

**Curso 2021-2022**

# Índice de contenidos

1. Introducción

2. Tipos de Datos Básicos

3. Entrada y Salida

4. Cadenas: el Tipo String

5. Operadores

6. Documentación

# Outline

1. **Introducción**
   - Elementos de un Programa en C++
   - Estructura de un Programa en C++
   - El Compilador
2. Tipos de Datos Básicos
3. Entrada y Salida
4. Cadenas: el Tipo String
5. Operadores
6. Documentación

# Elementos de un programa en C++

- **Tipos de datos**:
  - Fundamentales.
  - Compuestos.
  - Definidos por el usuario.
- **Variables**.
- **Operadores**.
- **Instrucciones o sentencias**:
  - Asignación.
  - Entrada/Salida.
- **Estructuras de control**:
  - Secuenciales.
  - Condicionales.
  - Iterativas.

# Otros elementos

- **Tokens**: carácter o conjunto de caracteres alfanuméricos o simbólicos que representan algo para el compilador.
- **Conjunto de palabras reservadas**: tokens de caracteres alfanuméricos que no se pueden utilizar para otra cosa.
- **Identificadores**:
  - Nombre de las variables, constantes, funciones, ...
  - Secuencia de caracteres, letras, dígitos y subrayados.
  - Deben comenzar por una letra o por _.
- **Signos de puntuación**: ;

# Otros elementos

- **Reglas gramaticales y sintaxis**: como crear sentencias del lenguaje correctas, deben respetarse.
- **Comentarios**: líneas que no son interpretadas por el compilador. En C++: /* */ //
- **Separadores**: espacios en blanco, tabuladores, retornos de carro (salto de línea).

# Estructura básica

Inclusión de bibliotecas
Definición de constantes
Prototipos de funciones
Espacio de nombres

```
int main ( ) {

    Declaración de variables
    Sentencias del programa

    return 0;

}
```

Funciones

# Estructura básica

- **Función main**: bloque de código encerrado entre llaves que incluye las sentencias que debe hacer el programa.
- **Inclusión de bibliotecas**: nos permiten usar funciones u operadores predefinidos por el compilador.

```
1 #include<iostream>
```

- **Definición de constantes**: nos permiten definir elementos que no cambiarán su valor durante la ejecución del programa.

```
1 #define PI 3.1416
```

# Estructura básica

- **Espacio de nombres**: muchas bibliotecas lo necesitan.

```
1 using namespace std;
```

- **Declaración de variables**: las que vamos a utilizar en nuestro programa.

```
1 int radio;
```

- **Sentencias del programa**: instrucciones que realiza el programa.

```
1 radio = 3 * 3;
```

# Estructura básica

- **Prototipos de funciones**: el nombre de la función debe ir antes de su declaración.

```
1 int area(int r);
```

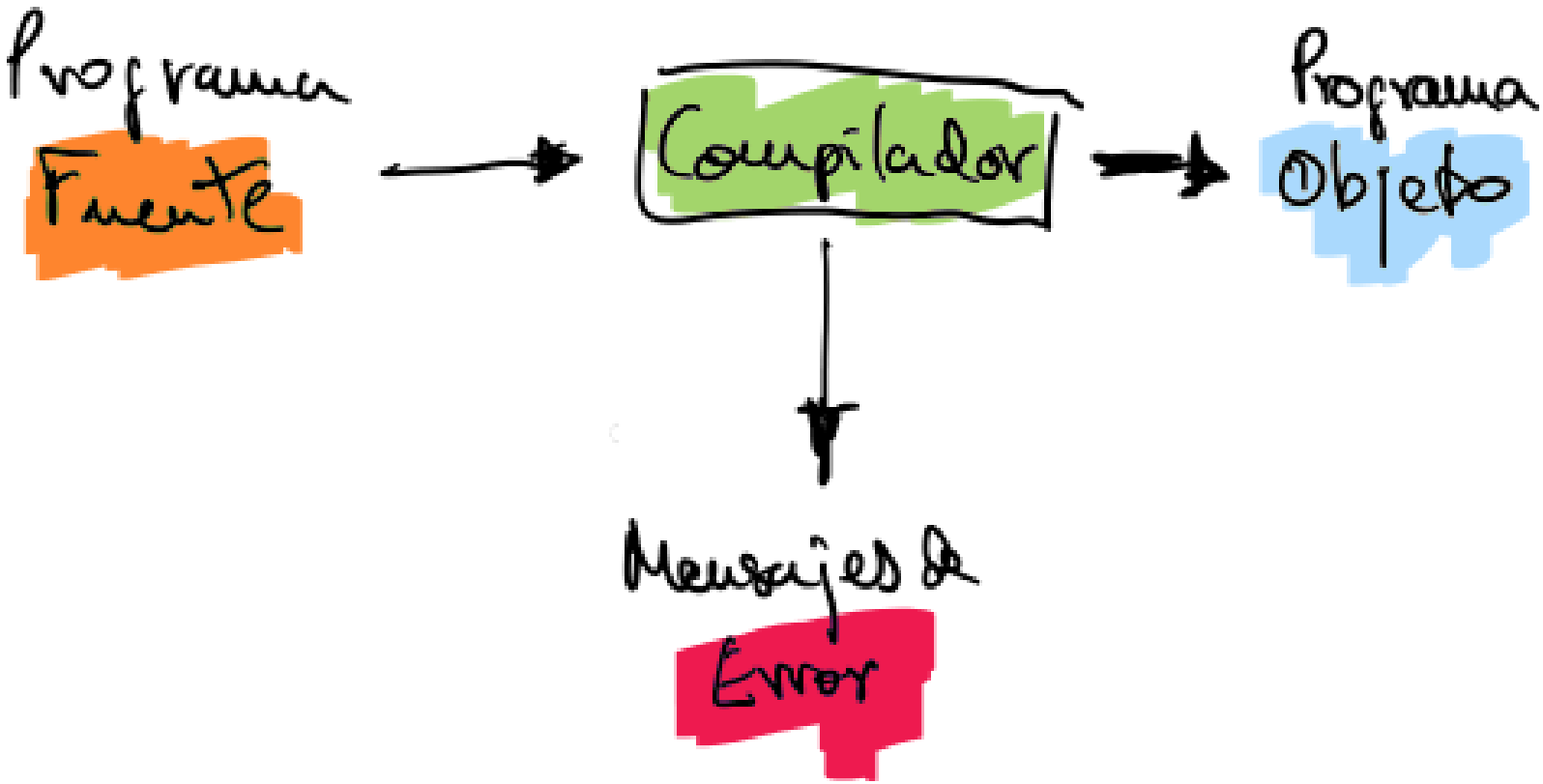
- **Funciones**: nos permiten agrupar trozos de código para utilizarlos varias veces.

```
1 int area(int r){
2   return PI*r*r;
3 }
```

# Ejemplo completo

```
1  #include<iostream>
2  #define PI 3.1416
3  using namespace std;
4
5  int main(){
6    int radio;
7    radio = 3*3;
8    int a = area(radio);
9    cout << a << endl;
10   return 0;
11 }
12
13 int area(int r)
14 {
15   return PI*r*r;
16 }
```

# El compilador



# Algunos ejemplos de compilación

- Compilación básica:

```
1 g++ main.cpp
```

- Compilación proporcionando nombre del programa:

```
1 g++ main.cpp -o programa.exe
```

- Compilación mostrando advertencias:

```
1 g++ main.cpp -Wall
```

A lo largo de la asignatura veremos distintas formas de compilar nuestros programas.

# Outline

1. Introducción

2. **Tipos de Datos Básicos**
   - Tipos de Datos
   - Tipo de Dato Entero
   - Tipo de Dato Real
   - Tipo de Dato Lógico
   - Tipo de Dato Carácter
   - Constantes
   - Variables
   - Operador Asignación
   - Conversiones

3. Entrada y Salida

4. Cadenas: el Tipo String

5. Operadores

6. Documentación

# Tipos

C++ tiene los siguientes tipos fundamentales:

- **Caracteres**: char.
- **Enteros**: short, int, long, long long.
- **Números en coma flotante**: float, double, long double.
- **Booleanos**: bool.
- **Vacío**: void.

El modificador **unsigned** se puede aplicar a enteros para obtener números sin signo con lo que se consigue un rango mayor de números naturales.

# Tipos

- **Tipos compuestos en C++**: cadenas de caracteres (lo simplificaremos con el uso de la clase String), vectores, matrices, registros y clases (se verán el próximo curso).
- **Características de los tipos de datos**:
  - El rango de valores que puede representar, que depende de la cantidad de memoria que dedique el compilador a su representación interna.
  - El conjunto de operadores que pueden aplicarse a los datos de este tipo.

# Enteros

Aunque el tamaño en bytes, y el rango de valores de cada tipo depende del SO y del compilador, por norma general tenemos lo siguiente:

| Tipo | Descripción | Número de bytes | Rango |
|---|---|---|---|
| short | Entero corto | 2 | -32768 a 32767 |
| int | Entero | 4 | -2147483648 a +2147483647 |
| long | Entero largo | 4 | -2147483648 a +2147483647 |
| char | Carácter | 1 | -128 a 127 |

# Enteros

Con los tipos enteros pueden utilizarse los calificadores **signed** y **unsigned**. Estos calificadores indican si el número tiene signo o no.

| Tipo | Descripción | Número de bytes | Rango |
|---|---|---|---|
| signed short | Entero corto | 2 | -32768 a 32767 |
| unsigned short | Entero corto sin signo | 2 | 0 a 65535 |
| signed int | Entero | 4 | -2147483648 a +2147483647 |
| unsigned int | Entero sin signo | 4 | 0 a 4294967295 |
| long | Entero largo | 4 | -2147483648 a +2147483647 |
| unsigned long | Entero largo sin signo | 4 | 0 a 4294967295 |
| signed char | Carácter | 1 | -128 a 127 |
| unsigned char | Carácter sin signo | 1 | 0 a 255 |

# Reales

| Tipo | Descripción | Número de bytes | Rango |
|---|---|---|---|
| float | Real | 4 | Positivos: 3.4E-38 a 3.4E28<br>Negativos: -3.4E-38 a -3.4E28 |
| double | Real doble | 8 | Positivos: 1.7E-308 a 1.7E308<br>Negativos: -1.7E-308 a -1.7E308 |
| long double | Real doble largo | 10 | Positivos: 3.4E-4932 a 1.1E4932<br>Negativos: -3.4E-4932 a -1.1E4932 |

# Booleanos

| Tipo | Descripción | Número de bytes | Rango |
|---|---|---|---|
| bool | Dato de tipo lógico | 1 | true(1) o false(0) |

# Caracteres

- Palabra reservada: char.
- Se utiliza para representar letras, dígitos, signos de puntuación, ...
  - Cualquier elemento de un alfabeto predefinido.
  - Código ASCII. https://aprende.olimpiada-informatica.org/cpp-char-ascii.
- Representación como un literal (entre comillas simples): 'A', '1', ';'.
- Internamente se representa como un entero.
- Se pueden realizar operaciones aritméticas, las mismas que con los enteros.
- Podemos representar cadenas de caracteres mediante vectores de caracteres, nosotros utilizaremos el tipo String que veremos más adelante.

# Constantes

- Objetos cuyo valor no se puede modificar a lo largo de un programa.
- Tipos constantes: definidas, declaradas y enumeradas.
- **Constantes definidas**: se definen mediante la directiva de precompilación #define.

```
1 #define PI 3.1416
```

- **Constantes declaradas**: usando el calificador const, se declara el tipo y el nombre.

```
1 const int meses=12;
```

- **Constantes enumeradas**: permiten crear listas de elementos afines.

```
1 enum Boolean {False, True}
```

# Variables

- Permiten almacenar información durante la ejecución de un programa: su valor puede ser modificado en el transcurso del mismo.
- Sirven para nombrar una posición de memoria: lugar donde se almacena la información.
- Deben declararse antes de ser utilizadas.
- Se declaran indicando el tipo, el nombre y opcionalmente un valor inicial.

```
1 int radio = 3;
```

# Tipos de variables

- **Variables globales**:
  - Se declaran al principio del programa, fuera de toda función.
  - Pueden utilizarse en cualquier punto del programa.
  - Su uso injustificado es una mala práctica de programación.
- **Variables locales**:
  - Tienen un ámbito de visibilidad.
  - Se crean al principio de un bloque y se destruyen al final.
- **Variables dinámicas**:
  - Se crean y liberan tras una petición explícita.

# Operador asignación

- Sirve para asignar un valor a una variable.
- Se utiliza el símbolo $=$.

```
1 a=3+2;
```

- Es asociativo por la derecha, se pueden encadenar operadores.

```
1 a=b=c=d=7;
```

- Se puede combinar con otros operadores aritméticos (lo veremos más adelante).

# Conversiones implícitas

- Si asignamos un tipo de dato pequeño a un tipo de dato grande no habrá problema.

```
1 int a = 45;
2 long b = a;
3 //b será igual a 45
```

- Si asignamos un tipo de dato grande a uno pequeño y cabe, tampoco habrá problemas.

```
1 int a;
2 long b = 45;
3 a = b;
4 //a será igual a 45
```

- Si no cabe, se almacenará basura.

# Conversión entero a flotante y viceversa

Flotante a entero:

```
1 int a;
2 float b=3.14;
3 a=b;
4 //a valdrá 3
```

Entero a flotante:

```
1 int a=3;
2 float b;
3 b=a;
4 //b valdrá 3.0
```

# Conversiones explícitas

- La solicita el programador.
- Se utiliza el operador cast: hacemos un casting.
- Ejemplos:

```
1 (float) i // Convierte i en un float
2 (int) 3.4 // Convierte 3.4 a entero con valor 3
3 k = int (1.7) * (int) masa //Convierte 1.7 a 1, convierte
      masa a entero, los multiplica y el valor se asigna a k
```

# Outline

1. Introducción

2. Tipos de Datos Básicos

**3. Entrada y Salida**
- Salida de Datos Estándar
- Entrada de Datos Estándar

4. Cadenas: el Tipo String

5. Operadores

6. Documentación

# Salida estándar

- Para mostrar datos por pantalla utilizaremos la funcion **cout** junto a los operadores $<<$.

```
1 int a = 3;
2 cout << a << endl;
```

- La palabra reservada **endl** introduce un salto de línea.
- **cout** es capaz de sacar por pantalla todos los datos primitivos vistos anteriormente, además del tipo de dato **string** que veremos en la siguiente sección de este tema.

# Salida estándar

Podemos mostrar caracteres especiales para mejorar el formato de la salida. Estos caracteres van precedidos de la barra inclinada \ y se llaman secuencias de escape.

| Secuencia | Significado |
|---|---|
| \n | Salto de línea |
| \t | Tabulador |
| \b | Retrocede un carácter |
| \r | Retorno de carro |
| \f | Salto de página |
| \' | Comilla simple |
| \" | Comilla doble |
| \\ | Barra inclinada |
| %% | Tanto por ciento |

# Entrada estándar (cin)

- Para introducir datos por teclado utilizaremos la función **cin** y el operador >>.

```
1 int a;
2 cin >> a;
```

- Es conveniente leer un sólo dato por línea.
- **cin** es capaz de leer por teclado todos los datos primitivos vistos anteriormente, además del tipo de dato **string** que veremos en la siguiente sección de este tema.

# Entrada estándar (getchar)

- Instrucción de entrada de datos especializada que se utiliza para leer un único carácter desde teclado.
- Es necesario pulsar **Intro** para que se produzca la operación de lectura.
- El resultado de la instrucción es un carácter que se puede asignar a una variable.

```
1 char c;
2 cout << "Introduzca un carácter: ";
3 c = getchar();
4 cout << c << endl;
```

# Entrada estándar (getchar)

- Los datos leídos no se almacenan directamente en la variable sino en una zona de memoria, llamada *buffer*.
- Se almacenan todas las pulsaciones, incluidas la tecla **Intro**.
- Por tanto, cada vez que se utiliza un getchar(); puede quedar un carácter adicional basura correspondiente a la tecla **Intro**.
- Es necesario eliminarlo usando otro getchar() adicional.

# Entrada estándar (getline)

- Lee una cadena de caracteres que puede contener espacios en blanco.
- El primer parámetro es el flujo cin para leer desde teclado y el segundo la cadena en la que queremos guardar la información

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main()
6  {
7    string nombre;
8    cout << "Escriba el nombre completo: ";
9    getline(cin,nombre);
10   cout << "Te llamas: " << nombre << endl;
11   return 0;
12 }
```

# Outline

1. Introducción

2. Tipos de Datos Básicos

3. Entrada y Salida

**4. Cadenas: el Tipo String**
Introducción
El Tipo String

5. Operadores

6. Documentación

# Cadenas

- Como hemos estado viendo en ejemplos anteriores, una cadena se define entre comillas dobles.
- Las cadenas son un vector de caracteres donde el elemento final es '\0'.
- Nosotros, para abstraernos de esta problemática utilizaremos un dato derivado denominado **string**.
- En C++, la primera posición de una cadena comienza en el índice 0.
- **string** se utiliza para representar cadenas de caracteres y cuenta con una serie de propiedades que pueden ser utilizadas.

# Acceso a un elemento de una cadena

- Para acceder a una posición concreta de una cadena se utiliza la función **.at()**.

```
1 string cadena = "hola";
2 cout << cadena.at(0) << endl; //Devuelve el primer elemento
      de la cadena, es decir, 'h'.
```

- El acceso se puede realizar con el operador corchetes [ ], es decir, lo anterior también se puede realizar con:

```
1 string cadena = "hola";
2 cout << cadena[0] << endl;
```

# Buscar una subcadena en una cadena

- Se puede llevar a acabo la búsqueda de subcadenas o caracteres dentro de nuestro **string**.
- Para ello, se hace uso de la función **.find()** que devuelve la posición del primer carácter de nuestra cadena en el que comienza la subcadena.

```
1 string cadena = "hola soy Antonio";
2 int posicion = cadena.find("soy");
3 cout << posicion << endl;
```

# Substraer una subcadena de una cadena

- Podemos extraer una subcadena entre dos posiciones con la función **.substr()**.
- El primer parámetro indica el carácter a partir empezamos a substraer.
- El segundo parámetro indica el número de caracteres a substraer.

```
1 string cadena = "hola soy Antonio";
2 string subcadena = cadena.substr(5,3);
```

# Obtener el tamaño de una cadena

- Para obtener el tamaño de una cadena se hará uso de la función **.length()**.
- También se puede obtener el tamaño con la función **.size()**.
- Ambos retornan un entero.

```
1 string cadena = "hola";
2 cout << "La longitud de la cadena es: " << cadena.length() <<
      endl;
3 cout << "La longitud de la cadena es: " << cadena.size() << endl;
```

# Variar el tamaño de una cadena

- Tenemos el método **.resize()**.
- Recibe como parámetro un número entero que indica cual es la nueva longitud de la cadena.
- Podemos acortarla o hacerla más larga dependiendo de nuestras necesidades.

```
1 string cadena = "hola";
2 cout << "La longitud de la cadena es: " << cadena.length() <<
      endl;
3 cadena.resize(8);
4 cout << "La nueva longitud es: " << cadena.length() << endl;
```

# Saber si una cadena esta vacía

- Se usará el método **.empty()**.
- No recibe ningún parámetro.
- Devuelve true si está vacía y false en caso contrario.

```
1    string cadena = "hola";
2    string cadena2 = "";
3    cout << "La cadena esta vacia: " << cadena.empty() << endl;
4    cout << "La cadena esta vacia: " << cadena2.empty() << endl;
5
```

# Intercambiar el contenido entre dos cadenas

- Para intercambiar el contenido de dos cadenas se llama a la función **.swap()**.

```
1 string cadena = "hola";
2 string cadena2 = "adios";
3 cadena.swap(cadena2);
4 cout << "La cadena es: " << cadena << endl;
5 cout << "La cadena2 es: " << cadena2 << endl;
```

# Aún hay más...

Esto es sólo una pequeña parte de todo el potencial de la clase **string**, para más información es muy recomendable visitar: https://en.cppreference.com/w/cpp/string/basic_string.

# Outline

1. Introducción

2. Tipos de Datos Básicos

3. Entrada y Salida

4. Cadenas: el Tipo String

**5. Operadores**
- Introducción
- Operador de Asignación
- Operadores Aritméticos
- Operadores Relacionales
- Operadores Lógicos
- Operadores a Nivel de Bits
- Otros Operadores
- Precedencia de Operadores

6. Documentación

# Operadores en C++

- Sirven para crear expresiones:
  - Unión de una serie de operadores y operandos (constantes y variables) que puede ser evaluada.
  - Su valor debe asignarse a una variable o mostrarse por pantalla para que no se pierda.
- Tipos de operadores:
  - Operador de asignación.
  - Operadores aritméticos.
  - Operadores lógicos.
  - Operadores relacionales.
  - Operadores a nivel de bits.

# Operador de asignación

- Asigna un valor a una variable.
  - Usa el símbolo =
- El operador = asigna el valor de la expresión de la derecha a la variable situada a la izquierda.

```
variable = 3 * 2;
b = variable + 5;
```

- Es asociativo por la derecha. Permite realizar asignaciones múltiples:

```
var1=var2=var3=20;
```

- Se puede combinar con otros operadores aritméticos para obtener un operador abreviado de asignación.

```
i *= 10; // es igual a
i = i * 10;
```

# Operadores aritméticos

| Operador | Tipos enteros | Tipos Reales | Ejemplo |
|---|---|---|---|
| + | Suma | Suma | $x + y$ |
| - | Resto | Resto | $x - y$ |
| * | Producto | Producto | $x * y$ |
| / | División entera: cociente | División en coma flotante | $x/5$ |
| % | División entera: resto | | $x\,\%5$ |

- Siguen las reglas de precedencia de las operaciones algebraicas.
- Pueden alterares con el uso del paréntesis.

# Operadores aritméticos

- Combinación de operadores aritméticos con el operador de asignación.

| Operador | Sentencia abreviada | Sentencia no abreviada |
|---|---|---|
| += | a+=b | a=a+b |
| -= | a-=b | a=a-b |
| *= | a*=b | a=a*b |
| /= | a/=b | a=a/b |
| %= | a %=b | a=a %b |

# Operadores aritméticos: incremento y decremento

- Incrementan o decrementan en una unidad el valor de una variable:
  - i++; equivale a i=i+1;
  - i- -; equivale a i=i-1;
- Pueden escribirse antes o después de la variable, pero no tienen el mismo significado.
  - **Prefijo**: primero se incrementa o decrementa y luego se hace la asignación. ++i;
  - **Postfijo**: primero se hace la asignación y luego se incrementa o decrementa; i++;

# Operadores aritméticos: incremento y decremento

## Postfijo

```
1 int a = 1;
2 b = a++;
3 // a = 2
4 // b = 1
```

## Prefijo

```
1 int a = 1;
2 b = ++a;
3 // a = 2
4 // b = 2
```

# Operadores aritméticos: operadores no definidos

- Se encuentran en la biblioteca cmath.
  - abs(x) → Valor absoluto de x.
  - pow(x,y) → Calcula x elevado a y.
  - cos(x) → Calcula el coseno de x (ángulo en radianes).
  - sin(x) → Calcula el seno x (ángulo en radianes).
  - sqrt(x) → Raíz cuadrada de x.
  - tan(x) → Calcula la tangente de x (ángulo en radianes).
  - log(x) → Calcula el logaritmo neperiano de x.
  - exp(x) → Calcula la función exponencial de x.

# Operadores relacionales

- Comprueban la relación entre dos expresiones.
- expresion1 operador expresion2
- Devuelve true si evalúa como verdadero.
- Devuelven false si evalúa como falso.

| Operador | Significado | Ejemplo |
|---|---|---|
| == | Igual a | a == b |
| != | Distinto de | a!=b |
| > | Mayor que | a>b |
| < | Menor que | a<b |
| >= | Mayor o igual que | a>=b |
| <= | Menor o igual que | a<=b |

# Operadores relacionales

```
1 int a = 1;
2 bool c;
3 c = a < 2;
4 // c es true
```

```
1 int a = 1;
2 bool c;
3 c = a > 2;
4 // c es false
```

# Operadores relacionales

No siempre se puede comprobar la igualdad de números flotantes con el operador ==.

- Sus valores pueden ser los mismos, pero debido al límite en la precisión y a errores de redondeo pueden diferir en una pequeña cantidad, y pueden ser considerados desiguales.
- Para comprobar la igualdad de dos números, a y b, en coma flotante, se comparará su diferencia con un valor en coma flotante muy pequeño (error permitido).

```
1 float error=0.000001;
2 float a, b;
3 bool c;
4 c=fabs(b-a)<error;
```

# Operadores lógicos

- Permiten combinar entre sí expresiones lógicas.
- Devuelven un booleano.
- true si es verdadero, o false si es falso.

| Operador | Operación Lógica | Ejemplo |
|---|---|---|
| Negación (!) | !(expresión) | !(x>=y) |
| Y lógica (&&) | expresion1 && expresion2 | a < b && b < c |
| O lógica(\|\|) | expresión1 \|\| expresión2 | a > 5 \|\| c < a |

# Operadores lógicos: tablas de verdad

| O lógica(\|\|) | **true** | **false** |
|---|---|---|
| **true** | true | true |
| **false** | true | false |

| Y lógica(&&) | **true** | **false** |
|---|---|---|
| **true** | true | false |
| **false** | false | false |

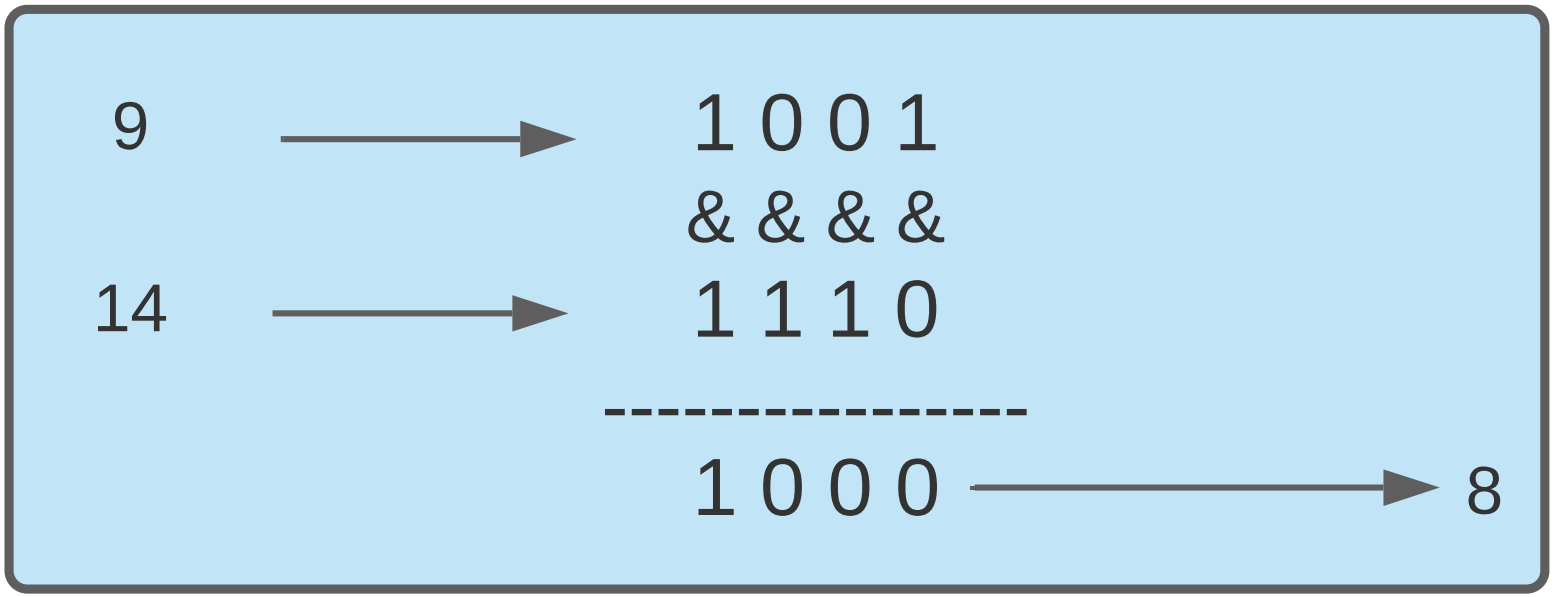
| Negación (!) | |
|---|---|
| **true** | false |
| **false** | true |

# Operadores a nivel de bits

| Operador | Operación |
|---|---|
| & | Y lógica bit a bit. |
| \| | O lógica bit a bit. |
| $\wedge$ | XOR lógica bit a bit. |
| $\sim$ | Complemento a uno (inversión de todos los bits). |
| << | Desplazamiento de bits a la izquierda. |
| >> | Desplazamiento de bits a la derecha. |

# Operadores a nivel de bits

Ejemplo: 9 & 14 = 8.



# Otros operadores

- **Operador condicional ternario**: puede sustituir a la estructura if-else en algunas expresiones.

```
1 expresion ? exp1 : exp2;
```

- **Operador coma**: permite combinar dos o más expresiones en una misma línea.

```
1 i++, j++, k++; // igual a i++; j++; k++;
```

- **Operador sizeof**: devuelve el tamaño que ocupa (en bytes) un tipo de dato o variable.

```
1 sizeof(int);
```

# Precedencia de operadores

| Orden | Categoría | Operadores |
|---|---|---|
| 1 | Más alta | ( ), [ ], ->, ., |
| 2 | Unarios | !, ++, - -, &, *, sizeof |
| 3 | Multiplicativos | *, /, % |
| 4 | Aditivos | +, - |
| 5 | Rotación | >>, << |
| 6 | Relacional | <, <=, >, >= |
| 7 | Igualdad | == |
| 8 | Y bit a bit | & |
| 9 | O bit a bit | ∧ |
| 10 | | \| |
| 11 | Y lógico | && |
| 12 | O lógico | \|\| |
| 13 | Condicional | ?: |
| 14 | Asignación | *=,=,+=,-=,... |
| 15 | Coma | , |

# Outline

1. Introducción

2. Tipos de Datos Básicos

3. Entrada y Salida

4. Cadenas: el Tipo String

5. Operadores

**6. Documentación**
- Introducción
- Documentación Interna
- Documentación Externa

# Documentación

- Un programa es escrito pr un programador, pero puede ser consultado o reutilizado por otras personas o programadores.
- De este modo, cualquier programa necesitará durante su vida un mantenimiento.
  - Un programa puede contener errores que pasen desapercibidos.
  - Es posible que sea revisado y modificado por otro programador.
  - También puede ocurrir que el cliente nos pida modificaciones del mismo.
- Surge la necesidad de **documentar** el código.

# Documentación

La documentación consiste en describir lo que hace el programa y como lo hace. Se distinguen:

- **Documentación interna**: aportada en el propio código.
- **Documentación externa**: fuera del código (manuales de usuario, código, etc).

# Comentarios

- Es lo más utilizado.
- Todo lenguaje proporciona la forma de incluir comentarios.
- Si un pseudo-código incorpora comentarios, se deben trasladar al código.
- Se recomienda comentar todo lo que plantee duda y no sobre-cargar el código de comentarios de forma excesiva.

# Comentarios

Cabeceras de las funciones o procedimientos:

- Nombre, Propósito, Autor.
- Parámetros de entrada y salida.
- Precondiciones y postcondiciones.
- Procedimientos o funciones a los que llama.
- Procedimientos y funciones por los que es llamado.
- Códigos de error.
- Fecha de creación.
- Fecha de modificaciones.
- Cambios realizados.

# Comentarios

Supongamos una función que calcula la suma cuadrada de dos elementos:

```
/*-----------------------------------------------------------
  Nombre: sumaCuadrados
  Tipo: entero (int)
  Objetivo: calcula la suma de los cuadrados de dos numeros
  Parámetros de entrada:
    int a: primer numero
    int b: segundo numero
  Precondiciones: Ninguna
  Valor devuelto: a*a + b*b
  Autor: Antonio Duran
  Fecha: 24-02-2021
-----------------------------------------------------------*/
int sumaCuadrados(int a, int b);
```

# Comentarios

- **Variables o tipos compuestos**: asignar un nombre adecuado a su función. Si no es posible, incluir un comentario aclaratorio.
- **Esquemas condicionales e iterativos**: clarificar su propósito, condiciones de salida o mantenimiento, etc.
- **Llamadas a subprogramas o funciones**: indicar la función y el efecto que tiene sobre las variables usadas por la misma.

# Presentación

- Para facilitar la compresión del código se deben identar el mismo según el nivel de anidamiento.
- Agrupar sentencias de E/S.
- Correspondencia entre el orden de ubicación de los bloques y el orden en que van a ser ejecutados.
- Usar líneas en blanco para separar los bloques.
- Utilizar espacios entre operadores y en los parámetros de las funciones.

# Presentación: ejemplos

**Nombre de las variables**: programa que calcula si la edad de un usuario es menor de 25 y su salario mayor de 10.000.

```
int a, b;
if (a < 25) && (b > 10000){
  return true;
}
else{
  return false;
}
```

```
int edad, salario;
if (edad < 25) && (salario > 10000){
  return true;
}
else{
  return false;
}
```

# Presentación: ejemplos

**Identación**: mismo programa que el anterior.

```
1 int edad, salario;
2 if (edad < 25) && (salario > 10000){
3 return true;}
4 else{return false;}
```

```
1 int edad, salario;
2 if (edad < 25) && (salario > 10000){
3   return true;
4 }
5 else{
6   return false;
7 }
```

# Presentación: ejemplos

**Espaciado**: mismo programa que el anterior.

```
1 int edad,salario;if(edad<25)&&(salario>10000){return true;}else{
      return false;}
```

```
1 int edad, salario;
2 if (edad < 25) && (salario > 10000){
3   return true;
4 }
5 else{
6   return false;
7 }
```

# Documentación externa

- **Manual de usuario**:
  - Destinado a facilitar el uso del programa a los usuarios finales.
  - Explicación detallada con gráficos de todas las opciones de la aplicación.
  - Debe contener ejemplos prácticos.
- **Manual de operador**:
  - Destinado al operador informático.
  - Instalación.
  - Entorno de funcionamiento.
  - Requerimientos hardware.
- **Manual de mantenimiento**:
  - Destinado a los programadores que se encargan del mantenimiento.

# Referencias

## Recursos electrónicos:


- cppreference. (2020). Referencia C++: https://en.cppreference.com/w/
- Goalkicker.com (2018). C++: Note for Professionals.
- Grimes, R. (2017). Beginning C++ Programming.
- Joyanes, L. (2000). Programación en C++: Algoritmos, Estructuras de datos y Objetos.
- Juneja, B.L., Seth, A. (2009). Programming with C++.

# Referencias

## Libros:


- Prieto, A., Lloris, A., Torres, J.C. (2008). Introducción a la Informática (4 ed).
- Joyanes, L. (2008). Fundamentos de programación : algoritmos, estructura de datos y objetos.
- Martí, N., Ortega, Y., Verdejo, J.A. (2003). Estructuras de datos y métodos algorítmicos: Ejercicios resueltos.
- Tapia, S., García-Beltrán, A., Martínez, R., Jaen, J.A., del Álamo, J. (2012). Fundamentos de programación en C.

# ¿Preguntas?