

CONTENIDOS

- 1. Estructura básica de un programa C++.**
 - 2. Tipos de datos simples.**
 - 3. Constantes y variables en C++. Declaración.**
 - 4. Operadores y expresiones.**
 - 5. Instrucciones de Entrada y Salida.**
- Algunas características de C++.**

1

Estructura básica de un programa C++

Estructura de un programa C++

Componente estructural básico: la función

Funciones

Una de las funciones ha de ser **main**

```
Directivas de preprocesador
Declaraciones globales ( variables globales, funciones, ...)

función main()
{
    secuencia de declaraciones e instrucciones
}

función1()
{
    secuencia de declaraciones e instrucciones
}

...
funciónN()
{
    secuencia de declaraciones e instrucciones
}
```

Estructura de un programa C++

Un ejemplo sencillo de un programa que intercambia el valor de 2 números:

```
Directiva de preprocesamiento
Biblioteca de E/S por consola
Este programa usa la definición de
cout para escribir por consola

#include <iostream.h>
int main() ← Cabecera de la función
{
    int x, y;
    int aux;
    cin >> x >> y;
    aux = x;
    x = y;
    y = aux;
    cout << x << y;
    return 0;
} ← Cuerpo de la función { ... }
```

Estructura de un programa C++

Un ejemplo sencillo de un programa que intercambia el valor de 2 números:

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

    cin >> x >> y;

    aux = x;
    x = y;
    y = aux;

    cout << x << y;

    return 0;
}
```

Declaración de variables locales

Flujo de entrada

Flujo de salida por estándar (pantalla)

<< : operador de inserción para flujos de salida

Devuelve el valor-resultado

Estructura de un programa C++

Directivas del preprocesador

Los compiladores de C++ proporcionan bibliotecas de funciones.

Cada biblioteca de funciones tiene asociada un archivo de definición que se denomina *cabecera*.

Para utilizar algo de una biblioteca en un programa, hay que colocar al principio del programa una *directiva de preprocesamiento* seguida de la cabecera de la biblioteca entre ángulos.

```
#include <iostream.h>
```

Indica al compilador que lea las directivas antes de compilar la función principal

Instrucciones al compilador antes de que se compile el programa principal

Las directivas más usuales son:
include
define

Estructura de un programa C++

Directivas del preprocesador

Constante de cadena de caracteres

```
#include <iostream.h>
```

```
int main()  
{  
    cout << "Hola amigos";  
    return 0;  
}
```

Por ejemplo, para mostrar datos en la pantalla podemos usar el operador << con el elemento **cout**,

```
cout << "Hola amigos";
```

Pero para poder usar **cout** y << debemos incluir la biblioteca donde están definidos.

Esta biblioteca tiene como archivo de cabecera **iostream.h**

Una vez que se incluye el archivo de cabecera, se puede utilizar todo lo que está definido en ella.

Estructura de un programa C++

Directivas del preprocesador

Existen archivos de cabecera estándar muy utilizados

Uso de funciones matemáticas

stdlib.h
string.h
math.h
conio.h
iostream.h
type.h

Uso de funciones de cadena

Uso de funciones de E/S

Funciones de clasificación de caracteres

El uso más frecuente en C++ de las directivas del preprocesador es la inclusión de archivos de cabecera, pero también se usan para definir macros, nombres de constantes, etc.

Estructura de un programa C++

La función main()

Una función C++ es un subprograma que devuelve un valor, un conjunto de valores o realiza una tarea específica.

Todo programa C++ tiene una única función main() que es el punto inicial de entrada al programa.

```
#include <iostream.h>
```

```
main()  
{  
  ...  
  ...  
}
```

Las sentencias escritas entre las llaves se denomina **BLOQUE**

Llamadas a otras funciones

```
#include <iostream.h>
```

```
int main()  
{  
  entrada_datos();  
  proceso_datos();  
  return 0;  
  ...  
}
```

Si se intenta declarar dos funciones main() dentro del programa se produce error.

Estructura de un programa C++

Comentarios

Un comentario es cualquier información que se escribe en el programa para proporcionar información de cualquier tipo.

```
#include <iostream.h>
```

```
/* podemos hacer  
comentarios que ocupen  
varias líneas */
```

Podemos escribir los comentarios de dos formas diferentes

```
int main()  
{
```

```
  int x, y;
```

```
  int aux;
```

```
  cin >> x >> y;
```

```
  ...
```

```
  return 0;
```

```
}
```

```
// éste es un comentario de una sola línea
```

2

Tipos de datos simples

Tipos de datos básicos en C++

El **tipo de dato** determina la naturaleza del valor que puede tomar una variable. Un tipo de dato define un *dominio* de valores y las *operaciones* que se pueden realizar con éstos valores.

C++ dispone de unos cuantos tipos de datos predefinidos (simples) y permite al programador crear otros tipos de datos

Tipo de datos básicos

- **int** (Números enteros)
- **float** (Números reales)
- **double** (Números reales más grandes que float)
- **bool** (Valores lógicos)
- **char** (Caracteres y cualquier cantidad de 8 bits)
- **void** (Nada. Sirve para indicar que una función no devuelve valores)

Tipos de datos básicos en C++

Tipo int

Números enteros

Tamaño en bytes: 2 bytes (16 bits)

Dominio: son todos los números enteros entre los valores
-32.768 y 32.767

Operaciones:

+	Suma	int × int → int
-	Resta	
*	Producto	
/	División entera	
%	Resto de la división entera (módulo)	
- , +	Signo negativo, positivo	
++	Incrementación	
--	Decrementación	int → int

Prioridad de los operadores:

++, --	10*5++
- , + (unario)	-3
* , / , %	3*5
+ , -	6+7

10×6

Tipos de datos básicos en C++

Operadores de incrementación y decrementación

Se trata de los operadores:

++

--

Son equivalentes

```
m = m + 1;  
m++;  
++m;
```

Suma una unidad a su argumento

Resta una unidad a su argumento

Si precede al operando, se realiza la operación ++ o -- y luego se realiza la asignación.

```
x = 10;  
y = ++x; // y vale 11
```

```
x = 10;  
y = x++; // y vale 10
```

Si sigue al operando, se realiza la asignación y posteriormente se realiza la operación ++ o --

Tipos de datos básicos en C++

Tipo float

Números reales

Tamaño en bytes: 4 bytes

Dominio: son todos los números reales que contienen una coma decimal comprendidos entre los valores:

$$3,4 \times 10^{-38} \quad \text{y} \quad 3,4 \times 10^{38}$$

Operaciones:

		float × float → float
+	Suma	
-	Resta	
*	Producto	
/	División en coma flotante	
		float → float
- , +	Signo negativo, positivo	
++	Incrementación	
--	Decrementación	

La prioridad de los operadores es la misma que para el tipo int

Tipos de datos básicos en C++

Tipo double

Igual que float pero más grandes

Números reales

Tamaño en bytes: 8 bytes

Dominio: son todos los números reales que contienen una coma decimal comprendidos entre los valores:

$$1,7 \times 10^{-308} \quad \text{y} \quad 1,7 \times 10^{308}$$

Operaciones:

		double × double → double
+	Suma	
-	Resta	
*	Producto	
/	División en coma flotante	
		double → double
- , +	Signo negativo, positivo	
++	Incrementación	
--	Decrementación	

La prioridad de los operadores es la misma que para el tipo int

Tipos de datos básicos en C++

Además de éstas operaciones, C++ dispone de un gran conjunto de **funciones matemáticas**.

Funciones:

abs: int → int	Calcula el valor absoluto de un número
ceil: double → double	Calcula el número entero mayor o igual que el dado
floor: double → double	Redondea por defecto el valor de un número
fmod: double × double → double	Calcula el resto de la división real de dos números
sqrt: double → double	Calcula la raíz cuadrada de un número

```
#include <math.h>
{
x = abs(-7)           // x vale 7
y = ceil(5.2)        // y vale 6
z = floor(5.2)       // z vale 5
resto = fmod(5.0, 2.0) // resto vale 1
}
```

Los archivos de cabecera que contienen éstas funciones son entre otras:

math.h
float.h
complex.h

Tipos de datos básicos en C++

Tipo **bool**

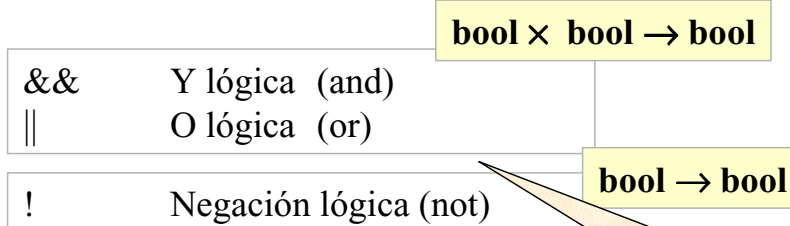
Tamaño en bytes: 1 byte

Dominio: dos únicos valores: { true, false }

No todos los compiladores de C++ tienen éste tipo de dato. En su lugar se utiliza el tipo **int** para representar el tipo de datos **bool**, de forma que el valor entero 0 representa false y cualquier otro valor representa true.

Falso	→	Cero
Verdadero	→	Distinto de cero

Operaciones:



Prioridad de los operadores

!
&&, ||

Operadores lógicos

Tipos de datos básicos en C++

Tabla de verdad:

A	B	! A	A && B	A B
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

Operadores relacionales:

==	Igual a
!=	Distinto
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Operadores relacionales

Los operadores relacionales se utilizan para comprobar una condición

Tipos de datos básicos en C++

Escritura de valores bool

Raramente se tiene la necesidad de escribir o leer valores de tipo bool ya que éste tipo de datos se utiliza sobre todo para evaluar expresiones lógicas.

En caso necesario, si escribimos un dato de tipo bool cuyo valor es true, en consola se visualiza el valor 1.

La lectura es análogo.

Tipo char

Tamaño en bytes: 1 byte

Dominio: dígitos, letras mayúsculas, letras minúsculas y signos de puntuación.

Internamente, los caracteres se almacenan como números. El tipo **char** representa valores en el rango -128 y 127 y se asocian con el código ASCII. Así, el carácter 'A' se almacena como el número 65, etc ...

Símbolo	Código ASCII		Símbolo	Código ASCII		Símbolo	Código ASCII	
0	0110000	48	A	1000001	65	a	1100001	97
1	0110001	49	B	1000010	66	b	1100010	98
2	0110010	50	C	1000011	67	c	1100011	99
3	0110011	51	D	1000100	68	d	1100100	100
4		52	F	1000101	69			
5	53						

$0 < 1 < 2 \dots < 9 < A < B < \dots < Z < a < b < \dots < z$

Tipos de datos básicos en C++

Operaciones:

Dado que los caracteres se almacenan internamente como números enteros, se pueden realizar operaciones aritméticas con los datos de tipo char. Se puede sumar un entero a un carácter para obtener otro código ASCII diferente.

Ejemplos:

- Para convertir una letra minúscula en mayúscula basta con restar 32.
 $'a' - 32 = 'A'$
- Para convertir una letra mayúscula en minúscula basta con sumar 32.
 $'B' + 32 = 'b'$
- Para convertir el carácter '4' en el número 4 basta con restar 48.
 $'4' - 48 = 4$

Más adelante veremos que, en unión con la estructura *array*, se pueden almacenar *cadena de caracteres*.

Tipos de datos básicos en C++

Funciones:

isdigit: char → int	Devuelve TRUE si el carácter es: '0', ... , '9'
isalpha: char → int	Devuelve TRUE si el carácter es: 'A', ... , 'Z', 'a', ... , 'z'.
islower: char → int	Devuelve TRUE si el carácter es una letra minúscula: 'a', ... , 'z'.
isupper: char → int	Devuelve TRUE si el carácter es una letra mayúscula: 'A', ... , 'Z'.
tolower: char → char	Convierte un carácter mayúscula en minúscula.
toupper: char → char	Convierte un carácter minúscula en mayúscula.

```
#include <ctype>
{
...
char c = 'A';
c = tolower (c);    // c vale 'a'
t = isdigit(c);    // t vale 0 (FALSE)
...
}
```

char c = 65;

El archivo de cabecera que contiene éstas funciones es:

ctype.h

Tipos de datos básicos en C++

Modificadores de tipos de datos

Los tipos de datos **int**, **double** y **char** tienen variaciones o *modificadores de tipos de datos*.

Son modificadores los siguientes:

signed short **unsigned long**

Permiten un uso más eficiente de los tipos de datos

Rango de valores

unsigned int	0 ... 65535
long double	$-3,37 \times 10^{-4932}$... $3,37 \times 10^{4932}$
long int	-2147483648 ... 2147483647

Tipos de datos básicos en C++

Las computadoras realizan numerosas operaciones para la resolución de problemas,

- Operaciones aritméticas y lógicas.
- Operaciones condicionales.

...

Además, puede ocurrir que en una misma expresión concurren varios tipos de datos. Ante ésta situación, debemos saber cómo se comporta el compilador.

Cuando los dos operandos son de tipos distintos, el de tipo *menor* se promociona al de tipo *mayor*.

+

long double
double
float
int
short int
char

Tipos de mayor a menor

-

Tipos de datos básicos en C++

En cuanto a la memoria que ocupa cada tipo:

Esto no siempre es cierto, depende del ordenador y del compilador.

```
#include <stdio.h>
#include <iostream.h>

int main()
{
    int i;
    i = sizeof( int )*8;

    cout <<"Tamaño (en bits) del tipo int = ";
    cout << i;
    return 0;
}
```

Tipo de datos	Datos almacenados	Nº de Bytes
char	caracteres	1
int	enteros	2
float	reales	4
double	reales	8
bool	lógicos	1

Para saber en nuestro caso qué tamaño tienen nuestros tipos de datos debemos hacer lo siguiente

3

Constantes y variables C++

Constantes y Variables

- Son *porciones de memoria que almacenan un valor*.
- Las **variables** son palabras que manipulan datos. Dicho valor puede ser modificado en cualquier momento durante la ejecución del programa.
- Una **constante** es una variable cuyo valor no puede ser modificado.
- Las variables pueden almacenar todo tipo de datos: caracteres, números, estructuras, etc ... Dependiendo del valor de la variable, decimos que dicha variable es de un tipo de dato.
- Tanto las variables como las constantes están constituidas por un **nombre** y un **valor**. El nombre lo llamaremos **identificador**.

Toda variable utilizada en un programa debe ser declarada previamente. En C++, ésta declaración puede situarse en cualquier parte del programa.

Dependiendo de dónde se definan,
tenemos varios tipos:

Variables globales
Variables locales
Parámetros

Constantes y Variables

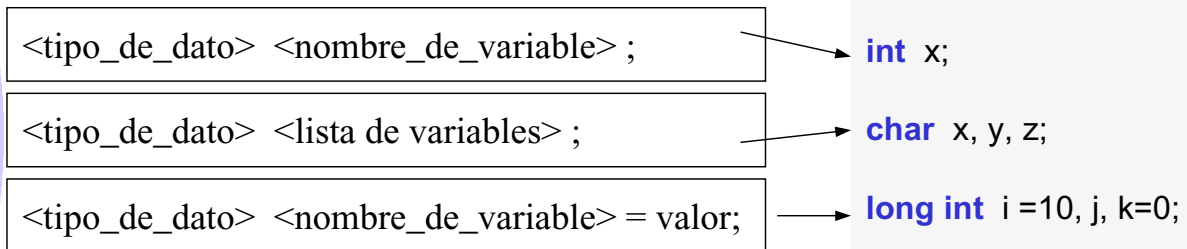
Declaración de variables

La declaración de una variable consiste en escribir un sentencia que proporciona información al compilador de C++.

- El compilador reserva un espacio de almacenamiento en memoria.
- Los nombres de las variables se suelen escribir en minúsculas.

En C++ las variables no se actualizan automáticamente

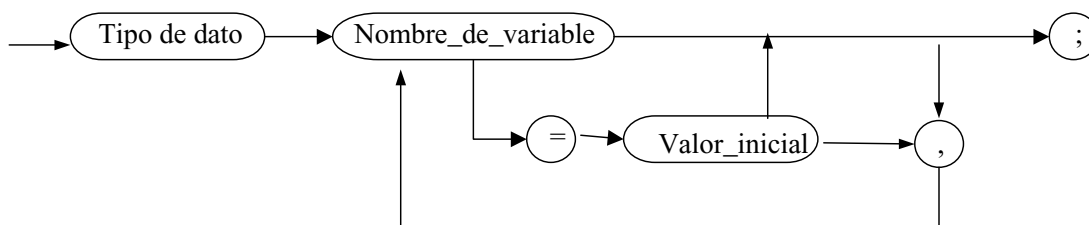
El procedimiento para declarar una variable:



Constantes y Variables

Una forma de expresar el procedimiento para declarar una variable es mediante los *diagramas sintácticos*:

Declaración de variables



Declaraciones locales

Son *variables locales* aquellas que están declaradas dentro de las funciones o de los bloques.

Constantes y Variables

➔ **Declaraciones globales** (*variables globales, funciones, ...*)

La zona de declaraciones globales de un programa puede incluir declaraciones de variables y declaraciones de funciones (*prototipos*).

```
Directivas de preprocesador  
Declaraciones globales ( variables globales, funciones, ...)
```

```
función main()  
{  
    secuencia de declaraciones e instrucciones  
}
```

```
función1()  
{  
    secuencia de de  
}
```

Las funciones y variables aquí declaradas, se pueden utilizar en cualquier punto del programa.

➔ **Parámetros**

Definidos en la lista de parámetros formales de las funciones.

Constantes y Variables

Ejemplos:

VARIABLES LOCALES

```
...  
int funcion1()  
{  
    int i;  
    if ( i != 1)  
    {  
        char m='s';  
        ....  
    }  
  
    /* aquí no se conoce a m */  
}  
...
```

← La variable **m** solo existe en éste bloque

PARÁMETROS

```
...  
int calcular(int i, float j)  
{  
    ...  
}  
...
```

Declaración de Constantes

Una **constante** es una variable cuyo valor no puede ser modificado.

Los nombres de las constantes se suelen escribir en mayúsculas.

1. Constantes declaradas **const**

La palabra reservada **const** es un calificador de tipo variable e indica que el valor de variable no se puede modificar.

```
const <tipo_de_dato> <nombre_de_constante> = <valor> ;
```

Ejemplos

```
...  
const int DIAS = 7;  
const char VACIO = ' ';  
const char PORCENTAJE = '% ' ;  
...
```

Si se intenta modificar una variable definida con **const**, se produce error.

2. Constantes definidas

Se declaran mediante la directiva **#define**

```
#define <nombre_de_constante> <valor>
```

Ejemplos

```
...  
#define pi 3.14  
#define fin 'F'  
...
```

No se especifica el tipo de dato

No aparece ; al final de la sentencia

No aparece el símbolo =

Es más recomendable utilizar **const** en lugar de **#define** ya que el compilador genera código más eficiente.

3. Constantes enumeradas

Las constantes enumeradas permiten crear listas de elementos afines.

Ejemplo de constante enumerada de una lista de colores

```
...  
enum Colores {Rojo, Verde, Azul, Amarillo};  
enum Botones {Salir, Jugar};  
...  
Colores favorito = Rojo;  
...
```

Se comporta como cualquier otro tipo de datos.

Se pueden declarar variables de tipo enumerado.

```
enum <nombre_de_constante> { <lista_de_valores> };
```

El compilador asigna un número a cada elemento del conjunto (comenzando con 0).

Ejemplo del funcionamiento de enumeraciones

```
#include <iostream.h>  
  
int main()  
{  
    enum Dias { Lunes, Martes, Miercoles, Jueves, Viernes };  
    Dias libre = Viernes;    // Dias libre = 4;  
    cout << libre;          // se visualiza por pantalla el número 4  
    return 0;  
}
```

4

Operadores y expresiones

Operadores y Expresiones

1. Instrucciones de asignación.

`<nombre_de_variable> = <expresión> ;`

Puede ser otra variable, una constante o una operación entre variables y constantes.

El operador asignación (=) asigna el valor de la expresión derecha a la variable situada en la izquierda de la instrucción.

Podemos tener en C++ varios operadores de asignación:

`= += -= *= /= %=`

Operadores y Expresiones

Ejemplos:

<code>m = n;</code>	<code>// asigna el valor de n a m</code>
<code>m += n;</code>	<code>m = m + n; // suma m y n y lo asigna a la variable m</code>
<code>m -= n;</code>	<code>m = m - n; // resta m menos n y lo asigna a la variable m</code>
<code>m *= n;</code>	<code>m = m * n; // multiplica m por n y lo asigna a la variable m</code>
<code>m /= n;</code>	<code>m = m / n; // divide m entre n y lo asigna a la variable m</code>
<code>m %= n;</code>	<code>m = m % n; // calcula el resto de la div. entera y lo asigna a la variable m</code>

Instrucción abreviada.

Operadores y Expresiones

Más ejemplos:

Podemos dar valores a varias variables a la vez

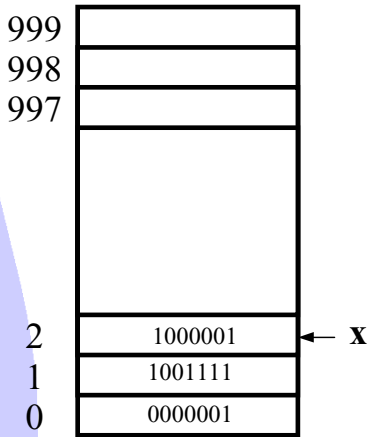
<code>m = n = t = 10;</code>	<code>// Damos a las variables m, n y t el valor 10</code>
<code>m = n = t = a;</code>	<code>// las variables m, n y t toman el valor de la variable a</code>

También podemos asignar a varias variables el valor de otra de un sólo golpe

2. Operador de dirección: &

Cuando se declara una variable, se le asocian tres características:

- nombre de la variable
- tipo de la variable
- dirección de memoria



```
...
char x = 'A';
...
```

El valor de la variable x es 'A'.
La dirección de memoria es 2.

Para conocer la dirección de memoria donde se encuentra almacenada la variable, se puede utilizar el operador &.

```
...
cout << x;
cout << &x;
...
```

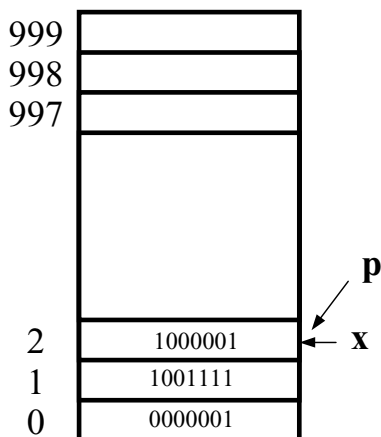
Se visualiza el valor: 'A'

Se visualiza la dirección: 2

Referencias

Puede ser interesante declarar dos variables con la misma dirección de memoria. Para realizar ésta tarea también utilizamos el operador &.

```
<tipo> & <variable> = <variable>;
```



```
#include <iostream.h>
```

```
int main()
{
    char x = 'A';
    char & p = x;
    cout << x << p;
    p = 'b';
    cout << x << p;
    ...
    return 0;
}
```

La dirección de la variable p es la misma que la dirección de la variable x

// x también vale 'b'

5

Instrucciones de Entrada / Salida

Instrucciones de Entrada / Salida

En C++ la entrada y salida se lee y escribe en flujos. Cuando se incluye la biblioteca **iostream.h** en el programa, se definen automáticamente dos flujos:

Flujo **cin** (se utiliza para la entrada de datos)

Flujo **cout** (se utiliza para la salida de datos)

Permiten la comunicación del ordenador con el exterior para tomar datos o devolver resultados

Esta biblioteca también nos proporciona dos operadores, uno de *inserción* (`<<`), que inserta datos en el flujo **cout** y otro operador de *extracción* (`>>`) para extraer valores del flujo **cin** y almacenarlos en variables.

```
...  
cin >> a;  
cin >> a >> b >> c;  
...
```

```
...  
cout << x;  
cout << x << y << z << endl;  
cout << " x vale: " << x;  
cout << "Hola\n";  
...
```

Salto de línea

\n también provoca salto de línea

C++ utiliza **secuencias de escape** para visualizar caracteres que no están representados por los símbolos tradicionales.

Las más utilizadas las mostramos en la siguiente tabla:

<code>\n</code>	Retorno de carro y avance de línea
<code>\t</code>	Tabulación
<code>\a</code>	Alarma
<code>\"</code>	Dobles comillas
<code>\\</code>	Barra inclinada

```
...
cout << "Hola\n";
cout << "Lunes\t Martes\t Miercoles\t ";
cout << "\a" ;
...
```

CARACTERÍSTICAS DEL LENGUAJE C++:

- Se distingue entre mayúsculas y minúsculas.
- Palabras clave: siempre en minúsculas.
- Lenguaje estructurado pero no estrictamente estructurado en bloques (no se pueden definir funciones dentro de otras funciones).
- Todas las sentencias y declaración de variables terminan en punto y coma.
- La ejecución siempre comienza con la función `main()`