



A C++ Intermediate Quick Review

Antonio José Alanís Bernal
Estudiante de Ingeniería de Software



<https://github.com/antonioalanxs>

C++

Índice

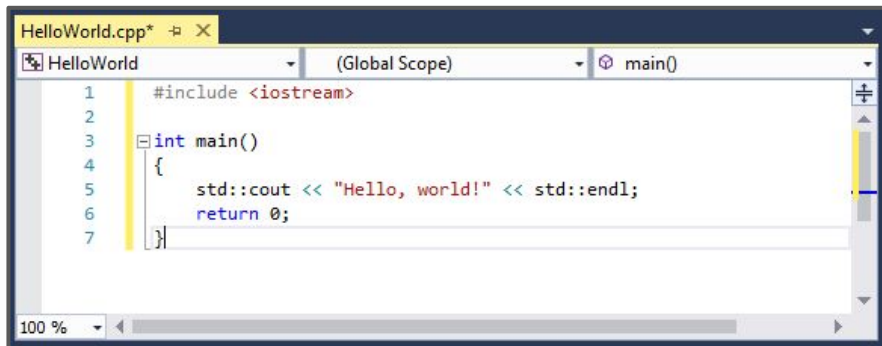
- ❑ Introducción | Estructura de un programa
- ❑ Preprocesador, librerías y espaciado de nombres
- ❑ Entrada y salida de datos (E/S)
- ❑ Programa principal “main()” | Subprogramación
- ❑ Estructuras de control
- ❑ Operadores y caracteres de control
- ❑ Tipos de datos

Introducción


Estructura de un programa

Aunque **C++** sea un lenguaje de programación **multiparadigma**, este es muy fiel al ideal de "Orientación a objetos", donde se busca la sencillez, organización y seguridad a la hora de estructurar el código mediante la **abstracción**, jerarquía, **encapsulación** y modularización. Por ello, nuestro programa se dividirá en dos partes:

- 1) **Interfaz pública**, donde **declaramos** previamente qué es lo que contendrá nuestro programa, es decir; mostramos al usuario qué es lo que hay.
- 2) **Implementación privada**, en la cual **ocultamos** los detalles de nuestra abstracción al usuario.



```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello, world!" << std::endl;
6     return 0;
7 }
```



```
sukumar@VBOX:~/Desktop/C++$ ls
HelloWorld.cpp
sukumar@VBOX:~/Desktop/C++$ g++ HelloWorld.cpp -o HelloWorld
sukumar@VBOX:~/Desktop/C++$ ls
HelloWorld HelloWorld.cpp
sukumar@VBOX:~/Desktop/C++$ ./HelloWorld
Hello World!
sukumar@VBOX:~/Desktop/C++$
```

"g++ <filename.cpp> -o <someName.exe>" invoca al compilador g++ para que convierta en un .exe al .cpp

Introducción

Estructura de un programa

```
//                                Interfaz pública:

<directivasPreprocesador (p.ej.: #include <iostream>; using namespace std;)>

<declaraciónConstantes>
<declaraciónTiposDeDatos>
<declaraciónVariablesGlobales>

<declaraciónPrototiposSubprogramas (p.ej.:
    .
    .
    .
tipoDeDato funcionEnésima(<tiposDeDatosParámetros>);>

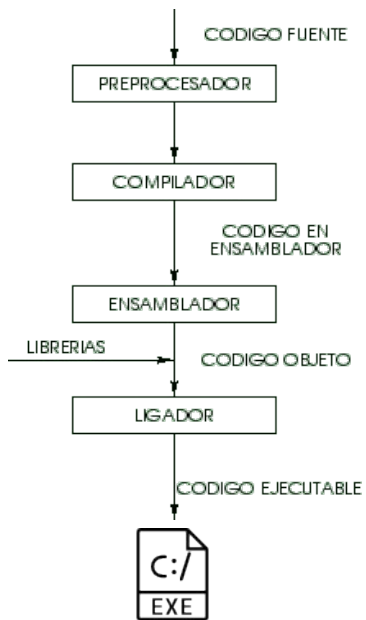
//                                Implementación privada:

int main() {<conjuntoInstrucciones; return 0;>}
    .
    .
    .
tipoDeDato funcionEnésima(<parametros>) {<conjuntoInstrucciones>}

// El programa principal no se define en la interfaz.
```

Preprocesador, librerías y espaciado de nombres

Preprocesador; programa separado que es invocado por el compilador antes de que comience la traducción real, y se encargará de tareas como **eliminar comentarios, incluir otros archivos, ejecutar sustituciones de macros, definir constantes**, etc.



```
// Comment to end of line
/* Multi-line comment */

#include <libraryHeaderFile> // Insert standard header file
#include "myfile.h"          // Insert file in current directory
#define X some text          // Replace X with some text (constants)
#define F(a,b) a+b           // Create F(a,b) defined by a+b
#define X \ some text        // Line continuation
#undef X                      // Remove definition
#ifdef X                      // Conditional compilation (#ifdef X)
    // ...
#elif defined(Y)
    // ...
#else
    // ...
#endif                        // Required after #if, #ifdef
```

<https://www.cplusplus.com/doc/tutorial/preprocessor/>

Preprocesador, librerías y espaciado de nombres

Biblioteca, llamada por vicio del lenguaje “**librería**” (del inglés library), es una colección de clases y subprogramas cuyo fin es ser utilizada por otro programa, independiente y de forma simultánea para facilitar su creación.

Biblioteca estándar de C++ (“**Standard Template Library**”) y sus diferentes clases (Tipos Abstractos de Datos).

Espaciado de nombres; contenedor abstracto en el que un grupo de uno o más identificadores únicos pueden existir.

→ Su existencia se debe a una mejor comprensión/organización del código y a la corrección de malas prácticas...

https://www.youtube.com/watch?v=etQX4Mme2f4&list=LL&index=2&ab_channel=CodeBeauty

```
// Using "namespace":
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    cout << "Hello world!";
```

```
    return 0;
```

```
}
```

```
// "namespace" isn't used:
```

```
#include <iostream>
```

```
// Alternativa más cómoda:
```

```
// using std::cout;
```

```
int main()
```

```
{
```

```
    std::cout << "Hello world!";
```

```
//    cout << "Hello world!";
```

```
    return 0;
```

```
}
```

Principales **ventajas** que nos ofrece la **segunda opción**:

- 1) **ahorrar memoria** (no cargamos **TODOS** los identificadores),
- 2) poder **definir subprogramas con identificadores ya reconocidos** en la biblioteca estándar del lenguaje.

Entrada y salida de datos (E/S)

"cin", "cout", "scanf" y "printf"

C Style (Formatting)

En la librería estándar de C++ encontramos “**stdio**” (**standard input/output stream**), perteneciente a la librería estándar de C; presentando por ello subprogramas primitivos con sus **limitaciones**.

- **printfQ**: print formatted, imprime por pantalla utilizando una “**cadena de formato**” (para la conversión al mismo tipo).
- **scanfQ**: Lee por teclado. Similar a la [clase Scanner de Java](#).

%d, %i	integer
%f	double (float)
%c	char
%s	char string
%p	pointer
%%	%

C++ Style

En la librería estándar de C++ encontramos a “**iostream**”, que nos ofrece una versión más actualizada de estos subprogramas, es decir; tendrán menos limitaciones.

- **cout**: console out, imprime por pantalla.
- **cin**: console in, lee por teclado **hasta el carácter de control “\n”**.
 - `cin.getline(<string>)`
 - `cin.ignore(<expression>)`

prompts	<code>int a, b;</code>
<code><< (console)</code>	<code>cout << ¿a? ¿b? << “\n”;</code>
<code>>> (user)</code>	<code>cin >> a >> b;</code>

Programa principal “main()”

Subprogramación

```
// Opción clásica:
#include <iostream>
using std::cout;
using std::cin;

int main()
{
    cout << "Hello world!";
    return 0; // convención
}
```

/*Realmente, todo programa retorna un valor de referencia para indicar que no ha habido errores. Por ello, para un mayor control de errores, elegimos esta.*/

```
// Opción alternativa:
// Alternativa más cómoda:
#include <iostream>
using std::cout;
using std::cin;
```

```
void main()
// main()
{
    cout << "Hello world!";
}
```

/*"void" ≈ "vacío", "nada". El programa principal no retorna nada, sino que sirve como flujo de todo el programa (inicializar variables, índice/organizar y resumir del código fuente).*/

La subprogramación permite que los algoritmos sean **simples** y **reutilizables**, además de que favorece una mayor organización y estructuración de los mismos (**modularización**).

// Paso de parámetros:

```
// Referencia:
void procedure(<TipoDeDato&>)
{
    <...>
    /* "&" indica la memory address para que no se copie el valor, actuando así C++ con punteros... */
}
```

```
/* Valor: igual, pero sin indicar la memory address ("&"). */
```

En C++ tratamos a todo bloque de código como una función, por lo que "main" se ha tomado como convención para distinguir al programa principal.

Estructuras de control

```
if (<condición>)  
{  
    <...>  
}  
else if (<condición>)  
{  
    <...>  
}  
else  
{  
    <...>  
}
```

```
switch (<expression>)  
{  
    case <constant>: <...> break;  
    // OR:  
    case <constant>:  
    case <constant>:  
    case <constant>: <...> break;  
    // defined range:  
    case <constant>...<constant>: <...> break;  
    // else:  
    default: <...> break;  
}
```

```
while (<condición>)  
{  
    <...>  
}
```

```
do  
{  
    <...>  
} while (<condición>)
```

```
for (i = <constant>; i ><(=) <count>; i++/--)  
{  
    <...>  
}
```

Operadores y caracteres de control

1	<ul style="list-style-type: none">. structure member access& memory address reference* access to value in &-> pointer reference:: scope~ complement to one <hr/>	= assignment	
2	<ul style="list-style-type: none">() parenthesis[] brackets(type) type castingsizeof(<var/type>) memory size?/: conditional <hr/>	5	<ul style="list-style-type: none">&& and ornot/!/^ not== equal to!= not equal to>/<(=) over/less (or equal) than <hr/>
3	<ul style="list-style-type: none">* multiply/ divide% modulus+/- addition/subtraction <hr/>	6	<ul style="list-style-type: none">\b backspace\n newlineendl \n and empty <u>output buffer</u>\t tab\f form feed\r return\" quote\' apostrophe\nnn octal character\NN hexadecimal character
4	<ul style="list-style-type: none"><u>++</u>/<u>--</u> increment/decrement(3)= assignment and (3)		

Type	Description	Size	Domain
char	Signed character/byte. Characters are enclosed in single quotes.	1	-128..127
double	Double precision number	8	ca. $10^{-308}..10^{308}$
int	Signed integer	4	$-2^{31}..2^{31} - 1$
float	Floating point number	4	ca. $10^{-38}..10^{48}$
long (int)	Signed long integer	4	$-2^{31}..2^{31} - 1$
long long (int)	Signed very long integer	8	$-2^{63}..2^{63} - 1$
short (int)	Short integer	2	$-2^{15}..2^{15} - 1$
unsigned char	Unsigned character/byte	1	0..255
unsigned (int)	Unsigned integer	4	$0..2^{32} - 1$
unsigned long (int)	Unsigned long integer	4	$0..2^{32} - 1$
unsigned long long (int)	Unsigned very long integer	8	$0..2^{64} - 1$
unsigned short (int)	Unsigned short integer	2	$0..2^{16} - 1$

Podemos encontrar funciones matemáticas en la librería [cmath](#).

Tipos de datos

Predefinidos

Abstractos

(parten de ellos)

Simples

Compuestos

Colección de valores con sus correspondientes operaciones definidos mediante una especificación que es independiente de cualquier representación.

Punteros

Enumerados

Subrangos

(no existen en C++)

Registros

Strings
(cadenas de caracteres)

Tipos de datos

/ Definir de tipos de datos
por el programador: */*

typedef <tipoDeDato> <id>;

/ Asignar un nombre alternativo
a un tipo ya existente: */*

// Ejemplo:
typedef unsigned int *tNaturales*;

// Registro o estructura:

typedef struct {
 <tipoDeDato> <campo1>
 <tipoDeDato> <campo2>
 .
 .
 .
 <tipoDeDato> <campoEnésimo>
} <id>; *// tEstructura*

// Enumerado:

typedef enum
{
 <Identificador1>
 .
 .
 .
 <IdentificadorEnésimo>
} <id>; *// tEnumerado*