



Declaración de variables y tipos de datos fundamentales en C++

Autor: Juan Camilo Correa Chica
Profesor Facultad de Ingeniería
Universidad de Antioquia

C++ es un lenguaje multiparadigma, compilado y de tipado estático, esto último quiere decir que el tipo de dato o valor que almacenará una variable debe ser definido explícitamente al momento de la declaración de dicha variable y es verificado en tiempo de compilación, antes de que pueda ejecutarse la aplicación [1].

Este fenómeno ocurre en C++ porque a pesar de ser un lenguaje de alto nivel tiene capacidades, que se verán más adelante, que le permiten trabajar a un bajo nivel. Esto afecta particularmente el almacenamiento de datos, ya que para C++ todos los datos son homogéneos en cuanto a formato, es decir, todos tienen el mismo, todos los datos son números que se almacenan en memoria. Es necesario dar dicho formato de algún modo, lo que se logra en el momento de la declaración de las variables al indicar el tipo de dato que se va a almacenar.

El tipo de dato no solo indica su formato (si es un número entero, decimal, o si es un carácter, etc.), sino también cuánto espacio en memoria debe ocupar [2]. Otra característica importante de los tipos de datos es que permiten que C++ sepa o infiera si cierta operación (matemática, lógica, etc.) está definida para los operandos que intervienen en ella. Por ejemplo: una operación lógica de negación tiene sentido y está definida para operandos de tipo numérico enteros, pero no está definida para tipos numéricos decimales.

Tipos de datos fundamentales de C++

1

Booleanos (bool)

Un dato tipo bool en C++ puede tener uno de dos valores: 0 o 1, o bien, falso o verdadero. Son utilizados para representar el resultado de una operación lógica. El tamaño en memoria de un tipo de dato bool es de un byte (8 bits).



2

Caracteres (char)

El tipo de dato **char** se utiliza en C++ para almacenar caracteres alfanuméricos o, en general, cualquier carácter del código estándar americano para intercambio de información, conocido más comúnmente como ASCII [3]. El tamaño en memoria de un tipo de dato char es de un byte (8 bits), por lo tanto, puede almacenar un valor numérico sin signo en el rango 0 a 255 (cada valor numérico corresponde a un carácter ASCII).

3

Números enteros (int)

Un valor numérico entero puede ser almacenado usando el tipo de dato **int** y sus diferentes variantes de acuerdo con la magnitud y el signo del número que se pretende almacenar. Al igual que con el tipo char, con el tipo int se puede especificar si la variable puede almacenar números negativos o exclusivamente números positivos. O bien, especificar la capacidad del dato que se espera almacenar en la variable. El tipo de dato int tiene una longitud de 4 bytes (32 bits), la variante **short** corresponde a un dato entero de 2 bytes, mientras que la variante **long** corresponde a un entero de 8 bytes.

4

Números decimales (float)

Los números reales (o de punto flotante) son representados utilizando el formato del estándar de aritmética binaria de punto flotante IEEE754 [5] y vienen en 3 longitudes: **float** de 4 bytes para decimales de precisión sencilla, **double** de 8 bytes para decimales de doble precisión y **long double** de 16 bytes para decimales de precisión extendida.

La **tabla 1** resume los tipos de datos fundamentales de C++, su longitud y el rango de valores que se puede almacenar en variables de cada tipo, respectivamente.

Tabla 1. Longitud y rango de operación de los tipos de datos fundamentales de C++

Tipo de dato	Longitud en bytes	Rango de operación
<i>bool</i>	1	0 o 1
<i>char</i>	1	Con signo: -128 a 127; sin signo: 0 a 255
<i>int</i>	4	Con signo: -2147483648 a 2147483647; sin signo: 0 a 4,294,967,295
<i>float</i>	4	+/- 3.4 e +/- 38 (aproximadamente 7 cifras decimales)
<i>double</i>	8	+/- 1.7 e +/- 308 (aproximadamente 15 cifras decimales)



Declaración de variables en C++

Toda variable en C++ debe cumplir, por lo menos, con 3 de 4 características indispensables:

1 **Tipo de dato:**

indica al compilador la naturaleza o formato del valor que almacena la variable, la longitud en memoria necesaria para almacenar el dato y las operaciones que están definidas para valores de ese tipo.

2 **Nombre:**

es el identificador que permite diferenciar a una variable de otras. Debe ser, en lo posible, un nombre nemotécnico que refleje el propósito o la naturaleza del valor que almacena la variable. Este nombre solo puede componerse de caracteres alfanuméricos incluyendo el guion (-) y la raya baja (_) y no puede empezar por un número. C++ es sensible a minúsculas y mayúsculas, por ejemplo, Rosa, rosa y ROSA, son nombres distintos para C++.

3 **Ubicación en memoria:**

es el lugar, a lo largo de la memoria y dependiendo de la naturaleza de la variable, donde se va a almacenar el valor asignado a ella. Por lo general, se representa con una dirección de memoria en formato hexadecimal.

4 **Valor de inicialización:**

es el valor inicial que se almacena en la variable y que se le asigna en el momento de su declaración. Para tipos *bool* solo puede ser 0 o 1. Para tipo *char* puede ser un carácter ASCII entre comillas simples o el valor numérico en el rango *char* correspondiente al carácter en alguno de los formatos numéricos permitidos por C++: decimal, hexadecimal u octal (los estándares modernos de C++ permiten también formato en binario). Para tipo *int*, el valor numérico en el rango *int* en alguno de los formatos numéricos permitidos por C++, mencionados anteriormente. Para los tipos de punto flotante debe ser un valor numérico, y si es un número racional o irracional debe usarse la notación de punto flotante (no utilizar comas) o la notación científica para números de punto flotante en C++ [6].

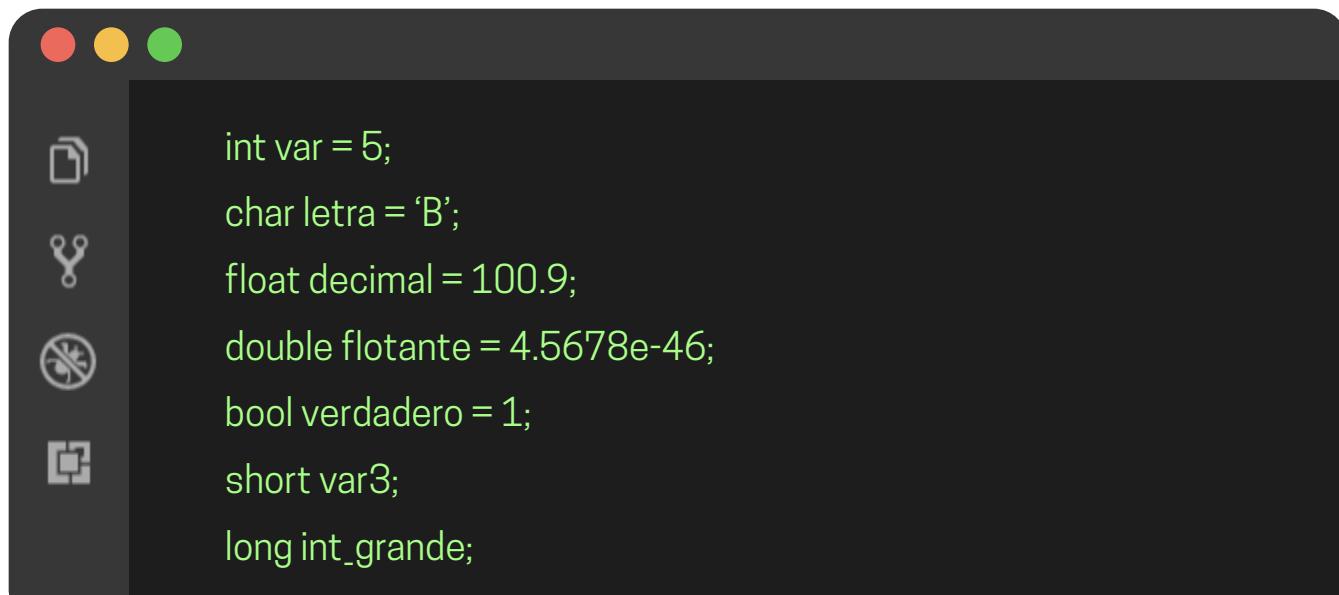


El orden de estas características en la declaración de una variable en C++ es el siguiente:

Tipo de dato + Nombre + Operador de asignación (=) + Valor de inicialización

Observe que la ubicación en memoria no está presente, ya que esta es asignada por el compilador y no por el programador. La parte de inicialización es opcional, se puede dar valor a una variable más adelante en el programa, sin embargo, es recomendable siempre inicializar las variables al momento de declararlas.

Ejemplos:



```
int var = 5;
char letra = 'B';
float decimal = 100.9;
double flotante = 4.5678e-46;
bool verdadero = 1;
short var3;
long int_grande;
```

Modificadores para tipos de datos enteros

Con ayuda de los modificadores de variable se pueden introducir variaciones a los tipos de datos enteros de C++, es decir, *char* e *int* y al tipo de punto flotante *double*. Cabe resaltar que los modificadores que se van a describir a continuación no tienen ningún sentido para el resto de los tipos de datos fundamentales de C++.

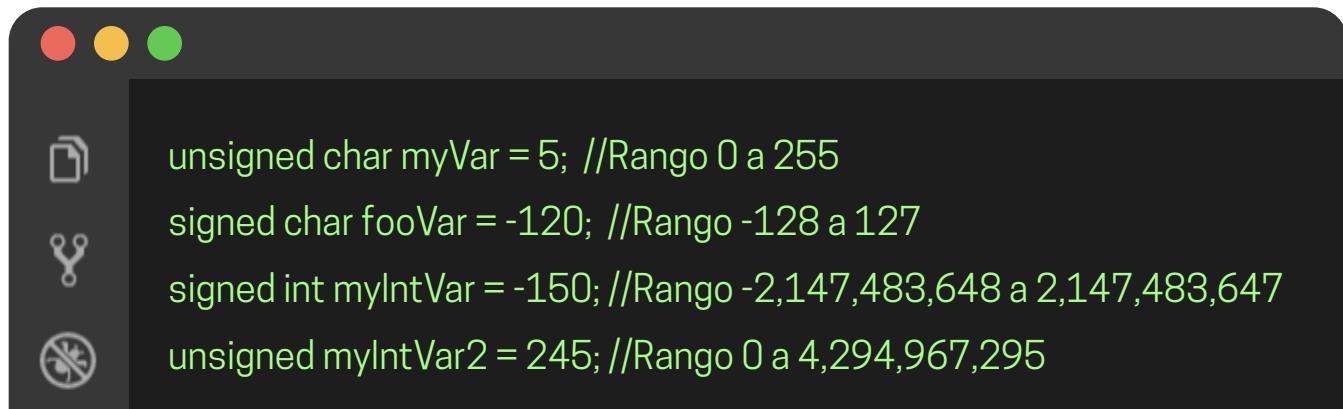
Modificadores para el signo

Los tipos *char* e *int* pueden declararse con signo o sin signo, dependiendo de si el valor que se va a almacenar es un número entero negativo o positivo respectivamente. Esto se logra agregando a la declaración las palabras ***signed*** o ***unsigned***, respectivamente. En C++ el modificador de signo por defecto es *signed*, es decir, cuando no se especifica un



modificador de signo, entonces C++ asume que es *signed*. Como se vio anteriormente, el tema del signo influye directamente en el rango de valores que se puede almacenar en la variable en cuestión.

Ejemplo:

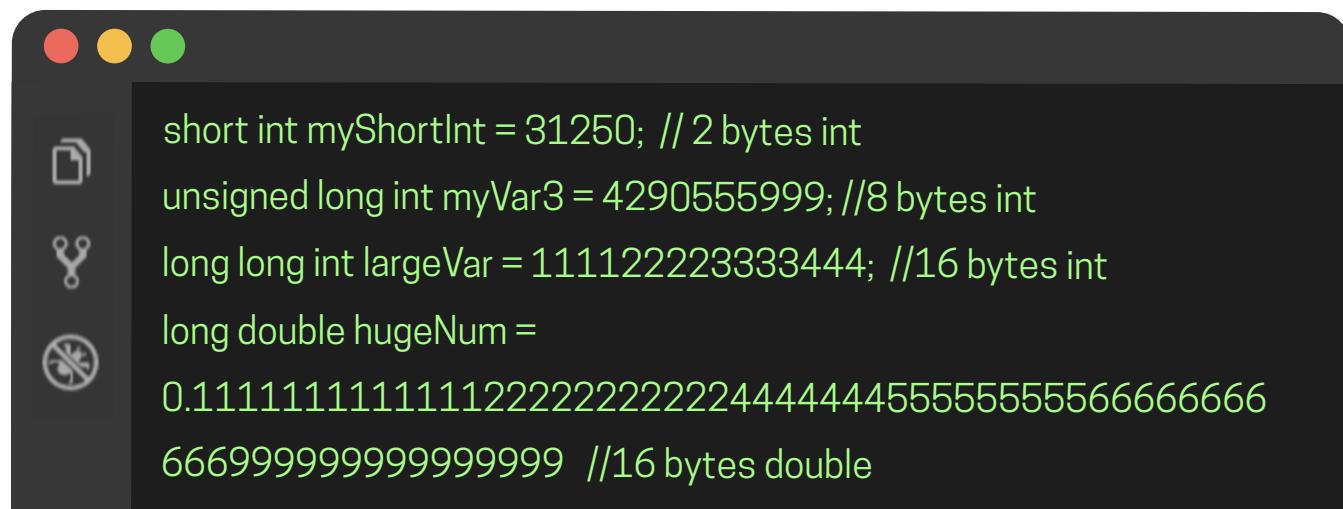


```
unsigned char myVar = 5; //Rango 0 a 255
signed char fooVar = -120; //Rango -128 a 127
signed int myIntVar = -150; //Rango -2,147,483,648 a 2,147,483,647
unsigned myIntVar2 = 245; //Rango 0 a 4,294,967,295
```

Modificadores para el tamaño de dato

Como se mencionó anteriormente el tipo **int** tiene una longitud de 4 bytes, pero pueden tenerse datos enteros de 2 bytes, 8 bytes y 16 bytes, mediante la utilización de los modificadores **short**, **long** y **long long**, respectivamente. El uso de tales modificadores impacta directamente en la cantidad de memoria que se asocia a una variable en el momento de su declaración. Asimismo, se puede tener un tipo de punto flotante de precisión extendida de 16 bytes con el uso del modificador **long** en conjunto con el tipo **double**.

Ejemplos:



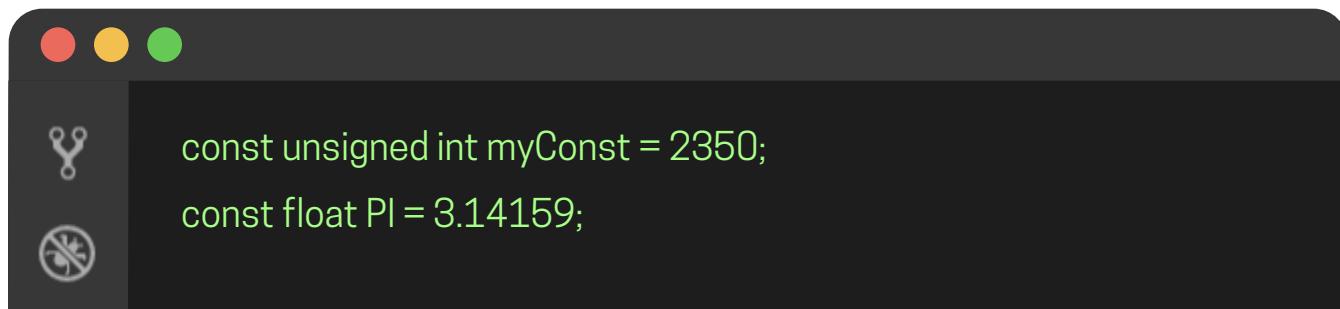
```
short int myShortInt = 31250; // 2 bytes int
unsigned long int myVar3 = 4290555999; //8 bytes int
long long int largeVar = 111122223333444; //16 bytes int
long double hugeNum =
0.11111111111122222222244444445555555666666666
66699999999999999999 //16 bytes double
```



Calificador const

El calificador **const** se utiliza para declarar constantes, que son todo lo contrario a una variable. Es un dato que conservará siempre el mismo valor que se le da en el momento mismo de su declaración y no puede ser cambiado en ningún otro momento en el programa. Puede usarse con todos los tipos de dato fundamentales de C++.

Ejemplos:



```
const unsigned int myConst = 2350;  
const float PI = 3.14159;
```

Arreglos

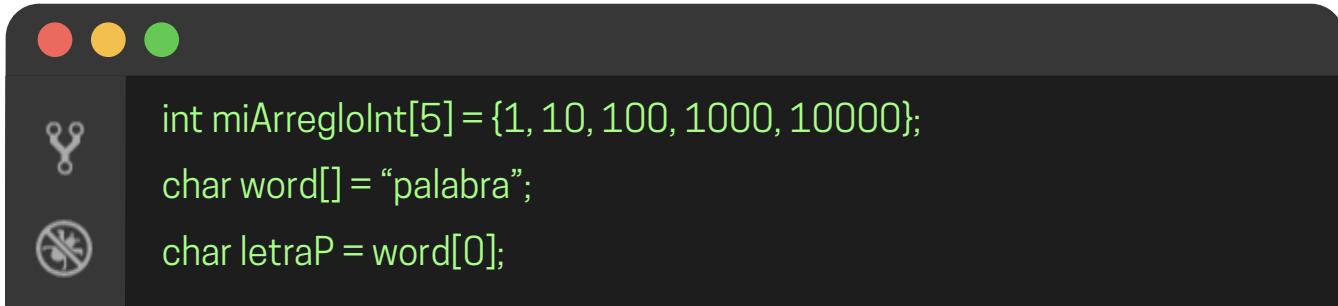
Los arreglos son colecciones de datos de un mismo tipo, se declaran de la misma forma que se declara una variable en C++ con la única diferencia de que se usa el operador [] (corchetes) para indicar que es un arreglo y opcionalmente para indicar el tamaño de la colección. Se pueden declarar arreglos de todos los tipos fundamentales de datos en C++, usando los modificadores y calificadores de variables.

El arreglo se inicializa agregando los elementos de la colección dentro de corchetes (si se especifica el tamaño entonces el número de elementos debe coincidir con dicho tamaño). Si no especifica en los corchetes el tamaño del arreglo entonces la inicialización es obligatoria para que el compilador de C++ pueda inferir el tamaño del arreglo en la compilación. Si especifica el tamaño, puede opcionalmente dejar el arreglo sin inicializar al hacer su declaración.

El operador corchetes también se utiliza para acceder, de manera individual, a los elementos del arreglo indicando su posición en el arreglo. Los elementos de un arreglo en C++ van desde la posición 0 para el primer elemento hasta la posición n-1 para el último elemento, y siendo n el tamaño del arreglo. A diferencia de otros lenguajes de programación, en C++ hay manera de sacar porciones (slices) de un arreglo [8][9].



Ejemplos:

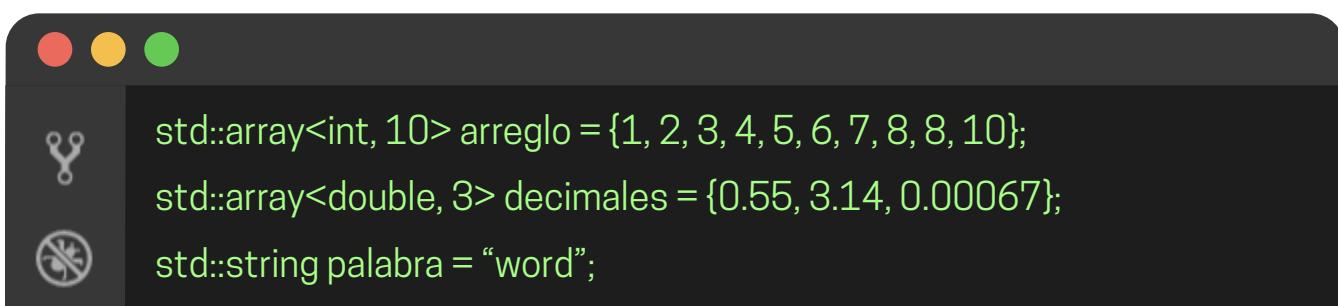


```
int miArregloInt[5] = {1, 10, 100, 1000, 10000};  
char word[] = "palabra";  
char letraP = word[0];
```

std::string y std::array

Las buenas prácticas de programación con C++ sugieren que se utilicen los contenedores (estructuras de datos) **std::array** [10] y **std::string** [11] de la biblioteca de plantillas estándar STL (por sus siglas en inglés Standard Template Library) de C++ para declarar colecciones (arreglos) de datos numéricos y cadenas (arreglos) de caracteres respectivamente. Al igual que en los arreglos y cadenas de caracteres convencionales, en los arreglos y cadenas con **std::array** y **std::string**, se puede utilizar el operador corchetes [] para acceder a los elementos del arreglo o de la cadena. En el caso de **std::array** es necesario especificar el tipo de dato del arreglo y la cantidad de elementos así: `std::array<tipo_dato, num_elementos> nombre_arreglo`

Ejemplo:



```
std::array<int, 10> arreglo = {1, 2, 3, 4, 5, 6, 7, 8, 8, 10};  
std::array<double, 3> decimales = {0.55, 3.14, 0.00067};  
std::string palabra = "word";
```



Referencias bibliográficas

1. Z. Grossbar. (2013, abril 18). An Introduction to Programming Type Systems. [En línea]. Disponible en <https://www.smashingmagazine.com/2013/04/introduction-to-programming-type-systems/>
2. B. Stroustrup. The C++ Programming Language. Addison Wesley, 2013. Disponible en <https://bit.ly/3oGAsrt>
3. Wikipedia. (2020). ASCII. [En línea]. Disponible en <https://es.wikipedia.org/wiki/ASCII>
4. Wikipedia. (2020). Unicode. [En línea]. Disponible en <https://es.wikipedia.org/wiki/Unicode>
5. Wikipedia. (2020). IEEE 754. [En línea]. Disponible en https://es.wikipedia.org/wiki/IEEE_754
6. Nascardriver. (2020). Introduction to scientific notation. [En línea]. Disponible en <https://www.learncpp.com/cpp-tutorial/introduction-to-scientific-notation/>
7. OpenGenus Foundation. (2020). Fundamental Data Types in C++. [En línea]. Disponible en <https://iq.opengenus.org/fundamental-data-types-in-cpp/>
8. Goalkicker. (2020). Programming Notes for Professionals books. [En línea]. Disponible en [C++ Notes for Professionals – GoalKicker.com](https://www.goalkicker.com/cpp-notes-for-professionals.html)
9. P. Deitel y H. Deitel. Como programar en C++. Pearson, 2014.
10. C Plus Plus. (2020). Array class. [En línea]. Disponible en [http://www.cplusplus.com/reference/array/array/](https://www.cplusplus.com/reference/array/array/)
11. C Plus Plus. (2020). Strings. [En línea]. Disponible en [http://www.cplusplus.com/reference/string/](https://www.cplusplus.com/reference/string/)