**Capitulos 20. Templantes**

En esta nota estudiaremos una característica genial de C++, se trata de las **plantillas**o***templates****.*Una plantilla es una manera especial de escribir funciones y clases para que estas puedan ser usadas con cualquier tipo de dato, similar a la sobrecarga, en el caso de las funciones, pero evitando el trabajo de escribir cada versión de la función. Las ventajas son mayores en el caso de las clases, ya que no se permite hacer sobrecarga de ellas y tendríamos que decidirnos por una sola o hacer especializaciones usando la herencia.

## ¿Cómo funcionan?

La magia de las plantillas está en no definir un tipo de dato desde el principio, sino dejar esto como algo pendiente y usar algo que permita manejar varias opciones, de hecho se usa una variable para este propósito. Veamos la sintaxis para el caso de las funciones:

1. // Para una función, ambas opciones son equivalentes
2. **template** <**class** identificador> definición\_de\_función;
3. **template** <**typename** identificador> definición\_de\_función;

El **identificador** es  el símbolo que guarda el tipo de dato que se ha de usar una vez elegido, por lo que en la definición de la función deberá utilizarse en lugar de los nombres de los tipos de datos, de esta manera queda hecha una función genérica a la cual podemos llamar **función-plantilla**.

en una función que nos retorne el mayor de dos datos que le demos como argumentos y que sirva con cualquier tipo:

1. **template** <**class** tipo>
2. tipo mayor(tipo dato1, tipo dato2){
3. **return** (dato1 > dato2 ? dato1 : dato2);
4. }

El identificador que usamos es “tipo”, por eso el tipo de dato de retorno y el de los parámetros debe ser ese identificador. Durante la compilación del programa, en la invocación a la función, se resuelve el tipo que se usará y **el compilador escribirá, con base en la plantilla, una función que sirva con el tipo de dato resuelto** y será la que realmente se utilice para dicha invocación.

Probemos nuestra función-plantilla y veamos lo que pasa:

1. **int** main(){
2. **int** a = 1, b = 2, n;
3. **float** c = 1.0, d = 0.5, m;
5. n = mayor <**int**> (a,b); // Usando enteros
6. m = mayor(c,d); // Usando reales
8. cout << "Entero mayor: " << n << endl;
9. cout << "Real mayor: " << m << endl;
11. **return** 0;
12. }

Se puede usar una especificación explícita para el tipo de dato que se usará en la función escribiendo  <tipo\_dato> o dejar que el compilador resuelva el tipo con los argumentos. Cabe destacar que las plantillas funcionan no solo con tipos de datos primitivos sino también con los estructurados como las clases, aunque para nuestro ejemplo debemos tener [sobrecargado](http://codingornot.com/cc-sobrecarga-de-operadores-en-c) el operador de comparaciónusado en la definición de la función **“>”**.

La función no sirve cuando los tipos de los datos son diferentes, entonces no podríamos usarla con un entero y un real, porque solamente se ha definido un **identificador de tipo.**Para usar más tipos basta con definir más identificadores, de esta manera podríamos escribir la función de la siguiente forma:

1. **template** <**class** tipo1, **class** tipo2>
2. tipo1 mayor(tipo1 dato1, tipo2 dato2){
3. **return** (dato1 > dato2 ? dato1 : dato2);
4. }

Con esta función es posible una llamada así:

1. **int** a = 10;
2. **float** b = 11.0;
3. **float** c = mayor(b,a); // como el primer argumento es un float, el resultado también lo es