Funciones genéricas

Las funciones genéricas son un mecanismo de C++ que permite crear una función en la cual es posible definir uno o varios parámetros que especifican el tipo de dato con el que se está trabajando.
Las funciones así generadas se denominan instancias o especializaciones de la plantilla. También versiones implícitas, para distinguirlas de las versiones codificadas manualmente (versiones explícitas).

Suponemos que se desea construir una función max(a, b) que calcule el mayor de dos valores, suponiendo que estos sean de cualquier tipo capaz de ser ordenado, es decir, cualquier tipo en el que se pueda establecer un criterio de ordenación (establecemos a > b. si a está después que b en el orden).

El problema que presenta C++ para esta propuesta es que, al ser un lenguaje fuertemente tipado la declaración max(a, b) requiere especificar el tipo de datos tanto para los argumentos como para el valor devuelto:

```
tipoT max(tipoT a, tipoT b);
```

La sintaxis del lenguaje no permite que tipoT sea una variable.
Una posible solución es sobrecargar la función max(), definiendo tantas versiones como tipos distintos debamos utilizar:

```
int max(int a, int b);
float max(float a, float b);
char max(char a, char b);
```

La solución más eficiente para este problema es utilizar una función genérica (plantilla). La sintaxis de su definición es la siguiente:

```
template <class T> T max(T a, T b)
 {
  if (a > b) return a;
    return b;
}
```

- template es un especificador de tipo, e indica que se trata de una plantilla (es una palabra clave C++)
- <class T> es la lista de parámetros; representa el/los parámetros de la plantilla (el tipo de dato).

Es importante considerar que utilizamos dos conceptos distintos, aunque relacionados: los parámetros de la plantilla (contenidos en la lista template <....>) y los parámetros de la función (argumentos con que se invoca la función en cada caso concreto). El compilador deduce los tipos concretos de los parámetros de la plantilla según el tipo de datos de los parámetros utilizados en la invocación de la función. Por ejemplo, la plantilla anterior puede ser utilizada mediante las siguientes sentencias:

```
int i, j;
UnaClase a, b;
...
int k = max(i, j);
UnaClase c = max(a, b);
```

En un caso los argumentos de la función son dos parámetros de tipo int; mientras en el otro son dos instancias (objetos) de la clase UnaClase. Es importante tener en cuenta que en el ejemplo anterior, plantilla max(a, b), el código generado utiliza el operador de comparación > para determinar cuál de los elementos es el de mayor valor. Esto significa que dicho operador debe estar definido para los tipos que se utilicen en la llamada a la función, en caso de usarse objetos de una clase, en esa clase el operador debe haber sido sobrecargado.

Clases genéricas

Las clases plantilla o clases genéricas son un artificio C++ que permite definir una clase en la cual el tipo de dato de uno o más de sus atributos no está definido explícitamente y se especifica mediante uno o varios parámetros pasados al momento de instanciar objetos. De este modo es posible que un mismo código pueda ser reutilizado con tipos de datos diferentes. Es decir, se crea el modelo de una clase que permitirá definir distintas instancias de la misma para diferentes tipos de datos.

La definición de una clase genérica tiene el siguiente aspecto:

```
template <class T>
class Nombreclase{
private:
 T Atributo;
 … // datos miembros de la clase
 public:
 Nombreclase ( );
 T Metodo1 ( );
 void Metodo2 (T valor); // funciones miembros de la clase
 ……
 };
                                                                                                    ...
};
```

La definición de una plantilla comienza siempre con **template <...>** y los parámetros de la lista **<...>** no son valores, sino tipos de datos.

Para declarar un objeto, a partir de una plantilla de clase, se aplica la siguiente sintaxis:

Nombreclase <tipo> Nombre del objeto;

Donde "tipo" indica el tipo de dato que reemplazará todas las ocurrencias de "T" en la definición de la clase.

Por ejemplo, si quisiera declarar dos objetos de tipo "Nombreclase", pero uno con el tipo "int" y otro con "float", entonces la definición seria la siguiente:

```
Nombreclase <int> Objeto1;
Nombreclase <float> Objeto2;
```

Para Objeto1, el atributo, el resultado del Metodo1 y el parámetro del Metodo2 serán de tipo "int", mientras que para el Objeto2, el atributo, el resultado del Metodo1 y el parámetro del Metodo2 serán de tipo "float".

En cuanto a la definición de métodos, la sintaxis que se aplica es:

```
template <class T>
T Plantilla de clase <T> :: Metodo1( )
 { … // cuerpo de método
 }
```

En este caso, el método de la clase da un resultado de tipo "T". Por lo tanto, el tipo de resultado se definirá en el momento de crear un objeto de dicha clase.

```
template <class T>
 void Plantilla de clase <T> :: Metodo2(T valor)
 { … // cuerpo de método
 }
```

En este caso, el método recibe un parámetro que será del tipo "T". Por lo tanto, el tipo se especifica en el momento de declarar el objeto.