Template

Template

Las plantillas son la base para la programación genérica en C++. Como lenguaje fuertemente tipado, C++ requiere que todas las variables tengan un tipo específico, ya sea declarado explícitamente por el programador o deducido por el compilador. Sin embargo, muchas estructuras de datos y algoritmos tienen el mismo aspecto sin importar en qué tipo estén operando. Las plantillas le permiten definir las operaciones de una clase o función, y permiten que el usuario especifique en qué tipos concretos deberían funcionar esas operaciones.

Esquema

```
template <class myType>

myType GetMax (myType a,
myType b) {

 return (a>b?a:b);

}
```

Aquí hemos creado una función de plantilla con myType como su parámetro de plantilla. Este parámetro de plantilla representa un tipo que aún no se ha especificado, pero que se puede utilizar en la función de plantilla como si fuera un tipo normal.

Definición y uso de plantillas

Una plantilla es una construcción que genera un tipo o función normal en tiempo de compilación en función de los argumentos que proporciona el usuario para los parámetros de la plantilla.

Definición y uso de plantillas

```
template <typename T>
T minimum(const T& lhs, const T& rhs)
{
    return lhs < rhs ? lhs : rhs;
}
```

Definición y uso de plantillas

El código anterior describe una plantilla para una función genérica con un único parámetro de tipo T , cuyo valor de retorno y parámetros de llamada (lhs y rhs) son todos de este tipo. Puede nombrar un parámetro de tipo como desee, pero, por convención, las letras mayúsculas simples son las más utilizadas.

Definición y uso de plantillas

T es un parámetro de plantilla; la typename palabra clave dice que este parámetro es un marcador de posición para un tipo. Cuando se llama a la función, el compilador reemplazará cada instancia de T con el argumento de tipo concreto especificado por el usuario o deducido por el compilador. El proceso en el que el compilador genera una clase o función a partir de una plantilla se denomina creación de instancias de plantilla ; minimum<int>es una instanciación de la plantilla minimum<T>.

Definición y uso de plantillas

En otros lugares, un usuario puede declarar una instancia de la plantilla que está especializada para int. Suponga que get_a() y get_b() son funciones que devuelven un int:

```
int a = get_a();

int b = get_b();

int i = minimum<int>(a, b);
```

Definición y uso de plantillas

```
template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}
```

Ejemplo

Ejemplo sin Identificador

```
template <class T>

T GetMax (T a, T b) {

  return (a>b?a:b);

}
```

Observe cómo, en este caso, llamamos a nuestra plantilla de función GetMax() sin especificar explícitamente el tipo entre paréntesis angulares <> . El compilador determina automáticamente qué tipo se necesita en cada llamada.

Ejemplo

Clases-plantilla

Podemos escribir también clases con plantillas, llamadas clases-plantilla, y crear objetos que sirvan con cualquier tipo de dato, esta característica es de gran ayuda al momento de escribir, por ejemplo, una clase para arreglos o para matrices que sirven con cualquier tipo y no solo con uno.

Clases-plantilla

En este caso, el identificador de tipo sirve en toda la definición de la clase, ya sea para los atributos o los métodos, a continuación tenemos la sintaxis para declarar una clase-plantilla:

template <class identificador> definición_de_clase;

// Para las funciones miembro definidas fuera de la clase

template <class identificador>

tipo_retorno nombre_clase<identificador>::nombre_función(argumentos){

// implementación

}

Clases-plantilla

```
template <class T> //identificador de tipo: T
class Coordenada{
private:
  T x; //atributos de tipo T
  T y;
 public:
  Coordenada(T x=0, T y=0); //parámetros de tipo T
 T dameX(){return x};    T dameY(){return y};      //retorna un tipo T
  void nuevoX(T x){this -> x = x};   void nuevoY(T y){this -> y = y};}
template <class T>
Coordenada<T>::Coordenada(T x, T y){
  this -> x = x;    this -> y = y; }
```

Ejemplo

Programación genérica

Gracias a las clases-plantilla es posible lograr un estilo de programación genérico, como es el caso de la biblioteca estándar de plantillas STL (Standard Template Library) de C++ que utiliza las plantillas para crear y proporcionar una gran colección de herramientas como lo son los contenedores (vector es uno de los más conocidos) y algoritmos (como los de ordenamiento y de búsqueda) que trabajan sobre estas estructuras de datos.

Bibliografia

https://cplusplus.com/doc/oldtutorial/templates/

https://docs.microsoft.com/en-us/cpp/cpp/templates-cpp?view=msvc-170

https://codingornot.com/cc-plantillas-templates-en-c#:~:text=Una%20plantilla%20es%20una%20manera,cada%20versi%C3%B3n%20de%20la%20funci%C3%B3n.