Plantillas de funciones

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.

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

template es la palabra clave que indica que se está declarando un template.

<typename **tipo**> es la declaración del parámetro de plantilla. typename puede ser reemplazado por class, y ambos se utilizan para declarar un tipo de parámetro.

tipo es el tipo de retorno de la función, y nombreFuncion es el nombre de la función.

tipo parametro es un ejemplo de cómo se utiliza el parámetro de plantilla en la función.

Ventajas

El uso de templates en C++ proporciona varias ventajas significativas, lo que contribuye a la flexibilidad y reutilización del código. Aquí algunas de las ventajas más destacadas:

1. **Generalización del Código:** Los templates permiten escribir código que funciona con varios tipos de datos sin tener que reescribir el código para cada tipo específico. Esto lleva a un código más general y reutilizable.
2. **Polimorfismo Paramétrico:** Los templates proporcionan polimorfismo paramétrico, lo que significa que puedes escribir código que es independiente del tipo de datos con el que trabajas. Esto es diferente del polimorfismo basado en herencia que se logra con clases y funciones virtuales.
3. **Eficiencia de Tiempo de Compilación:** El código generado a partir de templates se crea en tiempo de compilación, lo que puede llevar a una mayor eficiencia en términos de rendimiento. Esto se debe a que el compilador puede realizar optimizaciones específicas para cada tipo de datos utilizado.

Desventajas

Aunque los templates en C++ tienen muchas ventajas, también existen algunas desventajas que es importante considerar:

1. **Mayor Tiempo de Compilación:** El uso extensivo de templates puede llevar a un aumento en el tiempo de compilación. Esto se debe a que el compilador debe generar y compilar código para cada instancia específica del template que se utilice en el programa.

2. **Complejidad de Error de Compilación:** Los mensajes de error del compilador relacionados con templates pueden ser largos y complejos. Interpretar y corregir estos mensajes puede resultar desafiante, especialmente para aquellos que no están familiarizados con el código base.
3. **Exceso de Generalización:** En algunos casos, la generalización excesiva a través de templates puede hacer que el código sea menos claro y más difícil de entender. El uso excesivo de templates puede llevar a una sobrecarga cognitiva para los desarrolladores.

Ejemplo

El siguiente ejemplo utiliza templates y eloperador sobrecargado *

Clase Operador.h

```h
#include<stdio.h>
#include<iostream>
#include<conio.h>

template<typename T>

class Numero {

    private:
        //Este constructor inicializa las dos variables privadas de la clase Numero
        //con los valores especificados en los argumentos v1 y v2 tipo template.
        T valor1,valor2;

    public:
        //Recibe dos valores de tipo T como argumentos (v1 y v2).
        //Asigna estos valores a las variables privadas valor1 y valor2 del objeto que se está creando.
        Numero(T v1,T v2):valor1(v1),valor2(v2){};
        void imprimir();
        T  operator * (Numero val);
        void UsarSobrecarga(void);

};
```

Clase Operador.cpp

```cpp
#include<stdio.h>
#include<iostream>
#include<conio.h>
#include "Operador.h"
#include <stdio.h>
using namespace std;

// Declaración de la plantilla (template)
    template<typename T>
    void Numero<T>::imprimir() {
    //imprime el valor tipo template
    cout << valor1 << " * " << valor2 << " = " << valor1 * valor2 << std::endl;

}
// Declaración de la plantilla (template)
    template <typename T>
    T Numero<T>:: operator*(Numero val) {
    //Se realiza una operacion como en cualquier funcion
    return Numero(valor1 * val.valor1, valor2 * val.valor2);
}
```

main.cpp

```cpp
#include<stdio.h>
#include"Operador.cpp"//llamamos al .h

int main() {
    float a,b;

    printf("ingrese un numero a:  ");
    scanf("%f", &a);

    printf("ingrese un numero b:  ");
    scanf("%f", &b);

    //Se declara el tipo de dato que queremos que tome el template
    //en este caso float y luego se llama a la funcion
    Numero<float> intCalculo(a,b);

    intCalculo.imprimir() ;

  return 0;
}
```

Fuentes:
https://codingornot.com/cc-plantillas-templates-en-c
https://www.linkedin.com/pulse/ctemplates-carlos-enrique-hernandez-ibarra/?originalSubdomain=es