Laboratorio

Plantillas, sobrecarga de operadores

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando plantillas, sobrecarga de operadores del lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando plantillas, sobrecarga de operadores del lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Introducción

Una plantilla o template 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.

3.2. ¿Cómo funcionan las plantillas?

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:

```
// Para una función, ambas opciones son equivalentes
template <class identificador> definición_de_función;
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.

3.3. Plantillas de funciones

Las plantillas de funciones son una de las propiedades más importante que tiene C++, con su uso podemos crear funciones genéricas que acepten cualquier tipo de dato en sus parámetros, si C++ no contara con esta propiedad, nos veríamos obligados a tener que escribir una función una y otra vez, generando código repetido.

Las plantillas son muy usadas en la programación y se emplean cuando se necesita utilizar la misma función, pero con diferentes argumentos. Por ejemplo, supongamos que tenemos una función que obtiene el valor absoluto de un número, si refrescamos la memoria, sabemos que el valor absoluto de x siempre será x, es decir, si x es negativo, su valor absoluto será el mismo número, pero en positivo, si x es positivo, su valor absoluto será el mismo. Una función que haga esto sería la siguiente:

```
int valorAbsoluto(int numero){
        return (numero < 0)? -numero : numero;
}
```

Ok, pero tenemos un problema, con esta función podemos obtener el valor absoluto de un número, pero sólo de un entero. Esto está mal porque sabemos que hay más tipos de números. Para aceptar otros tipos de datos como un float, double, long lo que podríamos hacer sería sobrecargar la función, pero vaya que no será lo más idóneo, es por eso que para este tipo de casos C++ proporciona un mecanismo más ágil y potente, las plantillas de funciones. Así que el lugar de escribir las cuatro funciones, lo que haremos sería lo siguiente:

```
template <class tipo>
tipo valorAbsoluto(tipo numero){
        return (numero < 0)? -numero : numero;
}
```

La palabra reservada template indica al compilador que estamos haciendo uso de una plantilla de función. El símbolo tipo indica al compilador que puede ser sustituido por cualquier tipo de dato apropiado: int, float, double, etc. El nombre de tipo puede ser cualquiera que queramos, siempre respetando que no sea una palabra reservada. Por ejemplo:

```
template <class T>
T ejemploDeFuncion(T parametro1, T parametro2){
        //Código de la función…
}
```

Pero también podemos usar plantillas de funciones de la siguiente manera:

```
template <class T1, class T2>
T2 ejemploDeFuncion(T1 parametro1, T2 parametro2){
        //Código de la función…
}
```

Algo que debemos tener en cuenta cuando hagamos uso de una plantilla de función es que la palabra template debe aparecer arriba del prototipo de la función, así como también arriba de la definición de la función. Veamos un mejor ejemplo:

```
#include "stdafx.h"
#include ”iostream”

using namespace std;

template <class P>
P valorAbsolutoDeUnNumero(P numero);

int main()
{
        int entero = -323;
        long largo = -838237237;
        float real = -3.1454;
```

```
        double realDoble = -0.232352;

        cout << valorAbsolutoDeUnNumero(entero) << endl;
        cout << valorAbsolutoDeUnNumero(largo) << endl;
        cout << valorAbsolutoDeUnNumero(real) << endl;
        cout << valorAbsolutoDeUnNumero(realDoble) << endl;
        system("pause");
        return 0;
}

template <class P>
P valorAbsolutoDeUnNumero(P numero) {
        return (numero < 0) ? -numero : numero;
}
```

3.4. 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.

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
}
```

Se entiende mejor la idea con un ejemplo, veamos algo simple:

```
// coordenada.h
template <class T>
class Coordenada {
private:
        T x;
        T y;

public:
        Coordenada(T x = 0, T y = 0);
```

```
        T getX() {
                return x;
        };

        T getY() {
                return y;
        }
};

template <class T>
Coordenada<T>::Coordenada(T x, T y) {
        this->x = x;
        this->y = y;
    }
```

Con esta clase podemos instanciar objetos que representan coordenadas de cualquier tipo de dato, por ejemplo:

```
// Función main
#include "stdafx.h"
#include "iostream"
#include "Coordenada.h"

using namespace std;

int main()
{
        Coordenada <int> coord1(2, 1); // Enteros
        Coordenada <float> coord2(1.5, 0.5); // Reales

        cout << "Primera coordenada de X es: " << coord1.getX() << endl;
        cout << "Segunda coordenada de Y es: " << coord2.getY() << endl;

        system("pause");
        return 0;
    }
```

El resultado sería lo siguiente:

Primera coordenada de X es: 2
Segunda coordenada de Y es: 0.5

4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Se pide escribir una función utilizando plantillas que tome tres argumentos genéricos y devuelva el menor y el máximo de ellos como valor de retorno. La función debe ser capaz de dar este tipo de resultados.
2. Se pide escribir una función utilizando plantillas que tome dos argumentos genéricos de tipo entero y flotante que devuelva las cuatro operaciones básicas.
3. Se pide escribir una función utilizando plantillas que tome dos valores genéricos de tipo char y string (5 veces); char corresponde a una letra y string corresponde al apellido. El programa debe mostrar por pantalla el siguiente formato de correo electrónico: char/string@unsa.edu.pe.
4. Implemente un programa que haga uso de plantillas para determinar el mínimo y máximo valor de un arreglo de elementos dado. Debe de existir dos funciones, la primera que retorne el mayor de los valores y la segunda que retorne el menor de los valores. Asimismo, en la función main, se hace una prueba de estas funciones, con arreglos de enteros y flotantes.

   int ArrayEntero [5] = {10,7,2, 8, 6};
   float ArrayFloat [5] = {12.1, 8.7, 5.6, 8.4, 1.2};

5. Realizar la implementación de un programa que haga uso de plantillas, para elaborar una función que permita ordenar ascendentemente y descendentemente los elementos de un arreglo de valores enteros y otro arreglo de valores flotantes. Las funciones deben recibir como parámetros, un puntero al tipo de elemento dado, y dos enteros que indican los índices del primero y último elemento.

   int ArrayEntero [5] = {5,7,2,8,6,1,3,4,9};
   float ArrayFloat [5] = {10.1, 8.4, 3.6, 4.4, 11.2};

5. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB09_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB09_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB09_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB09_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.