Laboratorio 17

Programación genérica

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando programación genérica del lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando programación genérica del lenguaje de programación C++.

2. Equipos y Materiales

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

3. Marco Teórico

3.1. Programación genérica

El método de Programación Genérica se implementa para aumentar la eficiencia del código. La programación genérica permite al programador escribir un algoritmo general que funcionará con todos los tipos de datos. Elimina la necesidad de crear diferentes algoritmos si el tipo de datos es un entero, una string o un carácter.

Las ventajas de la programación genérica son

- Reutilización de código
- Evite la sobrecarga de funciones
- Una vez escrito, puede usarse para múltiples ocasiones y casos.

Los genéricos se pueden implementar en C++ usando plantillas o también llamados templates. Template es una herramienta simple pero muy poderosa en C++. La idea simple es pasar el tipo de datos como parámetro para que no necesitemos escribir el mismo código para diferentes tipos de datos. Por ejemplo, una empresa de software puede necesitar ordenar() para diferentes tipos de datos. En lugar de escribir y mantener varios códigos, podemos escribir un sort() y pasar el tipo de datos como parámetro.

3.2. Funciones genéricas usando plantilla:

Escribimos una función genérica que se puede utilizar para diferentes tipos de datos. Ejemplos de plantillas de funciones son sort(), max(), min(), printArray()

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// funciona para todos los tipos de datos.
template <typename T>

T maximo(T x, T y)
{
	return (x > y) ? x : y;
}

int main()
{
	cout << maximo<int>(3, 7) << endl;
	cout << maximo<double>(3.0, 7.0) << endl;
	cout << maximo<char>('g', 'e') << endl;
	system("pause");
	return 0;
}
```

El resultado sería el siguiente:

```
7
7
g
```

3.3. Clase genérica usando plantilla:

Al igual que las plantillas de funciones, las plantillas de clases son útiles cuando una clase define algo que es independiente del tipo de datos.

A continuación, se muestra un ejemplo simple de la clase de matriz de plantilla.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

template <typename T>
class Array {
private:
	T* ptr;
	int size;

public:
	Array(T arr[], int s);
	void print();
};

template <typename T>
Array<T>::Array(T arr[], int s)
{
	ptr = new T[s];
	size = s;
	for (int i = 0; i < size; i++)
		ptr[i] = arr[i];
}

template <typename T>
void Array<T>::print()
{
	for (int i = 0; i < size; i++)
		cout << " " << *(ptr + i);
	cout << endl;
}

int main()
{
	int arr[5] = { 1, 2, 3, 4, 5 };
	Array<int> a(arr, 5);
	a.print();
	system("pause");
	return 0;
}
```

El resultado sería el siguiente:

1 2 3 4 5

3.4. Trabajar con genéricos de varios tipos:

Podemos pasar más de un tipo de datos como argumentos a las plantillas. El siguiente ejemplo demuestra lo mismo.

```
#include <iostream>
using namespace std;

template <class T, class U>
class A {
	T x;
	U y;

public:
	A()
	{
		cout << "Constructor llamado" << endl;
	}
};

int main()
{
	A<char, char> a;
	A<int, double> b;
	system("pause");
	return 0;
}
```

El resultado sería el siguiente:

Constructor llamado
Constructor llamado

4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Desarrolle un programa de calculadora simple (operaciones básicas) que utilice clases con plantillas
2. Definir una clase utilizando plantillas que permita almacenar datos en un árbol binario. Por el momento solo se insertarán elementos en la estructura. Simule el proceso de almacenar 100 datos y verifique que la estructura no tenga problemas.
3. Analice y describa el siguiente comportamiento del siguiente código:

```cpp
#include <iostream>

template <class T>
class Contendor {
    T elemento;
    public :
    Contendor (T arg ) {
        elemento = arg ;
    }
    T add() { return ++elemento; }
};

template <>
class Contendor<char> {
    char elemento;
    public :
    Contendor ( char arg ) {
        elemento = arg ;
    }
    char uppercase() {
    if ((elemento >= 'a') && (elemento <= 'z')) {
        elemento += 'A'-'a'; }
    return elemento ;
    }
};

int main() {
    Contendor<int> cint (5) ;
    Contendor<char> cchar('t');
    std::cout << cint.add() << std::endl;
    std::cout << cchar.uppercase() << std::endl;
    return 0;
}
```

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 LAB17_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB17_GRUPO_A _2022123_PEDRO_VASQUEZ).

4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB17_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB17_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

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