**Name:**

**Advanced Programming in C++**

**Lab Exercise 4/7/2023**

In this lab exercise, you will modify the NumberList class to be a template class. A class template is a method of creating a class or a function that can accept a variety of data types.

A function template is a “generic” function that can work with any data type. As you might remember, we have in the past used function overloading for this behavior.

int square(int number)

{

return number \* number;

}

double square( double number)

{

return number \* number;

}

Here is a function template for a square function:

template <class T>

T square(T number)

{

return number \* number;

}

Here is an example program:

// template example.cpp : Defines the entry point for the console application.

#include <iostream>

using namespace std;

template <class T> //Function prototype

void operate(T, T, char);

int main()

{

int a = 50, b = 7;

char c = '+';

operate(a, b, c);

return 0;

}

template <class T>

void operate(T x, T y, char ch)

{

switch (ch)

{

case '+':

cout << x + y << endl;

break;

case '-':

cout << x - y << endl;

break;

case '\*':

cout << x \* y << endl;

break;

case '/':

cout << x / y << endl;

break;

}

}

Templates may also be used to create generic classes and abstract data types. Class templates allow you to create one general version of a class without having to duplicate the code to handle multiple data types.

Here is an example of a template linked list class:

// A class template for holding a linked list.

#ifndef LINKEDLIST\_H

#define LINKEDLIST\_H

template<class T>

class LinkedList

{

private:

// Declare a structure for the list

struct ListNode

{

T value;

struct ListNode \*next;

};

ListNode \*head; // List head pointer

public:

LinkedList(); // Constructor

~LinkedList(); // Destructor

void appendNode(T);

void insertNode(T);

void deleteNode(T);

void displayList();

T getNode();

void prependNode(T);

void displayListBackwards();

void destroyList();

void reverseList();

void copyList(LinkedList&);

};

#include "LinkedList.cpp"

#endif

**Programming Exercises**

1. Write templates for two functions mini and maxi. The mini function should accept two arguments and return the value of the argument that is the lesser of the two. The maxi function should accept two arguments and return the value of the argument that is the greater of the two. Write test programs to test it for integer and double values.
2. Write a template function called total that has 2 parameters (a reference parameter that is sum of values and the number of values). The function should keep a running total of the values entered by the user, then “return” the sum using a reference parameter. The argument sent into the function should be the number of values the function is to read. Test the template in a simple driver program.
3. Modify the NumberList class to be a template LinkedList class (LinkedList.h) as shown in the above example. Note: The LinkedList.cpp is not to be added to the project but must be in the project folder. The LinkedList.cpp file is brought in with an include at the bottom of the LinkedList.h file.

Note: The LinkedList class is exactly the same as the NumberList class but has been renamed. The LinkedList class will be converted to a Template class capable of holding different data types.

The following line must be added at the beginning of each function in the implementation file (LinkedList.cpp).

template <class T>

You must also add <T> after the reference to the linked list class. For example signature lines for the appendNode function should look like

template <class T>

void LinkedList<T>::appendNode(T num)

You must also change any reference to a type double to T.

Here is a test program to test your linked list:

// This program demonstrates the linked list template.

#include <iostream>

#include <string>

using namespace std;

#include "LinkedList.h"

int main()

{

LinkedList<string> list;

list.insertNode("Alice");

list.insertNode("Mary");

list.insertNode("Fred");

cout << "Here is the list of strings" << endl;

list.displayList();

LinkedList<double> list2;

list2.insertNode(23.65);

list2.insertNode(12.935);

list2.insertNode(4.768);

cout << "Here is the list of doubles" << endl;

list2.displayList();

cout << endl;

return 0;

}