//Demo use of template function

//Author: nmessa

#include <iostream>

#include <string>

using namespace std;

//Template function prototype

template <class T>

T operate(T, T, char);

int main()

{

//int a = 32, b = 7;

double a = 57.3, b = 12.8;

//char a = 'A', b = 'B';

//string a = "A", b = "B";

char ch = '-';

cout << a << ch << b << " = " << operate(a, b, ch) << endl;

return 0;

}

template <class T>

T operate(T x, T y, char c)

{

switch (c)

{

case '+': return x + y;

case '-': return x - y;

case '\*': return x \* y;

case '/': return x / y;

}

}

//Demo use of template function

//Author: nmessa

#include <iostream>

using namespace std;

template<class T>

T square(T);

int main()

{

double number = 7.93;

int iNumber = 7;

cout << square(number) << endl;

cout << square(iNumber) << endl;

return 0;

}

template<class T>

T square(T num)

{

return num \* num;

}

//Demo use of template class

//Author: nmessa

#include <iostream>

#include <string>

#include "LinkedList.h"

using namespace std;

int main()

{

LinkedList<double> list;

list.appendNode(4.6);

list.appendNode(7.5);

cout << "Here are the initial values: \n";

list.displayList();

cout << endl;

cout << "Now inserting a value \n";

list.insertNode(5.1);

cout << "Here are the nodes now\n";

list.displayList();

cout << endl;

cout << "Now deleting a node\n";

list.deleteNode(4.6);

cout << "Here is the list\n";

list.displayList();

}

// Specification file for the LinkedList class

#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