CSE225L – Data Structures and Algorithms Lab
Lab 08
Stack (Linked List)

In today's lab we will design and implement the Stack ADT using linked list.

stacktype.h

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED
class FullStack
{};
class EmptyStack
{};
template <class ItemType>
class StackType
{
    struct NodeType
    {
        ItemType info;
        NodeType* next;
    };
    public:
        StackType();
        ~StackType();
        void Push(ItemType);
        void Pop();
        ItemType Top();
        bool IsEmpty();
        bool IsFull();
    private:
        NodeType* topPtr;
};
#endif // STACKTYPE_H_INCLUDED
```

stacktype.cpp

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

template <class ItemType>
StackType<ItemType>::StackType()
{
    topPtr = NULL;
}

template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (topPtr == NULL);
}

template <class ItemType>
ItemType StackType<ItemType>::Top()
{
    if (IsEmpty())
        throw EmptyStack();
    else
        return topPtr->info;
}
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    NodeType* location;
    try
    {
        location = new NodeType;
        delete location;
        return false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}
template <class ItemType>
void StackType<ItemType>::Push(ItemType newItem)
{
    if (IsFull())
        throw FullStack();
    else
    {
        NodeType* location;
        location = new NodeType;
        location->info = newItem;
        location->next = topPtr;
        topPtr = location;
    }
}
template <class ItemType>
void StackType<ItemType>::Pop()
{
    if (IsEmpty())
        throw EmptyStack();
    else
    {
        NodeType* tempPtr;
        tempPtr = topPtr;
        topPtr = topPtr->next;
        delete tempPtr;
    }
}
template <class ItemType>
StackType<ItemType>::~StackType()
{
    NodeType* tempPtr;
    while (topPtr != NULL)
    {
        tempPtr = topPtr;
        topPtr = topPtr->next;
        delete tempPtr;
    }
}
```

Generate the **driver file (main.cpp)** where you perform the following tasks. Note that you cannot make any change to the header file or the source file.

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**

• Take infix expressions from the user as input, determine the outcome of the expression and gives that back to user as output, or the text "Invalid expression" if the expression is not a valid one. You will have to solve this problem in two steps. First, you have to convert the expression from infix notation to postfix notation. You are going to need a stack in order to do so. In the next step, you will have to evaluate the postfix expression and determine the final result. Again, you will need a stack in order to do this. All the operands in the infix expressions are single digit non-negative operands and the operators include addition (+), subtraction (-), multiplication (*) and division (/).	`10 + 3 * 5 / (16 - 4)`
	`(5 + 3) * 12 / 3`
	`3 + 4 / (2 - 3) * / 5`
	`7 / 5 + (4 - (2) * 3`
		`11.25`
		`32`
		`Invalid expression`
		`Invalid expression`