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)** and perform the following tasks:

Operation to Be Tested and Description of Action	Input Values	Expected Output
• Create a stack		
• Check if the stack is empty		Stack is Empty
• Push four items	5 7 4 2	
• Check if the stack is empty		Stack is not Empty
• Check if the stack is full		Stack is not full
• Print the values in the stack		2 4 7 5
• Push another item	3	
• Print the values in the stack		2 4 7 5 3
• Check if the stack is full		Stack is not full
• Pop two items		
• Print top item		7

- Add a function **`ReplaceItem`** to the `StackType` class which replaces all occurrences of **`oldItem`** with **`newItem`** in the Queue.

  ```
  void ReplaceItem(int oldItem, int newItem);
  ```

  Sample Input &Output:

  Stack items: 21 26 13 26 29 —**`ReplaceItem(26, 9)`**→ Stack items: 21 9 13 9 29