# North South University
# CSE-225L(Data Structures & Algorithm)
# Fall - 2018
# Lab-9 (Stack – Linked List Based)

**Class "StackType":**

**stacktype.h**

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED
#include <iostream>
using namespace std;

class FullStack{};

class EmptyStack{};

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

**stacktype.cpp**

```
#include "stacktype.h"

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

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

```
template <class DataType>
DataType StackType<DataType>::Top()
{
      if (IsEmpty())
         throw EmptyStack();// throwing an object as an 'exception'
      else
        return topPtr->info;
}

template <class DataType>
bool StackType<DataType>::IsFull()
{
           NodeType* location;

           try
           {
                 location = new NodeType;
                 delete location;
                 return false;
           }
           catch(bad_alloc& exception)
           {
             return true;
           }
}

template <class DataType>
void StackType<DataType>::Push(DataType newItem)
{
      if (IsFull())
                 throw FullStack();
      else
      {
           NodeType* location;
           location = new NodeType;
           location->info = newItem;
           location->next = topPtr;
           topPtr = location;
      }
}
```

```
template <class DataType>
void StackType<DataType>::Pop()
{
      if (IsEmpty())
                  throw EmptyStack();
      else
      {
            NodeType* tempPtr;
            tempPtr = topPtr;
            topPtr = topPtr->next;
            delete tempPtr;
      }
}



template <class DataType>
StackType<DataType>::~StackType()
{
      NodeType* tempPtr;

      while (topPtr != NULL)
      {
            tempPtr = topPtr;
            topPtr = topPtr->next;
            delete tempPtr;
      }
}

template class StackType<int>; // so CodeBlocks can compile the
                        // template for int type data

template class StackType<double>;// so CodeBlocks can compile the
                              // template for double type data

template class StackType<char>;// so CodeBlocks can compile the
                              // template for char type data
```