North South University

Department of Electrical and Computer Engineering

CSE 225L.13 (Data Structures and Algorithms Lab)

Lab 8: Stacks (Array Based)

Instructor: Syed Shahir Ahmed Rakin, Arfana Rahman

Objective:

- Learn how the Stacks work when made with arrays.

Remember the Arrays:

Go check the theory slides and the previous manuals to have an idea about the arrays

Stack:

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It is a collection of elements with two main operations: **push**, which adds an element to the top of the stack, and **pop**, which removes the top element from the stack. The element that was added last is the one that is removed first. Additionally, a common operation is **top**, where you can view the top element without removing it.

Let us push the values in this order, thus, the stack will look like this, the leftmost value is the first value to be inserted, and the rightmost value is the last value to be inserted.

5	3	7	6	1

Now the stack will look like this after we use the pop function

5	3	7	6	

Prototype of Sorted List:

The header and source file of the Array-based Sorted List is given as follows.

stacktype.h

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED
const int MAX_ITEMS = 5;
class FullStack
// Exception class thrown by Push when stack is full.
{};
class EmptyStack
// Exception class thrown by Pop and Top when stack is emtpy.
{};

template <class ItemType>
class StackType
{
public:
    StackType(); bool IsFull(); bool IsEmpty();
    void Push(ItemType); void Pop(); ItemType Top();
private:
    int top; ItemType items[MAX_ITEMS];
};
#endif // STACKTYPE_H_INCLUDED
```

stacktype.cpp

```
template <class ItemType>
StackType<ItemType>::StackType()
{
    top = -1;
}

template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (top == -1);
}

template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    return (top == MAX_ITEMS-1);
}
```

```
template <class ItemType>
void StackType<ItemType>::Push(ItemType newItem)
{
    if( IsFull() ) throw FullStack();
    top++;
    items[top] = newItem;
}

template <class ItemType>
void StackType<ItemType>::Pop()
{
    if( IsEmpty() ) throw EmptyStack();
    top--;
}

template <class ItemType>
ItemType StackType<ItemType>::Top()
{
    if (IsEmpty()) throw EmptyStack();
    return items[top];
}
```

Tasks:

Generate the **driver file (main.cpp)** where you perform the following tasks. However, there is a restriction that *you cannot make any changes to the header file or the source file*.

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
▪ Create a stack of integers		
▪ 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 (in the order the values are given as input)		5 7 4 2
▪ Push another item	3	
▪ Print the values in the stack		5 7 4 2 3
▪ Check if the stack is full		Stack is full
▪ Pop two items		
▪ Print top item		4
▪ Take strings of parentheses from the user as input and use a stack to check if the string of parentheses is balanced or not	( )	Balanced
	(( )) ( ) (( ) ( )) ( )	Balanced
	(( )) ( ) ((( )	Not balanced
	(( )))) ((( )	Not balanced