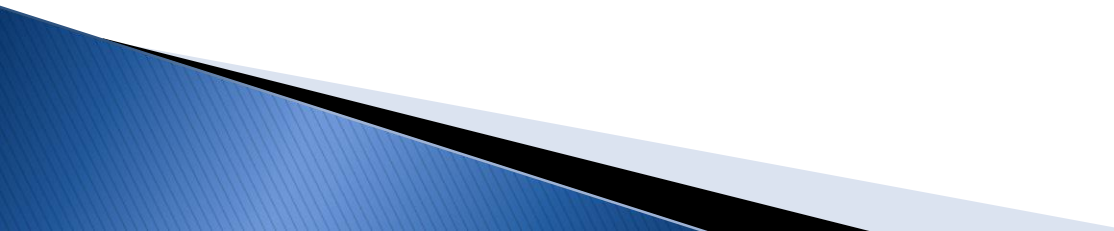


Data Structures and Algorithms– Lab 4

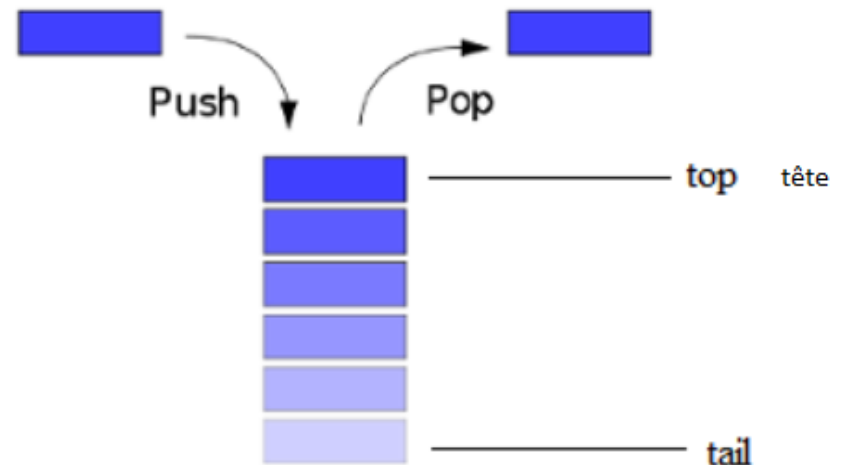
Iulia-Cristina Stanica
iulia.stanica@gmail.com

Roadmap

- ▶ Stack
 - ▶ Using headers
 - ▶ Applications of stack
- 

Stack

- ▶ Instance of an abstract data type (ADT)
- ▶ Formalizes the concept of the LIFO collection (last in first out)



Basic Operations

- ▶ **push(x)**
 - Adds the element x at the top of the stack
- ▶ **pop()**
 - Removes the element from the top of the stack and returns it
 - Returns an error if the stack is empty
- ▶ **peek()**
 - Returns (but does not remove) the element at the top of the stack
- ▶ **isEmpty()**
 - Returns 1 if the stack is empty and 0 otherwise

Stack: Array-based Implementation

```
#include <iostream>
using namespace std;
#define NMAX 10
template<typename T>
class Stack {
private:
    T stackArray[NMAX]; // an array of NMAX dimension
    int topLevel; // the top of the stack, representing the INDEX of last element of the
                  // stackArray:0, 1, 3,....

public:
    void push(T x) // puts an element in the stack array
    {
        if (topLevel >= NMAX-1) //check if the stack array has the maximum dimension
        {
            cout<<"The stack is full: we have already NMAX elements!\n";
            //exit the function without making anything
            return;
        }
        /*add an element=> the index of the last element of the stack Array
        increases and put the value of the new element in the stack array*/
        stackArray[++topLevel] = x;
    }
}
```

Stack: Array-based Implementation

```
int isEmpty()
{
    //returns 1, if topLevel>=0, meaning the stack array has
    elements
    // returns 0, otherwise
    return (topLevel < 0);
}

T pop() // extracts and element from the stack array and returns the
new top
{
    if (isEmpty()) {
        // the extraction is made only if the array is not empty
        cout<<"The stack is empty! \n";
        T x;
        return x;
    }
    return stackArray[topLevel--];    // the topLevel decreases and the
new top is changed
    //difference return stackArray[--topLevel] ?
}
```

```
T peek()
{
    // returns the top of the stack
    if (isEmpty())
    {
        // the extraction is made only if the array is
        not empty
        cout<<"The stack is empty! \n";
        T x;
        return x;
    }
    return stackArray[topLevel];
}

Stack()
{ // constructor
topLevel = -1; //the stack is empty in the
beginning
}

~Stack() { // destructor
}
};
```

Separating declaration – implementation

```
#include <iostream>
using namespace std;
#define NMAX 10 // pre-processing directive
```

```
template<typename T>
class Stack {
private:
    // an array of NMAX dimension
    T stackArray[NMAX];
    /* the top of the stack, representing the INDEX of last element of the
    stackArray:0, 1, 3,...*/
    int topLevel;
public:
    void push(T x);
    int isEmpty();
    T pop();
    T peek();
    Stack();
    ~Stack();
};
```

```
template<typename T>
void Stack<T>::push(T x) {
    //puts an element in the stack array

    //check if the stack array has the maximum dimension
    if (topLevel >= NMAX - 1)
    {
        cout<<"The stack is full: we have already NMAX elements!\n";
        //exit the function without making anything
        return;
    }
    /*add an element=> the index of the last element of the stack Array
    increases and put the value of the new element in the stack array*/
    stackArray[++topLevel] = x;
}
```

```
template<typename T>
int Stack<T>::isEmpty() {
    //returns 1, if topLevel>=0, meaning the stack array has elements
    // returns 0, otherwise
    return (topLevel < 0);
}
```

```
template<typename T>
T Stack<T>::pop() {
    // extracts and element from the stack array and returns the new top
    if (isEmpty()) {
        // the extraction is made only if the array is not empty
        cout<<"The stack is empty! \n";

        T x;
        return x;
    }
    // the topLevel decreases and
    the new top is changed
    return stackArray[--topLevel];
}
```

```
template<typename T>
T Stack<T>::peek() {
    // returns the top of the stack
    if (isEmpty()) {
        // the extraction is made only if the array is not empty
        cout<<"The stack is empty! \n";

        T x;
        return x;
    }
    return stackArray[topLevel];
}
```

```
template<typename T>
Stack<T>::Stack() { // constructor
    topLevel = -1; // the stack is empty in the beginning
}
```

```
template<typename T>
Stack<T>::~~Stack() { // destructor
}
```

Main function

```
int main()
{
    Stack<int> myStack;
    myStack.peek();
    myStack.push(5);
    myStack.push(2);
    myStack.push(3);
    cout<<myStack.peek()<<"\n";
    cout<<myStack.pop();
    myStack.push(1);
    myStack.push(4);
    cout<<myStack.pop();
    return 0;
}
```


Ex1.

- a) Test the implementation of the stack.
- b) Define the Stack class in a **header** file (e.g. mystack.h) and test it in a different file which contains the main function and includes the header:

```
#include "mystack.h"
```
- c) Add the following method to the class Stack:
 - Display all the elements of a stack.

Ex2.

- ▶ Write a program which reads a number n and n numbers of type double. The numbers are displayed in reversed order using the stack.

Ex 3.

- ▶ Given a string check if it's a palindrome or not using Stack.
Ex: “a santa at nasa”, “a nut for a jar of tuna” are palindromes.

HINT: (create a char array)

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    string s = "Hello";
    char suite[10];
    for(int i=0; i<s.length(); i++){
        suite[i]=s[i];
        cout<<suite[i]<<" ";
    }
}
```

Homework:

- ▶ Implement a template class called **LargeStack** which can store elements of type T. The class will have the following members:

Stack<T> Smain, Saux;

- ▶ **Smain** is the main stack which allows storing the values added in the LargeStack. Saux is an auxiliary stack which must be empty before and after the call of any function from the class LargeStack (Saux is used only for internal operations)
- ▶ The class LargeStack has the following methods:
 - **void push(T x):** add the element x at the top of the Smain stack.
 - **T pop() :** deletes and returns the element situated at the top of the Smain stack
 - **void swap(int i, int j):** changes the values from the levels i and j of the Smain stack (The levels are numbered starting from 0.

HINT: You can use the auxiliary stack Saux in order to store temporarily the elements of the Smain stack. You can use all the methods from the Stack class (pop, push, isEmpty etc.) for the variables Smain and Saux.