Lab Manual for Data Structures

Lab-3

Array based Stack implementation

Table of Contents

1. Introduction 29
1.1 Stack implementation as a static array 29
1.2 Stack implementation as a dynamic array 29
1.3 Stack as a class template 29
1.4Relevant Lecture Readings 29

2. Activity Time boxing 30

3. Objectives of the experiment 30

4. Concept Map 30
4.1 stack class using a static array 30
4.2Stacks using dynamic arrays 32
4.3Stack using a class template 34

5. Homework before Lab 35
5.1 Problem Solution Modeling 35
5.2Problem description: 35
5.3Practices from home 36
5.3.1 Task-1 36
5.3.2 Task-2 36

6. Procedure& Tools 36
6.1 Tools 36
6.2 Walk through Tasks [Expected time = 20mins] 36

7. Practice Tasks 38
7.1 Practice Task 1 [Expected time = 15mins] 38
7.2 Practice Task 2 [Expected time = 20mins + 20mins + 15mins] 38
7.4Out comes 39
7.5Testing 39

8.Evaluation Task (Unseen) [Expected time = 60mins] 40

9. Evaluation criteria 40

10. Further Readings 41
10.1 Web sites related to C++ tutorials related to stacks 41
10.2 Web sites containing supporting material 41

Lab 3: Array based Stack implementation with template class

1. Introduction

Objective of this lab is to introduce the implementation of stack in C++ and further introduce concept of class templates and defining a stack as a class template or generic class. A stack allows last in first out (LIFO) operations. We can place a new element at top of stack and we may remove an element from top of stack. Stacks can be implemented using static as well as dynamic arrays. Further we may implement a stack as a generic class/class template.

1.1 Stack implementation as a static array

A stack can be implemented using a static array in C++. We need to define an array with a maximum size and a variable of integer type normally called "top" for such stack. Stack is initially set to empty when program is executed, so at start of program top is set to -1 which represents an empty stack. There are two operations associated with a stack; push operation—used to add an element at top of stack and pop operation – to remove an element from top of stack. When push operation is performed we need to check that whether stack exceeds its maximum size or not, this is called stack overflow condition check. When pop operation is performed we need to test that whether stack is empty or not, this is called stack underflow condition check.

1.2 Stack implementation as a dynamic array

When maximum size of stack is to be decided at program execution time, then we need a dynamic array to represent the stack. Size of such array is specified at program execution time, a pointer is defined in program which points to an array dynamically created. Even the stack with dynamic array is implemented for stack overflow and underflow conditions in push and pop operations respectively.

1.3 Stack as a class template

A class template is a generic class which is defined as a template, from which multiple different classes can be generated. If we need to design a stack which can be used for integer, float, string and other primitive data types, we can define this stack as a class template. From this class template we may generate different stack classes which can be used to represent integers, floats, strings and other primitive data types as well for user defined data types.

1.4Relevant Lecture Readings

a) Revise Lecture No. 7 and 8 available at \\fs\lectures$ in instructor's folder.
b) From books: C++ Data Structures by Nell Dale (Page 196-225) and Data structures using C++ by D. S. Malik (Page 405-411).

2. Activity Time boxing

Table 1: Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5.1	Design Evaluation	20mins	20mins
6.2	Walk through tasks	20mins	20mins
7	Practice tasks	15 mins for task 1, 20 mins for task 2 and 35 mins for task 3	70mins
9	Evaluation Task	60mins for all assigned tasks	60mins

3. Objectives of the experiment

- To get knowledge of implementing stacks using static arrays in C++.
- To get knowledge of implementing stacks using dynamic arrays in C++.
- To get knowledge of implementing stacks as stack class template in C++.

4. Concept Map

This concept map will help students to understand the main concepts of topic covered in lab.

4.1 stack class using a static array

Following is an implementation of stack using static array in C++.

```
#include <iostream>
using namespace std;

const int STACKSIZE = 10;
class stack{
private:
int arr[STACKSIZE];
int tos;
public:
stack();
void push(int x);
int  pop();
bool empty();
bool full();
int  size();
void display();
        };

stack::stack()
        {
tos = -1;
```

```
        }

void stack::push(int x)
        {
if(!full())
arr[++tos] = x;
else
cout<< "Stack is full, Cannot push " << x <<endl;
        }

int stack::pop()
        {
if(!empty())
return arr[tos--];
else
cout<< "Stack is empty, cannot pop" <<endl;
return -1;
        }

bool stack::empty()
        {
if(tos == -1)
return true;
else
return false;
        }

bool stack::full()
        {
if(tos+1 == STACKSIZE)
return true;
else
return false;
        }

int stack::size()
        {
return tos;
        }

void stack::display()
        {
if(tos == -1)
                {
cout<< "No elements to display" <<endl;
return;
                }
```

```
for(int i=0;i < STACKSIZE; i++)
cout<<arr[i] << " ";
cout<<endl;
        }

int main()
        {
stackmystack;
cout<<mystack.size() <<endl;
mystack.push(1);
if(mystack.full())
cout<< "stack is full" <<endl;
mystack.pop();
if(mystack.empty())
cout<< "stack is empty" <<endl;
        }
```

Following figure 1 represents the functionality of pop and push operations on a stack implemented using a static array.

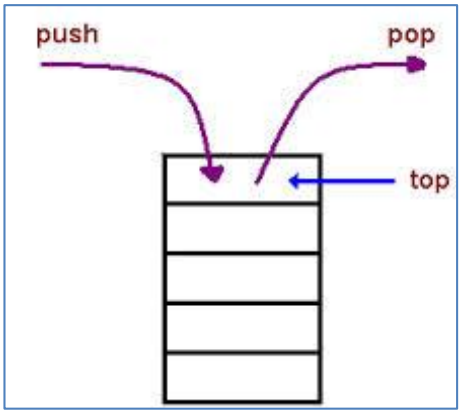

Figure 1: push and pop operations on a stack

4.2Stacks using dynamic arrays

A stack which is implemented using an array whose size is determined at program execution time is implemented using a dynamic array.

For the array implementation, we need to keep track of (at least) the array contents and a top index. Definition of stack class will be as:

```
class Stack{
public:
Stack(int stack_size);
~Stack();
void push(int x);
```

```
int pop();
bool isEmpty();
bool isFull();

private:
char *contents;
int top;
};

Stack::Stack(int stack_size)
{

contents = new char[stack_size];

if (contents == NULL) {
cout<< "Insufficient memory to initialize stack.\n";
exit(1);
  }

maxSize = stack_size;
top = -1;
}

Stack::~Stack()
{
delete [] contents;

contents = NULL;
maxSize = 0;
top = -1;
}

bool Stack::isEmpty() const
{
return top < 0;
}

void Stack::push(char element)
{
if (isFull()) {
cout<< "Can't push element on stack: stack is full.\n";
exit(1);
}

contents[++top] = element;
}
```

```
char Stack::pop()
{
if(!isEmpty())
return contents[--top];
else
cout<< "Stack is empty, cannot pop" <<endl;
return -1;
}
```

4.3Stack using a class template

A stack using a class template provides a generic implementation of stack which provides facility to use stack with many primitive and user defined data types. Following is an implementation of stack using templates.

```
template <class Typ, intMaxStack>
class Stack {
int EmptyStack;
Typ items[MaxStack];
int top;
public:
Stack();
~Stack();
void push(Typ);
Typ pop();
int empty();
int full();
};

Template < class Typ, intMaxStack>
Stack <Typ, MaxStack>::Stack() {
EmptyStack = -1;
top = EmptyStack;
}

template< class Typ, intMaxStack>
Stack<Typ, MaxStack>::~Stack()
{  delete[] items; }

template< class Typ, intMaxStack>
void Stack<Typ, MaxStack>::push(Typ c)
{  items[ ++top ] = c; }

template< class Typ, intMaxStack>
Typ Stack<Typ, MaxStack>::pop()
{  return items[ top-- ]; }

template< class Typ, intMaxStack>
```

```
int Stack<Typ, MaxStack>::full()
{  return top + 1 == MaxStack; }

template< class Typ, intMaxStack>
int Stack<Typ, MaxStack>::empty()
{  return top == EmptyStack; }

int main() {
   Stack<char, 10> s; // 10 chars
Char ch;
while ((ch = cin.get()) != '\n')
if (!s.full()) s.push(ch);
while (!s.empty())
cout<<s.pop();
cout<<endl;
   Stack<double, 4> ds; // 4 doubles
double d[] =
      {1.0, 3.0, 5.0, 7.0, 9.0, 0.0};
int i = 0;
while (d[i] != 0.0 && !ds.full())
if (!ds.full()) ds.push(d[i++]);
while (!ds.empty())
cout<<ds.pop() << "  ";
cout<<endl;
return 0;
}
```

5. Homework before Lab

This homework will help students to study the concepts of topic before start of lab.

5.1 Problem Solution Modeling

After studying the introduction and concept map sections you should be ready to provide the solution of following problems. Design the solutions of the problems in C++ and bring your code to lab so that lab instructor should assess and grade it.

5.2Problem description:

Write a program to implement a stack to store the objects of "person" class. "person" class should contain attributes (privately defined) per_ssn (string), per_name (string), per_age (int). "person" class should also contain member functions (publicly defined); constructor, input and output functions. You are required to design two stack class, one should use static array and other one should use dynamic array implementation. Your stack classes should contain constructors, destructors (if required), push and pop operations and functions to check empty or full stack.

5.3Practices from home

5.3.1 Task-1

Compare implementations of stack with static array and dynamic array and provide at least three differences between these two implementations.

5.3.2 Task-2

List three advantages of using stack as a class template.

6. Procedure& Tools

This section provides information about tools and programming procedures used for the lab.

6.1 Tools

Microsoft Visual Studio 2017 with Visual C++ compiler configured.

6.2 Walk through Tasks [Expected time = 20mins]

We will discuss the demonstration of stack using static array in Microsoft Visual Studio 2017. You are required to open an empty Win32 console application. You have already knowledge of creating a console application in Microsoft Visual Studio 2017 in your courses: computer programming and object oriented programming, which you have previously studied.

Following figure 2 represents the implementation of class in editor window.

```
#include <iostream>
using namespace std;

const int max=10;
class stack{
private:
    int arr[max];
    int top;
public:
    stack(){  top=-1;//empty initialy stack }

    void push(int i){
         top++;
         if (top<max)
            {   arr[top]=i;    }
         else
            {
             cout<<"stack full"<<endl;
             top--;
            }
         }
```

Figure 2: Stack definition using static array in Microsoft Visual Studio 2017.

Figure 3 shows push and pop operations implementation.

```
    int pop(){
         if (top==-1)
            cout<<"stack is emphty";
         else{
              int data=arr[top];
              top--;
              return data;
             }
         }

     bool empty(){
              return (top==-1);
         }
     };

int main(){
    stack a;
    a.push(12);a.push(30);a.push(23);a.push(42);a.push(100);
    while (!a.empty()){
        cout<<a.pop();
    }
    getche();
    return 0;
}
```

Figure 3: Implementation of push and pop operations for stack using static array in Microsoft Visual Studio 2017

Figure 4 represents the output of this program when executed.

Figure 4: Output window displaying values from stack in Microsoft Visual Studio 2017

7. Practice Tasks

This section will provide information about the practice tasks which are required to be performed in lab session. Design solutions of problems discussed in each task and placesolution code in a folder specified by your lab instructor.

Lab instructors are guided to help students learn how ACM problems work and provide students with certain practice/home tasks on the pattern of ACM Problems.

7.1 Practice Task 1 [Expected time = 15mins]

Write a program to implement a Stack using dynamic array of Character values. Implement the following functions in Stack with necessary checks:

a) Insert value in Stack (PUSH)
b) Delete value from Stack (POP)
c) Display the Stack.

7.2 Practice Task 2 [Expected time = 20mins]

Create a Stack of integer using arrays. Give the following menu to the user and then implement the given operations:

Press 1 to PUSH a value in Stack
Press 2 to POP a value from Stack
Press 3 to display the Stack
Press 4 to Show the next greater element in the Stack.

Sample Input:

Stack

3	2	6	7	8	13	

Press 1 to PUSH a value in Stack
Press 2 to POP a value from Stack
Press 3 to display the Stack
Press 4 to Show the next greater element in the Stack.

4

Sample Output:

The next greater elements for the given Stack are:
3→6
2→6
6→7
7→8
8→13
13→-1

7.3 Practice Task 3 [Expected time = 20mins + 20mins]

Use stack to solve the following problems:

a) Find the Minimum number of bracket reversals needed to make an expression balanced. A balanced expression is an expression in which all opening brackets have respective closing brackets. For example **2+{3*(7-2)}** is a balanced expression where as **2+{3*(7-2}**is not a balanced expression. Write a program to ask user to enter an expression. Expression must consists of only brackets { or }, but the expression may not be balanced. Find minimum number of bracket reversals to make the expression balanced.

b) Delete consecutive same characters in a sequence using stack. Ask user to enter a series of characters until user enters 'F' or 'f'. Now the task is to check if any two similar characters come together then they destroy each other. Print the new sequence after this pair wise destruction.

7.4Out comes

After completing this lab, student will be able to understand and develop programs related stacks using static and dynamic arrays and using stack as class template in C++ using Microsoft Visual Studio 2017 environment.

7.5Testing

Test Cases for Practice Task-1

Sample Inputs-1	Sample Outputs-1
Push elements for stack:	
2	
3	
4	
5	
8	
Enter number of elements to be popped from stack1: 3	
	Elements of stack popped:

	8
	5
	4

Test Cases for Practice Task-3 a

Sample Inputs-1	Sample Outputs-1
Enter expression:	
{{{	
	Can't be made balanced using reversals
Enter expression:	
{{{{	
	2
Enter expression:	
{{{{}}	
	1

Test Cases for Practice Task-3 b

Sample Inputs-1	Sample Outputs-1
Input : abaabcdabf	
	new sequence is: acdab

8.Evaluation Task (Unseen) [Expected time = 60mins]

The lab instructor will give you unseen task depending upon the progress of the class.

9. Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 2: Evaluation of the Lab

Sr. No.	Task No	Description	Marks
1	4	Problem Modeling	20
2	6	Procedures and Tools	10
3	7,8	Practice tasks and Testing	35
4	8.1	Evaluation Tasks (Unseen)	20
5		Comments	5
6		Good Programming Practices	10

10. Further Readings

10.1 Web sites related to C++ tutorials related to stacks

1. http://www.cplusplus.com/reference/stack/stack/
2. http://www.cppforschool.com/tutorial/dynamic-stack.html

10.2 Web sites containing supporting material

1. http://www.slideshare.net/nita23arora/stacks-queues-presentation