**Department of Computing**

**School of Electrical Engineering and Computer Science**

**CS250 – Data Structures and Algorithms**



**Lab 6: Implementation of Stacks in Different Problems**

**Submission Details**

|  |  |
| --- | --- |
| Name | CMS ID |
| Muhammad Umer | 345834 |
| Group | GP – 1 |
| Lab Engineer | Anum Asif |
| Faculty Member | Bostan Khan |
| Class | BEE12 |
| Date | 11/03/2024 |
| Time | 10:00 am – 12:50 pm |

**Table of Contents**

[2 Implementation of Stacks in Different Problems 3](#_Toc161015435)

[2.1 Introduction 3](#_Toc161015436)

[2.2 Objectives 3](#_Toc161015437)

[2.3 Tools/Software Requirement 3](#_Toc161015438)

[2.4 Description 3](#_Toc161015439)

[2.5 Deliverables 3](#_Toc161015440)

[3 Lab Task 4](#_Toc161015441)

[4 Conclusion 7](#_Toc161015442)

# Implementation of Stacks in Different Problems

## Introduction

This lab consists of stacks implementation and some of its applications.

## Objectives

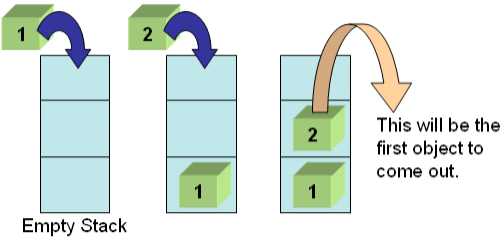
The objective of this lab is to enable students to build stack ADT using linked list and arrays, perform the following tasks on it and analyze the performance of each implementation.

## Tools/Software Requirement

* Visual Studio C++

## Description

A stack meant to mimic the information storage and retrieval in LIFO (Last In First Out) order.



**Stack Operations**

1. void push(element) – pushes an element on the top of stack
2. element pop() – removes and display the element on the top of stack
3. bool isEmpty() – checks if the stack is empty or not
4. bool isFull() – checks if the stack is full or not
5. void clear() – release the memory allocated by stack
6. void peek() – display the contents of the top element of stack

## Deliverables

Compile a single word document by filling in the solution parts and submit this file on LMS. The name of word document should follow this format. i.e., YourFullName(reg)\_Lab#. You must show the implementation of the tasks in a complete manner to get your work graded.

# Lab Task

The idea is rather simple: You keep a Stack of braces, and every time you encounter an open brace, you push it into your stack. Every time you encounter a close brace, you pop the top element from your stack. At the end, you check your stack for being empty. If so, indeed your input string contained balanced braces. Otherwise, it didn't.

**Expected Input**

1 + 2 \* (3 / 4)

1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10) – 11 + (12\*8)] + 14

1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10)} – 11 + (12\*8) / {13 +13}] + 14

Your program will determine whether the open brackets (the square brackets, curly braces and the parentheses) are closed in the correct order.

**Expected Output**

This expression is correct.

This expression is NOT correct e.g. error at character # 10. ‘{‘- not closed.

This expression is correct.

Your program should be able to take generic input expressions from user.

Solve the above problem using an **array-based stack**.

Code

*/\**

*C++ Program to implement Circular Doubly Linked List*

*See unsolved code from the lab manual.*

*\*/*

#include <iostream>

#include <string>

using *namespace* std;

*struct* BracketInfo {*// to be able to correctly print the position of the error*

*char* bracket;

*int* position;

};

*class* Stack {

*private:*

*int* top;

*int* capacity;

    BracketInfo\* stack;

*public:*

    Stack(*int* *size*) {

        capacity = *size*;

        stack = new BracketInfo[capacity];

        top = -1;

    }

    ~Stack() { delete[] stack; }

*void* push(BracketInfo *elem*) {

        if (top == capacity - 1) {

            cout << "Stack overflow" << endl;

            return;

        }

        stack[++top] = *elem*;

    }

    BracketInfo pop() {

        if (top == -1) {

            cout << "Stack underflow" << endl;

            return {'\0', -1};

        }

        return stack[top--];

    }

    BracketInfo peek() {

        if (top == -1) {

            cout << "Stack is empty" << endl;

            return {'\0', -1};

        }

        return stack[top];

    }

*bool* is\_empty() { return top == -1; }

*bool* is\_full() { return top == capacity - 1; }

*void* clear() { top = -1; }

};

*bool* check\_expr(string *expr*) {

    BracketInfo ch;

    Stack stack(*expr*.length());

    for (*int* i = 0; i < *expr*.length(); i++) {

        if (*expr*[i] == '(' || *expr*[i] == '[' || *expr*[i] == '{') {

            stack.push({*expr*[i], i});

        } else if (*expr*[i] == ')' || *expr*[i] == ']' || *expr*[i] == '}') {

            if (stack.is\_empty()) {

                cout << "This expression is NOT correct at character # " << i

                     << ". `" << *expr*[i] << "`- not opened." << endl;

                return false;

            } else {

                ch = stack.pop();

                if ((*expr*[i] == ')' && ch.bracket != '(') ||

                    (*expr*[i] == ']' && ch.bracket != '[') ||

                    (*expr*[i] == '}' && ch.bracket != '{')) {

                    cout << "This expression is NOT correct at character # "

                         << ch.position << ". `" << ch.bracket

                         << "`- not closed." << endl;

                    return false;

                }

            }

        }

    }

    if (!stack.is\_empty()) {

        ch = stack.peek();

        cout << "This expression is NOT correct. `" << ch.bracket

             << "` at character # " << ch.position << " - not closed." << endl;

        return false;

    }

    return true;

}

*int* main() {

    string expr;

    cout << "Enter an expression: ";

    getline(cin, expr);

    if (check\_expr(expr)) {

        cout << "This expression is correct" << endl;

    }

    return 0;

}

Output

root@Zonularity:/home/zonularity/dsa/lab\_6# ./task

Enter an expression: 1 + 2 \* (3 / 4)

This expression is correct

root@Zonularity:/home/zonularity/dsa/lab\_6# ./task

Enter an expression: 1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10) – 11 + (12\*8)] + 14

This expression is NOT correct at character # 17. `{`- not closed.

root@Zonularity:/home/zonularity/dsa/lab\_6# ./task

Enter an expression: 1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10)} – 11 + (12\*8) / {13 +13}] + 14

This expression is correct

root@Zonularity:/home/zonularity/dsa/lab\_6#

# Conclusion

In this lab, we successfully implemented a stack ADT using arrays. This data structure effectively followed the LIFO (Last In, First Out) principle, allowing for element addition (push) and removal (pop) from the top of the stack. We further utilized the stack to evaluate mathematical expressions, performing a basic check for balanced parentheses. This hands-on experience solidified our understanding of stacks and their applications in expression manipulation.