Data Structures And Algorithms
Lab Task 5

SUBMITTED BY:

Hasaan Ahmad SP22-BSE-017

SUBMITTED TO: Sir Syed Ahmad Qasim

Code:

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

class Node
{
public:
    char data;
    Node *next;
    Node(char data)
    {
        this->data = data;
        next = NULL;
    }
};
class Stack
{
public:
    Node *head;
    Stack()
    {
        head = NULL;
    }
    void push(char data)
    {
        Node *newNode = new Node(data);
        if (head == NULL)
        {
            head = newNode;
        }
        else
        {
            newNode->next = head;
            head = newNode;
        }
    }
    void pop()
    {
        if (head == NULL)
        {
            cout << "Stack is empty" << endl;
        }
```

```
        else
        {
            Node *temp = head;
            head = head->next;
            delete temp;
        }
    }
    void display()
    {
        Node *temp = head;
        while (temp != NULL)
        {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
    bool isEmpty()
    {
        if (head == NULL)
            return true;
        else
            return false;
    }
    char peek()
    {
        if (head == NULL)
        {
            cout << "Stack is empty" << endl;
            return 0;
        }
        else
        {
            return head->data;
        }
    }

    char top()
    {
        return head->data;
    }

    // Graded Lab 1
```

```
    // Write an application using Stack that checks whether the entered
string of brackets is balanced, e.g.
    // [{( )}] is Balanced while {[( )] } is not Balanced because the
priority of the pranthesis is not maintained.

    bool isBalanced(string expression)
    {
        char character;
        char character1;
        char character2;
        bool flag;
        int count = 0;
        for (int i = 0; i < expression.length(); i++)
        {
            if (expression[i] == '(' || expression[i] == '{' ||
expression[i] == '[')
            {
                push(expression[i]);
                continue;
            }
            if (head == NULL)
            {
                flag = true;
            }
            if (count == 0)
            {
                character2 = top();
            }
            character1 = top();
            if (count == 1)
            {
                if (character2 == '[' && (character1 == '(' || character1
== '{'))
                {
                    flag = false;
                    break;
                }
                else if (character2 == '{' && character1 == '(')
                {
                    flag = false;
                    break;
                }
                else
```

```
                {
                    flag = true;
                }
                count = 0;
            }
            switch (expression[i])
            {
            case ')':
                character = top();
                if (character == '{' || character == '[')
                {
                    flag = false;
                }
                pop();
                break;
            case '}':
                character = top();
                if (character == '(' || character == '[')
                {
                    flag = false;
                }
                pop();
                break;
            case ']':
                character = top();
                if (character == '{' || character == '(')
                {
                    flag = false;
                }
                pop();
                break;
            }
            count++;
            character2 = character1;
        }
        return flag;
    }
};
// Graded 2
// Use dynamic stack and implement Infix to Postfix conversion algorithm
and test it for various inputs.
string infixToPostfix(string expression)
{
```

```
    Stack s;
    string postfix = "";
    for (int i = 0; i < expression.length(); i++)
    {
        if (expression[i] == ' ' || expression[i] == ',')
        {
            continue;
        }
        else if (expression[i] >= '0' && expression[i] <= '9')
        {
            postfix += expression[i];
        }
        else if (expression[i] == '(')
        {
            s.push(expression[i]);
        }
        else if (expression[i] == ')')
        {
            while (s.top() != '(')
            {
                postfix += s.top();
                s.pop();
            }
            s.pop();
        }
        else
        {
            while (!s.isEmpty() && s.top() != '(' && expression[i] <=
s.top())
            {
                postfix += s.top();
                s.pop();
            }
            s.push(expression[i]);
        }
    }
    while (!s.isEmpty())
    {
        postfix += s.top();
        s.pop();
    }
    return postfix;
}
```

```
// Graded 3
// For a given postfix expression, use dynamic stack to evaluate a
numerical result for given values of variables.

int evaluatePostfix(string expression)
{
    Stack s;
    for (int i = 0; i < expression.length(); i++)
    {
        if (expression[i] >= '0' && expression[i] <= '9')
        {
            s.push(expression[i] - '0');
        }
        else
        {
            int val1 = s.top();
            s.pop();
            int val2 = s.top();
            s.pop();
            switch (expression[i])
            {
            case '+':
                s.push(val2 + val1);
                break;
            case '-':
                s.push(val2 - val1);
                break;
            case '*':
                s.push(val2 * val1);
                break;
            case '/':
                s.push(val2 / val1);
                break;
            }
        }
    }
    return s.top();
}

int main()
{
    // Testing graded 1
    Stack s;
```

```
    string expression = "{[( )]}";
    if (s.isBalanced(expression))
    {
        cout << "Balanced" << endl;
    }
    else
    {
        cout << "Not Balanced" << endl;
    }
    // More tests for Graded 1
    string expression1 = "(){([])}";
    if (s.isBalanced(expression1))
    {
        cout << "Balanced" << endl;
    }
    else
    {
        cout << "Not Balanced" << endl;
    }
    string expression2 = "({)}";
    if (s.isBalanced(expression2))
    {
        cout << "Balanced" << endl;
    }
    else
    {
        cout << "Not Balanced" << endl;
    }
    cout << endl;
    // Testing graded 2
    string infix = "a+b*(d-e)+(f+g*h)-i";
    cout << infixToPostfix(infix) << endl;
    // More Tests For Graded 2
    string infix1 = "a-b*c-(d+e)";
    cout << infixToPostfix(infix1) << endl;

    cout << endl;
    // Testing graded 3
    string postfix = "231*+9-";
    cout << evaluatePostfix(postfix) << endl;
    // More Test for Graded 3
    string postfix1 = "123*+";
    cout << evaluatePostfix(postfix1) << endl;
```

```
    return 0;
}
```

Output:

```
Not Balanced
Not Balanced
Not Balanced

ab+de-fg+h*i-+*
ab-cde+-*

-4
7
```