**PRACTICAL ASSIGNMENT NO::3**

**GITHUB:https**[**Aditya6768/DSAL: all LAB work 2024-25 (github.com)**](https://github.com/Aditya6768/DSAL)

**Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix**

**expression to postfix, prefix and evaluation of postfix and prefix expression.**

#include <iostream>

#include <cctype>

#include <cstdlib>

#include <string>

#include <algorithm>

using namespace std;

// Node for singly linked list stack

struct Node {

char data;

Node\* next;

};

// Stack class

class Stack {

private:

Node\* topNode;

public:

Stack() : topNode(NULL) {}

// Push an element onto the stack

void push(char value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = topNode;

topNode = newNode;

}

// Pop an element from the stack

char pop() {

if (isEmpty()) {

cerr << "Stack underflow" << endl;

exit(EXIT\_FAILURE);

}

char value = topNode->data;

Node\* temp = topNode;

topNode = topNode->next;

delete temp;

return value;

}

// Peek at the top element

char top() const {

if (isEmpty()) {

cerr << "Stack is empty" << endl;

exit(EXIT\_FAILURE);

}

return topNode->data;

}

// Check if the stack is empty

bool isEmpty() const {

return topNode == NULL;

}

// Destructor to clear the stack

~Stack() {

while (!isEmpty()) {

pop();

}

}

};

// Utility functions for infix to postfix and prefix conversion

int precedence(char op) {

switch (op) {

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^': return 3;

default: return 0;

}

}

string infixToPostfix(const string& infix) {

Stack stack;

string postfix;

for (size\_t i = 0; i < infix.size(); ++i) {

char ch = infix[i];

if (isalnum(ch)) {

postfix += ch;

} else if (ch == '(') {

stack.push(ch);

} else if (ch == ')') {

while (!stack.isEmpty() && stack.top() != '(') {

postfix += stack.pop();

}

stack.pop(); // Remove '('

} else {

while (!stack.isEmpty() && precedence(stack.top()) >= precedence(ch)) {

postfix += stack.pop();

}

stack.push(ch);

}

}

while (!stack.isEmpty()) {

postfix += stack.pop();

}

return postfix;

}

string infixToPrefix(const string& infix) {

string reversedInfix = infix;

reverse(reversedInfix.begin(), reversedInfix.end());

for (size\_t i = 0; i < reversedInfix.size(); ++i) {

if (reversedInfix[i] == '(') {

reversedInfix[i] = ')';

} else if (reversedInfix[i] == ')') {

reversedInfix[i] = '(';

}

}

string reversedPostfix = infixToPostfix(reversedInfix);

reverse(reversedPostfix.begin(), reversedPostfix.end());

return reversedPostfix;

}

int evaluatePostfix(const string& postfix) {

Stack stack;

for (size\_t i = 0; i < postfix.size(); ++i) {

char ch = postfix[i];

if (isdigit(ch)) {

stack.push(ch - '0');

} else {

int right = stack.pop();

int left = stack.pop();

switch (ch) {

case '+': stack.push(left + right); break;

case '-': stack.push(left - right); break;

case '\*': stack.push(left \* right); break;

case '/': stack.push(left / right); break;

}

}

}

return stack.top();

}

int evaluatePrefix(const string& prefix) {

Stack stack;

for (size\_t i = prefix.size(); i-- > 0; ) {

char ch = prefix[i];

if (isdigit(ch)) {

stack.push(ch - '0');

} else {

int left = stack.pop();

int right = stack.pop();

switch (ch) {

case '+': stack.push(left + right); break;

case '-': stack.push(left - right); break;

case '\*': stack.push(left \* right); break;

case '/': stack.push(left / right); break;

}

}

}

return stack.top();

}

void menu() {

string infix;

int choice;

while (true) {

cout << "\nMenu:\n";

cout << "1. Infix to postfix and evaluate postfix\n";

cout << "2. Infix to prefix and evaluate prefix\n";

cout << "3. Quit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter infix expression: ";

cin >> infix;

{

string postfix = infixToPostfix(infix);

cout << "Postfix: " << postfix << endl;

cout << "Postfix evaluation: " << evaluatePostfix(postfix) << endl;

}

break;

case 2:

cout << "Enter infix expression: ";

cin >> infix;

{

string prefix = infixToPrefix(infix);

cout << "Prefix: " << prefix << endl;

cout << "Prefix evaluation: " << evaluatePrefix(prefix) << endl;

}

break;

case 3:

cout << "Exiting..." << endl;

return; // Exit the menu function and end the program

default:

cout << "Invalid choice. Please try again." << endl;

}

}

}

int main() {

menu(); // Start the menu loop

return 0;

}

