/\*

ASSIGNMENT NO. 2

NAME- ABRAR SHAIKH ROLL NO. - 23570

TOPIC- Stack using linked-list

\*/

#include <iostream>

using namespace std;

// Node class for the linked list

class Node {

public:

int data;

Node\* next;

Node(int data) {

this->data = data;

this->next = NULL;

}

};

// Stack ADT using singly linked list

class Stack {

private:

Node\* top; // Pointer to the top of the stack

public:

Stack() {

top = NULL;

}

// Push an element to the stack

void push(int value) {

Node\* newNode = new Node(value);

if (top == NULL) {

top = newNode;

} else {

newNode->next = top;

top = newNode;

}

}

// Pop an element from the stack

int pop() {

if (top == NULL) {

cout << "Stack Underflow!" << endl;

return -1; // returning an error value

} else {

Node\* temp = top;

top = top->next;

int poppedValue = temp->data;

delete temp;

return poppedValue;

}

}

// Peek at the top element of the stack

int peek() {

if (top == NULL) {

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

return -1;

}

return top->data;

}

// Check if the stack is empty

bool isEmpty() {

return top == NULL;

}

};

// Function to check if the character is an operator

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

// Function to get precedence of operators

int precedence(char c) {

if (c == '^')

return 3;

else if (c == '\*' || c == '/')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return -1;

}

// Function to convert infix expression to postfix

void infixToPostfix(char infix[], char postfix[]) {

Stack s;

int j = 0;

for (int i = 0; infix[i] != '\0'; i++) {

char c = infix[i];

// If the character is an operand, add it to the output

if (c >= '0' && c <= '9') {

postfix[j++] = c;

}

// If the character is '(', push it to the stack

else if (c == '(') {

s.push(c);

}

// If the character is ')', pop and output until '(' is encountered

else if (c == ')') {

while (s.peek() != '(') {

postfix[j++] = s.pop();

}

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

}

// If the character is an operator

else if (isOperator(c)) {

while (!s.isEmpty() && precedence(c) <= precedence(s.peek())) {

postfix[j++] = s.pop();

}

s.push(c);

}

}

// Pop all the operators from the stack

while (!s.isEmpty()) {

postfix[j++] = s.pop();

}

postfix[j] = '\0'; // Null terminate the postfix expression

}

// Function to convert infix expression to prefix

void infixToPrefix(char infix[], char prefix[]) {

Stack s;

char reversedInfix[100], reversedPostfix[100];

// Reverse the infix expression

int length = 0;

for (int i = 0; infix[i] != '\0'; i++) {

length++;

}

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

if (infix[length - i - 1] == '(')

reversedInfix[i] = ')';

else if (infix[length - i - 1] == ')')

reversedInfix[i] = '(';

else

reversedInfix[i] = infix[length - i - 1];

}

reversedInfix[length] = '\0';

// Convert the reversed infix to postfix

infixToPostfix(reversedInfix, reversedPostfix);

// Reverse the postfix to get the prefix

for (int i = 0; reversedPostfix[i] != '\0'; i++) {

prefix[i] = reversedPostfix[length - i - 1];

}

prefix[length] = '\0'; // Null terminate the prefix expression

}

// Function to evaluate a postfix expression

int evaluatePostfix(char postfix[]) {

Stack s;

for (int i = 0; postfix[i] != '\0'; i++) {

char c = postfix[i];

// If the character is an operand, push it to the stack

if (c >= '0' && c <= '9') {

s.push(c - '0');

}

// If the character is an operator

else if (isOperator(c)) {

int val1 = s.pop();

int val2 = s.pop();

switch (c) {

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.pop();

}

//function for prefix evaluation

int evaluatePrefix(char prefix[]) {

Stack s;

int length = 0;

// Find the length of the prefix expression

for (int i = 0; prefix[i] != '\0'; i++) {

length++;

}

// Traverse the prefix expression from right to left

for (int i = length - 1; i >= 0; i--) {

char c = prefix[i];

// If the character is an operand, push it to the stack

if (c >= '0' && c <= '9') {

s.push(c - '0'); // Convert char to int

}

// If the character is an operator

else if (isOperator(c)) {

int val1 = s.pop();

int val2 = s.pop();

switch (c) {

case '+': s.push(val1 + val2); break;

case '-': s.push(val1 - val2); break;

case '\*': s.push(val1 \* val2); break;

case '/': s.push(val1 / val2); break;

}

}

}

// The final result will be the only element left in the stack

return s.pop();

}

int main() {

char infix[100], postfix[100], prefix[100];

cout << "Enter an infix expression: ";

cin >> infix;

infixToPostfix(infix, postfix);

infixToPrefix(infix, prefix);

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

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

char postfixEval[100];

cout << "Enter a postfix expression for evaluation: ";

cin >> postfixEval;

cout << "Postfix Evaluation Result: " << evaluatePostfix(postfixEval) << endl;

char prefixEval[100];

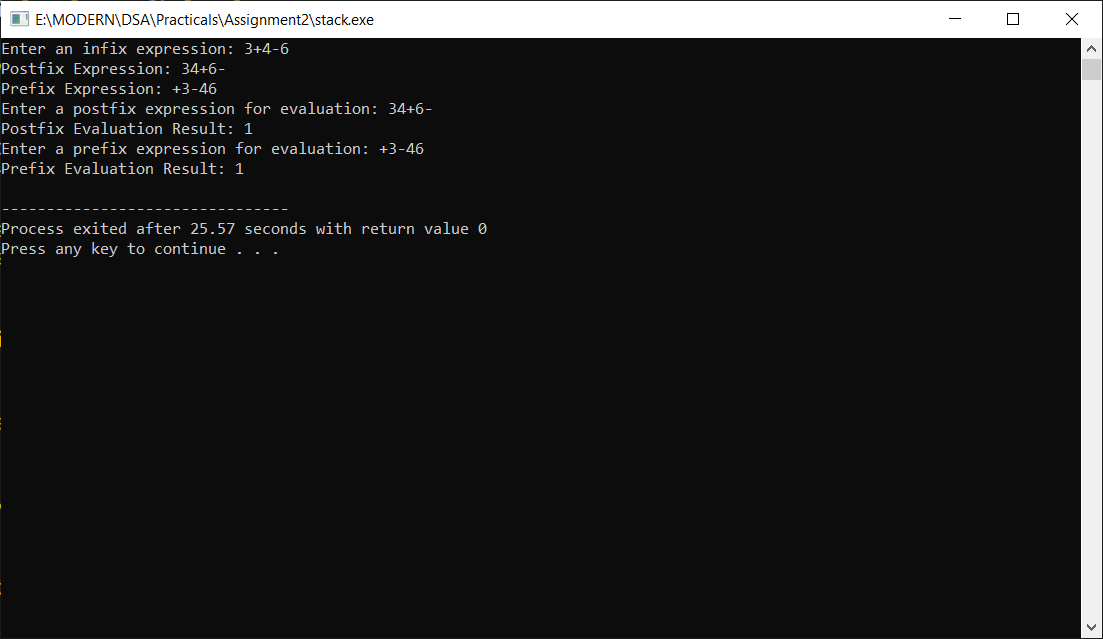
cout << "Enter a prefix expression for evaluation: ";

cin >> prefixEval;

cout << "Prefix Evaluation Result: " << evaluatePrefix(prefixEval) << endl;

return 0;

}



Github Repository - https://github.com/abssha/DSA.git