**Lab 06 Data Structures Lab**

**Date: 09 Sept 2024**

Write a C++ program for the following tasks:

Q1) Conversion of infix expression to postfix and prefix forms

**CODE:**

#include <iostream>

#include <cstring>

#include<algorithm>

using namespace std;

class Stack {

int top;

char arr[100];

public:

Stack() { top = -1; }

void push(char c) {

if (top >= 99) {

cout << “Stack Overflow\n”;

return;

}

arr[++top] = c;

}

char pop() {

if (top < 0) {

cout << “Stack Underflow\n”;

return ‘\0’;

}

return arr[top--];

}

char peek() {

if (top < 0) return ‘\0’;

return arr[top];

}

bool isEmpty() {

return top == -1;

}

};

bool isOperand(char c) {

return (c >= ‘a’ && c <= ‘z’) || (c >= ‘a’ && c <= ‘z’);

}

int precedence(char c) {

if (c == ‘^’) return 3;

if (c == ‘\*’ || c == ‘/’) return 2;

if (c == ‘+’ || c == ‘-‘) return 1;

return -1;

}

string inf\_to\_post(string infix) {

stack s;

string postfix = “”;

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

char c = infix[i];

if (isOperand(c)) {

postfix += c;

}

else if (c == ‘(‘) {

s.push(c);

}

else if (c == ‘)’) {

while (!s.isEmpty() && s.peek() != ‘(‘) {

postfix += s.pop();

}

s.pop(); // remove ‘(‘ from stack

}

else {

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

postfix += s.pop();

}

s.push(c);

}

}

// pop all remaining operators from the stack

while (!s.isEmpty()) {

postfix += s.pop();

}

return postfix;

}

string infixtoprefix(string infix) {

Stack s;

reverse(infix.begin(), infix.end()); // reverse infix expression

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

if (infix[i] == ‘(‘) infix[i] = ‘)’;

else if (infix[i] == ‘)’) infix[i] = ‘(‘;

}

string prefix = “”;

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

char c = infix[i];

if (isOperand(c)) {

prefix += c;

}

else if (c == ‘(‘) {

s.push(c);

}

else if (c == ‘)’) {

while (!s.isEmpty() && s.peek() != ‘(‘) {

prefix += s.pop();

}

s.pop();

}

else {

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

prefix += s.pop();

}

s.push(c);

}

}

while (!s.isEmpty()) {

prefix += s.pop();

}

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

return prefix;

}

int main() {

string infix;

cout << “enter infix expression: “;

cin >> infix;

cout << “postfix expression: “ << inf\_to\_post(infix) << endl;

cout << “prefix expression: “ << infixtoprefix(infix) << endl;

return 0;

}

**OUTPUT:**

Enter infix expression: a+b^c^d\*(e-f)

Postfix Expression: abc^d^ef-\*+

Prefix Expression: +a\*^^bcd-ef

Q2) Evaluation of postfix and prefix expressions

**CODE:**

#include <iostream>

#include <string.h>

Using namespace std;

#define MAX 100

class Stack {

int top;

double arr[MAX];

public:

Stack() : top(-1) {}

bool push(double x) {

if (top >= MAX – 1) {

cout << “Stack Overflow”;

return false;

}

arr[++top] = x;

return true;

}

double pop() {

if (top < 0) {

cout << “Stack Underflow”;

return 0;

}

return arr[top--];

}

bool isEmpty() { return (top < 0); }

};

double evaluatePostfix(const std::string& expr) {

Stack stack;

string num;

for (size\_t I = 0; I < expr.length(); i++) {

char c = expr[i];

if (c == ‘ ‘) {

continue;

}

if (isdigit(c)) {

num.clear();

while (i < expr.length() && isdigit(expr[i])) {

num += expr[i++];

}

i--;

stack.push(stod(num)); // Convert the string to number and push

} else {

double val2 = stack.pop();

double val1 = stack.pop();

switch (c) {

case ‘+’: stack.push(val1 + val2); break;

case ‘-‘: stack.push(val1 – val2); break;

case ‘\*’: stack.push(val1 \* val2); break;

case ‘/’: stack.push(val1 / val2); break;

}

}

}

return stack.pop();

}

double evaluatePrefix(const std::string& expr) {

Stack stack;

string num;

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

char c = expr[i];

if (c == ‘ ‘) {

continue;

}

if (isdigit(c)) {

num.clear();

while (i >= 0 && isdigit(expr[i])) {

num = expr[i--] + num;

}

i++;

stack.push(stod(num));

} else {

double val1 = stack.pop();

double val2 = stack.pop();

switch (c) {

case ‘+’: stack.push(val1 + val2); break;

case ‘-‘: stack.push(val1 – val2); break;

case ‘\*’: stack.push(val1 \* val2); break;

case ‘/’: stack.push(val1 / val2); break;

}

}

}

return stack.pop();

}

int main() {

string postfix = “100 200 + 2 / 5 \* 7 +”;

cout << “Postfix Evaluation of (“<<postfix<<”) : “<< evaluatePostfix(postfix) << endl;

string prefix = “+ \* 5 6 7”;

cout << “Prefix Evaluation of (“<<prefix<<”) : “<< evaluatePrefix(prefix) << endl;

return 0;

}

**OUTPUT:**

Postfix Evaluation of(100 200 + 2 / 5 \* 7 +): 757

Prefix Evaluation of(“+ \* 5 6 7): 37