|  |  |  |
| --- | --- | --- |
| Image result for latest marwadi university logo | **Marwadi University**  **Faculty of Technology**  **Department of Information and Communication Technology** | |
| **Subject: DSC  (01CT0308)** | Aim: Implementations of Infix to Postfix Transformation and of Infix to Prefix Transformation and their evaluation program. | |
| **Experiment No: 3** | **Date: 28- 10 - 2023** | **Enrolment No:-** 92200133030 |

**Experiment – 3**

**Objective:** Implementations of Infix to Postfix Transformation and of Infix to Prefix Transformation and their evaluation program.

**Code :-**

#include <iostream>

#include <stack>

#include <string>

#include <cctype>

#include <algorithm>

using namespace std;

bool isOperator(char c) {

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

}

int getPrecedence(char op) {

if (op == '+' || op == '-')

return 1;

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

return 2;

return 0;

}

string infixToPrefix(const string& infix) {

string reversedInfix = infix;

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

stack<char> operators;

string prefix;

for (char c : reversedInfix) {

if (isalnum(c)) {

prefix += c;

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

operators.push(c);

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

while (!operators.empty() && operators.top() != ')') {

prefix += operators.top();

operators.pop();

}

if (operators.empty()) {

cerr << "Unbalanced parentheses in the expression." << endl;

return ""; // Exit with an error

}

operators.pop(); // Pop ')'

} else if (isOperator(c)) {

while (!operators.empty() && getPrecedence(c) < getPrecedence(operators.top())) {

prefix += operators.top();

operators.pop();

}

operators.push(c);

}

}

while (!operators.empty()) {

if (operators.top() == '(') {

cerr << "Unbalanced parentheses in the expression." << endl;

return ""; // Exit with an error

}

prefix += operators.top();

operators.pop();

}

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

return prefix;

}

string infixToPostfix(const string& infix) {

stack<char> operators;

string postfix;

for (char c : infix) {

if (isalnum(c)) {

postfix += c;

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

operators.push(c);

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

while (!operators.empty() && operators.top() != '(') {

postfix += operators.top();

operators.pop();

}

operators.pop(); // Pop '('

} else if (isOperator(c)) {

while (!operators.empty() && getPrecedence(c) <= getPrecedence(operators.top())) {

postfix += operators.top();

operators.pop();

}

operators.push(c);

}

}

while (!operators.empty()) {

postfix += operators.top();

operators.pop();

}

return postfix;

}

int evaluatePostfix(const string& postfix) {

stack<int> operands;

for (char c : postfix) {

if (isdigit(c)) {

operands.push(c - '0');

} else {

int operand2 = operands.top();

operands.pop();

int operand1 = operands.top();

operands.pop();

switch (c) {

case '+':

operands.push(operand1 + operand2);

break;

case '-':

operands.push(operand1 - operand2);

break;

case '\*':

operands.push(operand1 \* operand2);

break;

case '/':

operands.push(operand1 / operand2);

break;

}

}

}

return operands.top();

}

int evaluatePrefix(const string& prefix) {

stack<int> operands;

for (char c : prefix) {

if (isdigit(c)) {

operands.push(c - '0');

} else {

int operand1 = operands.top();

operands.pop();

int operand2 = operands.top();

operands.pop();

switch (c) {

case '+':

operands.push(operand1 + operand2);

break;

case '-':

operands.push(operand1 - operand2);

break;

case '\*':

operands.push(operand1 \* operand2);

break;

case '/':

operands.push(operand1 / operand2);

break;

}

}

}

if (operands.size() != 1) {

cerr << "Invalid prefix expression." << endl;

return -1;

}

return operands.top();

}

void printPostfix(const string& postfix) {

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

}

void printPrefix(const string& prefix) {

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

}

int main() {

string infix\_expression = "5 + 6 \* (7 - 8) / 9";

string postfix\_expression = infixToPostfix(infix\_expression);

string prefix\_expression = infixToPrefix(infix\_expression);

printPostfix(postfix\_expression);

printPrefix(prefix\_expression);

int PrefixResult = evaluatePrefix(prefix\_expression);

int PostfixResult = evaluatePostfix(postfix\_expression);

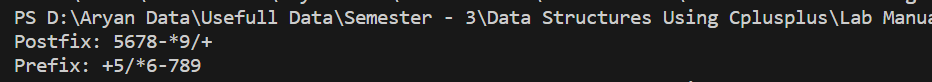
cout << "Prefix Result: " << PrefixResult << endl;

cout << "Postfix Result: " << PostfixResult << endl;

return 0;

}

**Output:**

****