10 - Intermediate code generation - Infix to Prefix and Postfix

Thursday, April 18, 2024 11:50 AM

Ashima Fatima Seik Mugibur Raghman, 21BAI1830

AIM: Write a program to generate prefix and postfix for the given infix expression

Sample Input:

a + b * c - (d / e + f * g * h)

Sample Output:

Postfix : abc * + de / fg * h * + -

Prefix : + a - * bc + / de * * f g h

Postfix Procedure:

1. Initialize an empty stack to hold the operators and a string to store the postfix expression.
2. Input the infix expression.
3. For each symbol in the infix expression:
   a. If the symbol is an operand, append it to the postfix expression
   b. If the symbol is an opening parenthesis '(', push it onto the stack
   c. If it is a closing parenthesis ')', pop operators from the stack and append them to the postfix expression until '(' is encountered.
   d. If it is an operator,
      - while the stack is not empty and the precedence of the operator at the top of the stack is ≥ precedence of current operator, pop the operator and append it to the postfix expression.
      - then, push the current operator onto the stack
   e. pop any remaining operators from the stack and

c. pop any remaining operators from the stack and append them to the postfix expression.

4. Print the postfix expression.

Prefix Procedure:

1. Reverse the infix expression
2. Replace '(' with ')' and vice versa in the reversed expression.
3. Apply the algorithm for infix to postfix on the modified expression.
4. Reverse the resulting postfix expression to obtain the prefix expression.

Sample Input and Output:

```
21BAI1830

Enter infix expression: a+b*c-(d/e+f*g*h)
Prefix expression: +a-*bc+/de*f*gh
Postfix expression: abc*+de/fg*h*+-
```

Code:

```
#include <iostream>
#include <stack>
#include <string>
#include <algorithm>
using namespace std;
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}
bool isOperand(char c) {
    return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');
}
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}
string infixToPrefix(string infix) {
    stack<char> operators;
    string prefix;
```

```
    reverse(infix.begin(), infix.end());
    for (char& c : infix) {
        if (c == '(') {
            c = ')';
        } else if (c == ')') {
            c = '(';
        }
    }
    for (char& c : infix) {
        if (isOperand(c)) {
            prefix += c;
        } else if (c == '(') {
            operators.push(c);
        } else if (c == ')') {
            while (!operators.empty() && operators.top() != '(') {
                prefix += operators.top();
                operators.pop();
            }
            operators.pop(); // Pop '('
        } else if (isOperator(c)) {
            while (!operators.empty() && precedence(operators.top()) >=
precedence(c)) {
                prefix += operators.top();
                operators.pop();
            }
            operators.push(c);
        }
    }
    while (!operators.empty()) {
        prefix += operators.top();
        operators.pop();
    }
    reverse(prefix.begin(), prefix.end());
    return prefix;
}
string infixToPostfix(string infix) {
    stack<char> operators;
    string postfix;
    for (char& c : infix) {
        if (isOperand(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() && precedence(operators.top()) >=
precedence(c)) {
                postfix += operators.top();
                operators.pop();
            }
            operators.push(c);
        }
    }
```

```
    while (!operators.empty()) {
        postfix += operators.top();
        operators.pop();
    }
    return postfix;
}
int main() {
    string infixExpression;
    cout << "21BAI1830\n" << endl;
    cout << "Enter infix expression: ";
    getline(cin, infixExpression);
    string prefix = infixToPrefix(infixExpression);
    string postfix = infixToPostfix(infixExpression);
    cout << "Prefix expression: " << prefix << endl;
    cout << "Postfix expression: " << postfix << endl;
    return 0;
}
```