Date 30-Nov-2020

Assignment - 10

21118

classmate Date Page

Problem Statement:

Implement C++ program for expression conversion as infix to postfix & it's evaluation using stack based on given conditions:

1) Operands & operators, both must be single character.
2) Input postfix expression must be in desired format
3) Only '+', '-', '*' & '/' operators are expected.

Objectives:

1) To understand types of expression representation along with adv & disadvantages of each type
2) Understand usage of stack in evaluating & converting one form of expression to another.

Outcomes:

1) To implement stack using linked lists/ arrays.
2) Use stack for calculation of postfix expression which contains simple operators.

Hardware Requirement:

Manufacturer & Model: Acer Swift-3
Processor: Intel core i-5 8th gen (8265U @ 1.6 GHz)
Installed Memory: 8GB RAM, 512 GB SSD.
Architecture: 64-bit

Software Requirement:

Operating System: Ubuntu 20.04 LTS on Oracle Virtual Machine (3 processors & 4096MB base memory is allocated)
C++ version used: C++14
Compiler for C++: g++ (version: 10.1.0)
Code Editor: Sublime Text (Build 2011)

```
for i ← 0 to length of infix {
    if exp[i] is number {
        append it to posfix }
    else if exp[i] is operator {
        while (!stk.empty() && stk.top() != '(' &&
                stk.top() has higher precedence than exp[i]) {
            posfix ← posfix + stk.top()
            stk.pop()
        }
    }
    else if exp[i] is '('
        stk.push('(')
    else if exp[i] is ')' {
        while (!stk.empty() && stk.top != '(') {
            posfix ← posfix + stk.top()
            stk.pop()
        }
    }
    while (!stk.empty()) {
        posfix ← posfix + stk.top();
        stk.pop();
    }
    return posfix
}
```

```
Algorithm Evaluate Postfix (* postfix) {
    stack<int> stk;
    for i ← 1 to length of postfix: {
        if (exp[i] is number) {
            stk.push(exp[i])
        }
        else if (exp[i] is operator) {
```

```
            op2 ← stk.top() ; stk.pop()
            op1 ← stk.top() ; stk.pop()
            stk.push (evaluate(op1, op2, expr[i]))
        {
    }
    return stk.top();
}
```

ADT of classes:

// Class Stack { (this will be a template class (say T))

```
private:
    Node* head;     // using linked list to implement stack
Public:
    stack() {       // Constructor function
      head = NULL;
    }
    int isEmpty() {
      // checks if stack is empty
    }
    void Push (T x) {
      // add x to stack
    }
    void Pop () {
      // remove element from top of stack
    }
    T Top() {
      // returns top of stack
    }
```

2) class Expression {

```
private:
  string exp;
```

```
bool isNumber(char c){
    // checks if char c is number or not
}
bool isOperator(char c){
// checks if c is operator or not
}
int evaluate(int op1, int op2, char oprt){
// evaluate op1 & op2 according to oprt.
}
int getPriority(char oprt){
// returns priority of i/p operator
}
bool isHigherPrecedence(char oprt1, char oprt2){
// check the precedance of oprt1 over oprt2
}

public:
    Expression(string str = ""){
    // constructor
    }
    string getExpression(){
    // returns expression
    }
    void setExpression(){
    // set value of exp
    }
    int evaluatePostfix(){
    // evaluates the postfix expression
    }
    string InfixToPostfix(string){
    // converts exp to postfix form
    }
};
```

Analysis of Algorithms:

→ Evaluation of postfix:
The time complexity of the algorithm is O(n) where n is the length of postfix expression.
~~(operands only)~~
The space complexity of the algorithm is O(n) where n is the operands present in postfix expression

→ Infix to postfix conversion:
The time complexity of the algorithm is O(n) where n is the length of infix expression.
The space complexity of the algorithm is O(n) where n is the operands + brackets present in ~~postfix~~ infix expression.

Test cases:

Sr.No.	Testcase Description	Expected output	Actual output
1)	infix → postfix i/p: (a+b)	a b +	a b +
2)	infix → postfix i/p: (a + b) * c	a b + c *	a b + c *
3)	postfix evaluation i/p: 2 3 + 5 /	1	1
4)	~~postfix evaluation~~		

* Applications

stacks are ideal data structure to calculate/evaluate prefix @@ postfix expressions.

prefix/postfix expressions are preferred in computer science due to their ease of calculation by computers.

* Conclusion

The assignment allows you to use stack to conversion from infix to postfix ~~efficiently~~ one can see that use of data structure can significantly reduce the complexity of problem.