frankSJSU DataStructure

Search this site

Labs and Homeworks >

Lab 6 Stack

Objective:

1. Understand how stack works
2. Exercise a stack application: Infix to Postfix algorithm.
3. Practice STL stack.

Overview:

1. Create a stack class using array (try to have just the minimum function, e.g. push, pop). If you need help, check out this page.
2. Do programming exercise #11 in chapter 7. Detail algorithm is described in exercise #9 (see below).
3. If you need sample code, the textbook solution is in the attachment. It is important that you understand the code and make whatever changes needed for the exercise.
4. Do the same program with STL stack.

9. **(Infix to Postfix)** Write a program that converts an infix expression into equivalent postfix expression.

The rules to convert an infix expression into an equivalent postfix express are as follows:

Suppose **infx** represents the infix expression and **pfx** represents the pos expression. The rules to convert **infx** into **pfx** are as follows:

a. Initialize **pfx** to an empty expression and also initialize the stack.
b. Get the next symbol, **sym**, from **infx**.
   b.1. If **sym** is an operand, append **sym** to **pfx**.
   b.2. If **sym** is **(**, push **sym** into the stack.
   b.3. If **sym** is **)**, pop and append all the symbols from the stack until most recent left parenthesis. Pop and discard the left parenthes
   b.4. If **sym** is an operator:
      b.4.1. Pop and append all the operators from the stack to **pfx** t are above the most recent left parenthesis and have p cedence greater than or equal to **sym**.
      b.4.2. Push **sym** onto the stack.
c. After processing **infx**, some operators might be left in the stack. P and append to **pfx** everything from the stack.

In this program, you will consider the following (binary) arithmetic ope tors: **+**, **-**, *****, and **/**. You may assume that the expressions you will process error free.

Design a class that stores the infix and postfix strings. The class must incl the following operations:

- **getInfix**—Stores the infix expression
- **showInfix**—Outputs the infix expression
- **showPostfix**—Outputs the postfix expression

Some other operations that you might need are the following:

- **convertToPostfix**—Converts the infix expression into a pos expression. The resulting postfix expression is stored in **pfx**.
- **precedence**—Determines the precedence between two operators. If first operator is of higher or equal precedence than the second operato returns the value **true**; otherwise, it returns the value **false**.

Include the constructors and destructors for automatic initialization dynamic memory deallocation.

11

Test your program on the following five expressions:

```
A + B - C;
(A + B ) * C;
(A + B) * (C - D);
A + ((B + C) * (E - F) - G) / (H - I);
A + B * (C + D ) - E / F * G + H;
```

For each expression, your answer must be in the following form:

```
Infix Expression: A + B - C;
Postfix Expression: A B + C -
```

InfixToPostfix.h (0k)	Frank sjsu Lin, Apr 5, 20…	v.1	
infixToPostFixImp.cpp (3k)	Frank sjsu Lin, Apr 5, 20…	v.1	

Comments

You do not have permission to add comments.