C Lab (Part 1): Infix-to-Postfix Converter

Name:
Student ID:

Stacks are used by compilers to help in the process of evaluating expressions and generating machine language code. In this lab project, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like 3 + 4 and 7 / 9 in which the operator (+ or / here) is written between its operands - this is called **infix notation**. Computers "prefer" **postfix notation** in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as 3 4 + and 7 9 /, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation, and then evaluate the postfix version of the expression. Each of these algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack in support of its operation, and in each the stack is used for a different purpose.

In this project, you'll write a version of the infix-to-postfix conversion algorithm. Write a program that converts an ordinary infix arithmetic expression (assume a valid expression is entered) with **single digit integers** such as

(6 + 2) * 5 - 8 / 4

to a postfix expression. The postfix version of the preceding infix expression is

6 2 + 5 * 8 4 / -

The program should read the expression into character array infix, and use stack functions to help create the postfix expression in character array postfix. The algorithm for creating a postfix expression is as follows:

1) Push a left parenthesis '(' onto the stack.
2) Append a right parenthesis ')' to the end of infix.
3) While the stack is not empty, read infix from left to right and do the following:
   If the current character in infix is a digit, copy it to the next element of postfix.
   If the current character in infix is a left parenthesis, push it onto the stack.
   If the current character in infix is an operator,
      Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the popped operators in postfix.
      Push the current character in infix onto the stack.
   If the current character in infix is a right parenthesis
      Pop operators from the top of the stack and insert them in postfix until a left parenthesis is at the top of the stack.
      Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

+ addition
- subtraction
* multiplication
/ division
^ exponentiation
% remainder

The stack should be maintained with the following declarations:

```
struct stackNode {
char data;
struct stackNode *nextPtr;
};
typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

The program should consist of main and eight other functions with the following function headers:

`void convertToPostfix( char infix[], char postfix[] )`
Convert the infix expression to postfix notation.

`int isOperator( char c )`
Determine if c is an operator.

`int precedence( char operator1, char operator2 )`
Determine if the precedence of operator1 is less than, equal to, or greater than the precedence of operator2. The function returns -1, 0 and 1, respectively.

`void push( StackNodePtr *topPtr, char value )`
Push a value on the stack.

`char pop( StackNodePtr *topPtr )`
Pop a value off the stack.

`char stackTop( StackNodePtr topPtr )`
Return the top value of the stack without popping the stack.

`int isEmpty( StackNodePtr topPtr )`
Determine if the stack is empty.

`void printStack( StackNodePtr topPtr )`
Print the stack.

The program should be able to give the following output:

```
Enter the infix expression.
1+(2*3-(4/5^6)*7)*8
The original infix expression is:
1+(2*3-(4/5^6)*7)*8
(       NULL
+       (       NULL
(       +       (       NULL
*       (       +       (       NULL
(       +       (       NULL
-       (       +       (       NULL
(       -       (       +       (       NULL
/       (       -       (       +       (       NULL
^       /       (       -       (       +       (       NULL
/       (       -       (       +       (       NULL
(       -       (       +       (       NULL
-       (       +       (       NULL
*       -       (       +       (       NULL
-       (       +       (       NULL
(       +       (       NULL
+       (       NULL
*       +       (       NULL
+       (       NULL
(       NULL
The stack is empty.
The expression in postfix notation is:
123*456^/7*-8*+
```

Submission Guidelines

a. Create a single file program. Name the file as “clab_p1.c”.
b. You need provide appropriate comments to make your code readable. (Note that if your code is not working and there are no comments, you may loss all the marks.)