Week 5: Stacks and Queues

Lab Tutor: For this lab-tutorial session, please discuss about the solution for each question in the lab. You may allocate about 30 minutes for each question. No need to discuss about the assignment questions and practice questions.

Questions:

1. Write a function `palindrome()` that determines whether a given string in a Queue is a palindrome. The prototype of the `palindrome()` function is given below:

   ```
   int palindrome(Queue* wordPtr);
   ```

   The function should return 1 if the string is a palindrome and 0 otherwise. You should ignore the space characters and the case of each letter. The stack and queue data structures and their interface functions are provided.

   Sample inputs and outputs:

   ```
   Enter a string of characters, terminated by a newline:
   On A Clover If Alive ERUPTS A VAST PURE EVIL A FIRE Volcano
   The string is a palindrome.

   Enter a string of characters, terminated by a newline:
   A   man    a    plan   a  CANAL    PANama
   The string is a palindrome.

   Enter a string of characters, terminated by a newline:
   Superman in the sky
   The string is not a palindrome.
   ```

2. Write a function `balanced()` that determines if an expression comprised of the characters, `()[]{}`, is balanced. The prototype of the `balanced()` function is given below:

   ```
   int balanced(char *expression);
   ```

   The function returns 1 if the expression is balanced. Otherwise, it return 0. The following expressions are balanced because the order and quantity of the parentheses match:

   ```
   ()
   ([])
   {[]()[]}
   ```

   The following expressions are not balanced:

   ```
   {{)]
   ```

```
[({{)])
```

Sample inputs and outputs:

```
Enter an expression, terminated by a newline:
[({{{}}})[[]]{}({})]
The expression is balanced.

Enter an expression, terminated by a newline:
{1+2+{5}*[6+x]+{4+5}(3+2)}
The expression is balanced.

Enter an expression, terminated by a newline:
[5[3(3)4{()]}]
The expression is not balanced.
```

3. Write a C program to evaluate a postfix expression. You may assume that operands are single-digit numbers. The function prototype is given below:

```
double exePostfix(char* postfix)
```

```
Some test sample cases:
Input: 987-654-/3+*+
Output: 18.00

Input: 25+89/9*7*+
Output: 63.00

Input: 88/8*77+7+5*6*-
Output: -622.00
```

4. Write a C program to convert an infix expression to a postfix expression. The input and the output of the function are character strings. The input expression contains only four possible operators: +, -, * and /. Operands can be any alphanumeric. Each operand is represented by a character symbol. The parentheses are allowed in the input expression. You may assume that the expression is always valid. The function prototype is given below:

```
void in2Post(char* infix, char* postfix)
```

Some test sample cases:

```
Infix: (A+B)+(C-D)
Postfix: AB+CD-+
```

```
Infix: a+b*c-d*(e/f)
Postfix: abc*+def/*-

Infix: 9+(8-7)*(6/(5-4)+3)
Postfix: 987-654-/3+*+
```

Practice Questions:

1. Write a c function reverseStack() that reverses a stack using a queue. Note that the reverseStack() function only uses push() and pop() when adding or removing integers from the stack, and only uses enqueue() and dequeue() when adding or removing integers from the queue.

   The function prototypes are given as follows:

   ```
   void reverseStack(Stack *s);
   ```

   For example: if the stack is <5, 4, 3, 2, 1>, the resulting stack will be <1, 2, 3, 4, 5>

   Sample test cases are given below:

   ```
   Enter a list of numbers, terminated by any non-digit
   character:
   1 2 3 4 5 a
   Before reverseStack() is called:
   Current List has 5 elements: 5 4 3 2 1
   After reverseStack() was called:
   Current List has 5 elements: 1 2 3 4 5
   ```

2. Write a c function reverseFirstKItems()that reverses the order of the first k elements of the queue using a stack, leaving the other elements in the same relative order for given an integer k and a queue of integers. Note that the reverseFirstKItems()function only uses push() and pop() when adding or removing integers from the stack and only uses enqueue() and dequeue() when adding or removing integers from the queue.

   The function prototypes are given as follows:

   ```
   void reverseFirstKItems(Queue *q, int k);
   ```

For example: if the queue is <1, 2, 3, 4, 5, 6> and k =3, the resulting queue will be <3, 2, 1, 4, 5, 6>

Sample test cases are given below:

```
Enter a list of numbers, terminated by any non-digit
character:
1 2 3 4 5 6 a
Enter the number of elements to be reversed their
order:
3
Before reverseFirstKItems() is called:
Current List has 6 elements: 1 2 3 4 5 6
After reverseFirstKItems() was called:
Current List has 6 elements: 3 2 1 4 5 6
```

3. Write a recursive function `recursiveReverse()` that reverses the order of items stored in a queue of integers. The prototype of the `recursiveReverse()` function is given below:

```
void recursiveReverse(Queue *qPtr);
```

Sample inputs and outputs:

```
Enter a list of numbers for a queue, terminated by any
non-digit character:
0 1 1 2 3 5 8 13 21 a

Before recursiveReverse() is called:
Current List: 0 1 1 2 3 5 8 13 21

After recursiveReverse() was called:
Current List: 21 13 8 5 3 2 1 1 0
```

4. Write a C program to convert an infix expression to a prefix expression.

```
void in2Pre(char* infix, char* prefix)
```

Some test sample cases:

```
Infix: (A+B)+(C-D)
Prefix: ++AB-CD

Infix: a+b*c-d*(e/f)
Prefix: -+a*bc*d/ef
```

```
Infix: 9+(8-7)*(6/(5-4)+3)
Prefix: +9*-87+/6-543
```