Stacks and Queues

---

1. Write a function `removeUntil()` that pops all values off a stack of integers down to but not including the first occurrence of the chosen value. The prototype of the `removeUntil()` function is given below:

```
void removeUntil(Stack *s, int value);
```

Given a stack [1 2 3 4 5 6 5 4 3 2 1] with the topmost number displayed on the left, calling `removeUntil()` with value = 5 will produce the stack [5 6 5 4 3 2 1]. If value =7, an empty stack will be produced.

Sample inputs and outputs:

Enter a list of numbers for a stack, terminated by any non-digit character:

```
1 2 5 6 5 7 8 a

Before removeUntil() is called:

Current List: 8 7 5 6 5 2 1

Enter an integer value for removeUntil()

5

After removeUntil() was called:

Current List: 5 6 5 2 1
```

2. 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 *q);
```

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
```

3. 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(char *word);
```

The function should return 0 if the string is a palindrome and -1 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:

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.
```

4. 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 0 if the expression is balanced. Otherwise, it returns -1. 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.
```