

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 for 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].

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

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

3. Write a function `palindrome()` that determines whether a given string is a palindrome. The prototype for 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 null terminator. Ignore the case of each letter.

Sample output:

```
Enter a string: A man a plan a canal Panama
The string is a palindrome.
```

```
Enter a string: 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 for the `balanced()` function is given below:

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

The following expressions are balanced because the order and quantity of the parentheses match:

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

The following expressions are not balanced:

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