

## 1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Lab07, put each programming exercise into its own subdirectory of this directory, zip the entire Lab07 directory up into the file Lab07.zip, and then submit this zip file to Lab #7.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyClasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. No inline coding in the (.h) files. All templated class structures are to have their own guarded specification/implementation file (.h) that has the same name as the class. As with non-templated class structures, no inline coding in the specification. In addition, you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

## 2 Programming Exercises

1. Ackermann's Function is a recursive mathematical algorithm that can be used to test how well a computer performs recursion. Write a function  $A(m, n)$  that solves Ackermann's Function. Use the following logic in your function:

If  $m = 0$  then return  $n + 1$

If  $n = 0$  then return  $A(m - 1, 1)$

Otherwise, return  $A(m - 1, A(m, n - 1))$

Test your function in a driver program that displays the following values:

$A(0, 0)$   $A(0, 1)$   $A(1, 1)$   $A(1, 2)$   $A(1, 3)$   $A(2, 2)$   $A(3, 2)$ ,  $A(4, 1)$ ,  $A(4, 2)$

2. In the study of chaos and dynamics there are two sequences, like the Fibonacci sequence but more complex, that come up. These are sometimes called  $Q$  numbers and  $D$  numbers. The  $D$  sequence is defined to be  $D(1) = 1$ ,  $D(2) = 1$ , and

$$D(n) = D(D(n - 1)) + D(n - 1 - D(n - 2))$$

The  $Q$  sequence is defined to be  $Q(1) = 1$ ,  $Q(2) = 1$ , and

$$Q(n) = Q(n - Q(n - 1)) + Q(n - Q(n - 2))$$

Obviously both of these functions are recursive.

Write a recursive functions  $D$  and  $Q$  that take in a single long parameter  $n$  and return a long that is the value of  $Q(n)$  and  $D(n)$ , respectively. Create a main that will ask the user for a value of  $n$  and print out each of the values of the  $Q$  function and  $D$  function from 1 to  $n$ . that is  $Q(1), Q(2), Q(3), \dots, Q(n)$ , and  $D(1), D(2), D(3), \dots, D(n)$ . All of the printing is to be done in the main and no printing is to be done in the function. A sample run is below,

```
Input n: 25
Q Numbers: 1 1 2 3 3 4 5 5 6 6 6 8 8 8 10 9 10 11 11 12 12 12 12 16 14
D Numbers: 1 1 2 2 3 4 4 4 4 5 6 7 8 8 8 8 8 9 10 10 10 11 13
```

3. As you know, a Palindrome is a string that is the same written in either direction. For example, "A Toyota" or "Eva, can I see bees in a cave?". We can determine if a string is a palindrome recursively. Take a string and compare the first letter with the last letter, if the letters are the same you compare the second and second to last, and so on until you get to the middle of the string. Create a function,

```
bool isPal(const string& str, int startIndex, int endIndex)
```

that will return false if the characters in the start and end are not the same, true is it makes it to the middle of the string and recurses to the next letter positions to check otherwise. Create a program to check input palindromes using this function. A couple runs are below.

```
Enter a string, no spaces and all lower case: evacaniseebeesinacave
evacaniseebeesinacave is a palindrome.
```

```
Enter a string, no spaces and all lower case: atoyota
atoyota is a palindrome.
```

```
Enter a string, no spaces and all lower case: help
help is not a palindrome.
```