CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12

This assignment is due **Friday, December 7th, by 11:55 pm**

- ***All components*** **(Cloud9 workspace and moodle quiz attempts) must be completed and submitted by Friday, December 7th, by 11:55 pm for your solution to receive points.**
- **Recitation attendance is required to receive credit.**

Please follow the same submission guidelines outlined in Homework 3 description regarding Style, Comments and Test Cases. Here's a review below on what you need to submit for Recitation 12.

Develop in Cloud9: For this recitation assignment, write and test your solution using Cloud9.

Submission: All three steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.

1. ***Make sure your Cloud 9 workspace is shared with your TA:*** Your recitation TA will review your code by going to your Cloud9 workspace. *TAs will check the last version that was saved before the submission deadline*.
    - Create a directory called **Rec12** and place all your file(s) for this assignment in this directory.
    - Make sure to *save* the final version of your code (File > Save). Verify that this version displays correctly by going to File > File Version History.
    - The file(s) should have all of your functions, test cases for the functions in main function(s), and adhere to the style guide. Please read the **Test Cases** and **Style and Comments** sections included in the **Homework 3** write up for more details.
2. ***Submit to the Moodle Autograder:*** Head over to Moodle to the link **Recitation 12**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.

CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12

Binary Numbers

Remember counting using your fingers? When you've counted to five and you want to proceed, what would you do? The five fingers on your left hand are not enough, so you borrow from your right hand to do the job.

The most familiar numeral system is decimal where we use symbols from 0 to 9. When we want to represent some number larger than 9, the existing digits are not sufficient, so we have to add more digits. This decimal representation is not the only numeral system, and in many fields, it is not an ideal one either. Which system to use usually depends on specific applications. For example, you might have a sixty-two-based numeral system, and the symbols are from:
0, 1, 2, …, 8, 9, a, b, c, …, x, y, z, A, B, C, …, X, Y, Z.
In decimal, we say this system is "62-based", or "base 62". Then how do we represent the decimal-based number 63? We start counting from 1 through 9. When we get to 10 we still haven't run out of symbols, so we keep counting from a to z, and from A to Z. Z is the last symbol, or the 61st. So how do we represent sixty-two in our 62-based system? We add a new digit to the number, so it is represented by **10** in our new system. And then, counting 63 is represented by **11** in our new system.

Binary Numeral System
In computer systems, a binary numeral system (with symbols only 0 and 1), is the most important representation, since it can be easily mapped to logic gates. 1 for on and 0 for off. Let's start counting in decimal first, and see how the number is represented in binary, as shown in the following table.

0	1	2	3	4	5	6	7	8
0	1	10	11	100	101	110	111	1000

When we're counting 2, we run out of symbols, so we have to add a new digit. Similarly, when we are counting 4, we add new digit. A simple math shows how to convert a number from binary to decimal. Say we have a binary number 11010. Below is a step-by-step guide:

CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12

Binary:	**1**	**1**	**0**	**1**	**0**
Location:	4	3	2	1	0
Power of 2:	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Multiply by the binary:	16 * 1 = 16	8 * 1 = 8	4 * 0 = 0	2 * 1 = 2	1 * 0 = 0
Add them up	16 + 8 + 0 + 2 + 0 = 26				

This procedure applies to any numeral system. If the system is based on 62, in the step of "power of 2", we replace "2" with "62", and the rest of it is the same.

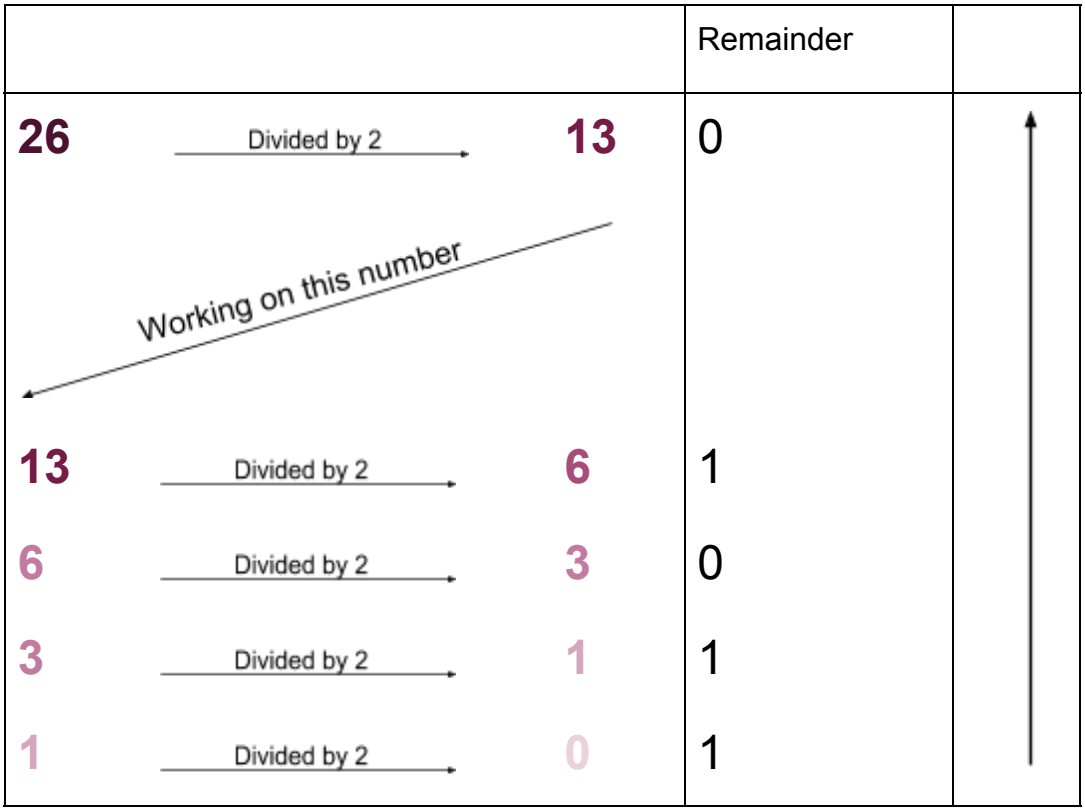
Here's a similar example in the more-familiar base 10:

Number:	**8**	**5**	**2**	**7**	**7**
Location:	4	3	2	1	0
Power of 2:	$10^4 = 10000$	$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
Multiply by the binary:	8 * 10000 = 80,000	5 * 1000 = 5,000	2 * 100 = 200	10 * 7 = 70	1 * 7 = 7
Add them up:	80,000 + 5,000 + 200 + 70 + 7 = 85,277				

Converting Decimal to Binary

To convert a decimal to binary, we simply keep dividing our number by 2, and get the remainder of the number. We won't go into details now, but just work on an example. Let's start with a decimal number 26.

CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12



We should stop this procedure when the last number we work on is 1 or 2. We get all the remainders, and align them from bottom to the top, which is **1 1 0 1 0**, and we get our binary number.

Now that we know how to do this by hand, think about how to do that in your code. If you have an integer 13, how do you get the integer result 6 instead of 6.5? Then how do you get the remainder? What structure came to your mind first about this repeating procedure? Then how do you store the remainders and go back from the bottom to the top once you're done?

CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12

Recursion

A recursive function is one which calls itself. Recursion can be used to accomplish a repetitive task, like loops. Indeed, it turns out that anything you can do with a loop, you could also do with recursion, and vice versa. However, some algorithms are much easier to express with loops, and others are much easier to express with recursion. You'll want both in your toolkit to write elegant, simplistic, short code.

Every recursive function includes two parts:

- base case: A simple non-recursive occurrence that can be solved directly.
- recursive case: A more complex occurrence that can be described using smaller chunks of the problem, closer to the base case.

To write a recursive function, we often use the following format:

```
returnType functionName(arguments){
     if (/*baseCase?*/){
          return /*baseCase result*/;
     }else{
          // some calculations, including a call to functionName
          // with "smaller" arguments.
          return /*general result*/
     }
}
```

Consider the following simple recursive function which calculates the sum of the numbers 1...n:

```
int numberSum(int n){
     if(n==0){ //base case
          return  0;
     }else{ //recursive case
          int  sumForSmallerNumbers = numberSum(n-1);
          return (n + sumForSmallerNumbers);
     }
}
```

CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12

Consider this example; what does this function do?

```
int findValue(int n){
     if (n < 10){
          return n;
     }else{
          int a = n/10;
          int b= n%10;
          return findValue(a+b);
     }
}
```

This table shows both the final returned value and intermediate recursive function calls. For example, the top right cell reads: "`findValue(27)` returns `findValue(9)` which returns `9`."

`findValue(8)`
`  8`
	`findValue(27)`
	`  findValue(9)`
	`  9`

`findValue(93)`
`  findValue(12)`
`  findValue(3)`
`  3`
	`findValue(84676)`
	`  findValue(8473)`
	`  findValue(850)`
	`  findValue(85)`
	`  findValue(13)`
	`  findValue(4)`
	`  4`

CSCI 1300 - Starting Computing
Instructor: Fleming
Recitation 12

Problem Set

Problem 1

Write a function **decimalToBinaryIterative** that converts a decimal value to binary using a loop. This function takes a single parameter, a non-negative integer, and returns a string corresponding to the binary representation of the given value.

- Your function should be named **decimalToBinaryIterative**
- Your function should take a single argument
    - An **integer** to be converted to binary
- Your function should not print anything
- Your function should use a loop
- Your function should return the binary representation of the given value as a string

For example, the call `decimalToBinaryIterative(5)` should return `101`.

Problem 2

Write a function **decimalToBinaryRecursive** that converts a decimal value to binary using recursion. This function takes a single parameter, a non-negative integer, and returns a string corresponding to the binary representation of the given value.

- Your function should be named **decimalToBinaryRecursive**
- Your function should take a single argument
    - An **integer** to be converted to binary
- Your function should not print anything
- Your function should use recursion **instead** of a loop
- Your function should return the binary representation of the given value as a string

For example, the call `decimalToBinaryRecursive(8)` should return `1000`.