CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 8
Due: Wednesday, Nov 20, by 6:00 pm
+5% bonus if submitted by Tuesday, Nov 19, 11:59 pm

All 2 components (Moodle Coderunner attempts, and zip file) must be completed and submitted by Wednesday, Nov 20, at 11 pm for your homework to receive points.

---

1. Objectives

- Learn binary numbers and how to use recursive functions
- Develop some methods that will be useful in your future!

---

2. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Work on questions on your Cloud 9 workspace***: You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **hmwk9** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner***: Head over to Moodle to the link **Homework 9 Coderunner** 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.

3. ***Submit a .zip file to Moodle***: After you have completed all questions from the Moodle assignment, zip all 8 files you compiled in Cloud9, and submit the zip file through the **Homework 9** link on Moodle.

---

3. Rubric

Aside from the points received from the **Homework 9 Coderunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 9** link on Moodle and assign points for the following:

Style, Comments, Algorithm (5 points):

Style:

- Your code should be well-styled, and we expect your code to follow some basic guidelines on whitespace, naming variables and indentation, to receive full credit. Please refer to the **CSCI 1300 Style Guide** on Moodle.

Comments:

- Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to help other developers understand how your code works. These comments should begin with two backslashes (`//`) or the multi-line comments (`/* … comments here… */`) .
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Homework 9 - Problem # ...
```

Algorithm:

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the function.
- This is an example C++ solution. Look at the code and the algorithm description for an example of what is expected.

Example 1:

```
/*
 * Algorithm: convert money from U.S. Dollars (USD) to Euros.
 *    1. Take the value of number of dollars involved
 *       in the transaction.
 *    2. Current value of 1 USD is equal to 0.86 euros
 *    3. Multiply the number of dollars got with the
 *       currency exchange rate to get Euros value
 *    4. Return the computed Euro value
 * Input parameters: Amount in USD (double)
 * Output (prints to screen): nothing
 * Returns: Amount in Euros (double)
 */
```

Example 2:

```
double convertUSDtoEuros(double dollars)
{
     double  exchange_rate  =  0.86;  //declaration  of  exchange
rate
     double euros = dollars * exchange_rate; //conversion
     return euros; //return the value in euros
}
```

The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

```
/*
 * conversion
 */
double convertUSDtoEuros(double dollars)
{
     double euros = dollars * 0.86;
     return euros;
}
```

Global variables (use will result in a 5 point deduction):

To keep things simple, straightforward, and easy to debug and test, you may not use global variables in this homework.

Test Cases (10 points)

Code compiles and runs (3 points):

The zip file you submit to Moodle should contain 7 full programs (with a main() function), saved as .cpp files. It is important that your programs can be compiled and run on Cloud9 with no errors. The functions included in these programs should match those submitted to the CodeRunner on Moodle.

Test cases (7 points):

For this week's homework, all 2 problems are asking you to create a function. In your solution file for each function, you should have at least 2 test cases present in their respective `main()` function, for a total of at least 4 test cases (see the diagram on the next page). Your test cases should follow the guidelines, Writing Test Cases, posted on Moodle under Week 3

---

4. Background

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 string some of the existing digits (0 to 9) together to represent numbers larger than 9. There are 10 symbols (0 to 9) in terms of which all other numbers are represented. The decimal system has a base of 10, or is "base 10" for this reason.

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, whose 62 symbols are:

0, 1, 2, …, 8, 9, a, b, c, …, x, y, z, A, B, C, …, X, Y, Z.

We say this system is “62-based”, or “base 62”. Since it has 62 unique symbols, we can represent the decimal numbers 0 to 61 using them. We start counting from 0 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 62nd. It represents the decimal number 61. Then how do we represent the decimal number 62 in our “62-based” system? We string together some of the existing symbols. For example, we can string 1 and 0 together, so **10** in our new system represents 62. And then, counting 63 is represented by **11** in our new system and so on.

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 represents “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

Note that when we are addressing numbers, leading 0s don’t matter. That is, 0001 is the same as 1.

When we’re counting 2, we run out of symbols, so we have to add a new bit (the ***bi***nary version of digit). Similarly, when we are counting 4, we add new bit. A bit simple math - pun intended, you’re welcome - shows how to convert a number from binary to decimal. Say we have a binary number 11010. Below is a step-by-step guide:

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				

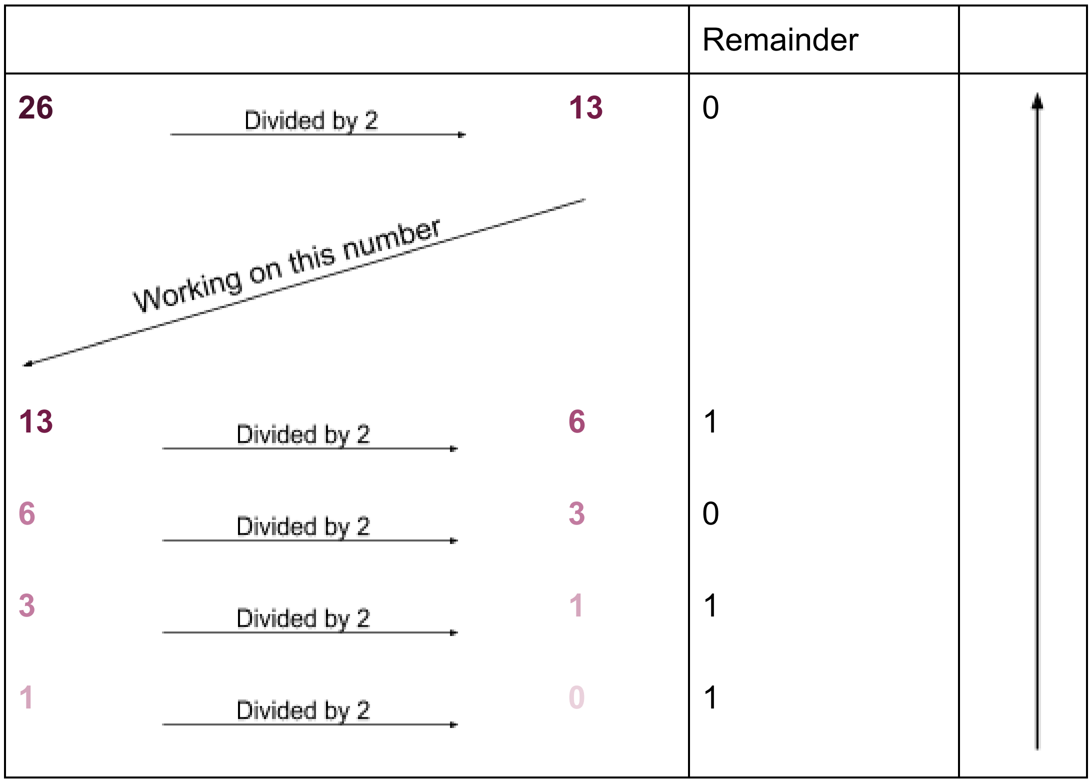
This procedure applies to any numeral system. If the system is based on 62, in the step of “power of 2”, we replace “2” with “62”, and the rest of it is the same.

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 number to its binary representation, we simply keep dividing our number by 2, and get the remainder of the number. We won’t go into details now, but let’s work on an example. For starters, consider the decimal number 26.



			Remainder	
26	Divided by 2	13	0	
	Working on this number			
13	Divided by 2	6	1	
6	Divided by 2	3	0	
3	Divided by 2	1	1	
1	Divided by 2	0	1	

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?

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. Sometimes, there are multiple base cases.

- **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, 2, 3, …, n:

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

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`

5. Problem Set

Problem 1 - binary!

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 - recursion!

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”.

5. Homework 9 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 9 Coderunner**
2. Submit one zip file to **Homework 9**. The zip file should be named, **hmwk9_lastname.zip**, and have following 2 files:

```
1. iterative.cpp
2. recursive.cpp
```

6. Homework 9 point summary

Criteria	**Pts**
Coderunner	35
Style, Comments, Algorithms	5
Test cases	10
Recitation attendance *	-15
Using global variables	-5
Total	50
5% early submission bonus	+5%

* If your attendance is not recorded, you will lose points.
Make sure your attendance is recorded on Moodle **before you leave recitation**.