CS 31 Solutions Week 3

This worksheet is entirely **optional**, and meant to prepare you for upcoming projects and exams. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. We encourage you to collaborate with your peers while completing this worksheet. If you finish the worksheet early, you are welcome to inquire with your LA for this week's Supplemental Problems.

Solutions are written in red. The solutions for **programming problems** are not absolute, it is okay if your code looks different; this is just one way to solve the specific problem.

If you have any questions or concerns please contact your LA or go to any of the LA office hours.

Concepts: Loops, If Statements, Cin, Variables, Doubles, Ints

Reading Problems

1. This code snippet tries to print all prime numbers between 3 (inclusive) and a given input *n* (exclusive). Find the 3 bugs contained in the code and fix them.

```
int n;
cin >> n;
for (int candidate = 3; candidate < n; ++candidate) {
        bool isPrime = true;
        for (int x = 2; x < n; x++) {
                if (candidate % x = 0){
                        isPrime = false;
                }
        }

        if (isPrime) {
                cout << n << " ";
        }
}
```

```
int n;
cin >> n;
// check the candidates: the integers in the interval of [3, n)
for (int candidate = 3; candidate < n; ++candidate) {
      bool isPrime = true;  // a flag for whether the candidate is
prime or not
      for (int x = 2; x < candidate; x++) {
            // if any number in between 2 (inclusive) and the
candidate
            // (exclusive) is the factor of the candidate,
            // then the candidate is not a prime.

            if (candidate % x == 0){
                  isPrime = false;
            }
      }

      if (isPrime) {
            cout << candidate << " ";
      }
}
```

++ before
Why use cand. Instead of n

Programming Problems

1. Write a program that takes in a number as an int and outputs the sum of all of the digits in that number.

Sample Output:
Enter a number: 184
The sum of the digits in your number is 13!

```
#include <iostream>
using namespace std;

int main() {
      /*
      NOTICE: the modulo (%) operator can be used to "extract" the
last digit of a number. For example, 803 % 10 = 3; 41 % 10 = 1; 12345
% 10 = 5.

      Similarly, the arithmetic division operator (/) can be used to
"remove" the last digit of a number. For example, 803 / 10 = 80; 41 /
10 = 4; 12345 / 10 = 1234

      We can exploit these facts to progressively extract and remove
the last digit of an integer to create a sum.
      */
      cout << "Enter a number: ";
```

```
	int num;
	cin >> num;
	int sum = 0;
	while (num > 0) {
		sum += num % 10;
		num /= 10;
	}
	cout << "The sum of the digits in your number is " << sum <<
	"!" << endl;
}
```

2. Write a program that takes in N numbers and writes their mean as a double.

Sample output:
How many numbers do you want to average? 5
Number: 4
Number: 2
Number: 8
Number: 9
Number: 7
The average is 6!
-> Change to decimal

```
#include <iostream>
using namespace std;

int main() {

	cout << "How many numbers do you want to average? ";
	int n;
	cin >> n; // take in a number as input
	double num;
	double total = 0; // store average as a double type; the
average should be a double so averages are not rounded to the nearest
whole number i.e. if total was of int type, the average of 1, 1, 2 =
1, not 1.3333

	for (int i = 0; i < n; i++) {
		cout << "Number: ";
		cin >> num;
		total += num; // add each number to your sum
	}

	cout << "The average is " << total/n << "!" << endl; // average
= total / n
}
```

3. Write a program that reads in an integer N and prints an NxN box where the (i,j)th character is as follows:

'.' if j > i
i + j otherwise

Where i is the row number and j is the column number (starting at 0, not 1). For Example, if the input is 4, it should print:

```
0 . . .
1 2 . .
2 3 4 .
3 4 5 6
```

Bonus:
Without using any explicit if/else statements, can you modify the previous program to only print out the sums and no '.'s?
Example, if the input is 4, it should print:

```
0
1 2
2 3 4
3 4 5 6
```

```
#include <iostream>
using namespace std;

int main() {
	int n;
	cout << "Enter a positive integer: ";
	cin >> n;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (j > i)
				cout << ". ";
			else
				cout << i + j << " ";
		}
		cout << endl;
	}
}
```

Bonus:

```
#include <iostream>
using namespace std;

int main() {
	int n;
	cout << "Enter a positive integer: ";
	cin >> n;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j <= i; j++) {
			cout << i + j << " ";
		}
	cout << endl;
	}
}
```

4. Write a program that reads in an integer and prints whether that number is a perfect number. A perfect number is defined as a number that is equal to the sum of all positive factors excluding itself.

Example:

4 != 1 + 2	=> Print “Not perfect.”
5 != 1	=> Print “Not perfect.”
6 = 1 + 2 + 3	=> Print “Perfect.”
12 != 1 + 2 + 3 + 4 + 6	=> Print “Not perfect.”
28 = 1 + 2 + 4 + 7 + 14	=> Print “Perfect.”

```
int perfect;
cout << "Enter a number: ";
cin >> perfect;
int sum = 0;

// any factor of perfect is smaller than perfect: loop through all
values smaller than perfect to find all its factors
for (int i = 1; i < perfect; i++) {
	// if the current iteration number is a factor of “perfect”
	if (perfect % i == 0) {
		// add the factor to the current total
		sum += i;
	}
}

// if the sum of all the factors is equal to perfect, it is a perfect
number
if (sum == perfect) {
	cout << "Perfect." << endl;
} else {
```

```
// else, it is not a perfect number
      cout << "Not perfect." << endl;
}
```

5. Write a program that takes in an integer N where N > 0, and outputs all its factors, each one separated by a comma. There should be no comma before the first number or after the last one.

Sample input:
12
Sample output:
1,2,3,4,6,12

```
#include <iostream>
using namespace std;

int main() {
      int n;
      cout << "Enter a number: ";
      cin >> n; //Takes in the integer N; saves to declared variable

      cout << "1"; // 1 is always a factor
      for (int i = 2; i <= n; i++) //Iterates through all integers
less than or equal to n
      {
            if (n % i == 0) //If i evenly divides into n, print it
      with a comma separation
                  cout << "," << i;
      }
      cout << endl;
}
```

6. Write a program that, given an input integer N, finds an integer x such that $2^x <= N < 2^{x+1}$. The program should ask for user input and print the integer x it finds. If there is no such x, it should print "error" and nothing more.

Sample Input:

200 => Should output 7, since $2^7 = 128 <= 200 < 2^8 = 256$.
20 => Should output 4, since $2^4 = 16 <= 20 < 2^5 = 32$.
8 => Should output 3, since $2^3 = 8 <= 8 < 2^4 = 16$.

```
#include <iostream>
using namespace std;

int main() {
	int number;
	cout << "Enter a number: ";
	cin >> number;

	if (number < 1) {	// check if the number is non-positive
		cout << "error" << endl;
	}
	else {
		int pow = 0, powOf2 = 1;
		while (powOf2 <= number) {
			pow++;
			powOf2 *= 2;
		}
		cout << pow - 1 << endl;
		// pow - 1 bc while loop terminates once powOf2 > number,
		// but we want powOf2 to still be less than number
	}
}
```

Check if the number is positive. Want to create a loop to find 2^x. To do this, you must keep track of both x, which is the exponent of 2, and powOf2 which is the evaluated value of 2^x. In your while loop, check if powOf2 is greater than N as per the prompt. If powOf2 is less than the number, you must increment the power x by one. You must also update powOf2. Once you find a powOf2 > N, then you have found the value of x that is greater than N, so you must return pow - 1 since that will be the closest value of x which has a result powOf2 <= N.

7. The Fibonacci series consists of the integers 0, 1, 1, 2, 3, 5, 8, … . With the initial values $n_1 = 0$ and $n_2 = 1$ it is possible to find the next number, because the next number is related to the preceding two by the formula $x_{n+1} = x_n + x_{n-1}$. For example, 1+1 = 2, the next number in the series. Based on this information, write a program that receives an integer n as an input and prints the n^{th} Fibonacci number. **What is the 10^{th} one?**
Your program should also check whether the integer provided is valid. If the user inputs zero or a negative number, the program should print *Error: The input must be positive* and nothing more.
If you haven't done so already, try to write the program using a do-while loop.

10th Fibonacci number: 34

```
#include <iostream>
using namespace std;

int main() {
      int n;
      cin >> n;
      if (n <= 0){
            cout << "Error: The input must be positive" <<endl;
            return 1;
      }
      if (n == 1)
            cout << 0 << endl;
      else if (n == 2)
            cout << 1 << endl;
      else {
            int xnext = 0;
            int x_last1 = 1;
            int x_last2 = 0;
            for (int i = 2; i < n; i++) {
                  xnext = x_last1 + x_last2;
                  x_last2 = x_last1;
                  x_last1 = xnext;
            }
            cout << xnext;
      }
}
```

Implement base cases with the first 2 if / else if. In the else branch, follow the formula to compute the next number in the Fibonacci series. The else branch must start from the base numbers, and compute every fibonacci number up until the nth integer in the series.

```
// alternative solution with a do-while loop

#include <iostream>
using namespace std;

int main() {
      // n is the fibonacci number we want
      int n;
      cout << "Enter a number " << endl;
      cin >> n;
      // check for non-positive n
      if (n <= 0) {
            cout << "Error: The input must be positive" <<endl;
      }
      else {
            // initialize
            int x_last2 = 0; // first fib num
            int x_last1 = 1; // second fib num
            switch(n) {
                  case 1:
                        cout << x_last2 << endl;
                        break;
                  case 2:
                        cout << x_last1 << endl;
                        break;
                  default:
                        // n is at least 3
                        int xnext;
                        // first 2 already computed, so init count to 2
                        int count = 2;
                        do {
                              // calculate the next term (sum of prev)
                              xnext = x_last1 + x_last2;
                              // push forward x_last2 and x_last1
                              x_last2 = x_last1;
                              x_last1 = xnext;
                              // increment the number of terms
                        calculated
                              count++;
                        }
                        while (count < n);
                        cout << xnext << endl;
            }
      }
}
```