Lab 6 - Functions

Directions

- The labs are marked based on attendance and effort.
- It is your responsibility to ensure the TA records your progress by the end of the lab.
- Do each step of the lab in order.
- While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments, however, are individual work.
  You may not work on assignments during your lab time.
- If you complete the lab early, you should experiment with C++; however, you may leave if you prefer.
- If you do not finish the lab exercises during your lab time, you are encouraged to complete them later to finish learning the material. You will still receive full marks if you arrived on-time and put in your best effort to complete the lab.

1. Output vs Return

1. Create a new project for this lab, and create a new program called `perfect.cpp`.
2. Add the usual `main()` function.
3. Create a function with the following header:
   `void holdWindowOpen();`
   - Make this function output a message to the screen and hold the window.
   - Have `main()` call this function and test your program.
4. Create a function with the following header:
   `void displaySumUptoN(int n);`
   - It should add up all integers from 1 to `n`.

     For example, if `n` is 4, it adds 1+2+3+4 = 10, and outputs the sum to the screen.
5. In `main()`, prompt the user for their favourite number (an integer), and pass it as an argument to the `displaySumUpToN()` function.
   - Run the program. Your output should look like:

```
Enter your favourite number: 5
Sum 1 .. 5 = 15

Holding window open...
```


6. Add a new function with the following header:
   `int getSumUpToN(int n);`
   - Make this function the same as `displaySumUpToN()`, except instead of printing to the screen, have it **return** the sum.
     Hint: Copy and paste part of the `displaySumUptoN()` function.
   - Have `main()` call this function, passing in the user's favourite number and print its return value to the screen:
     ```
     int sum = getSumUpToN(favNum);
     cout << "Returned sum is: " << sum << endl;
     ```
   - Your output should now be:
     ```
     Enter your favourite number: 5
     Sum 1 .. 5 = 15
     Returned sum is: 15

     Holding window open...
     ```
7. In `main()`, write code to outputs a number of stars ('`*`') equal to the sum from 1 to `n`.
   - For example, if `n` is 5, the sum from 1 to 5 is 15, so output 15 *'s:
     `***************`
   - Use one of the functions you wrote in this section to help accomplish this.
8. **Understanding:**
   - What is the fundamental difference between a function *returning a value*, and a function *displaying a value*?
   - For outputting the stars to the screen, will either one of the functions in this section work? Why or why not?

2. Finding the perfect number

Let's check if the number is "perfect". In math, a perfect number is one which equals the sum of its "proper" factors. For example, 6 = 1 + 2 + 3, where 1, 2, and 3 are all the proper factors of 6.

1. Overview:
   - First we'll create a function to test for proper factors: `isProperFactor()`
   - Then we'll sum up proper factors: `sumProperFactors()`
   - Finally, we'll test if a number is perfect: `isPerfectNumber()`
2. Crate the following function in `perfect.cpp`:
   `bool isProperFactor(int num, int divisor);`
   - Make this function return `true` if both of the following are true:
     - `divisor` < `num`, and
     - The `divisor` divides evenly into `num`: `(num % divisor) == 0`
   - otherwise, return `false`.

Math theory: *d* is a proper factor of *n* if $d < n$, and *d* is a factor of *n* (which means, *d* divides evenly into *n*).

3. Write a couple test statements in `main()` to test `isProperFactor()`:
   - Output to the screen if 3 is a proper factor of 6:
     ```
     cout << "3 a proper factor of 6? "
          << isProperFactor(6, 3) << endl;
     ```
   - Repeat the test for the following pairs:
     - 4 a proper factor of 6?
       1 a proper factor of 6?
       6 a proper factor of 6?
   - Your output should look like:
     ```
     Enter your favourite number: 5
     Sum 1 .. 5 = 15
     Returned sum is: 15
     ***************
     3 a proper factor of 6? 1
     4 a proper factor of 6? 0
     1 a proper factor of 6? 1
     6 a proper factor of 6? 0

     Holding window open...
     ```
   - Comment out these 4 lines of test code when `isProperFactor()` works.
4. Create a function to sum up the proper factors of a number:
   ```
   int sumProperFactors(int n);
   ```
   - Return the sum of all numbers between 1 and `n` which are proper factors of `n`.
   - Hints:
     - Create the loop first.
     - In the loop, use the isProperFactor() function to check each number if it is a proper factor. If so, add it to the sum.
     - **Return** the sum from the function.
5. Add a test call in `main()` to call `sumProperFactors()` passing in the user's favourite number.
   - Display the sum to the screen, which should look like:
     ```
     Enter your favourite number: 5
     Sum 1 .. 5 = 15
     Returned sum is: 15
     ***************
     Sum of proper factors: 1

     Holding window open...
     ```

6. Finally, create a function with the following header:
   `bool isPerfectNumber(int n);`
   - Make this function return true if n is a perfect number (i.e. where sumProperFactors(n) == n); otherwise return false.
   - Call `isPerfectNumber()` from `main()`, using the favourite number as an argument. Print a message stating if it is, or is not, a perfect number. Here are two sample outputs:

```
Enter your favourite number: 5
Sum 1 .. 5 = 15
Returned sum is: 15
***************
Sum of proper factors: 1
I'm sorry, that's not a perfect answer.

Holding window open...


Enter your favourite number: 6
Sum 1 .. 6 = 21
Returned sum is: 21
***************
Sum of proper factors: 6
Very good! That's a perfect answer!

Holding window open...
```

7. Using your program, try and find a perfect number between 490 and 500.
   - You might start by typing in the numbers to see if you can find it.
   - Is there a better way you could find perfect numbers? Modify your program to output every perfect number between 1 and 10000. (You should find 4, slowly...).

8. **Understanding:**
   - Describe what your algorithm would look like if you implemented it without using functions. What is the advantage of using functions?
   - Inside `sumProperFactor()`, why is a for loop a good type of loop to use?
   - Why are the following two lines of code interchangeable?

```
if (isProperFactor(n, i)) {...}
if (isProperFactor(n, i) == true) {...}
```


3. Good test values

Many people debug their programs as fast as possible in order to show that there are no bugs. **Debugging is trying to find bugs, not hide from them.**

1. From the course website, download the file `debugging.cpp` and add it to your project. This program reads in two test scores and averages the two.
2. Test the program with the following 4 separate test runs:

	Name	**Score 1**	**Score 2**	**Expected Outcome**
Test 1	Mary Q	80	80	80.0 Program quits.
Test 2	Bill Q	70	80	75.0 Program quits.
Test 3	Tom q	80	90	85.0 Program quits.
Test 4	Sam q	-1, then 1	999 then 99	50.0 Program quits.

3. Did following these test find any bugs?
   No? But, there are five logic errors in the program!
   - Spend a moment and try and find the logic errors.
4. Each test you do should check for some specific possible failure. It is better to have a few good tests that many many ineffective tests.
5. Run the three tests (below) to help you narrow down the bugs. Fix all the bugs you find.
   - Notice how each of these tests is targeted at one aspect of the program: each test is trying to break one thing.

	Name	**Score 1**	**Score 2**	**Purpose**	**Expected Outcome**
Test 1	Mary Bill Tom Q	80 70 80	80 80 91	Handles whole number results (80.0), and decimal results (85.5) And, works with multiple students.	80.0 75.0 85.5 Program quits.
Test 2	Sam Ted Q	-1, then 1 -1 then -2 then 1	101 then 99 200 then 500 then 99	Handles one and more than one bad input. Catches numbers just invalid (101), and way out of range (500).	50.0 50.0 Program quits.
Test 3	Bob q	0	100	Handles values at extreme of valid range.	50.0 Program quits.

Understanding:

- Explain why the initial 4 tests did not find any of the bugs.
- Explain how the second set of tests exercises different parts of the code than the first set.
- A few good, well thought-out test cases are better than 100 poorly thought out ones. Here we saw that 4 poorly thought out cases did not find any of the 5 bugs!
- When testing, you are searching for bugs. Try and break it!

4. Extra Challenges

- Create a program which keeps reading in integers from the user until the user enters -1.
  - Add a function which checks if all the digits of the integer are odd. Give it the header:
    `bool areAllDigitsOdd(int num);`
  - You may find the following code useful (requires `#using <cmath>`)

```
/*
     Return the number of digits in a number.
*/
int getNumberDigits(int num) {
     return log10((double)num) + 1;
}
/*
     Return the n'th digit from num.
*/
int getNthDigit(int num, int n) {
     int removedBottomDigits = num / pow(10.0, n-1);
     return removedBottomDigits % 10;
}
```

  - Your program might look like:

```
Enter your favourite number (-1 to exit): 1
All digits of your number are odd.
Enter your favourite number (-1 to exit): 2
All digits of your number are NOT odd.
Enter your favourite number (-1 to exit): 10
All digits of your number are NOT odd.
Enter your favourite number (-1 to exit): 11
All digits of your number are odd.
Enter your favourite number (-1 to exit): 13579
All digits of your number are odd.
Enter your favourite number (-1 to exit): 23579
All digits of your number are NOT odd.
Enter your favourite number (-1 to exit): -1
```

5. Skills and Understanding

You should now be able to answer all the "understanding" questions in the previous sections. Complete the following to get credit for the lab:

- Show the TA the following:
  - Your operational programs which complete all of the above tasks.
  - The TA may ask you to explain any section of the lab, or answer any of the "Understanding" questions.
- **Nothing** is to be submitted electronically or in hard-copy for this lab.

