Lab 7: More Functions and Variables

Directions

• While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments are individual work. You may not work on assignments during your lab time.
• If you are done early, please help your classmates.
• 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. Getting rid of global variables

Create a new project for the lab.

1. Copy the reVariable.cpp file provided on the course website into your project.
   - This file contains a small program which reads some information in from the user and displays it in a summary.
   - It works fine; however, the problem is it's written using a number of global variables. Read through the code and understand where each variable is used, and what it does.

2. Change the reVariable.cpp program to get rid of all global variables.
   - You may need to change the parameters and/or return types of functions.
   - Do not remove any of the current functions.
   - The program should seem identical to the user (same functionality).
   - Do not remove any of the global constants; they are fine.
   - Hints:
     - Any function which changes the value of a global variable should instead return that value.
       - When changing a function to return a value, you'll have to change all code which calls that function to “catch” the returned value and do something with it. This will likely be storing the returned value in a local variable (instead of a global variable).
     - Any function which requires the value from a global variable should instead be passed that value as a parameter.
     - In the end, each of the global variables will likely become a local variable inside a function. You might set it with the return value from a function, and then pass it as an argument into other functions.
     - If stuck, first look for global variables which are only used inside a single function. Switch those to local variables first.
3. **Understanding**
   - How to use local variables instead of global variables.
   - How to pass multiple values to a function.

4. **Optional**
   - Change the program to use prototypes and put the functions below `main()`.

---

### 2. Finding the perfect number *(Sorta Optional)*

Let's check if a number is "perfect". In math, a perfect number is one which equals the sum of its "proper" factors. For example, 6 has proper factors 1, 2, and 3; and as it happens $6 = 1 + 2 + 3$ so 6 is a perfect number. Another example is 12 with proper factors 1, 2, 3, 4, 6 but since $12 \neq 1 + 2 + 3 + 4 + 6$, it means 12 is not a perfect number.

For this activity, you are strongly encourage to work with another student. You can help each other write the code, and figure out how to make it work. **Plus, coding together is more fun!**

1. Overview of what we are going to do (detailed step-by-step instructions are below):
   - First we'll create a function to test for proper factors:
     ```cpp
     bool isProperFactor(int num, int divisor)
     ```
   - Then we'll sum up proper factors:
     ```cpp
     int sumProperFactors(int n)
     ```
   - Finally, we'll test if a number is perfect:
     ```cpp
     bool isPerfectNumber(int n)
     ```
   - These functions will call one another as needed!

2. Create a new file in your project name `perfect.cpp`.

3. Crate the following function:
   ```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: i.e.:
       ```cpp
       (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 $n$).

4. Write a couple test statements in `main()` to test `isProperFactor()`
   - First output to the screen if 3 is a proper factor of 6:
     ```cpp
     cout << "3 proper factor of 6? " << isProperFactor(6, 3) << endl;
     ```
   - Repeat the test for the following pairs:

---

1 “Sorta Optional” means “it covers skills you must be comfortable with for the assignments and exams, and will help you understand functions better; however, you may not have time to finish it in the lab.”
4 a proper factor of 6?
1 a proper factor of 6?
6 a proper factor of 6?

Your output should look similar to this:
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

5. **Exam Note**
You'll likely be asked to write functions on exams. For example, if you were asked to implement the `isProperFactor()` function on an exam, the question might read as follows:

Implement the `isProperFactor()` function which accepts two integers (int) in the order: 1) the number-to-test, and 2) the divisor. It returns `true` if both of the following are true:

- divisor < number-to-test, and
- the divisor divides evenly into the number-to-test.

Otherwise return `false`.

For example:

```
isProperFactor(30, 5) → returns true
isProperFactor(30, 7) → returns true
isProperFactor(8, 8) → returns false
```

- Note that you are asked to just implement the `isProperFactors()` function:
  - You would not do any input from the keyboard or output to the screen.
  - You would not write the code for the `main()` function.
  - Depending on how the question is worded, you may or may not have to write any necessary `#includes` (in this case, none are needed).
  - You likely wouldn't need to write comments during the exam unless told to do so.
  - As always, good variable and function names matter.

6. Create a function to sum up the proper factors of a number:

```cpp
int sumProperFactors(int n)
```

- Return the sum of all numbers between 1 and n which are proper factors of n.

- **Hints:**
  - Create a local variable to hold the sum of the values. Try naming it `sum`.
  - Create the loop. Try out a `for` loop because you know where it starts and stops.
  - In the loop, call the `isProperFactor()` function to check each number if it is a proper factor. If the number is a proper factor, add it to the sum.
  - **Return** the sum from the function.
7. In `main()`, prompt the user and read in an integer. This value will be used to see if it's a perfect number.
   - Hint: Name the variable something like `checkNum`.

8. In `main()`, print the sum of proper factors of `checkNum` by doing the following:
   - In `main()`, call `sumProperFactors()` passing in `checkNum`
   - In `main()`, display the sum to the screen. Complete output should now look like:
     - 3 proper factor of 6? 1
     - 4 proper factor of 6? 0
     - 1 proper factor of 6? 1
     - 6 proper factor of 6? 0
     - Enter a number to check: 15
     - Sum of proper factors: 9

9. Finally, create a function with the following header:
   ```cpp
   bool isPerfectNumber(int n)
   ```
   - Make this function return `true` if `n` is a perfect number (i.e. where `sumProperFactors(n) == n`); otherwise return `false`.
     - **Challenge:** How short and concise can you make this function’s implementation? Don’t make it cryptic, just make it concise!
   - Call `isPerfectNumber()` from `main()`, passing in the value the user entered. In `main()`, print a message stating if it is, or is not, a perfect number. Here are two sample outputs:
     - 3 proper factor of 6? 1
     - 4 proper factor of 6? 0
     - 1 proper factor of 6? 1
     - 6 proper factor of 6? 0
     - Enter a number to check: 15
     - Sum of proper factors: 9
     - I'm sorry, that's not a perfect number.
     - 3 proper factor of 6? 1
     - 4 proper factor of 6? 0
     - 1 proper factor of 6? 1
     - 6 proper factor of 6? 0
     - Enter a number to check: 6
     - Sum of proper factors: 6
     - Amazing! It's perfect!

10. Using your program, 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 10,000. (You should find 4, slowly...).
11. **Challenge**: How long does it take your program to find the 5th perfect number? Hint: Use `INT_MAX` from `climits` as the upper bounds for where to stop looking.

- You may want to have your program output something to the screen after every 100,000 numbers checked, just so you know it's alive.

- Extra challenge: Make your program display its status every 10 seconds to see what number it's checked. Hint: use `time(0)`, which gives the time in seconds.

- While it's running, look up online what number is the 5th perfect number. Guess how long it will be before your computer finds the answer.
  - Let your program run for a while; does it seem to speed up or slow down? Explain why it does this.
  - This highlights the need for efficient algorithms. Fast computers are not the answer: we need to compute smarter, not faster. It can make a HUGE difference.
  - For fun, compute how old you'll be when your computer finds the 6th, 7th, or 8th perfect number! Change your program to output how many numbers it checked in the last 10 seconds (count each number checked, every 10 seconds print the count and reset the count).

- How can you speed up the algorithms used here? Are there any optimizations you can think of?

12. **Understanding**

- Describe what your algorithm would look like if you implemented it without using functions. What is the advantage of using functions?

- What would happen if the function `isProperFactor()` *displayed* the answer instead of *returned* the answer?

- Inside `sumProperFactor()`, why is a for loop a good type of loop to use?

- Why are the following two lines of code interchangeable? Which is better? Why?
  ```java
  if (isProperFactor(n, i)) {...}
  if (isProperFactor(n, i) == true) {...}
  ```

---

**Nothing** is to be submitted electronically or in hard-copy for this lab unless you are completing the lab on your own time and submitting it via CourSys for marks.