MP1: Loops, Conditionals, Integer Math

Objectives:

- Write a program that uses integer arithmetic, conditional statements, and iterative statements.
- Compile, test, and debug your program.

Tasks:

1. Create a source code file, using Template 1 (main only).
2. Read a value from the user, and print it.
3. Use the debugger (gdb) to step through the program.
4. Change the program to print the factors of a number.

For mini-programs, you can work with other students, but every student will be graded individually, based on his/her own submitted work.

These instructions will assume that you are using the CLion integrated development environment. If you are using another platform, the steps will be slightly different, but you should be able to do everything described in this assignment. The C source code file should compile and run in any environment.

Task 1: Create a C Source Code File

In CLion, create a **New Project** (under the File menu). The type is "C Executable" and the Language Standard is "C90". Use **mp1** as the last part of the project location (replacing "untitled").

CLion will automatically create a file named main.c, and will put a "Hello, World!" print statement in the main function. If this is your first time using CLion, build and run the project to make sure your environment is set up properly. (Click on the green triangle at the top right of the window.) You should see "Hello, World!" printed in the output area.

Task 2: Get a number from the user, and print it

Inside the main function, delete the printf statement.

Write one or more statements to read a decimal integer value from the user's console. (Hint: scanf.) (Hint: You'll need to declare a variable in which to store the input value.) (Hint: Don't forget the "magical ampersand".)

Write another statement to print that value to the screen as a decimal integer, followed by a linefeed. (Hint: printf.)

Compile (build) your program without running it. (The build icon is in the top right corner of the window, with little 1's and 0's.) If you get errors or warnings, try to understand what they are telling you, and make corrections appropriately. If you click on the error, CLion will take you to the line where it thinks the error occurred. (Keep in mind that the real error might come earlier in the program, but this is where the compiler figured out something is wrong.)

Once it compilers, run the program. (Click on the green triangle: Run.)

When your program runs, the output area will be blank – it is waiting for you to type a number, followed by linefeed (Enter). Once you type a number and hit Enter, the program should read it, print it, and then exit.[1]

Task 4: Use the debugger

When you write a program, it is unlikely that you'll get it completely right the first time. It is critical that you know how to test and debug your program to make sure that it performs the desired function.

In CLion, as with most integrated development environments (IDEs), the debugger is integrated with the editor. (That's what the "I" in "IDE" means!) This makes it easy to set breakpoints, step through your code, and make changes to the program.

First, set a breakpoint. We'll stop the program after the number is read, but before it is printed. To do this, click in the gray lefthand margin right next to your printf statement. You should see a red circle appear, and the statement will be highlighted in pink.

Now, click the green arrow (Run). What happens? Nothing – the program runs normally and completely ignores the breakpoint. That's because you are in "run" mode and not in "debug" mode.

Click on the Debug icon – it looks like a green ladybug. Again the program runs, and waits for you to enter a number. Then, something very different happens. The printf statement is highlighted in blue, to show that the program has stopped executing at this point. Also, the display at the bottom changes.

The window on the right shows all the local variables and their values. If you were really debugging the program, seeing the values in local variables will be very helpful. You will often find that a variable's value is not what you expected, so your job is then to figure out why.

When you're ready to execute again, you have several options:

- To continue, click on the green triangle (Resume) at the bottom left. This will keep going from where the program stopped.

[1] By default, CLion will echo your input twice. To avoid this, turn off PTY in the Registry. (1) Go to Help -> Find Action -> Registry… (2) Uncheck the box next to **run.processes.with.pty**.

- To start the program over from the beginning (without recompiling), click on the Rerun icon just to the left of the Debugger label. (This is different from clicking the Run icon at the top right, because that will rebuild and run without entering debug mode.)
- Above the debug window, there are a number of different arrow that allow you to step through the program, statement by statement. On the left is Step Over – this will execute the current statement and stop. The next one is Step Into – if the current statement includes a function call, this one will stop at the first statement within that called function. You'll have time to try this out, and the other options, as you write more complex programs.
- If you want to quit running the program, without continuing, just click on the red box (Stop).

You can switch from the debugger view to the console view (where your program reads and prints data) by clicking on the Console and Debugger tabs.

Use the debugger to answer this question: If the user types a non-number (like *abc*) or a non-integer (like *123.45*), what happens to the variable in the scanf statement? Try a few different inputs (e.g., *ab33*, *123xyz*) and see if you can figure out what's going on.

Task 5: Modify your program to calculate the factors of the number

Change your program to do the following:

- Print a prompt for the user before waiting for the number. Something like "Enter a positive integer:".[2] For this program, assume that the user will enter a positive integer greater than 1. Your program does NOT need to work for any other kind of input.
- Instead of just printing the number, print a line that says "The factors of XX are:" where XX is replaced by the number that was entered by the user. Print a linefeed at the end of this string, so that the factors will start printing on the next line.
- Print all of the factors of the number, including 1 and the number itself. A factor is a value that divides evenly into the number. For example, the factors of 45 are 1, 3, 5, 9, 15, and 45. Print a single space (not a comma) between each factor. You can print them all on one line, no matter how many there are. You can print the factors in any order, as long as they are all there. It's even OK to print a factor twice (but no more than twice, please).

For example, here is a sample run of the program:

```
Enter a number: 45
The factors of 45 are:
1 3 5 9 15 45
```

[2] To make this work correctly, put the following statement after the printf statement: **fflush(stdout);** Without this statement, the system may wait to print the prompt until after you have entered the number. The fflush function forces the characters to be printed.

How do we check for "evenly divided"? Use the modulo operator (%), which gives the remainder after integer division. (What should the remainder be, if the first number is divisible by the second?)

NOTE: Do not Google code for factoring. Don't try to be very sophisticated -- a "brute force" check of all values between 2 and number-1 is fine. The point is to design and write your own code, not to solve the factoring problem in a clever way.

Compile and execute your program. Try different values to make sure that it behaves correctly in all scenarios.

Question: What should happen if the number is prime -- that is, there is no factor other than 1 and the number itself? Try a few prime numbers to see if your program does what you expect — for example, 113 is a prime number.

Question: Do we have to check every value between 2 and number-1? When can we stop checking and know that we've got all the factors?

Extra challenge (but not extra credit): Try to reduce the number of numbers that you have to check. See if it makes your program run faster for large values. (Hint: Only work on this after you have your program working. Submit the working program BEFORE working on this. If you're convinced that the program still works correctly, you can submit this version, but there's no need to submit it — the first working version of your program will be fine.)

Extra challenge (but not extra credit): Print only the prime factors of the number. (Do not submit this code. Only work on it if you have time, or you can try it after class.)

Submit your source code (**main.c**) to the Moodle MP1 assignment. Do not submit your executable program, just the source code.

Where is your **main.c** file? It is in the directory that you specified as the "location" when you created the new project. For example, all of my CLion projects are in this directory on my Windows laptop:

```
C:\Users\Greg\CLionProjects\
```

So the location of the main.c for this project is:

```
C:\Users\Greg\CLionProjects\mp1\main.c
```