Assignment #3

CSC111: Introduction to Programming

Learning Outcomes

When you have completed this assignment, you should understand:

- How to write and call functions that return values.
- How to write count driven loops
- How to accumulate results over a loop
- How to define functions with nested loops
- How to establish a correlation between the control of outer inner loops

Assignment Overview

1. Create a single new file called `assignment3.c` in Jupyter Hub
2. Design the functions according to the specifications in the Function Specifications section.
3. Test to make sure your code produces the expected output.
4. Ensure that the filename is `assignment3.c` (files with an incorrect name will not be marked).
5. Download your `assignment3.c` file from Jupyter Hub (File→Download) and submit this single file to the Assignment 3 dropbox (under Assignments) on the CSC111 BrightSpace Page.
6. Download your file from BrightSpace to confirm that name and file contents are as expected.

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee.

Grading

1. Late submissions will not be accepted
2. You must submit a single file with the name `assignment3.c` or you will be given a **zero grade**.
3. Your function names must match the specification **exactly** or you will be given a **zero grade**.
4. Any code written in the main function may not be marked.
5. We will do spot-check grading for code quality. All assignments are graded BUT only a subset of your code may be graded for code quality. You will not know which portions of the code will be graded, so all of your code must be complete and adhere to specifications to receive marks.
6. Your code must run without errors in the Computer Science department's Jupyter Hub environment using the compilation command provided in this document.
7. We reserve the right to replace your main function with our own, allowing it to better interact with our testing software.
8. It is the responsibility of the student to submit any and all correct files. Only submitted files will be marked. Submitting an incorrect file or forgetting a file is not grounds for a regrade.

Marks will be given for:

- Your code producing the correct output.
- Your code following good coding conventions
  - Proper indentation
  - Documentation (comments)
  - Use of whitespace to improve readability
  - Names of variables should have meaning relevant to what they store

Compiling and Running

Use the following commands to compile and run your code when you are testing your work.

```
gcc -Wall -Werror -pedantic -std=c18 -o assignment3 assignment3.c
```

and the following to run the code:

```
./assignment3
```

NOTE: Where examples are provided for clarity in the problem descriptions, they do not necessarily consider all edge cases – your testing must take into account all edge cases as we will when grading your submissions.

Functions Specifications

IMPORTANT: Be careful with your choice of types! All floating point return types and argument types **must** use type **double**. When the return type or argument type is clearly not a fractional number, you **must** use the type **int**.

When testing your code type the following command to compile your code. Every line of output must have a new line after it. This section will describe the functions that you are tasked to create:

1. Design a function, called `median`, that meets the following criteria:
   - The function must take as arguments three floating point numbers (doubles)
   - The function should **return** the median value computed as a double.
   - **RECALL:** the median of a set of numbers is the middle number of that set in sorted order.
   - **Example:** If the function is called with `median(3.2, 20.22, 3.2)` it must return: 3.2000.
   - **Note:** The definition of a median can be found here
2. Design a function called `is_multiple_of` that meets the following criteria:
   - The function must take two integers (in the following order): n1 and n2 as arguments.
   - The function must determine whether the second argument (n2) is a multiple of the first argument (n1).
   - If n2 is a multiple of n1, the function must return 1.
   - If n2 is not a multiple of n1, the function must return 0.
   - **Note:** The definition of a multiple can be found here
   - **Example:** If the function is called as: `is_multiple_of(3,12)`. 3 divides evenly into 12, 4 times, 12 is therefore a multiple of 3. The function returns 1.
3. Design a function called `permitted_to_register` that meets the following criteria:
   - The function must take two arguments (in the following order):, the student's sessional GPA and a value representing the student's standing (good standing will be a value of 1, representing true, and a value of 0 indicates false).
   - The function must return 1 if the student is permitted to register and 0 if they are not permitted to register.
   - The function must make sure that the GPA is a value between 0.0 and 9.0 (inclusive). If a value outside this range is passed, the function must return -1.
   - A student is permitted to register if their sessional GPA is at least 2.0
   - A student is permitted to register if their sessional GPA is less than 2.0 but they are in good standing.
   - If the student is not in good standing and their sessional GPA is less than 2.0 they are not permitted to register.
4. Design a function called `print_multiples` that meets the following criteria:
   - The function must take two arguments (in the following order): an integer (n) representing the number of multiples and an integer (num) to compute the multiples of.
   - The function must print n + 2 increasing multiples of num, separated by commas, starting at n.

- The first value in the result must be num.
- Your output must have a comma and space between each value, with no additional spaces or commas before or after the values. There must be **no** trailing comma.
- The output must end with a newline
- **Examples:**

Function Call	Result
`print_multiples(0,4)`	
`print_multiples(3,0)`	`0, 0, 0, 0, 0`
`print_multiples(3,4)`	`4, 8, 12, 16, 20`
`print_multiples(6,-5)`	`-5, 0, 5, 10, 15, 20, 25, 30`

Note: `print_multiples(0,4)` prints out nothing.

5. Design a function called `factorial_sum` that meets the following criteria:
   - The function must take one argument, an integer (n).
   - The function may assume that the integer passed will be larger than 0.
   - The function must return the sum of all of the factorial numbers from 0 to n (inclusive).
   - The factorial of n (i.e. n!) is computed as:
     $n! = n * (n-1) * (n-2) * (n-3) * ... * 2 * 1$
   - NOTE: $0! = 1$.
   - There are other ways to compute the factorial, but to receive full marks on this question your implementation must use the formula shown above.
   - RECALL: integer values have a limit, in testing this will put a limit on the highest value of n your function can take as an argument.
6. Design a function called `print_number_triangle` that meets the following criteria:
   - The function must take two arguments (in the following order): an integer max and an integer reps.
   - The function must print a right able triangle with the numbers, in order and separated by single spaces, across multiple lines. The width of the triangle will be controlled by the max argument.
   - The triangle will be repeated `reps` times.
   - **Examples:**

Function Call	Result
`print_number_triangle(0,4)`	0 0 0 0
`print_number_triangle(3,1)`	0 0 1 0 1 2 0 1 2 3
`print_number_triangle(4,2)`	0 0 1 0 1 2 0 1 2 3 0 1 2 3 4 0 0 1 0 1 2 0 1 2 3 0 1 2 3 4