Assignment #4

CSC111: Introduction to Programming

Learning Outcomes

When you have completed this assignment, you should understand:

- How to define functions with condition driven loops.
- How to define and call functions that take pointers are arguments.

Assignment Overview

1. Upload the provided file called `assignment4.c` to Jupyter Hub (Drag the file from your downloads into the file section of Jupyter Hub).
2. Compile the code, using the provided command, to ensure the upload was successful.
3. Design the functions according to the specifications in the Function Specifications section.
4. Test to make sure your code produces the expected output.
5. Ensure that the filename is `assignment4.c` (files with an incorrect name will not be marked).
6. Download your `assignment4.c` file from Jupyter Hub (File→Download) and submit this single file to the Assignment 4 dropbox (under Assignments) on the CSC111 BrightSpace Page.
7. 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 `assignment4.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 **compile and run, as submitted,** 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, as submitted, compiling without errors.
- 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. Please note the removal of the -Pedantic flag

```
gcc -Wall -Werror -std=c18 -o assignment4 assignment4.c
```

and the following to run the code:

```
./assignment4
```

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**.

NOTE: Where looping is required to solve the given problem, you must use a while loop. Solutions using for-loops will receive a zero grade. 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 `sum_fib_sequence`, that meets the following criteria:
   - The function must take as arguments a single integer value, limit.
   - The function can assume the given limit is not negative.
   - You may **not** assume that only numbers in the Fibonacci sequence will be used to test the function.
   - The function should **return** the sum of each value in the Fibonacci sequence, up to and including the given value (if it is part of the sequence), see example three.
   - **Note:** The definition of a Fibonacci sequence can be found here
   - **Examples:**

Function Call	Fibonacci Sequence Values	Result
`sum_fib_sequence(0)`	0	0
`sum_fib_sequence(21)`	0,1,1,2,3,5,8,13,21	54
`sum_fib_sequence(20)`	0,1,1,2,3,5,8,13	33

2. Design a function called `digit_product` that meets the following criteria:
   - The function must accept any valid integer (positive, or negative). If the value is negative, the negative sign should be ignored.
   - The function must return the product of each digit in the integer.
   - You should use the division and modulo operations to solve this problem. No marks will be given to solutions that convert the numbers to strings and access each digit of the string.
   - Hint: Work through an example:
     What is the result of: 432%10
     What is the result of: 432/10
     What is the result of: 43%10
     What is the result of: 43/10
   - **Examples:**

Function Call	Values	Result
`digit_product(0)`	0	0
`digit_product(432)`	4*3*2	24
`digit_product(-571)`	5*7*1	35

3. Design a function called `get_number` that meets the following criteria:
    - The function must take two arguments (in the following order): a maximum integer and a minimum integer.
    - The function must ask the user to enter numbers between the minimum and maximum values.
    - If the user enters a value larger than maximum or smaller than minimum the function should not accept the value and ask the user to enter another value.
    - If the user enters a value smaller than the maximum and larger than the minimum the function should return that value.
4. Design a function called `roll_dice_pair` that meets the following criteria:
    - The function simulates rolling a pair of dice, to generate two random numbers (from 1 to 6 inclusive).
    - The function must accept two pointers to integers that represent the locations that will store the result of the two dice that are being simulated.
    - The function must call the provided `roll_one_die` that is provided. This will generate random numbers to be stored at the addresses given to the function arguments.
    - NOTE: we have called srand for you in the main function provided. You DO NOT need to call it again in this function. See the comments in roll_one_die to help with testing versus running a realistic version of the function.
5. Design a function called `play_one_round` that meets the following criteria:
    - This function must simulate the playing one round of a two dice game, rules below.
    - The function must take as an argument an integer representing the number the user guesses will be rolled on one of the two dice.
    - Your function may assume the guess is a number from 1 to 6 inclusive.
    - The function must return 1 if the player won and 0 if they lost.
    - The function must print the output specified in the examples.
    - Rules:
        - A pair of dice are rolled for the player (using the already built functions)
        - If either dice rolled matches the guess, the round is automatically won.
        - If neither dice rolled matches the guess, and the two dice rolled are not equal, the round is automatically lost.
        - If the values of the two dice rolled are equal, the sum of the 2 dice values should be calculated, it is called the losing target. Only in this case is a second chance given.
        - In a second chance, the pair of dice continue to be rolled until either:
            * The sum of the dice rolled matches the losing target. In this case the player loses the round.
            * One of the dice matches the guess, in which case the round is won.
            * In the case both the sum of the dice rolled matches the losing target, and one of the dice matches the guess, the round is won.

The following sample runs are using a call to the function `play_one_round` with a guess of 4. These samples are provided for clarification purposes. Your function must work for all other expected inputs and possible dice rolls. You must NOT prompt the user for input anywhere within this function – doing so will result in our tester timing out and you will receive a zero grade.

Sample Run A, should return 1.

```
You guessed 4. Rolling!

Rolled 4 and 2.

You won!
```

Sample Run B, should return 1.

```
You guessed 4. Rolling!

Rolled 3 and 4.

You won!
```

Sample Run C, should return 1.

```
You guessed 4. Rolling!

Rolled 4 and 4.

You won!
```

Sample Run D, should return 0.

```
You guessed 4. Rolling!

Rolled 3 and 2.

You lost!
```

Sample Run E, should return 1.

```
You guessed 4. Rolling!

Rolled 3 and 3.

You are getting a second chance!

You want one of your dice to be 4.  You don't want the total to equal 6 or your second chance is over.

Rolled 4 and 1.

You won!
```

Sample Run F, should return 0.

```
You guessed 4. Rolling!

Rolled 3 and 3.

You are getting a second chance!

You want one of your dice to be 4.  You don't want the total to equal 6 or your second chance is over.

Rolled 3 and 6.

Rolled 6 and 2.

Rolled 1 and 5.

You lost!
```

Sample Run G, should return 1.

```
You guessed 4. Rolling!

Rolled 3 and 3.

You are getting a second chance!

You want one of your dice to be 4.  You don't want the total to equal 6 or your second chance is over.

Rolled 3 and 6.

Rolled 4 and 2.

You won!
```

Sample Run H, should return 1.

```
You guessed 4. Rolling!

Rolled 3 and 3.

You are getting a second chance!

You want one of your dice to be 4.  You don't want the total to equal 6 or your second chance is over.

Rolled 3 and 6.

Rolled 2 and 5.

Rolled 6 and 1.

Rolled 1 and 4.

You won!
```