CSc 352 (Fall 25): Assignment 1

Due Date: Friday, Sep12, 11:59PM

The purpose of this assignment is to get you started with writing some simple C programs.

For some of the problems below, you need to read in integer values from **stdin**. You've seen in class how you can use the library function **scanf()** to do this. Remember %d is the code to use in the format string to read a decimal (integer) number. (For example, **`scanf("%d", &num);`**)

General Requirements

1. Your C code should adhere to the coding standards for this class as listed in the codingstandards document found in the Misc Documents section on the Content tab of D2L.
2. Your programs should indicate if they executed without any problems via their *exit status*, i.e., the value returned by the program when it terminates:

Execution	Exit Status
Normal, no problems	**0**
Error or problem encountered	**1**

3. Under *bash* you can check the exit status of a command or program ***cmd*** by typing the command "**echo $?**" immediately after the execution of ***cmd***. A program can exit with status *n* by executing "**`exit(`***n*`)`**" anywhere in the program, or by having **`main()`** execute the statement "**`return(`***n*`)`**".
4. Remember your code will be graded on lectura using a grading script. You should test your code on lectura using the **`diff`** command to compare your output to that of the example executable.

Testing

Example executables of the programs will be made available. You should copy and run these programs on lectura to test your program's output and to answer questions you might have about how the program is supposed to operate. Our class has a home directory on lectura which is:

`/home/cs352/fall25`

You all have access to this directory. The example programs will always be in the appropriate **`assignments/assg#/prob#`** subdirectory of this directory. They will have the same name as the assigned program with "ex" added to the start and the capitalization changed to maintain camelback. So, for example, if the assigned program is **`theBigProgram`**, then the example executable will be named **`exTheBigProgram`**. You should use the appropriate UNIX commands to copy these executables to your own directory.

So, for this assignment, the paths to the example executables are:

```
/home/cs352/fall25/assignments/assg1/prob1/exCountPrimes
/home/cs352/fall25/assignments/assg1/prob2/exCalcFactorial
/home/cs352/fall25/assignments/assg1/prob3/exMirrorSum
```

Your program's output to **`stdout`** should match exactly the output to **`stdout`** of the example executable. You should test this by redirecting **`stdout`** from both your program and the example executables to a file, and then using the UNIX **`diff`** command to compare the outputs. There should be no differences. For example, you should do something like:

```
exCountPrimes <test1 >goodOut
countPrimes <test1 >myOut
diff -Z goodOut myOut
```

Of course, you should do this for multiple test cases.

If your program encounters an error, it should print an error message to **`stderr`**. The error message does NOT have to match the error message printed by the example program. It should just be somewhat informative of what the error was, and it must be printed to **`stderr`** and not **`stdout`**. When the grading script evaluates your program, it will only make sure that an error message was printed when an error occurs. It will not check the text of the message. So, when testing your code, just make sure that for any test case that causes the example executable to print to **`stderr`** also causes your program to print to **`stderr`**.

Lastly, don't forget to test your program's return code as specified in the prior section.

Submission Instructions

Your solutions are to be turned in on the host **lectura.cs.arizona.edu**. Since the assignment will be graded by a script, it is important you have the directory structure, and the names of the files exactly as specified in this document. Remember that UNIX is case sensitive, so make sure the capitalization is also correct. For all our assignments the directory structure should be as follows: The root directory will be named **assg#**, where # is the number of the current assignment. Inside that directory should be a subdirectory for each problem. These directories will be named **prob#** where # is the number of the problem within the assignment. Inside these directories there should

be any files required by the problem descriptions. For this assignment the directory structure should look like:

```
assg1
   prob1
      countPrimes.c
   prob2
      calcFactorial.c
   prob3
      mirrorSum.c
```

To submit your solutions, go to the directory containing your assg1 directory and use the following command:
turnin cs352f25-assg1 assg1

Problems:

Prob1: countPrimes

Write a program, in a file **countPrimes.c**, as specified below:

- **Behavior:** The program reads in two positive integers *m* and *n* from **`stdin`** and counts how many prime numbers p exist such that m < p < n. For example, if 2 and 9 are entered, the output would be 3 because there are three primes (3, 5, 7) between 2 and 9.

  If $m \geq n$ then the interval is empty and 0 is printed out.

- **Input:** Two positive integer values specifying, respectively, the beginning and end of an integer interval.

- **Output:** A single integer indicated the number of primes, printed out using the statement

  ```
  printf("%d\n", num)
  ```

  where **`num`** is the number of primes greater than the first number and less than the second number.

- **Error Conditions:** It is an error if your program cannot read two integers from **`stdin`** or if either of the values read is not positive. If an error occurs, your program should print an appropriate error message to **`stderr`** and return a 1 when it exits.

- **Hint:** You may find the function **`sqrt`** helpful for this program. You can use the man command to find information on this function. It is part of the math library. Using functions from this library in your program requires specifying to the compiler/linker that the math library needs to be linked in. You do this by adding "-lm" at the end of the compiler invocation: **gcc -Wall *`programName.c`* -lm** You don't have to use the sqrt function. The program can be written without it if you choose.

Prob2 calcFactorial

Write a program, in a file **`calcFactorial.c`** contained in the directory **`prob2`**, as specified below:

- **Program behavior:** The program reads in a sequence of 0 or more integer values between 0 and 12 (inclusive) from stdin until no more can be read in; and for each value N so read, prints out
  "**`N! = x`**" where N and x are replaced by the values of N and N! respectively.

- **Input format:** The input is a sequence of integers. These may or may not be on the same line. If you use

  ```
  scanf("%d", ...)
  ```

  it shouldn't matter whether they are on the same line or not.

  You can provide input to your program in a couple of different ways:

  1. put the input numbers in a file and use I/O redirection; or
  2. type in the numbers from the keyboard.

  In the latter case, use **`Ctrl-D`** to end the input stream (not **`Ctrl-C`**).

- **Error Conditions:** It is an error if the input contains integers smaller than 0 or larger than 12 or any non-integer values. Integers that are out of range shouldn't produce any output on stdout (but you should produce an error message on stderr). If the input contains something that cannot be read as an integer (i.e., **`scanf("%d", ...)`** is unable to read anything), your program should exit at that point.

  The exit status of your program should indicate whether or not an error was encountered at any point during processing.

  Example: Given the input sequence

  ```
  0 4 2
  ```

  the program should print out

  ```
  0! = 1
  4! = 24
  2! = 2
  ```

  Note if the input was coming from a keyboard, the program would produce output every time the Enter key is pressed.

Prob3: mirrorSum

The sum of the reverse of a positive integer is the sum of that number and the number formed by listing its digits in the reverse order. For example:

```
mirrorSum(2) = 4                (2 + 2)
mirrorSum(61) = 77              (61 + 16)
mirrorSum(7553190) = 8466747    (7553190 + 913557)
```

Write a program, in a file **`mirrorSum.c`** in the directory **`prob3`**, that behaves as specified below.

- **Input**: The input is a sequence of positive integers. These numbers may or may not be on the same line.
- **Behavior**: Your program should read in the input numbers, and for each number calculate the mirror sum as defined above. Your program should print out

  mirrorSum

  on separate lines where ***mirrorSum*** is the calculated number.
- **Error Conditions**: It is an error if the input contains any non-positive integers or any non-integer values. Non-positive values in the input should not produce any output. If the input contains something that cannot be read as an integer (i.e., **`scanf("%d", ...)`** is unable to read anything), your program should print an error to stderr and exit.

  The exit status of your program should indicate whether or not an error was encountered at any point during processing.
- **Assumptions**: You do not need to worry about the size of the numbers input. You may assume they are in valid range for integers.
- **Comments**: This is a straightforward problem of reading in numbers and calculating their reverse by performing a series of integer arithmetic operations on it. Your solution should not involve printing out the number into a string.

- **Example**: Given the input

  34 55 90

  the output should be
  77
  110
  99