CSc 352 (Fall 2018): Assignment 1

Due Date: 11:59PM Saturday, Sep 8

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 Documents section on the Resources tab for Piazza.
2. Your programs should indicate whether or not 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.
5. To get full points your code should compile without warnings or errors when the **`-Wall`** flag is set in **`gcc`**

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/fall18`

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/fall18/assignments/assg1/prob1/exIsFib
/home/cs352/fall18/assignments/assg1/prob2/exSumDigits
/home/cs352/fall18/assignments/assg1/prob3/exSumSquares
```

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:

```
exIsFib <test1  >goodOut
isFib <test1 >myOut
diff 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 programs 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 exact. 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 should be any files required by the problem descriptions. For this assignment the directory structure should look like:

```
assg1
   prob1
      isFib.c
   prob2
      sumDigits.c
   prob3
      sumSquares.c
```

To submit your solutions, go to the directory containing your assg1 directory and use the following command:
turnin cs352f18-assg1 assg1

Problems

prob1: isFib

A Fibonacci number is one that appears in the Fibonacci sequence (see https://en.wikipedia.org/wiki/Fibonacci_number). The Fibonacci sequence looks like:
1, 1, 2, 3, 5, 8, 13, 21, 34, . . .
where the next number in the sequence is obtained by adding the prior two numbers together ($F_{n+1} = F_n + F_{n-1}$). Some people include 0 in this sequence but we will ignore it. Our legal input is positive integers and so 0 is not a legal input.

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

- **Behavior:** The program reads in a sequence of 0 or more positive integer values from `stdin` until no more can be read in; and for each value *N* so read, prints out "*N* **is fib**" [use **printf**("%d is fib\n", n)] if *N* is a Fibonacci number, and "*N* **is not fib**" [use **printf**("%d is not fib\n", n)] if it is not. Nothing else should be sent to stdout.

- **Input:** The input is a sequence of positive 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 any non-positive integers or any non-integer values. Non-positive values in the input should not produce any output (but you should produce an error mesage on stderr), and then should continue reading input. If the input contains something that cannot be read as an integer, your program should print an error message to stderr and exit.

- The exit status of your program should indicate whether or not an error was encountered at any point during processing. If no error is encountered, the exit status should be 0. If an error is encountered, the exit status should be 1.

- **Assumptions**: You may assume the integers input are not too large to fit into a type **int**. (In other words, don't worry about overflow.) You may make this assumption about integers for all programs unless I specify otherwise. When I say you may "assume"

something it means that case will not be tested and that your program behavior in such cases may be anything.

- **Algorithm:** The algorithm for this program should be as follows:

  repeat

  1. read a number from **`stdin`**;
  2. if a number could be read, and the value read is positive: print out whether or not it is a Fibonacci number;

  until no more values can be read;

- **Example:** Given the input sequence

  1 12 4
  13
  34

  the program should print out

  1 is fib
  12 is not fib
  4 is not fib
  13 is fib
  34 is fib

prob2: sumDigits

This assignment you will write a program to input some integers and for each one calculate the sum of its digits. For example, given 8762 you would calculate 8 + 7 + 6 + 2 = 23

Write a program, in a file **sumDigits.c**, 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 n calculate the sum of its defined above. Your program should print out each of these sums on a separate line using the command

  ```
  printf("%d\n", num);
  ```
  where num is the sum of digits calculated. Nothing else should be sent to stdin!

- **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 to **stdout**, but anytime an error is encountered a message should be sent to **stderr**. A non-positive input should not cause the program to exit. If the input contains something that cannot be read as an integer, then your program should print an error message to stderr and exit.

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

Comments: This is a straightforward problem of reading in numbers and calculating the digits of each one (and then summing those digits) by performing a series of arithmetic operations on it. Your solution should <u>not</u> use strings or arrays.

prob3: sumSquares

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

- **Behavior:** The program reads in two positive integers *m* and *n* from **`stdin`** and prints out all the numbers in the interval **[***m*, *n***]** (i.e., *m* and *n* are included in the range of values considered) which can be written as the sum of two squares. A number N can be written as the sum of two squares if there exists two positive integers p and r such that $N = p^2 + r^2$. For example, 5 can be written as the sum of two squares since $5 = 1 + 4 = 1^2 + 2^2$. 6, on the other hand, cannot be written as the sum of two squares. Also notice that by this definition 4 is not the sum of two squares since neither p or r are allowed to be 0. (0 is not positive.)

  If *m* > *n* then the interval is empty and nothing is printed out.

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

- **Output:** A sequence of numbers in ascending order, one per line, printed out using the statement

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

  where **`num`** is a number that can be written as the sum of two squares which is being printed out. Notice that you don't have to print out the squares.

- **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 exit with the appropriate status value.

- **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**