Homework 2
Due 2021-03-25

CSSE/MA 335 Spring AY 2020/2021

Implement each program in C. For each program you must hand in the source code. Do not compress your files into a zip file; just submit the files to Moodle.

Note: Your code must compile with the command
`$ gcc` *filename*
where *filename* is the specified file name. No Makefiles.

Exercises

1. Write a program in `summer.c` that prompts the user for three numbers and outputs the sum.

2. Write a program in `u.c` that prompts the user for a non-negative integer N prints $u(N)$, where $u(N)$ is defined as

$$u(0) = 3$$
$$u(n+1) = 3u(n) + 4.$$

   Your program should re-prompt the user if they enter a negative number, you can assume the user will enter an integer.

3. Write a program called `pancakeanalyzer.c` that asks the user to enter the number of pancakes eaten for breakfast by 10 different people (Person 1, Person 2, ..., Person 10) Once the data has been entered the program must analyze the data and output which person ate the most and fewest pancakes for breakfast. Assume that the user inputs integers, and negative integers are fine.

4. Write a program in file `triangles.c` that prompts the user for a positive integer and prints a triangle of stars for output. For example, if $N = 1$ then print

   *

   if $N = 2$ then print

```
N=2
**
 *
```

if $N = 3$ then print

```
N=3
***
 **
  *
```

and so on. Assume the user will enter an integer, and re-prompt until the user enters a positive integer.

5. Modify `pancakeanalyzer` so that it outputs a list in order of number of pancakes eaten of all 10 people. Call the file `pancakeanlayzerpro.c`. Sort the data using your own bubble-sort implementation; do not use `qsort` or anything similar.

6. Write the "Guess my number" game in a file called `numberguess.c`. The user has a number in his or her head, the computer tries to guess the number. After each guess the computer asks the user if its guess is correct; if it is incorrect the computer asks if it guessed too low or too high. Stop when the guess is correct or the player has been shown to lie.

   Your code should accept 'H' or 'h' for high, 'L' or 'l' for low, and should re-prompt if none of those are entered. Your code should accept 'Y' or 'y' for yes and 'N' or 'n' for no. Your code should assume that the number is between 1 and 100 inclusive. Your code should follow the standard "bisection" approach for guessing the target number, and should exit if the player is proven to lie.

Testing Directions

Each problem has associated testing code, and an associated "standard". Your goal is to make your code match the output of the standard in every test case.

Each test case is a sequence of inputs stored in a text file. For example, the first test case for problem 1 is `summer_input_1.txt`, and the second test case is in `summer_input_2.txt`. By checking the input files you can see that the first input sequence is 1,2,3. The second input sequence is 1,-2,3, and so on.

You can check to see how your code behaves with the input in `summer_input_1.txt` by using *input redirection*, like this:

```
$ ./summer < summer_input_1.txt
```

The standard is found in `summer_standard`, you can test to see how it behaves with the same input with:

```
$ ./summer_standard < summer_input_1.txt
```

Finally, you can test all the inputs, and see whether your code is fully acceptable, by using the command

```
$ ./test_summer.py
```

That command will compare the output of your code and the standard code in all the test cases. It will compare the results character for character, ignoring whitespaces (tabs, multiple spaces, etc. don't matter). It then determines whether the outputs match, and makes a proclamation about the suitability of your code.

Testing all the other problems is nearly identical. You must compile your code to have the name specified in the problem, matching the standard code.