MA 335 Homework 3
Due 4/3

Spring AY 2019/2020

- You should submit two source files, `randomstuff.c` and `primestuff.c`.
- The first file, `randomstuff.c`, may contain a `main` function that you write for your own testing purposes. However, this `main` function will not be tested – only the assigned functions will be tested.
- The second file, `primestuff.c`, needs to have a `main` function that meets the requirements laid out. `main` and each of the assigned functions are all tested.
- There are three testing functions – `test_randomstuff.py`, which tests all the functions in `randomstuff.c` directly, without using your `main`, `test_primestuff_functions.py`, which tests the functions in `primestuff.c` without calling your `main`, and `test_primestuff.py`, which calls your `main` and compares its output (the text), to the output of `primestuff_soln`. Notice that it is possible that your `test_primestuff` will succeed, while your `test_primestuff_functions` does not succeed. **You must be successful with all three tests.**

All of these functions belong in a file named `randomstuff.c`.

1. Write a function `int countEven(int* arr, int size)` which receives an integer array and its size, and returns the number of even numbers in the array.
2. Write a function `double* maximum(double* a, int size)`. The input `double* a` is a pointer to an array of `double`s of length `size`. The return value is a pointer to the largest value in `a`.
3. Strings are terminated with the special character `'\0'`. Write a function `int myStrLen (char* s)` which returns the length of a properly terminated string. Write the function without using the function `strlen`.
4. Write a function `void revString(char* s)` which reverses the parameter `s`. The array `s` should actually be changed in this example.

5. Write a function `void delEven(int* arr,int size)` so that it sets all the even entries in `arr` to `-1`. It should not return anything, and it should actually modify the array pointed to by `arr`.

All of these functions belong in a file named `primestuff.c`

1. Write a function
   `int isprime(int n)`
   that returns 1 if the input integer n is prime, and 0 if it is not prime. Your function should indicate that numbers less than 2 are not prime, by definition. You do not need to use anything fancy to determine whether or not n is prime. For reference, % is the `mod` operator in C.

2. An integer m is a *Mersenne prime* if m is prime and there exists an integer p such that $m = 2^p - 1$. For example 3 is a Mersenne prime since 3 is prime and $3 = 2^2 - 1$. However, 2 is not a Mersenne prime, despite being prime.

   Write a function
   `int isMersenne(int n)`
   that returns 1 if n is a Mersenne prime, and 0 otherwise. Your code should call the `isprime` function written above.

3. Write a function
   ```
   void prime_info(int* ns,int l, int* num_prime,int* primes, int* num_mersenne
   , int* mersennes)
   ```
   That takes an array of integers, `ns`, and the length of that array `l`, and returns in `num_prime` the number of those integers that are prime, returns in `primes` the actual array of prime elements of `ns`, returns in `num_mersenne` the number of Mersenne primes in `ns` and returns in `mersennes` the actual list of Mersenne primes present in `ns`. Assume that the pointers `primes` and `mersennes` already point to arrays that are large enough to hold all the numbers you wish to store in them.

4. Add a `main` function to `primestuff.c`. The final program should take arguments from the command line, and interpret them thusly:

   - If the first argument is exactly `--list` then all following command line arguments are treated as an array, `prime_info` is called, and information regarding the result is printed.
   - If the first argument is exactly `--range` then the following two numbers are treated as a range, for example,
     `primestuff --range 10 15` should print the `prime_info` regarding the array `[10,11,12,13,14,15]`.
   - If anything else is the first argument, or if any necessary arguments are missing, then the program simply prints `See usage information.`

   You may use the `strlen` and/or `strcmp` and/or `strncmp` functions useful. In order to use `strlen`, `strcmp`, or `strncmp` you will need `#include<string.h>`.