Assignment 1 (10%)

You are given a 2D array of size **n*n**, where **n** is a positive non-zero number and **4 <= n <= 10**. Each number (say x) in the 2D array is a positive non-zero number such that **a <= x <=b** and **p** is the count of prime numbers on either side of **x**, we want to accomplish the following complex mathematical operation as follows.

*The sample data assumes **n**=4, **a**=10, **b**=99, **p**=5*

- Step 1. Read in the input (Say the 2D array is)

17	28	67	65
22	19	11	77
89	78	45	40
20	10	90	76

- Step 2. We fetch **p** positive prime numbers before & after **x**, and including **x**, such that we have the following set **px**
  - `For x=17; px={3,5,7,11,13,17, 19,23,29,31,37}`
  - `For x=28; px={11,13,17, 19,23,29,31,37,41,43}`
  - `For x=10; px={2,3,5,11,13,17,19,23}` *(Note: In this case, there are only 3 prime numbers before **x** and only these 3 have been included in **px**. However, we have included **p=5** prime numbers after **x** in the set **px.** Observe, we do not go less than 0 in the **px**.)*
- Step 3. Calculate average (ignore any decimal portion) of each **px**, say we call it **thapx**, such that now we have 1 **thapx** for each **x**.
  - `For x=17; px= {3,5,7,11,13,17, 19,23,29,31,37}; thapx=17`
  - `For x=28; px={11,13,17, 19,23,29,31,37,41,43}; thapx=26`
  - `For x=10; px={2,3,5,11,13,17,19,23}; thapx=11`
- Step 4. Calculate the average (ignore any decimal portion) of **n** such values of **thapx**, say we call it **wpapx**
- Step 5. Calculate the average of **n** such values of **wpapx,** say we call it **fapx**

The value of **fapx** is our desired result.

Write a **POSIX compliant** program in C where the above steps are accomplished in row major approach, with the following considerations.

1. The program should execute with the following arguments: <prog_name> n a b p e x1 x2 .. xn^2
    a. Example `./prog_name` **`4 10 99 5`** `17 28 67 65 22 19 11 77 89 78 45 40 20 10 90 76`
        i. In the above example, n is 4, a & b are 10 and 99 respectively and p is 5
        ii. The numbers are the values of x in the 2D array, where the count of such numbers is n^2, that is 16 in this case.
2. The process (say controller/parent/main) executed from <prog_name> will generate the n*n 2D array using the command line input. The program should validate the count of x and construct the n*n array, however the validation of individual x (that it is in the range of a and b) will be done at a later stage by the worker (child) process.
3. The controller will spawn **n** worker processes in a loop and have a 1-1 pipe communication link with the worker processes. Now, the controller will wait for all the **n** worker processes to finish execution.
4. Each worker process i is aware of the index i of the loop when it is forked and will be responsible for validating the contents of only row number i of the 2D array. Assume 0-based indexing in this case. If any of the row values is invalid, the worker process will report error and terminate.
5. Each worker process i after being spawned will do the following:
    a. Create **n** threads (say worker threads) where each thread will take a value **x** from row i and
        i. Create the set **px**, subject to the value of **p**.
        ii. Calculate the **thapx**
        iii. Report it back to the worker process and terminate the thread gracefully.
    b. The main thread of each worker process i will wait for all spawned worker threads to return the **thapx**.
    c. Once all **n** values of **thapx** are available, and all worker threads have joined, the main thread in the worker process i will calculate the **wpapx** and write it back to the controller process in the pipe.
6. Next, the controller which was waiting to read/receive **n** values of **wpapx** from the worker processes will get unblocked after receiving all the **n** values of **wpapx**.
7. If a worker process is terminated before reporting the **wpapx**, the same should be handled by the controller by handling the **SIGCHLD** signal. The controller then will report the error and kill and clean up all worker processes.
8. Finally, the controller process will calculate the average of **n** values of **wpapx** as **fapx** and report/print to the console the value of **fapx**.

The program should report the intermediate results/ progress at the following stages.

1. After reading the input.
2. After creating the pipe and the worker process.
3. After the worker process begins execution, it should report the row it is processing.
4. After creation of worker threads.
5. After the worker thread starts executing.
6. At each level of discovery of a prime number.
7. After all, px is calculated.
8. After thapx is calculated.
9. After thread termination / thread join.
10. After all thapx is calculated
11. After wpapx is calculated
12. After wpapx is written to the controller
13. After Controller captures a wpapx
14. After the controller calculates the fapx
15. All invalidation errors.

Marking scheme will be uploaded shortly.

Plagiarism Policy:

The source code submitted by each group will be run through a code plagiarism checker. If the similarity indices of the submitted source codes exceed certain thresholds (which will be decided and notified later), marks penalty will be imposed on the concerned groups irrespective of the correctness of the submitted solutions. Note that merely renaming variables and keeping the flow of logic same including the order of declaration of variables and data structures does not make your program is different from another one and ultimately adds up to the similarity index. Remember that if 2 groups are individually writing a program, it is highly improbable that a large portion of the code will be the same. If you are attempting the assignment honestly, you will not have to worry about the similarity percentages.

Note that penalty will be levied on both the source and destination groups.