Division of Engineering Science
Faculty of Applied Science & Engineering
University of Toronto – St. George
Semester: Winter 2014
CSC190H1 – Computer Algorithms and Data Structures

Lab 2: Week of Monday January 20th, 2014 – 0.5%
Due: Friday January 24th, 2014 @ 24:00

Objectives

Apply and augment your knowledge of the following C concepts: loops, reading from a file, arrays, structs.

Assumptions

You have basic knowledge of the following C concepts: `fprintf` and `fscanf` – see King sections 3.1, 3.2, 22.3), conditional statements (see King chapter 5), and loops (see King sections 6.1, 6.2, 6.3, 6.4), arrays (see King chapter 8), and `struct` (see King sections 16.1, 16.2)

The sieve of Eratosthenes [`q1.c`]

The "sieve of Eratosthenes" method is an algorithm that dates back to the 3rd century B. C., which prints all the prime numbers less than 1000. Implement the "sieve of Eratosthenes" algorithm in a C program. However, for your program, specify lower and upper inclusive bounds between 0 and 1000 (read these bounds from the keyboard; two separate entries). If the entries for the bounds are invalid, print "Invalid Bound Entries!" and terminate the program (`return 1`). Print all the prime numbers *between the lower bound and the upper bound inclusively*. Upon successful completion, your `main` function should `return 0`.

Your program must use an array consisting of the very simplest type of elements: 0 and 1 values. The goal of the algorithm is to set `a[i]` to 1 if `i` is prime and to 0 if it is not. It does so by, for each `i`, setting the array element corresponding to each multiple of `i` to 0, since any number that is a multiple of any other number cannot be prime. Then, is goes through the array once more, printing out the prime numbers *only*. The full array is first initialized with the value 1, indicating that all numbers are prime. The algorithm then sets to 0 array elements corresponding to positions `i` that are known to be nonprime.

HINTS: An array `a[upperBound+1]` should be declared in the `main()` function. To apply the algorithm, start with all the integers between `2` and `upperBound`. Ignore positions 0 and 1 of the array. Print all prime numbers between the `lowerBound` and `upperBound` one after the other (each number is formatted to take 4 spaces and there are no new lines). Below is an output example for a `lowerBound` of 40 and an `upperBound` of 345 (see sample output file on the portal):

```
 41  43  47  53  59  61  67  71  73  79  83  89  97 101 103 107
109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337
```

Files, Structures and Arrays [`q2.c`]

For this problem, you will read data from a file, store the data in structures, and print the content of the structures into a file. Go on the portal (in lab2) and download the file `input.txt` (do not modify the file). This file contains students' records. The file is formatted in the following way:

- The first line indicates the total number of records
- The records then follow. Each record contains the following information:
    - Student ID
    - Student age
    - Day of birth (e.g., 25)
    - Month of birth (e.g., 3)
    - Year of birth (e.g., 1991)
    - Number of courses the student took
    - Grades for the courses

Read all data from the `input.txt` file and store each record using an array of `struct`. Individual course grades must be stored in an array that will part of the student `struct` (you can assume the maximum number of courses for a student is 10). The student `struct` must also contain a `gpa` variable to store the calculated gpa of the student (sum of all courses divided by total number of courses).

Print the entire array of `struct` to a text file named `output.txt` (printing to the output file must be done by code in your C program, not simply by redirecting the outputs of your executable to a file).

The content of your `output.txt` file should look like this (see sample output file on the portal):

```
Student #1:

ID: 987637847
Age: 25
Birthday (DD-MM-YYYY): 5-7-1988
Number of courses: 5
GPA: 61.00


Student #2:

ID: 946913287
```

```
Age: 22
Birthday (DD-MM-YYYY): 25-3-1991
Number of courses: 8
GPA: 68.45


Student #3:

ID: 934286547
Age: 18
Birthday (DD-MM-YYYY): 17-1-1996
Number of courses: 3
GPA: 56.53
```