CS 420
PROGRAM 3P

Purpose

The purpose of these programs is for you to gain more familiarity with C language constructs and working with files programmatically. This submission is worth 100 points (50 points for each program).

Program 1

Filename to be submitted: **birthday.c**

(You will only write a function called birthdayGame() with the signature as indicated below in its prototype declaration. It will be tested by my main function. If you write a main() function and place it in this file for your own testing, you will need to comment it out when you submit it otherwise your program will not build. If your program does not build you get a zero for it).

Simulations that involve the repeated use of a random number generator to reproduce a probabilistic event are called Monte Carlo simulations, so called because Monte Carlo has some of the world's most famous gaming casinos. We can use this technique to find the break-even point in a variation of the birthday game. In a roomful of people, at least two of them can be expected to have birthdays on the same day of the year. We wish to find the probability that any two people in a room with ***n*** people will have been born in the same day. Simulate the probability by running several hundreds of thousands of trials (configurable) with 2, 3, … 50 …up to 100 people (configurable) in the room. When any variable (could be an array element) gets to two, that trial is true meaning that at least two people in the room were born on the same day.
To find a break-even point we should see the "n" value (people) after which there is no appreciable difference of the probability going any higher. The "estimated probability" is simply the number-of-true-trials/number-of-trials. You don't need to show this estimated probability as a percentage.

In order to simulate the birthday of a person you will generate a random number for the day of the year. For the purposes of this program assume that we are working only with non-leap years i.e. a year has 365 days.

Extend this to also during the same trial to simulate the probability for the same month instead of same day (must be done with the same random number generated for the birthday i.e. you should be generating only one set of random numbers per trial).

The output from the program is from my main() function, for me to display the estimated probability for both birthday and birth month that you are going to compute and return from your function. *You don't need to figure out what the break-even point is, that's something you can experiment on your own if you are curious. We will check this with one of our tests.*

Prototype declaration of function birthdayGame:
*double birthdayGame(int numberOfTrials, int numberOfPeople, double *monthProbability);*

numberOfTrials – input argument (int), total number of trials
numberOfPeople – input argument (int), total number of people in the room
*monthProbability – output argument (double *), estimated probability of having a birthday on the same month*
return value (double) – estimated probability of having the same birthday

If the input arguments are not positive integers, the return value and output argument should return -1

Sample runs:

```
Trials: 100000, People: 7
Estimated probability for the same birthday: 0.055860
Estimated probability for the same birth month: 0.888160
```

```
Trials: 100000, People: 10
Estimated probability for the same birthday: 0.116980
Estimated probability for the same birth month: 0.995930
```

```
Trials: 200000, People: 39
Estimated probability for the same birthday: 0.877960
Estimated probability for the same birth month: 1.000000
```

Program 2

Filenames to be submitted: **stats.c, stats_functions.c**

stats.h is available in the public folder, do not modify it but include it in both your source files.

To build (compile and link) both your source files and create an executable, use the following:
gcc stats.c stats_functions.c -o stats

This will create an executable called **stats**

The program is invoked as shown below from the command line with one or more file names passed as command line arguments:

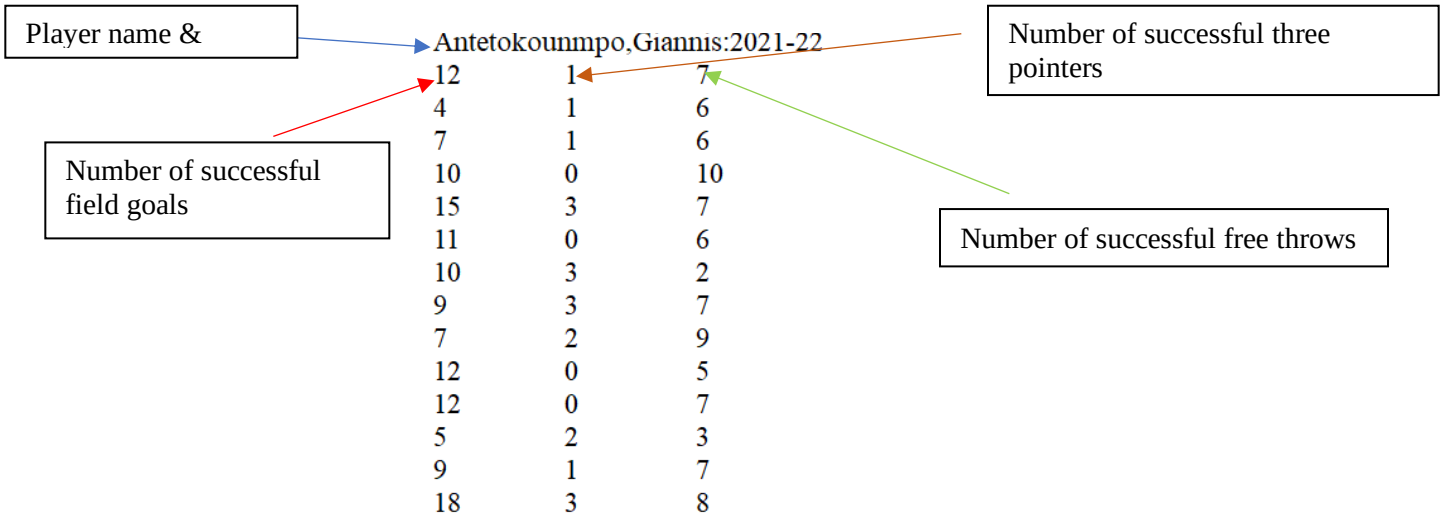
./stats in1.txt

./stats in2.txt in1.txt

The objective is to read **programmatically** from a data file (**not using input redirection but using a file pointer to open and read from a file within your program**) which contains scoring statistics for NBA players for a specific season, and perform some basic computations with it.

The first line of the input file contains information about the player and the season. Following this are several rows of data given to you in 3 columns per row, each row represents a game played in the season by the player. The first column indicates number of successful field goals, the second column is the number of successful three point plays, and the third column is the number of successful free throws.

Here's a sample of an input file:




```
Antetokounmpo,Giannis:2021-22
12      1       7
4       1       6
7       1       6
10      0       10
15      3       7
11      0       6
10      3       2
9       3       7
7       2       9
12      0       5
12      0       7
5       2       3
9       1       7
18      3       8
```

The goal is to read all input data from the file into a few data structures and come up with some scoring statistics. You can assume that the input file format is the same for any player, and the data is error free. The maximum number of games in a season is 82 so if the input file has more than 82 rows of scoring data you need to ignore the rest.

The results to be printed are:

- Information about player and season
- Average field goals per game for the season
- Average three pointers per game for the season
- Average free throws per game for the season
- Average points per game for the season (Points make-up: field goal 2 points, three pointer 3 points, free throw 1 point)

You are supplied with a header file called stats.h that contains a constant (for maximum number of games in a season) and some prototype declarations for functions that you will need to define in stats_functions.c. The source file stats.c must contain only your main() function which supports command-line arguments.

The data structures you will need to use are three single dimensional arrays, one for each column (each of the same size – use the the constant in your header file stats.h). The three arrays are declared in your main() function.The main() function extracts the filename from the command line argument and calls the function read_data() with the appropriate arguments to set up the three arrays and also returns the player name and actual number of games played. Then the main() function calls the compute_stats() function by supplying the necessary arguments and returning with all the results that are eventually printed in the main() function. **No printing must be done anywhere else other than your main() function**.

Let's keep the output of your program simple as shown below:

```
Statistics for: Antetokounmpo,Giannis:2021-22
Total games played: 67
Season Averages:
Field Goals: 10.28    3 Points: 1.06        Free Throws: 8.25    Points per game: 32.00
```

If there are multiple files supplied on the command line your program has to process one file after another and the output results are displayed one below the other:

Statistics for: Tatum,Jayson:2021-22
Total games played: 76
Season Averages:
Field Goals: 9.32 3 Pointers: 3.03 Free Throws: 5.26 Points per game: 32.97

Statistics for: Antetokounmpo,Giannis:2021-22
Total games played: 67
Season Averages:
Field Goals: 10.28 3 Points: 1.06 Free Throws: 8.25 Points per game: 32.00

stats.c: contains only main()
stats_functions.c: contains function definitions for
read_data()
compute_stats()

Look at the two prototype declarations that are present in the supplied file stats.h. Understand what the signature of these functions are. stats.h should be included in both your .c source files. You cannot modify stats.h, and you don't submit it either. **Keep in mind that you never include .c files in other .c files (only .h files are included).**

1. To read data:
   Prototype declaration of function read_data():
   *int read_data(char *filename, char player[], int field_goals[], int three_pts[], int free_throws[]);*
   filename – input argument – character string, input file name (pointer to character is similar to character array)
   player – output argument – character array, holding player name and season
   field_goals – output argument, integer array holding number of field goals
   three_pts – output argument, integer array holding number of three pointers
   free_throws – output argument, integer array holding number of free throws

   function return value – integer – number of games played

2. To compute stats
   Prototype declaration of function compute_stats():
   *float compute_stats(int field_goals[], int three_pts[], int free_throws[], int size, float *avg_fg, float *avg_tpt, float *avg_ft);*

   field_goals – input argument, integer array holding number of field goals
   three_pts – input argument, integer array holding number of three pointers
   free_throws – input argument, integer array holding number of free throws
   size – input argument, size of the arrays (related to first three parameters)
   avg_fg – output argument, average field goals per game for the season
   avg_tpt – output argument, average three pointers per game for the season
   avg_ft – output argument, average free throws per game for the season

   function return value – average total points per game for the season

Grade Key

Birthday: birth day probability accuracy	**20**
Birthday: birth month probability accuracy (same random numbers as for birth day, otherwise -4)	**8**
Birthday: Input validation	**6**
Birthday: Finding break-even point indicates correct progression of probability	**16**
Birthday: Multiple sets of random numbers generated per trial (for computing both birth day and month probabilities)	**-10**
Stats: player name and season	**6**
Stats: total games played	**6**
Stats: average field goals per game	**6**
Stats: average three pointers per game	**6**
Stats: average free throws per game	**6**
Stats: average total points per game	**6**
Stats: command line arguments implemented (3), multiple files on command line processed correctly (7)	**10**
Stats: output format reasonable	**4**