Computer Science 352 Spring 2023
Programming Assignment 5
Due 3/3/2023 by 7pm

This PA for CS 352 has 2 parts, and part 2 depends on code from part 1. The first part will require you to write a simple C library for helping with arrays. For this, you should name the files **arrayz.h** and **arrayz.c**. The next part will require you to write a C program named **basketball.c** that will rely on the arrayz library. This program will be responsible for doing some analysis on basketball based on content from a data file.

For the project, you should end up with the following directory / file structure:

```
pa5
├── test.c   (optional)
├── makefile
├── arrayz.c
├── arrayz.h
└── basketball.c
```

The **makefile** should have three rules: one for **arrayz.o**, one for **basketball**, and one for **clean**. The **arrayz.o** rule should compile **arrayz.c** with the required gcc flags and produce a library file named **arrayz.o** in the current working directory. The **basketball** rule should compile **basketball.c** with the required gcc flags and produce an executable named **basketball** in the current working directory. The **clean** rule should delete the **arrayz.o** and **basketball** files from the current working directory.

Arrayz

For this part of the assignment, you must implement a C library named arrayz (**array.h** for the header information, **array.c** for the implementation). The purpose of this library is to have a collection of functions that can perform calculations on C arrays. For this library, you should implement the following:

- **long sum(long values[]);** - Return the sum of every element in **values**
- **long minil(long values[]);** - Return the index of the minimum value in **values**
- **long minid(double values[]);** - Return the index of the minimum value in **values**
- **long maxil(long values[]);** - Return the index of the maximum value in **values**
- **long maxid(double values[]);** - Return the index of the maximum value in **values**
- **void printal(long values[]);** - Print the elements in **values**
    - Formatted as: **length ARRAY_LENGTH array: EL1, EL2, ..., ELN**
- **void printad(double values[]);** - Print the elements in **values**
    - Formatted as: **length ARRAY_LENGTH array: EL1, EL2, ..., ELN**
- **double mean(long values[]);** - Return the mean (average) of the elements in **values**
    - Note: return a double, the result may not be a whole number.
- **double sdev(long values[]);** - Return the standard deviation of the elements in **values**
    - If you don't know what standard deviation is, or how to calculate it, see this site: https://www.mathsisfun.com/data/standard-deviation-formulas.html

You should add each of these function ***declarations*** into the header file named **`arrayz.h`**. It is also up to you to ensure that you have a clear comment in the header file for each of these functions. You may add more functions if you want, but you must have at least these. You should then **`#include`** this header file into a file named **`arrayz.c`**, which is where you should implement each function.

For all of these functions, you should assume that the first element in the array is used to store the number of remaining elements in the array. For example, say we wanted to pass the data **20, 25, 20, 30** into one of these functions. The actual array should be length 5, and should have as its content: **`{4, 20, 25, 20, 30}`**. The 4 as the first element represents the number of elements in the array following. This is used so that the functions can avoid over-writing or accessing memory that it should not.

The autograder may have tests for some or all of these individual functions, you should implement and test each thoroughly. If you want, you can create another file called **`test.c`** and put code in there to test the library.

Basketball Stats

For the next part, you should write a C program named **`basketball.c`**. This program will generate a report about NBA basketball players, based on data that it will read in from a text file. This program can and should use several functions from the arrayz library. In particular, it must at least use the **`min`** / **`max`** functions, the **`mean`** function, and the **`sdev`** function. You may also want to use some of the print functions from the library as you develop and debug your program.

This program should expect to have *exactly* one command-line argument, that being the path (absolute or relative) to the file that the program will open and process. If the program does get exactly one argument, it should print **`“expects 1 command line argument”`** to standard error and then return a nonzero. The program should expect that the file will be a text file with zero or more lines, and each line will be formatted as follows:

```
PLAYER_NAME[P1, R1, A1][P2, R2, A2],...,[PN, RN, AN]
```

The program is allowed to expect that the file is correctly formatted. It does not have to handle incorrectly formatted data. The **PLAYER_NAME** will be the name of the NBA player, first and last, separated by a space. Immediately following the player will be a sequence of three numbers within brackets. The three numbers represent that player's points scored, rebounds, and assists for one game played. Thus, each triple of numbers represents a simplified stat line from one game.

A simple example of what in input file’s contents could look like:

```
Devin Booker[25,5,7][27,8,5][29,5,5][28,3,10]
James Harden[10,2,4][32,8,10][30,13,3][40,5,15]
Deandre Ayton[10,15,2][15,10,3][20,9,1][25,15,5]
Chris Paul[10,5,15][20,4,14][17,2,15][15,5,13]
```

A few other things you may assume about these input files:

- A player name will not be longer than 50 characters.
- A player will never have 100 or more points, rebounds, or assists in a single game. In other words, the points, rebounds, and assists will always be a 1 or 2 digit number. (Wilt Chamberlain breaks this rule)
- A given line will not have more than 100 games on it
- The file will contain 100 players (lines) or less

As a part of the program, you should open up the input file and read through it line-by-line. As you read through the stats for each player, you should calculate the mean (average) and standard deviation for each stat category (sdev points, sdev rebounds, sdev assists, mean points, mean rebounds, mean assists) and store these into an array for future use.

You should calculate these values for every player in the file, and as you go you should store the results for each player into various array(s). After you have done so, you will need to determine the player the meets the criteria for the following twelve categories:

- The ***most*** consistent scorer, rebounder, and assister (The players with ***lowest*** standard deviation in these categories)
- The ***least*** consistent scorer, rebounder, and assister (The players with ***highest*** standard deviation in these categories)
- The ***best*** scorer, rebounder, and assister (players with the ***highest*** average in these categories)
- The ***worst*** scorer, rebounder, and assister (players with the ***lowest*** average in these categories)

Say that we have a file named **`players.data`** that contains the example player content shown earlier in the spec with Devin Booker, James Harden, Deandre Ayton, and Chris Paul. If we were to implement the program correctly and then run it with this data file, we should get the following:

```
$ ./basketball players.data
most consistent scorer: Devin Booker
most inconsistent scorer: James Harden
highest scorer: James Harden
lowest scorer: Chris Paul
most inconsistent rebounder: James Harden
most consistent rebounder: Chris Paul
highest rebounder: Deandre Ayton
lowest rebounder: Chris Paul
most inconsistent assister: James Harden
most consistent assister: Chris Paul
lowest assister: Deandre Ayton
highest assister: Chris Paul
```

Every time the program is run with an input file that has at least one player in it, it should print out exactly 12 lines of output. The formatting of each line must match what is shown in this spec, though the ordering does not matter. It is your responsibility to ensure that your program is well-tested with various input files.

There will be a small number of test cases visible before grades are published. You should do your best to pass each one. Also keep in mind that there will be some test cases that get revealed after it is graded, so you should try your best to test your code thoroughly.

Compiling

This PA is a bit unique since you are going to be writing a library, and then using the functionality from that library in your basketball program. Due to this, compilation can happen in two separate steps. The first step is to compile the **arrayz.c** file with a special flag, -c. This option tells the compiler to compile and assemble the program, but to skip the **linking** step. So, you should run:

```
$ gcc -Wall -Werror -std=c11 -c arrayz.c
```

This should produce a file named **`arrayz.o`**. After this file has been generated, you can compile basketball.c and link it with the **`arrayz.o`** library:

```
$ gcc -Wall -Werror -std=c11 -o basketball basketball.c arrayz.o -lm
```

As long as there were no errors, you should have a file named **basketball** in the current directory that is the executable file for the basketball program. You can run **`./basketball file_name`** to test it out!

Submitting Your PA

After you have completed PA, double check that the file structure and file / directory names match what is shown in the project overview on the first page. Before you submit, you should ensure that your code compiles and runs correctly on lectura, ensure you follow the rules from the style guide, and remove all files from the pa5 directory and subdirectories other than the ones shown on the first page of this spec. Do not includes test files, executable files, and other file types. If you are on a mac, you should avoid zipping the file on the mac because it might include one or more hidden / extra files. Instead, zip on lectura.

Once you are ready to submit, zip up the **`pa5`** directory by running:

```
$ zip -r pa5.zip ./pa5
```

Then, turn this file to the PA 5 dropbox on gradescope. There will be some test cases that will not be visible until after the grades get published. Test your program thoroughly!