# CS 3335: C Programming
Project - 3

## Submission Guidelines

- Programs must utilize proper and consistent brace style and indention style.

- Programs must be easy to read. In other words, use blank lines and white space as appropriate to make code read nicely.

- Programs must be commented and must use meaningful variable names.

Distribution of points:

- 5% for good programming practices like indentation, comments, good variable names, consistent bracing etc.
- 10% for compilation
- 10% for execution with a good representation of inputs
- 10% for correct main function
- 10% for correctly defining the structs
- 15% for correct readFile function
- 15% for correct computeLengths function
- 15% for correct saveLengths function
- 10% using dynamic memory allocation

- Extra Credit:
    - 15% for correct printStats function

**I want one submission:**

**a. Submit the program (only the source file, for example – LastName_FirstName_project2.c) via BlazeVIEW.**

**Problem:**

In this program you will be reading from and writing to files.

**All files used are text files. Please do NOT write code for binary files**

Your task is to read a file that has the following format:

---

File format:

5

10 20 30 40

3 50 20 4

5 4 6 7

10 15 20 10

20 10 4 5

The first line of the file stores the number of lines represented by the file. The first line of each line represents the information about the number of geometric lines for which information is stored in the file. Each subsequent line stores the following information:

x y x1 y1

where x is the x co-ordinate of the first point of the line, y is the y coordinate of the first point of the line, x1 is the x co-ordinate of the second point of the line, y1 is the y coordinate of the second point of the line.

**Main Assignment:**

- Define two structs: Point and Line, ○ Point stores the x and y coordinates (integers) of points and is used to create variables of type Point
  - Line is made up of two Points: a and b and a float: length.

- Define a function **readFile** that has the following signature:

  ```
  struct Line * readFile(char file_name[], int *n)
  ```

  - The function reads the number of lines (first line of the file) into the variable pointed to by `n`, allocates memory dynamically by allocating an array of type Line struct, and prints "Array of size x created".
  - Int *n is a pointer to the variable that stores the number of elements.
  - It then reads each subsequent line of the file and stores the information of each Line as an element of the array.
  - It prints "Data read" and returns a pointer to the first element of the array

- Define a function **computeLengths** that has the following signature:

  ```
  void computeLengths(struct Line *lines, int n)
  ```

  - The argument is a pointer to the first element of the array and number of array elements.
  - The function first prints "Computing lengths…"
  - This function computes the length of each Line in the array and stores it in Line.length for each Line variable in the array.
  - It then prints "Length computation completed"

- Define a function **saveLengths** that has the following signature:

  ```
  void saveLengths(struct Line* lines, int n)
  ```

  - The argument is a pointer to the first element of the array and number of array elements.
  - The function first prints "Saving lengths…"
  - This function prints the lengths of the lines upto 1 decimal place (in the order in which lines were read from the file) to a file named lengths_LastName_firstName.txt. Print one Line per line of the file. ○ Example:

    ```
    28.3
    49.0
    3.2
    11.2
    16.8
    ```

- Your main function should do the following:

○ Take the name of the input file as a command-line argument ○ Call the function readFile ○ Call the function computeLengths ○ Call the function saveLengths

○ If you are implementing the extra credit option

 Call function printStats (Print “printStats not implemented” if you have not done the extra credit)

## Extra Credit

• Define a function **printStats** that has the following signature:

```
void printStats(struct Line* lines, int n)
```

○ **Assume that there is a difference of 0.5 between any two Line lengths and no duplicates**

○ Compute and print to terminal the length of the longest line ○ Compute and print to terminal the length of the shortest line

○ Compute and print to terminal the average of the lengths of all the Lines in the array

A sample run of the code is below – **Sample Run 1 using the above sample file:**

```
Array of size 5 created

Data read

Computing lengths…

Length computation completed

Saving lengths… printStats not

implemented
```

**Sample Run 2 using the above sample file (EC option):**

```
Array of size 5 created

Data read

Computing lengths…

Length computation completed
```

```
Saving lengths…

Line of max length: 49.0

Line    with    min    length:    3.2

Average line length: 21.7
```

**Sample Input file:**

```
5

10 20 30 40

3 50 20 4

5 4 6 7

10 15 20 10

20 10 4 5
```

**Sample lengths_LastName_firstName.txt:**

```
28.3
49.0
3.2
11.2
16.8
```