Project 3

150 – 200 lines of code

**Sports analysis: With Structs!**

Since writing your program to analyze the results of [sports/debate/robotics] events, you’ve decided you want to be able to keep track of team member statistics throughout the season. You have the idea of creating a struct that will store information for each team member. Then your program will do the following:

1. Read the existing team member data from a file
2. Read an update file and combine it into the initial data.
3. Output a new data file.

**Learning Objectives covered:**

1. I can write a program that adheres to a style guide.
2. I can write high quality documentation.
3. I can write a program that reads from files and writes formatted output to files.
4. I can write programs using 1D and 2D arrays with functions.
5. I can write a program that uses command line arguments.
6. I can write and use a makefile.
7. I can write a program that defines a struct and uses it in functions.
8. I can write a program that allocates memory on the heap (including arrays and structs).

**New Details:**

**Program structure:**

* analyze\_team.c
  + main()
* team\_member.c
  + 2 or more functions
* team\_member.h
  + Defines the TeamMember struct
  + Contains documentation and prototypes
* makefile

Before you start programming, fully read the program description and outline your program using pseudocode. You must separate the program into multiple functions.

**Compiling and running:**

Your program will be compiled using your makefile.

Your program should be run with the following command line arguments:

./teamData oldData.txt newData.txt outName.txt numStats numPlayers

* oldData.txt: file containing team data
* newData.txt: new team data to add
* outName.txt: where to save the data
* numStats: Number of columns of stats in both data files (does not include name and ID columns)
* numPlayers: number of team members in oldData.txt

**The TeamMember struct:**

Define a struct that will store data for each member of the team. Your struct **must** have the following member variables:

* int ID; // Team member #
* char lastName[50]; //Team member last name
* char firstName[50]; // Team member first name
* float\* memberData; // Will point to an array of data for the team member

**Storing team members:**

Store all team members in an array of TeamMember structs, such as the following:

TeamMember theTeam[numPlayers];

**Input files:**

Each input file is formatted as follows:

ID FirstName LastName points catches throws games

1 Jennifer Mortensen 2 6 6 1

2 Joe Smith 4 7 12 1

Where the first 3 columns have the ID, first and last name and the remaining columns have whatever stats are stored for the team.

**Output:**

Your output file should look just like the input files, but with updated stats data.

**Additional details:**

1. Check that the user has included enough command line arguments. If the user uses incorrect command line arguments, print the following:

Missing command line arguments

./teamData oldData.txt newData.txt outName.txt numStats numPlayers

1. Not all team members will be included in newData.txt.
2. The columns in newData.txt and oldData.txt will be the same.
3. After creating an array of TeamMember structs, allocate memory for each person’s memberData on the and storing the pointer in memberData in the struct
4. Read the first line of the input file into a character array. Keep that array to print to the first line of the output file.

**Example functions:**

If you are struggling to separate the tasks into functions, here are some ideas.

// load data from the file at name into theTeam

char\* load\_data(int size, TeamMember theTeam[], char\* name, int numStats);

// add the data from the file at name into theTeam

char\* add\_data(int size, TeamMember theTeam[], char\* name, int numStats);

// print the information in theTeam to the file at fileName

void print\_data(int size, TeamMember theTeam[], char\* fileName, int numStats, char\* firstLine);

### **Deliverables:**

1. **README.txt** that describes the function of your program, how to compile and how to run it. If you used any outside resources for this assignment, be sure to cite your sources *and* explain in detail how the code works. An example is on Canvas.
2. All source code and header files needed to compile your program. You must have at least 2 .c files and at least 1 .h file.
3. Your makefile which will be used to compile your program. The executable must be called teamData

**Restricted material:**

You may not hard-code the output of the program and you may not use any of the following programming concepts in this assignment: - global variables - static

**Grading:**

The full grading rubric is explained in the CS 2303 Project Guidelines document. Five additional rubric items are listed below.

|  |  |
| --- | --- |
| **Met** | **Rubric Item** |
|  | Program does not contain restricted items |
|  | Makefile contains 2 or more keywords |
|  | Program defines the TeamMember struct as required |
|  | Function prototypes are in the .h file and the .h file included correctly in the .c files |
|  | Program uses malloc or calloc to place information on the heap |

The grading breakdown is shown in the following table. To earn a particular grade, you must meet all criteria in the row corresponding to that grade. There are 7 core rubric items and 10 supplemental items (5 in the Rubric document and 5 listed above).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Rubric Items Met | |
| **Autograder tests** | **Core** | **Supplemental** |
| Excellent | All Passed | 7 | 9+ |
| Meets Expectations | All Passed | 7 | 8+ |
| Needs Revision | All Passed | N/A | N/A |
| Not Graded | Failed 1 or More test | N/A | N/A |