Project 4

150 – 200 lines of code

**Sports analysis in C++!**

Remember the program you wrote analyzing the team member statistics using structs? In this project you will write a similar program in C++, where each team member is part of the Player class. Your class will have member functions for setting and updating the player stats. And 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 and compile C++ programs that communicate with the terminal and with files.
2. I can write a program that adheres to a style guide.
3. I can write high quality documentation.
4. I can write a class in C++ with constructors and operator overloading.
5. I can write C++ code that uses auto and arrays.

**New Details:**

**Program structure:**

* analyze\_team.cpp
  + main()
* Player.cpp
  + 2 or more functions
* Player.h
  + Defines the Player class
  + Contains documentation and prototypes
* makefile

Before you start programming, fully read the program description and outline your program using pseudocode.

**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 Player class:**

Define a class that will have private member variables and public functions and store the data for each member of the team. Your class **must** have the following member variables:

* int ID; // Player #
* string lastName; // Player last name
* string firstName; // Player first name
* float memberData[20]; // An array of stats. Because the heap is allocated differently in C++, we will hardcode this array to be of size 20.
* int numStat; // the number of stats we will read (must be <= 20)

In the last assignment, we allocated memory for memberData on the heap. Because the best way to use the heap in C++ is to use smart pointers, which we haven’t covered yet, we are gong to hard-code the size to be 20. numStats should still store the actual number of stats available and should be <= 20.

Member functions: Your Player class must have the 5 member functions listed below, along with any additional functions you decide to create.

// Default constructor

Player()

// copy the player data from stats into memberData

int setStats(float\* stats, int statSize);

//update the Players data using the data in stats

   int updateStats(float\* newData, int statSize);

  // print the Player to a file (where out is the file or the terminal)

  void printPlayer(ostream& out);

   // set all member variables

   int setAll(int num, string fname, string lname, float\* stats, int statSize);

**Storing team members:**

Store all team members in an array of Player objects, such as the following:

Player theTeam[numPlayers];

Note: this line of code will automatically call the constructor for each Player object in the array

**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.

When updating stats, the new stat are the original data + the new 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. Check that numStats is <= 20. If not, print the following message:

Too many player stats for this program (max of 20).

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. Read the first line of the input file into a string. Keep that array to print to the first line of the output file.

### **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 .cpp files and at least 1 .h file.
3. Your makefile which will be used to compile your program. The executable must be called teamData

**Grading:**

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

|  |  |
| --- | --- |
| **Met** | **Rubric Item** |
|  | The Player class contains the required member functions and variables |
|  | Function prototypes are in the .h file and the .h file included correctly in the .cpp files |

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 7 supplemental items (5 in the Rubric document and 2 listed above).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Rubric Items Met | |
| **Autograder tests** | **Core** | **Supplemental** |
| Excellent | All Passed | 7 | 6+ |
| Meets Expectations | All Passed | 7 | 5+ |
| Needs Revision | All Passed | <7 | N/A |
| Not Graded | Failed 1 or More test | N/A | N/A |