**CIT120 Final Project – NFL Stat Calculator**

For my final project, I elected to combine two of my interests into a project that I would find more directly interesting than some of the more “generic” labs and projects we had to do throughout the semester. In particular, I decided to do something related to my sports fandom and interest in sports statistics. For these reasons I elected to create a program that would utilize National Football League (NFL) statistics. My original intent was to create a program that accepted a file to input that would contain statistics for the 32 starting quarterbacks (QBs) in the NFL and use those statistics to perform various ranking methods, calculate additional metrics, and display these to the user either through the console or an output file. This intent was only somewhat met due to the (apparently) overly ambitious nature of the original project.

My statistics are read into the program by default from a file called “stats.txt” which contains a quasi-spreadsheet of statistics. The user has the option to input custom files if he so chooses. The program then iterates over the file and stores each row from the input file into a row of a two dimensional array. The file looks as such (as an example):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 400 | 600 | 4700 | 42 | 5 |
| 412 | 550 | 4215 | 32 | 15 |
| 369 | 532 | 3254 | 10 | 16 |

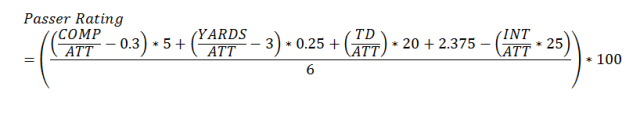
While the above numbers are not accurate numbers as they appear in the file, they are representative of the general format. There are 32 rows like those above. The stats are, respectively, passes completed (C), passes attempted (A), yards gained (YDS), passing touchdowns scored (TD), and interceptions thrown (INT). These represent the most basic of passing statistics for any given QB. The input file is not specific about what characters are used as delimiters and will work with any whitespace character. However, it does not specifically look for line breaks. Instead the program will load 5 stats into a given row in the two dimensional array and then move to the next row. This is a potential source of logical error and is addressed in a special “readme.txt” that I have included with the program. The program by default loads in a pre-formatted list of 2014 season statistics.

The second file that is inputted is “names.txt” which contains a list of names of the QBs, to be stored in a string array parallel to the first dimension of the stats array. The default argument for the load function accepts ‘|’ as the delimiter but can be changed to accept any specific character. I chose this character as the delimiter because it is unlikely to appear “accidentally” in any names and is unlikely to cause formatting issues between systems. I chose not to use the line break character as this caused strange issues when switching systems and loading “names.txt”. Because of this choice, the names.txt file is not particularly easy for a human to read, but inputs smoothly into the program. The input file looks as such:

Tom Brady|Aaron Rodgers|Andrew Luck|Russell Wilson|Ryan Tannehill|Derek Carr|Alex Smith|

Again, this example is not an accurate representation of the contents of the file, but rather an example of the general format. Due to the fact that there are 32 names, this file is very inconvenient for a human to read. I do not feel this is an issue as the file exists only to be read by the program itself. These names, as mentioned above, are stored in parallel to the first dimension of the array of statistics. For example, if stats[0][] held Tom Brady’s statistics, names[0] would be equal to the string “Tom Brady.” This is done so it is always easy to tie a QB’s statistics back to that QB’s name. There does not exist a file outside of the program that correlates the names to statistics, so it is important to ensure no changes are made to either file that would break these associations, such as inserting or removing names without adding or deleting the corresponding stats to ensure the files remain parallel.

I could, and perhaps should, have used the .peek() and .putback() method to use one file to fill both arrays, but I felt that using a two dimensional array had already complicated the issue enough and I chose to restrict my input format more in order to make loading my arrays simpler.

After the stats array is filled with the inputted stats the program will calculate additional statistics at runtime. These statistics are more advanced than the inputted ones, but rely on the inputted ones to be calculated. They are, in order, completion percentage (C/A), yards per attempt (YDS/A), yards per completion (Y/C), touchdowns per attempt (TD/A), interceptions per attempt (INT/A), touchdown to interception ratio (TD/INT) and passer rating (a somewhat complicated metric, pictured below). These stats are then loaded into the stats array at indexes 5 through 11.

Once the files are done being used for loading arrays they are closed and the user is prompted to select a statistic (of the 12 that are stored per QB) to get more information about. After the user selects a statistic, the program will prompt the user to select either the console or a file to print to.

If the user selects console, the name and value of the league leader and last place player in the selected stat are printed to the console. There is currently no logic for handling ties as I could not come up with a simple way to handle that. Given more time, I feel I may have been able to implement something, but it was not a priority for this project as there exist only two ties for a maximum or minimum value, which is a tie for league lead in yardage, and the most interceptions. After printing these two players to the console, the program continues on to print the league average (mean) value. Last, the program will print to the console a list of all players that are above the average value followed by all players who are below average. If the user selects to print to a file, they are prompted to enter the file name for their output and the same information as above is printed to that file.

Though this is the current state of the program and it accomplishes mentioned so far, I will now discuss briefly what I originally intended to do that eventually proved difficult to the point of near impossibility.

As mentioned before, the statists are stored in a two dimensional array. This array, if visualized, would look something like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 425 | 536 | 4752 | 42 | 8 |
| 1 | 461 | 612 | 3654 | 30 | 12 |
| 2 | 385 | 588 | 4152 | 26 | 5 |
| 3 | 345 | 601 | 4411 | 38 | 3 |

My original intent was to sort the rows of the array based upon the values in one of the columns. I tried many different methods, Googled tips, and talked to multiple friends who have computer science degrees, until I eventually decided, after consulting with the professor, that there was no easy method to use to sort an array thusly. For this reason, I changed the purpose of the program and was able to effectively accomplish that purpose.