**Racing\_Stats**

1**)Research**

*Racing statistics play a crucial role in evaluating driver and vehicle performance across various motorsport disciplines. Key metrics include* ***lap times****, which measure the time taken to complete one lap of the race, and are used to gauge speed and consistency****. Fuel consumption*** *is also tracked, as efficiency in fuel use can impact race strategy, especially in endurance racing.*

*Additional metrics include* ***fastest lap times****, total race distance, and* ***pit stop efficiency****, which can significantly affect overall race outcomes. Modern racing stats also incorporate* ***telemetry data****, tracking variables like tire wear, engine performance, and aerodynamics to provide a more granular analysis of vehicle behavior. Furthermore,* ***overtakes, pole positions,*** *and* ***race finishes*** *contribute to ranking drivers and teams, helping to identify trends, areas for improvement, and competitive advantage.*

**2)Analysis**

1. Error Handling

While there is some input validation, additional error handling could be implemented for file operations (e.g., checking if the file was written successfully).

2. Dynamic Memory Management The program uses static arrays for storing driver data (drivers[MAX\_DRIVERS]). Using dynamic memory allocation (e.g., malloc) could allow for more flexible handling of varying numbers of drivers and races.

3. Sorting Algorithm Efficiency

The sorting function employs bubble sort, which is inefficient for larger datasets. Implementing a more efficient sorting algorithm like quicksort or mergesort would improve performance.

4. Code Modularity

The code could benefit from further modularization by breaking down some functions into smaller sub-functions or creating additional helper functions to enhance readability and maintainability.

5. User Interface Enhancements

The user interface could be improved by providing clearer instructions or options (e.g., using menus) instead of relying solely on prompts.

**3)Ideate**

*The use of a struct to encapsulate all relevant data for a driver (name, car details, race statistics) promotes organized data management and enhances code readability.*

*User Input Handling:*

*The program includes robust input validation to ensure that user inputs are within acceptable ranges (e.g., number of races, lap times, fuel consumption). This prevents invalid data from causing errors or incorrect calculations.*

*Statistical Calculations:*

*Functions are implemented to perform essential calculations, such as average lap time, fastest and slowest lap times, total distance covered, and average fuel consumption. This modular approach allows for easy updates and maintenance.*

4)Build

*We have built this code using VS code and is working as per our knowledge, And code is based on f1 racing, which contains data like fuel usage no of laps*

***5)Test***

*Testing Points for the Racing Statistics Program*

*Unit Testing:Functionality Tests: Write unit tests for each function (e.g., calculateAverageLapTime, findFastestAndSlowestLap, sortLapTimes) to verify their correctness with a variety of input scenarios.*

*Edge Cases: Test edge cases, such as:*

*Minimum and maximum values for lap times and fuel consumption.Handling of zero or negative values (which should be invalid).Empty lap times array (e.g., when no races have been entered).*

*Input Validation Testing:*

*Valid Inputs: Test the program with valid inputs to ensure it processes them correctly.*

*Invalid Inputs: Test with invalid inputs (e.g., negative lap times, non-numeric values) to confirm that the program handles them gracefully without crashing.*

*Boundary Values: Check how the program behaves at boundary values (e.g., entering exactly 1 or MAX\_RACES for the number of races).*

*Integration Testing:*

*Test how different components of the program work together. For instance, validate that input functions correctly populate the RaceStats structure and that calculations reflect those inputs accurately.*

*Ensure that file I/O operations work as expected when saving data to a CSV file.*

*Performance Testing:*

*Assess the performance of sorting algorithms used in sortLapTimes with large datasets (e.g., maximum number of drivers and races) to ensure they execute within acceptable time limits.*

*Evaluate memory usage during execution, especially if dynamic memory allocation is implemented in future enhancements.*

**6)Implement**

*Code Modularity:*

*Function Separation: Ensure that each function has a single responsibility. For example, separate functions for input handling, calculations, and output display can make the code easier to manage and test.*

*File Organization: Consider splitting the code into multiple files (e.g., race\_stats.c, file\_io.c, statistics.c) for better organization and maintainability.*

*Dynamic Memory Allocation:*

*Instead of using fixed-size arrays (e.g., MAX\_DRIVERS, MAX\_RACES), consider using dynamic memory allocation with malloc and free. This allows for more flexibility in handling varying numbers of drivers and races.*

*Ensure proper memory management to avoid memory leaks by freeing allocated memory when it is no longer needed.*

*Input Validation:*

*Improve input validation by creating dedicated functions that handle specific types of input (e.g., validating race counts, lap times, fuel consumption). This reduces redundancy and enhances readability.*

*Consider using a loop to retry input until valid data is provided, ensuring that the program does not proceed with invalid values.*

**GIT HUB LINK-** <https://github.com/Omkar23748ghb/Racing_Stats-Sim>