WORLD RALLY CROSS CHAMPIONSHIP:

REPORT

NAME	: HAMMAAD RIZWAN
RGU ID	: 2237928
IIT NO	: 20221729
COURSE	: Bcs AI and Data Science
MODULE	: CM1601 (Programming Fundamentals)

EXECUTIVE SUMMARY

This is detailed documentation about our application which was created to manage the world rally cross championship season.

There are a variety of functions designed to suit the needs of this grand competition, one of the functions such as VCT (code for displaying the points table in descending order) allows teams and drivers to know their position among their rival competitors.

Moreover, the SRR function is the main highlight of our project where it allows races to be simulated randomly, with random dates, across stunning race locations around the globe. This would be useful in case if the race cannot be conducted physically, such as during the pandemic.

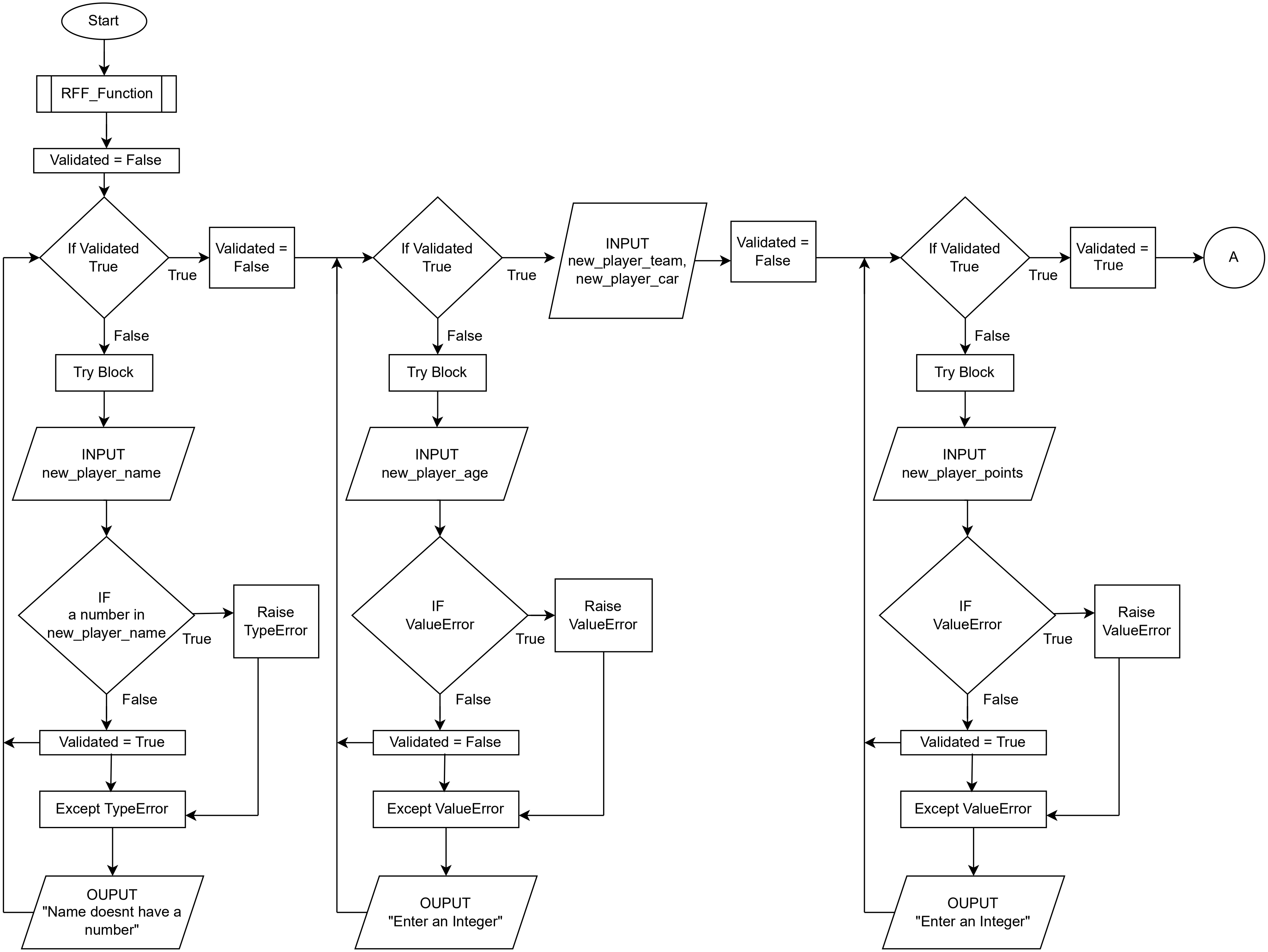
Furthermore, other basic functions such as add, delete, and update details of a driver have been updated to meet the technological advances in the racing industry, teams could also view detailed reports on the race by using the VRL function, as a result the teams could improve their positions on the championship standings allowing them to emerge victorious and title them as champions for this year’s world rally cross championship.

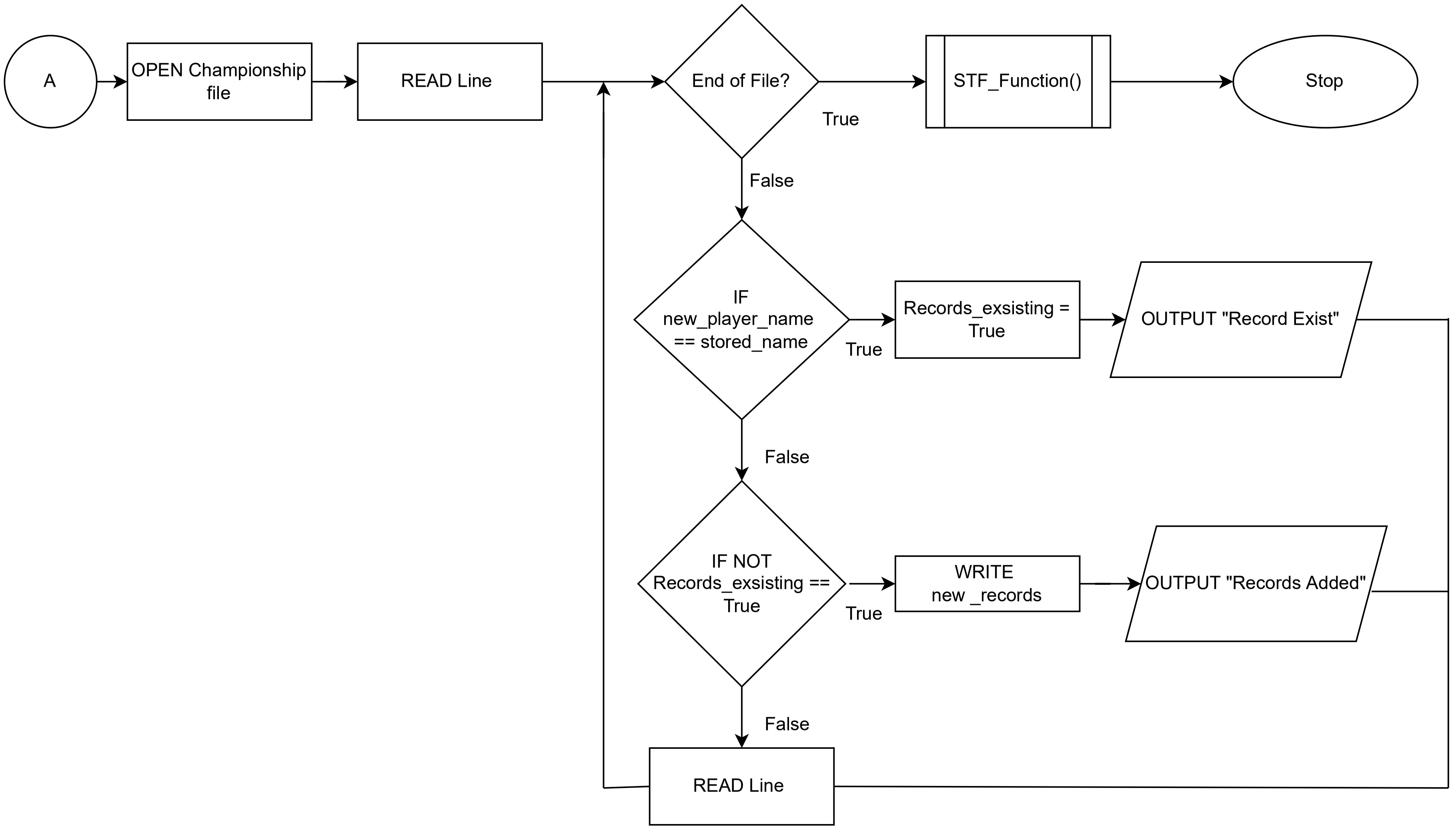
CONTENTS

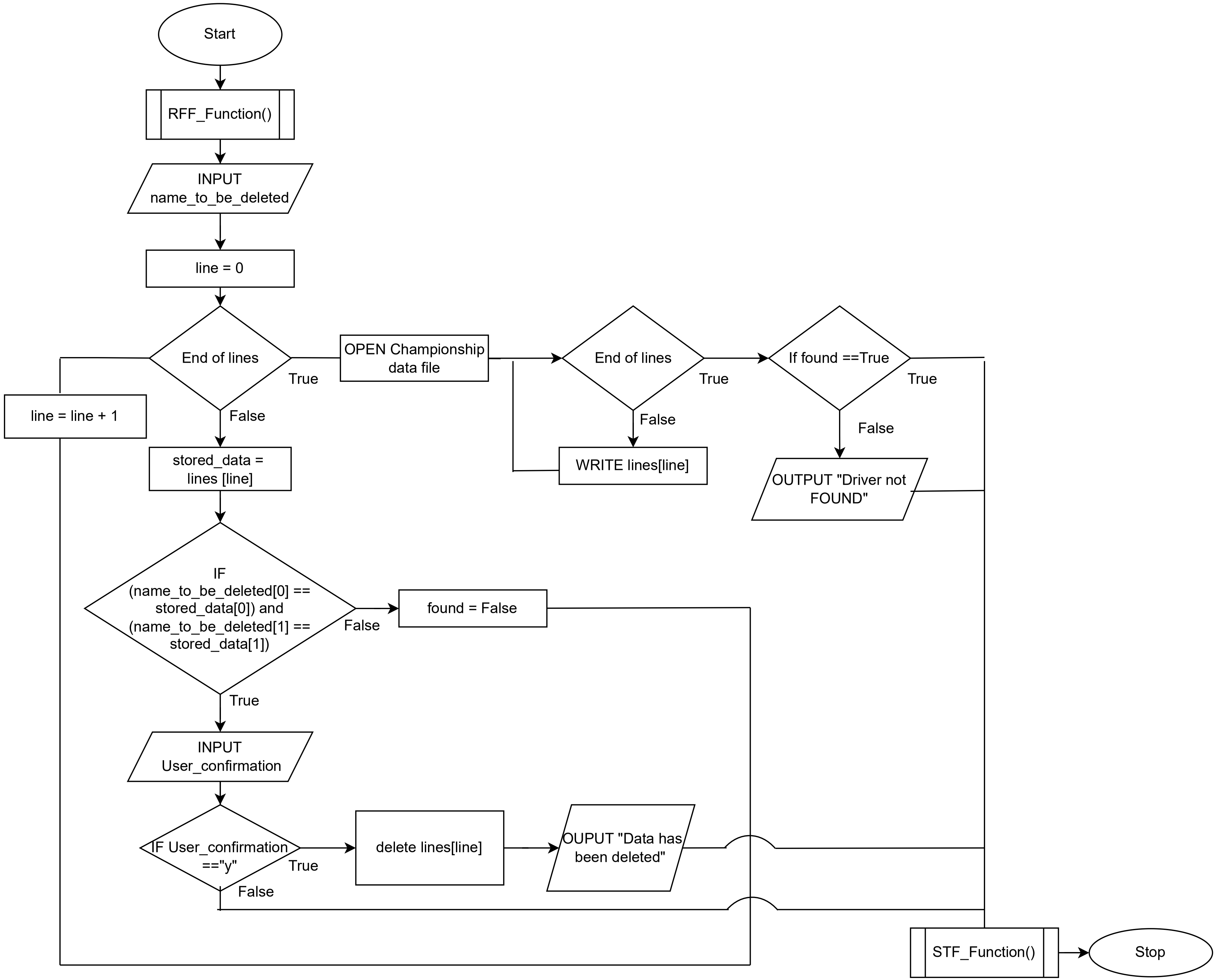
Flowcharts:

Documentation:

Test Plan:


Start
RFF_Function
Validated = False
If Validated True
True
Validated = False
False
Try Block
INPUT new_player_name
IF a number in new_player_name
True
Raise TypeError
False
Validated = True
Except TypeError
OUPUT "Name doesnt have a number"
If Validated True
True
False
Try Block
INPUT new_player_age
IF ValueError
True
Raise ValueError
False
Validated = False
Except ValueError
OUPUT "Enter an Integer"
INPUT new_player_team, new_player_car
Validated = False
If Validated True
True
Validated = True
A
False
Try Block
INPUT new_player_points
IF ValueError
True
Raise ValueError
False
Validated = True
Except ValueError
OUPUT "Enter an Integer"


A
OPEN Championship file
READ Line
End of File?
True
STF_Function()
Stop
False
IF new_player_name == stored_name
True
Records_exsisting = True
OUTPUT "Record Exist"
False
IF NOT Records_exsisting == True
True
WRITE new _records
OUTPUT "Records Added"
False
READ Line


Start
RFF_Function()
INPUT
name_to_be_deleted
line = 0
End of lines
True
False
line = line + 1
OPEN Championship data file
End of lines
True
False
WRITE lines[line]
If found ==True
True
False
OUTPUT "Driver not FOUND"
stored_data = lines [line]
IF (name_to_be_deleted[0] == stored_data[0]) and (name_to_be_deleted[1] == stored_data[1])
False
found = False
True
INPUT User_confirmation
IF User_confirmation =="y"
True
False
delete lines[line]
OUPUT "Data has been deleted"
STF_Function()
Stop

Documentation: **ADD Function**

```python
def ADD_Function():
    lines_in_championship_file,lines_in_race_file = RFF_Function()
    while True:
        print("Enter Name:")
        try:
            new_player_name = input("> ").title()
            if new_player_name == "":
                print("\33[91m" + "Field cannot be empty!" + "\33[0m")
                continue
            elif any((character.isdigit() for character in
                      new_player_name)) == True:
                raise TypeError("Name cannot contain numbers")
            break
        except TypeError:
            print(
                "\33[91m" + "Name cannot contain numbers, Try Again" + "\33[0m")
        print("")

    while True:
        print("Enter Age:")
        try:
            new_player_age = int(input("> "))
            break
        except ValueError:
            print("\33[91m" + "Requires an Integer, Try Again" + "\33[0m")

    while True:
        print("Enter Team:")
        new_player_team = input("> ").title()
        if new_player_team != "":
            break
        else:
            print("\33[91m" + "Field cannot be empty!" + "\33[0m")

    while True:
        print("Enter Car:")
        new_player_car = input("> ").title()
        if new_player_car != "":
            break
        else:
            print("\33[91m" + "Field cannot be empty!" + "\33[0m")

    while True:

        try:
            print("Enter Current Points:")
            new_player_current_points = int(input(">  "))
            if new_player_current_points == "":
                print("\33[91m" + "Field cannot be empty!" + "\33[0m")
            break
        except ValueError:
            print("\33[91m" + "Requires an Integer, Try Again" + "\33[0m")  # outputs in RED
    print("")
    championship_data_file = open("championship_data.txt", "r+")
    line_in_championship_data = championship_data_file.readline()


    if line_in_championship_data == "":
        header_championship_data = '{:<22} {:<12} {:<22} {:<18} {:<12}\n'.format("NAME","AGE","TEAM","CAR","POINTS")
        championship_data_file.write(header_championship_data)
        STF_Function()

    championship_data_file = open("championship_data.txt", "r+")
    line_in_championship_data = championship_data_file.readline()
    record_exsisting=False
    while line_in_championship_data != "":
        stored_data = line_in_championship_data.strip().split()
        stored_first_name = stored_data[0]
        stored_last_name = stored_data[1]
        new_name = new_player_name.strip().split()
        new_player_first_name = new_name[0]
        new_player_last_name = new_name[1]
        if (new_player_first_name == stored_first_name) and (new_player_last_name == stored_last_name):
            record_exsisting = True
            print("'{}' Exists already".format(new_player_name))
            print("\033[91m"+"Invalid Input"+"\033[0m")
            break
        line_in_championship_data = championship_data_file.readline()

    if record_exsisting == False:
        new_records = '{:<22} {:<12} {:<22} {:<18} {:<12}\n'.format(new_player_name, new_player_age, new_player_team, new_player_car,new_player_current_points)
        championship_data_file.write(new_records)
        print(("\33[32m"+" {}  has been added.."+"\33[0m").format(new_player_name))
    STF_Function()
```

The system loads the data from both text files using the RFF function. Using exception handling throughout user inputs the system makes sure there cannot be any anomalies in data as it would be crucial for the entire program. For example, if the name contains any numbers, the program would raise an error as the name could not contain a number, therefore using this all inputs could be validated.

Once all inputs of the driver details have been entered into the system, the file is open in '*r+*' which allows us to read and write at the same time. If the line is empty at first a header is assigned, and then the new driver name is searched throughout the system to reduce duplication of records.

If there are no existing records then the program writes the new driver's details into the system, thus displaying a message to the user to inform them that the operation was successful.

STF Function is called to save the contents of the file since details have been added.

Add Function is the most important component of the system, as all other functions depend the data which has been entered here. Therefore all inputs are being validated strictly and no field can be ignored by the user when registering their driver to the world rally cross championship system. Users will have to check their inputs, as the system might not proceed if the inputs aren't met to the required standard.

Documentation: **DDD Function**

```python
def DDD_Function():
    lines_in_championship_file,lines_in_race_file = RFF_Function()
    print("Enter the Name To be deleted:")
    name_to_be_deleted = input("> ").title()
    while name_to_be_deleted == "":
            print("\33[91m" + "Field cannot be empty!" + "\33[0m")
            print("Enter the Name To be deleted:")
            name_to_be_deleted = input("> ").title()



       delete_name = name_to_be_deleted.strip().split()
       if len(delete_name)!= 2:
           print("\033[91m"+"Driver not Found!"+"\33[0m")
       else:
           delete_firstname = delete_name[0]
           delete_lastname = delete_name[1]

           lines = lines_in_championship_file
           found = False

           for records in range(len(lines)):
               stored_data = lines[records].split()
               stored_firstname = stored_data[0]
               stored_lastname = stored_data[1]
               if stored_firstname == delete_firstname and stored_lastname == delete_lastname:
                   found=True
                   user_confirmation = input("You are about to delete '{} {}' Records (Y/N): ".format(delete_firstname,delete_lastname)).lower()
                   if "y" == user_confirmation:
                       del(lines[records])
                       print("")
                       print("{}'s data has been deleted..".format(name_to_be_deleted))
                       print("")
                       break
                   elif "n" == user_confirmation:
                       break
                   else:
                       print("Enter (Y/N)")
           if found == False:
               print("\033[91m"+"Driver not Found!"+"\33[0m")

           with open("championship_data.txt","w") as file :
               for line in lines:
                   file.write(line)
       STF_Function()
```

Lines are being read from the championship file, if the record has been found the user will have to confirm that they would want to delete the records. This is vital as this information is important for future races and standings table, thereby reducing careless mistakes from the user. If the user doesn't want to delete the record, they can simply enter *"n"* thereby the system takes the user back to the main screen.

If the record is not found the system would inform the user, all the lines will be re-written to the system, and changes have been saved using the STF function.

To be more technical, we mainly focus on list manipulation where we read the entire file and all records are stored in the list, in order to remove a driver from the system, the program first checks whether the name to be deleted is present in the championship file. If it is true, using the *del()* command we remove the records associated with that specific driver.

Once the list is updated the file is re-opened and is being rewritten using the *'w'* mode.

```python
def UDD_Function():
    lines_in_championship_file,lines_in_race_file = RFF_Function()
    while True:
        print("Enter the Driver's Name for which details needs to be updated: ")
        driver_to_be_updated = input("> ").title()
        driver_name = driver_to_be_updated.strip().split()
        if len(driver_name) !=2:
            print("Enter a valid name..")

        else:
            break
    driver_firstname = driver_name[0]
    driver_lastname = driver_name[1]
    update_successful = False

    lines = lines_in_championship_file
    for records in range(len(lines)):
        stored_data = lines[records].split()
        stored_firstname = stored_data[0]
        stored_lastname = stored_data[1]
        if stored_firstname == driver_firstname and stored_lastname == driver_lastname:
            print("Exsisting Record for '{}'".format(driver_to_be_updated))
            print("Name ...........: {}".format(driver_to_be_updated))
            print("Age ............: {}".format(stored_data[2]))
            print("Team ...........: {}".format(str(stored_data[3]+" "+stored_data[4])))
            print("Car ............: {}".format(stored_data[5]))
            print("Current Points .: {}".format(stored_data[-1]))
            print("")

            updated_player_name = driver_to_be_updated
            print("Which records do you need to update? ")
            print("(type 'all' if you want to update everything)")
            update_option = input("> ").lower()

            if update_option == "age":
                while True:
                    try:
                        updated_player_age = int(input("Enter Age: "))
                        break
                    except ValueError:
                        print("\33[91m"+"Requires an Integer, Try Again"+"\33[0m")
                print("")
                stored_data[2] = updated_player_age

            elif update_option == "team":
                updated_player_team = input("Enter Team: ").title()
                updated_player_team_split = updated_player_team.split()
                stored_data[3] = updated_player_team_split[0]
                stored_data[4] = updated_player_team_split[1]

            elif update_option == "car":
                updated_player_car = input("Enter Car: ").title()
                stored_data[5] = updated_player_car

            elif "points" in update_option.split() or "current" in update_option.split():
                while True:
                    try:
                        updated_player_current_points = int(input("Enter Current Points: "))
                        break
                    except ValueError:
                        print("\33[91m"+"Requires an Integer, Try Again"+"\33[0m")
                stored_data[-1] = updated_player_current_points

            elif update_option=="all":
                while True:
                    try:
                        updated_player_age = int(input("Enter Age: "))
                        break
                    except ValueError:
                        print("\33[91m"+"Requires an Integer, Try Again"+"\33[0m")
```

```python
                updated_player_team = input("Enter Team: ").title()
                updated_player_car = input("Enter Car: ").title()
                while True:
                    try:
                        updated_player_current_points = int(input("Enter Current Points: "))
                        break
                    except ValueError:
                        print("\33[91m"+"Requires an Integer, Try Again"+"\33[0m")
                stored_data[2] = updated_player_age

                updated_player_team_split = updated_player_team.split()
                stored_data[3] = updated_player_team_split[0]
                stored_data[4] = updated_player_team_split[1]
                stored_data[5] = updated_player_car
                stored_data[-1] = updated_player_current_points
            else:
                print("\33[91m"+"Wrong input"+"\33[0m")
                print("Choose the correct field. eg-'Team' if the details of driver's team should be updated..")
            updated_records = '{:<22} {:<12} {:<22} {:<18} {:<12}\n'.format( updated_player_name, stored_data[2],(stored_data[3] +" "+stored_data[4]),stored_data[5],stored_data[6])
            lines[records] = updated_records
            update_successful=True
            print("")
            print(("\33[32m"+"{}'s data has been updated.."+"\33[32m").format(driver_to_be_updated))
            print("")
    if update_successful == False:
        print("\33[91m"+"Driver not found.."+"\33[91m")
    with open("championship_data.txt","w") as file :
        for line in lines:
            file.write(line)
    STF_Function()
```

Reads the entire file using RFF function and checks whether there is a driver that the user is searching for. If the driver is found all his details are displayed to the user which will allow the user to know which of the details needs to be updated.

Users could update all the details of the driver other than the name (assuming the name doesn't need to be changed throughout the campaign).

For example, if drivers transfer to a different team during the course of the season, these details could be updated easily in the UDD function by entering "*team*" when asked which details need to be updated.

Exception handling is vital in these stages as the system is completely relying on the inputs taken from the user, therefore same as in ADD function all inputs have been validated to minimize Runtime and Syntax errors.

Not to forget, we had made an improvement to the update function where users/ admins would be able to update all fields at once of their driver, rather than repeating the process several times. To use this function, users need to type *'all'* when asked for which data to be updated, thereby if the driver to be updated is found in the championship file, the data would be updated and a message would be displayed to the user that the process was successful.

Users/ admins could even type VCT to get a clear view of all the changes they made.

Documentation: **VCT Function**

```
def VCT_Function():
    lines_in_championship_file,lines_in_race_file = RFF_Function()
    lines = lines_in_championship_file


    for outer_loop in range(1, len(lines)):
        for records in range(1, len(lines)-1):
            current_player_points = int(lines[records].strip().split()[-1])
            next_player_points = int(lines[records+1].strip().split()[-1])
            if current_player_points < next_player_points:
                temp = lines[records]
                lines[records] = lines[records + 1]
                lines[records + 1] = temp
    STF_Function()

    print("")
    print("░░░░░▒▒▒▒▒▒▒▒▒▒▒▒▒███████████████ CHAMPIONSHIP STANDINGS ███████████████▒▒▒▒▒▒▒▒▒▒▒▒▒░░░░░")
    print("")
    rank = 0
if len(lines)==0:
        print("There are no drivers in the system...")
        print("Use the ADD function to get started!")
    else:
        for line in lines:
            if rank == 0:
                print("   "+line)
            else:
                print(("{}. "+line).format(rank))
            rank += 1
```

Using a for loop until the end of the list, the current driver points is being stored and checked with the next driver points. If the next driver's points are greater than the current driver points a temporary variable will store the current driver’s records (name, age, points and etc).

The current driver records are being replaced with the next driver's records, while the temporary variable will replace the next driver's records (In simple terms using a third variable it allows us to swap values between them.

The list of the championship records are being sorted in descending order of points. A header is being assigned to allow the user to identify records more easily and gives a tabular format to the Championship display.

Positions are being assigned based on the points which will allow teams and drivers to be constantly updated on the standings and allow them to remaster their strategies in order to climb up the table.

We got the format for our header by using font keyboard's wide range of symbols.

If the records in the files are empty, when VCT is being called; the user will get a message that there are no data to be displayed, thereby in order to view the data user should add the drivers into the championship. Thereby even if the files aren't available our system would be able to start a new campaign in a few clicks.

Documentation: **SRR Function**

```
def SRR_Function():
    lines_in_championship_file,lines_in_race_file = RFF_Function()
    race_data_file = open("race_data.txt", "r+")
    line_in_race_data = race_data_file.readline()
    if line_in_race_data == "":
        header_race_data = '{:<12} {:<12} {:<22} {:<18} {:<12}\n'.format("DATE","LOCATION","DRIVER","POSITION","POINTS")
        race_data_file.write(header_race_data)

    def Random_Race_location():
        race_locations= ["Nyirad","Holjes","Montalegre","Barcelona","Riga","Norway"]
        random_location= random.randint(0,len(race_locations)-1)
        location_random = race_locations[random_location]
        return location_random

    def Random_Race_date():
        month = random.randint(1, 12)
        if month == 4 or month == 6 or month == 9 or month == 10:
            day = random.randint(1, 30)
        else:
            day = random.randint(1, 31)
        race_date_random = ("{}/{}/22".format(day, month))
        return race_date_random
    race_location = Random_Race_location()
    race_date = Random_Race_date()
    driver_available = []
    exsisting_dates = []

    lines = lines_in_championship_file
    for records in range(1,len(lines)):
        stored_data = lines[records].split()
        stored_firstname = stored_data[0]
        stored_lastname = stored_data[1]
        driver = (stored_firstname+" "+stored_lastname)
        driver_available.append(driver)

    contestants = driver_available
    contestants_copy = contestants.copy()
    random.shuffle(contestants_copy)

```

```
        race_points = 0
        for position in range(len(contestants_copy)):
            if position == 0:
                race_points = 10
            elif position == 1:
                race_points = 7
            elif position == 2:
                race_points = 5
            else:
                race_points=0
            drivers_position = position+1
            drivers_points=race_points
            driver_name=contestants_copy[position].strip().split()
            driver_firstname=driver_name[0]
            driver_lastname=driver_name[1]
            STF_Function()

            lines = lines_in_championship_file
            for records in range(len(lines)):
                stored_data = lines[records].split()
                stored_firstname = stored_data[0]
                stored_lastname = stored_data[1]
                if stored_firstname == driver_firstname and stored_lastname == driver_lastname:
                    updated_player_name = stored_firstname+" "+stored_lastname
                    updated_player_age = stored_data[2]
                    updated_player_team = stored_data[3]+" "+stored_data[4]
                    updated_player_car = stored_data[5]
                    updated_player_current_points = int(stored_data[-1]) + race_points
                    updated_records = '{:<22} {:<12} {:<22} {:<18} {:<12}\n'.format(updated_player_name,updated_player_age,updated_player_team,updated_player_car,updated_player_current_points)
                    lines[records] = updated_records
            with open("championship_data.txt", "w") as file:
                for line in lines:
                    file.write(line)


            lines = lines_in_race_file
            occurrence = 0
            for records in range(1,len(lines)):
                stored_date = lines[records].split()[0]
                occurrence = occurrence + 1
                if occurrence == len(contestants_copy):

                    exsisting_dates.append(stored_date)
                    occurrence =0

            date_exists = True
            while date_exists == False:
                if race_date in exsisting_dates:
                    date_exists = True
                    race_date = Random_Race_date()


            race_data_file.write('{:<12} {:<12} {:<22} {:<18} {:<12}\n'.format(race_date,race_location,updated_player_name,drivers_position,drivers_points))
        STF_Function()
```

SRR Function is one of the vital components of our program. It allows races to be simulated randomly from a range of stunning locations around the world.

Reads the entire file to get the records of the contestants in order to simulate a random race. If the race data file is empty then headers are assigned as required and are written to the race data file (this stores the details of all the races and the positions of each driver in the championship).

A location is generated randomly by getting a random element from the list of possible locations. All races will be happening in 2022 as it is assumed to be one season therefore a month is generated using random numbers between (1-12), and if the months are either (4,6,9,10) the dates possible could be only till 30th while the other months will be able to host races till 31st.

The driver positions are being simulated using "*.shuffle()*" command which randomizes the location of elements on the list, thereby we could consider that the order the elements are in after using the shuffle method would be the final positions for the race.

The first 3 positions will get points added to their championship standings. However, before appending these details to the race data file the program checks whether there were any races with the same date, if so a new race date is generated.

```
def VRL_Function():
    print("Race Table Loading..")
    print("")
    lines_in_championship_file,lines_in_race_file = RFF_Function()

    lines = lines_in_race_file

    for outer_loop in range(1, len(lines)):
        for records in range(1, len(lines) - 1):
            current_month = int(lines[records].split()[0].split("/")[1])
            next_month = int(lines[records + 1].split()[0].split("/")[1])
            current_day = int(lines[records].split()[0].split("/")[0])
            next_day = int(lines[records + 1].split()[0].split("/")[0])
            if current_month > next_month:
                temp = lines[records]
                lines[records] = lines[records + 1]
                lines[records + 1] = temp
            if current_month == next_month:
                if current_day > next_day:
                    temp = lines[records]
                    lines[records] = lines[records + 1]
                    lines[records + 1] = temp
    STF_Function()


    print("")
    print("░░░░░░▒▒▒▒▒▒▒▒▒▒▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ RACE TABLE ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▒▒▒▒▒▒▒▒▒▒░░░░░░")
    print("")
if len(lines)==0:
    print("No previous data to be displayed")
    print(" ")
    print("Inorder to view the race table..")
    print("Use the SRR function to get started!")
else:
    for line in lines:
        print(line)

```

By reading the entire race data file it stores all the details into a list which is identified as lines and for each line, the current month has been taken into consideration where if the current month is greater than the next month, the current line is being swapped with the next line.

If the current month and the next month are equal then we look at the day of that specific month and sort them in ascending order. This allows the lines to be sorted according to the date and will be easy to read when it comes to displaying it on the console.

Once the lines are sorted the document is closed and saved using the STF function as there are no changes needed to make from the file.

Records are printed according to the line until the end of the entire list, thereby neatly formatting the race table on the console.

If the records in the files are empty, when VRL is being called; user will get a message that there are no previous race data to display this could be because the first race of the season has not been yet begun. Thereby user/ admin could enter SRR inorder to see the true standings of the championship season.

Documentation: **STF Function**

```
def STF_Function():
    race_data_file.close()
    championship_data_file.close()
```

In order to save the file we have closed both the files which were opened earlier, thereby no further changes can be made to the document.

Documentation: **RFF Function**

```
def RFF_Function():
    race_data_filename = "race_data.txt"
    championship_data_filename = "championship_data.txt"
    race_data_file = open(race_data_filename, "r")
    championship_data_file = open(championship_data_filename, "r")
    lines_in_race_file = race_data_file.readlines()
    lines_in_championship_file = championship_data_file.readlines()
    return (lines_in_championship_file,lines_in_race_file)
```

To load the contents of the files and to resume capabilities all files are opened in read mode and entire records of the files are being stored in a list, which allows other functions to use these data and carry out their operations.

Test Plan: **ADD Function**

TEST NAME	TEST CASE	INPUT*	EXPECTED RESULT	ACTUAL OUTPUT
Name with one number	Entering a single number for the driver's name	`> Max Verstappen 8`	Numbers should not be present in a name	`Name cannot contain numbers, Try Again`
Name with only numbers	Entering multiple numbers for the driver's name	`> 67667`	Not a single number should be in a name	`Name cannot contain numbers, Try Again`
Empty name	Leaving driver's name field empty	`>`	Cannot continue until a name is entered	`Field cannot be empty!`
Incorrect data type for age	Entering a noninteger value for age	`> twenty two`	Takes only integers for age	`Requires an Integer, Try Again`
Empty age	Leaving driver's age field empty	`>`	Cannot continue until age is entered	`Field cannot be empty!`
Incorrect data type for points	Entering a string for current points	`> t200`	Current points can only be an integer	`Requires an Integer, Try Again`
Empty points	Leaving driver's current points field empty	`>`	Cannot continue until a point is entered	`Field cannot be empty!`
Empty team	Leaving driver's team field empty	`>`	Cannot continue until a team is entered	`Field cannot be empty!`
Empty car	Leaving driver's car field empty	`>`	Cannot continue until a car is entered	`Field cannot be empty!`

*Note: "**>**" symbol is used to show that it requires a user input into the system, this is present throughout the code. If inputs are empty it is shown as '**>**   '*

Test Plan: **DDD Function**

TEST NAME	TEST CASE	INPUT*	EXPECTED RESULT	ACTUAL OUTPUT
Wrong name	Entering the name of a driver who is not in the championship	**>** `Fernando Alonso`	User should know that driver is not available	`Driver not Found!`
Only first name	User inputs only the first name of a driver	**>** `Haadiya`	Cannot proceed unless the entire name has been entered	`Driver not Found!`
More than 2 names	If the driver has more than 2 parts to their name	**>** `Bradyn Alvaro Kramer`	All drivers have only first name and last name	`Driver not Found!`
Empty Field	User doesn't enter a name when requested	**>**	Cannot continue until the user enters a name	`Field cannot be empty!`

CONCLUSION

World rally cross championship is an emerging event during the past few years and in order to match the high standards, we had been asked to create software to manage this grand event.
Our system is a CLI (Command line interface) driven model, in simple terms the user interacts with the console, rather than in a complex interface. All instructions are displayed clearly on the screen when the program is launched.

The menu screen allows the user to access different avenues, such as:

- *Add* a new driver to the competition
- *Delete* a specific driver from the tournament
- *Update* a driver in case he gets transferred to a new team during the course of the season
- *View championship standings* which give a clear idea where the drivers are ranked among their rivals
- *Simulating random races*
- *View Race table* which gives an in-depth report of all races according to the date it was held
- *Store data* safely into the text file
- *Read data* from multiple text files

We believe that using this management system teams and drivers would be able to identify where they could improve thus allowing them to reach to their maximum potential. This system would be a breakthrough not only in the rally cross championship but also in the entire racing industry.

ASSUMPTIONS:IMPROVEMENTS

- All drivers have a first name and the last name (2 parts to their name)
- Races will be held in 2022 as the season goes only for one year.
- Races could be held on any day in any month. This takes into consideration the months that have 30 days and 31 days where duplicate and unmeaningful data can be handled
- System displays an error or success message using red and green colors respectively
- UDD function allows users to not only update one field but all, except the name. This would save plenty of time and effort for the user.
- HLP function helps the user if they are stuck. For example, if they aren't sure how to access the functions, there are detailed explanations on how to get started.
- Drivers are being ranked according to the points they achieve from races, sorted in descending order.
- Assumes the user inputs only Y or N for the confirmation of deletion of records in the DDD function.
- Colored syntax had been visible throughout the program- this would allow admins/ users to identify between the outputs and inputs.
- The program would sleep for 2 seconds after each function is successful, this would allow the user to read the displayed text for a bit longer, than moving to the next step at once.
- All inputs for each field will have this symbol '>'- which is done to identify that an input is required for that specific field.

REFERENCES

1. Shubham Singh,2022. “Read a file line by line in Python”.[online]. India: *GeeksforGeeks*. Available from https://www.geeksforgeeks.org/read-a-file-line-by-line-in-python/ [Accessed 29 November 2022]
2. Font Keyboard,2022. “Cool Symbols & Cool Fonts - Symbols, Emoji & Fonts ”. [online]. Font Keyboard LLC. Available from https://coolsymbol.com/ [Accessed 25 November 2022]
3. Peter Mortensen,2021. “How do I print colored text in terminal”. [online].Stack overflow.Available from https://stackoverflow.com/questions/287871/how-do-i-print-colored-text-to-the-terminal [Accessed 25 November 2022]
4. Adam McQuistan, 2017. “Formatting Strings with Python”. [online]. Stack abuse. Available from https://stackabuse.com/formatting-strings-with-python/ [Accessed 12 November 2022]
5. W3Schools, 2022. “Python Random shuffle() Method”. [online]. W3Schools. Available from https://www.w3schools.com/python/ref_random_shuffle.asp [Accessed 7 November 2022]
6. Portfolio Courses, 2022. “Replace A Specific Line In A File | Python Examples”. [online]. Portfolio Courses. Available from https://www.youtube.com/watch?v=6v1bVWI8nl8&ab_channel=PortfolioCourses [Accessed 28 October 2022]