Program Specifications

|  |  |
| --- | --- |
| **Chosen Project Assignment:**  Sports application analyzer. | |
| **Program Requirements:**   * It needs to be programmed in C++ * References need to be listed. * It needs to be efficient and it also needs to accurately detect useful information from the txt file. * The application should be able to examine multiple files while in run time. | |
| **Goal:**   1. My aim is to make the program as accurate as my Garmin Connect application. 2. I would also like to use an appealing interface like QT for my GUI 3. My second most important goal is to focus on Design and implementation. | |
| **Key Features of the program:**   * Import a list of files to examine * Overall Run Statistics * Average pace (min/mile) **with stops** * Average Moving Pace (min/mile) **without stops** * Best Pace (min /mile) * Average Speed (mph) * Max Speed (mph) * Total Time of Run * Total Moving Time of Run * Distance ran * Average Run Cadence * Max Run Cadence * Elevation * Min. Elevation * Max Elevation | * Individual information for each lap   -average pace (time) min/mile  - best pace (time) min/mile  - average speed (mph)  - Max speed (mph)  - Average Cadence Steps per Minute  - Max Cadence Steps per Minute |

DESIGN AND IMPLEMENTATION

# Stages

1. Analyze Raw File Data
2. Form Program Structure
3. Extract Data

|  |  |
| --- | --- |
| **Analyze Raw File Format** (TCX. File) | |
| File Tags   1. Important Tags in the Document   Below are snips of a tcx file. I made sure to only screenshot the important tags needed to perform my data analysis.    <TotalTimeSeconds>406.0</TotalTimeSeconds>   * + This Tag shows how long the lap took in seconds. This tag was not used. Because it was not reliable in determining how long the mile took.   <DistanceMeters>1117.53</DistanceMeters>   * + This tag I did use. The first Distance meters tag in each lap tag shows indicates how far that specific lap was.   <MaximumSpeed> 3.231343</MaximumSpeed>   * + This tag detects and reveals the highest speed ran in the lap. | <AltitudeMeters> 44.32</AltitudeMeters>   * + This tags detects the current elevation the runner is running on in meters.   <Speed>2.4332</Speed>   * + This tag indicates the speed the runner is running at. This is measured   By meters per second.  <RunCadence>54<RunCadence>   * + This tag captures the run cadence at that given moment.      * + **AvgSpeed** tag calculates the average speed ran in that lap.   + **AvgRunCadence** tag does the same and determines what was the average cadence for that lap.   + **MaxRunCadence** is the same; however, it determines only the MaxRunCadence. |

|  |
| --- |
| Forming Program StructureDATA STRUCTURES USED  1. STRUCT TIME 2. STRUCT LAP 3. STRCUT RUNNERSTATS  Data members for the Time Structure  1. Variable of ***type integer*** **minutes = 0;** 2. Variable of ***type integer*** seconds = 0;  Data Members for the Lap Structure  1. Variables of ***type Double*** are **lapDistance, averageSpeed, maxSpeed = 0**; 2. Variables of ***type integer*** **averageCadence, maxCadence, moveCount = 0;** 3. Variables of ***type struct Time:*** **lapTime, averagePaceInTime**;  Data members for the RUNNERSTATS  1. Vector that has the ***data type Lap*** of splitInformation; 2. Variables of ***type struct Time*** **overalPace overalMovingPace bestPace totalRunTime;** 3. Variables of ***type Integer***: **totalAverageOfCadence = 0, MaxCadence = 0,**   **highestAltitude = 0, lowestAltitude = 200, stopCount = 0, moveCount = 0;**   1. Variables of ***type Double***: **double totalRan = 0, bestSpeedMph, avgMph = 0,**   **pace = 0** |
| Extract Data  1. Step 1: Read the names in the file named database.txt and place them in a vector of type String. 2. Step 2: Erase any previous values left behind from the last query. 3. Step 3: Present Menu interface. This gives the user look at a file or quit the program. 4. Step 4: display the available files for the program to analyze. This task is put through a function that iterates through a vector that contains the different filenames. The user then selects a file and it is passed to the **ifstream** to be read and analyzed. if by chance the file is not found in the directory the method will display that the file could not be found and the function will call itself again to let the user pick a different file to view. 5. Step 5: Like the last step this one and the previous one are intertwined. Before Ifstream can read a file, the index of the file must be decided. This step thus sends an index back so the ifstream can locate the selected file to be read. 6. Step 6: This step takes place after the user indicates which index in the file vector to read from. The file is then placed in a function to determine which information is important. 7. Step 7: If it is identified to be important the information is sent to another function to remove the tags in the original file. 8. Step 8: In this step the data is then sent to separate functions depending on the type of data they are. In this step data is converted, and placed into data members up above. 9. Step 9: This step includes performing mathematical algorithms on data due to the format of the original file. (Unit Measurements |

Test Cases

|  |
| --- |
| **Expectation from the test:**  In this test case I am figure out the best algorithm to perform to get the average pace of the whole run.  Average Pace:  21/3/2017,  Example:  I ran 1.88 mile in 17:49.3 minutes. My average pace was  9:28 minutes / mile. |
| Experiment :  ( Theory 2 )   1. Convert each split time into seconds 2. Add them together 3. Divide them by the distance ran 4. Divide by 60 to get the average pace time in minutes 5. Multiply the digits behind the decimal point by 60 to get the amount in seconds | Code Algorithm:   1. Split a. 8:31 in seconds = 511 sec   Split b. 9:18 in seconds = 558 sec   1. 1069 sec 2. 1069 / 1.88 = 568.6170213 3. 598 / 60 = 9 4. .46666667 \* 60 = 28   Full average pace time 9:28  **CORRECT !!!** |

|  |  |  |
| --- | --- | --- |
| **Average Moving Pace**:  Example:  17:49  Time  15:21  Moving Time | Theory:   1. Scan through the list and look at the speed per meter. 2. If the speed is logged at 0, it indicates the runner was not active. 3. To calculate average moving speed, add any speed that is greater than 0. For each speed track above 0 add to a counter variable. 4. Divide the total by the counter value. | Result:   1. When performing these test I discovered a discrepancy with the data and the calculations with the applications. I originally discovered that some speed tags had 0 which indicated the runner was not moving. So I found the average speed ran by the runner excluding the moments the runner stopped. In doing this I found that my results were different from the Garmin Connect application by 10 seconds sometimes. 2. Other discrepancies I found is that the application presented the average speed ran in two formats. One was in miles per hour, and the second was the average time per mile. After taking a closer examination in the data I noticed that time format is wrong when comparing it with its’ equivalent format in miles per hour. |
| **Best Pace:** | Theory:   1. Scan the speeds and compare it with the adjacent speed. 2. If the adjacent speed is greater than the previous speed, add that speed to be compared with the rest of the speed tags. | C++ Algorithm  Double bestPace = 0;  Loop while list is not empty:     1. Add string into variable 2. If variable is speed 3. Is speed greater than bestPace, if so replace bestPace with variable |

|  |  |  |
| --- | --- | --- |
| Calculating Speed and finding the time equivalent | | |
| Average Speed:  Original format Meters Per second = 2.4570000171661377  Average Speed Is already given under the  <ns3:AvgSpeed>2.5439999103546143</ns3:AvgSpeed>  1 meter = 2.2369 mph  Double Convert2mph = 2.2369; | Step 1. Convert the speed in meters to mile per second. This is done by multiplying it by 2.2369.  Step 2. Divide 1 by this number and you will get a long floating point number.  Step 3. To find the time in minutes multiply by 60.  Step 4. To fin the time in seconds remove any number in front of the decimal point and multiply by 60. | Result:  This method does work; however, this method does not account for the fact the runner may have not completed a full mile. If the runner only runs part way and the formula presented in the steps do not account for this, the lap time will not be accurate.  A resolution to this problem is determine if the runner ran a full mile, if he or she does not then instead of dividing 1 by the miles per hour, you will divide the percentage of the run completed by the miles per. After doing this step the program can continue with the other steps mention in step 3, and 4. |

# Pictures:







