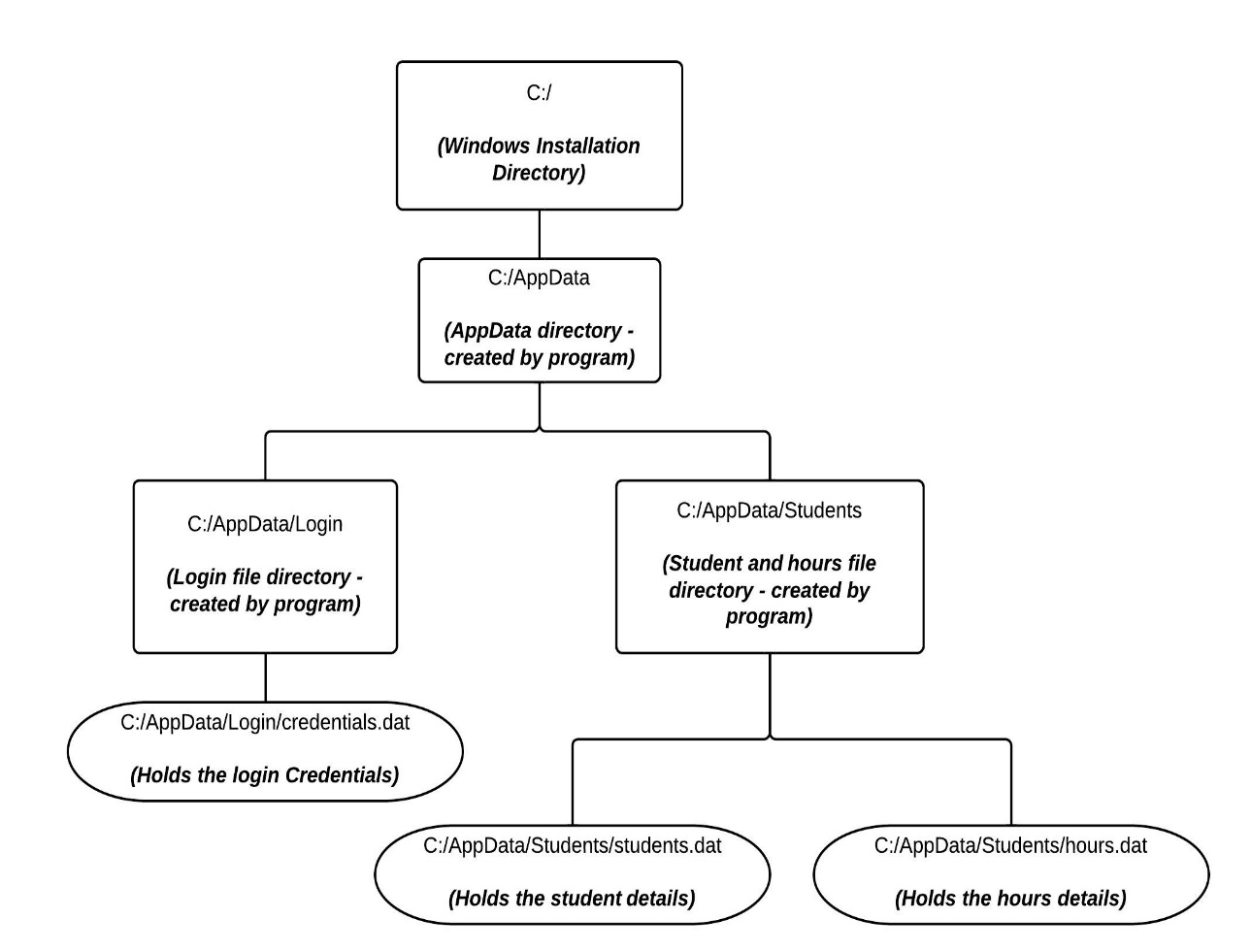
# Criterion B: Design Overview

## B1. File Directories

The program is designed in such a way that when it is executed for the first time on the client’s computer, it will create all the necessary directories. First the program creates an “AppData” folder in the client’s “C:/” drive giving it a directory of “C:/AppData”. This directory will be used by the program to store its files and folders. In the “AppData” folder, there are two more folders called “Login” and “Students”. The login folder is used to hold the file which has the login credentials – the username and the password. The “Students” folder is used to hold the “students” file and the “hours” file. The “students” file holds the ids, first names and last names of all the students while as the “hours” file holds all the hours (creativity, action and service) that are input into the system. The entire file structure/map can be seen below:



## B2. File Structures

Each of the three files: the login file, the students file and the hours file follow a different file structure and length. These file structures and lengths can be seen below:

|  |  |  |  |
| --- | --- | --- | --- |
| **The Login File** *(Total bytes not fixed – user dependent)* | | | |
| **Username** | | **Password** | |
| Data Type:  **String** | Bytes: **n/a** | Data Type:  **String** | Bytes:  **n/a** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **The Students File** *(Total bytes: 26 bytes)* | | | | | |
| **Student ID** | | **First Name** | | **Last Name** | |
| Data Type: **Integer** | Bytes: **4 bytes** | Data Type:  **String** | Bytes:  **11 bytes** *(10 Characters)* | Data Type:  **String** | Bytes:  **11 bytes** *(10 Characters)* |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **The Hours File** *(Total bytes: 26 bytes)* | | | | | | | |
| **Student ID** | | **Creativity Hours** | | **Action Hours** | | **Service Hours** | |
| Data Type: **Integer** | Bytes: **4 bytes** | Data Type:  **Integer** | Bytes: **4 bytes** | Data Type: **Integer** | Bytes: **4 bytes** | Data Type: **Integer** | Bytes: **4 bytes** |

## program_flowB3. Program Flow

## B4. Algorithms

1. ***Directory Creating***

This is the first method that is called by the main(). The main function of this method is to check if the directories and files required for login are present. Usually when the program is to be executed for the first time, these will not be present so it creates them. At the same time it also asks the client for the username and password that he would like to set which will be used henceforth and writes them to disk in the “credentials.dat” file.

1. ***Login***

The login method is the second method to be executed. As the name suggest this methods performs the login operation. It asks the user to input the username and password. It then reads the saved username and password from disk (credentials.dat file) and then compares the two to see if they match. If the two match, it returns a Boolean true value and if not a false value. If the value returned is true, the login is successful and control is transferred to the main menu while as if the return value is false, the login will be unsuccessful and the login method will be recalled until the user inputs the correct username and password.

1. ***Add students***

The add students method is executed when the user presses “1” from the main menu. Its main function is to add more students to the students file. When it is executed, it asks the user to input the first name and last name for the student. It then assigns him the next student ID and writes the three attributes to the students.dat file. It also writes the student ID, 0 creativity, 0 action and 0 service hours to the hours file to initialize CAS hours record. After execution it will ask the user if he wants to add more students or stop execution and return to the main menu. The user can then press “1” if he wants to add more students or any other number to return to the main menu.

1. ***Add hours***

The add hours method is executed when the user presses “2” from the main menu. Its main function is to add CAS hours for an existing student. When it is executed, it asks the user to input the ID for the student for which the CAS hours are to be added to. After a correct student ID is entered it will ask him for the creativity, action and service hours that the user wished to add and write them to the “hours.dat” file. However if an invalid or non existent student ID is entered, an error message will be displayed and the ID will be asked again until a valid one is entered. After execution it will ask the user if he wants to add more hours or stop execution and return to the main menu. The user can then press “1” if he wants to add more students or any other number to return to the main menu.

1. ***View Students***

The view students method is executed when the user presses the number “3” from the main menu. It’s main function is to view the details of saved students. Therefore it first accesses the “students.dat” file and reads all the student IDs, first names and last names and stores them in parallel arrays. It then accesses the “hours.dat” file and reads all the IDs, creativity, action and service hours for all students from it and stores them in parallel arrays as well. If there are several entries for a specific student ID, it sums them up and then stores them. After both the names and the hours are in parallel arrays, it uses a loop to display them for each student and then returns to the main menu.

1. ***Calculation #1: Total CAS Hours per student***

This method is executed when the user presses the number “1” from the calculations menu. It’s main function is to calculate the display the total number of CAS hours that each student has on record. It does this by accessing all the names and IDs from the students.dat file and then using the hours.dat file to read the hours for each student after which everything is stored in parallel arrays. A loop is then used to run through the arrays and calculate the total for each student and display it at the same time.

1. ***Calculation #2: Students who have failed to complete CAS***

This method is executed when the user presses the number “2” from the calculations menu. It’s main function is to display students who have not completed all their CAS hours – do not have at least 50 hours in creativity, action and service. When this method is executed, it reads the students’ IDs and names from the students.dat file and their hours from the hours.dat file and stores them in parallel arrays. It then uses a new loop to transverse the array and using a conditional if checks if any of the three hours (creativity, action and service) are less than 50. If so it displays them otherwise it continues with the loop until it is complete.

1. ***Calculation #3: Students who have completed CAS hours***

This method is executed when the user presses the number “3” from the calculations menu. It’s main function is to display students who have completed all their CAS hours – have at least 50 hours in creativity, action and service. When this method is executed, it reads the students’ IDs and names from the students.dat file and their hours from the hours.dat file and stores them in parallel arrays. It then uses a new loop to transverse the array and using a conditional if checks if all the three hours (creativity, action and service) are greater than or equal to 50. If so it displays them otherwise it continues with the loop until it is complete.

1. ***Calculation #4: Students with under certain criteria***

This method is executed when the user presses the number “4” from the calculations menu. It’s main function is to display students who have a total number of CAS hours under a certain criteria number. When this method is executed, it asks the user what the criteria number is and stores it in a variable. After the user inputs the number, it reads the students’ IDs and names from the students.dat file and their hours from the hours.dat file and stores them in parallel arrays. It then uses a new loop to transverse the hours arrays (creativity, action and service) and calculate the total number of CAS hours. Within the same loop it then uses a conditional if check to see if the total number of CAS hours is less than the criteria number. If so it displays the student and his details and if not it does not display and continues with the next student until the loop is complete.

1. ***Calculation #5: Highest CAS - top 3***

This method is executed when the user presses the number “5” from the calculations menu. It’s main function is to view firstly calculate the value of total CAS hours of each student. After this it stores the value of the total CAS hours in an array. It then runs a bubble sort algorithm to rearrange parallel arrays of the totals, ids, first name and last name arrays in descending order according to totals. It then runs a loop in descending order to display the students with the 3 highest CAS hours after which it breaks the execution and returns control to the main menu.

## B4. Testing

The testing of the program was carried out in three main categories:

1. **Functional Testing**

Functional testing on the other hand involved testing the program to see whether the program did what it was supposed to or not – i.e. if it produced the correct output that it was supposed to. This type of testing helped us to determine if the logical coding of the program was done correctly.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Description** | **Example** | **Result** |
| Directories & Files Creation Test | This test involved running the program to see if it was able to create the required directories and files which it would need to store data. | Executing the program to see if it could create the “students.dat” file in the correct “C:\AppData\Students” directory. | The program was able to create all the required files and directories every time. No errors encountered. |
| Data Writing Test | This test involved running the program to see if it was able to successfully save/write the data entered by the user to disk. | Executing the add students method to add new students to file and then checking if the students have been successfully saved in file. (manual check) | The program was able to successfully write data to disk every time. No errors encountered. |
| Data Reading Test | This test involved running the program to see if it was able to successfully access and read the data that had been previously saved to disk. | Executing the view students method to view all the ids, names and hours details of students who had been already saved to disk. | The program was able to successfully read data from disk every time. No errors encountered. |
| Calculations Test | This test involved running the program to see if it was able to use the data saved on the disk to successfully perform calculations. | Executing the calculation methods to see if the output they generated was correct and as expected. | Initially encountered some logical errors in earlier stages of the program which were then fixed. |
| Display Test | This test involved running the program to see if the output it generated was appropriately formatted and fit to be presented to the client. | Constantly executing the view student method and calculation methods to see how they displayed the reports to the user. | Initially reports generated were not properly formatted. Required tweaking and is now fixed. |

1. **Data Input Testing**

The data input testing involved testing the program with various inputs such as data which was expected, as well as data which was totally not expected. This type of testing helped to see how the program would respond to various types of inputs.

|  |  |  |
| --- | --- | --- |
| **Test** | **Description** | **Result** |
| Blank Name Inputs | Entering blank first or last name when adding students using the addStudents() method. [1 from main menu] | Program does not crash, instead does not allow the user to proceed until he enters some string value. |
| Invalid ID Inputs | Entering a non existent or invalid student ID when adding CAS hours using the addHours() method. [2 from main menu] | Program does not crash, instead displays an error message and tells the user to reenter the ID not allowing him to proceed until he enters the correct ID |
| Invalid credentials Input | Entering invalid username or password during login | Program displays an error message and tells the user to reenter username and password not allowing him to continue until the correct credentials are entered. |
| Entering string data instead of int | Entering an alphanumeric/string input where an numerical/int value is expected. | Program does crash. Error not as yet fixed. |

1. **Runtime Testing**

Lastly runtime testing of the program involved testing the program in variety of runtime environment such as on different computers with different specifications, with different users, etc. This was done to make sure that the program worked correctly in all situations.

|  |  |  |
| --- | --- | --- |
| **Specifications** | **Result** | **Error Cause** |
| Manufacturer: Lenovo Intel Core i7 4702MQ 2.2 -2.3 Ghz 8GB DDR3 RAM @ 1600Mhz Nvidia Gefore GT 740M 1TB Toshiba HDD 5400RPM MICROSOFT WINDOWS 8.1 | Program Executed as expected. | n/a (No Problem Encountered) |
| Manufacturer: Apple Intel Core i5 4250U 1.3Ghz 4GB LPDDR3 RAM @ 1600Mhz Intel HD Graphics 5000 128GB Apple SATA 3 SSD MAC OS X | Program did not execute at all – did not even start. | Failed due to the “.exe” executable not being supported/recognized by Apple’s MAC operating system. Needs a windows runtime environment. |
| Manufacturer: HP AMD E-350 1.6Ghz 3GB DDR3 RAM @ 1333Mhz AMD Radeon HD 6310 320GB Hitachi HDD 7200RPM MICROSOFT WINDOWS 7 – HOME PREMIUM | Program executed as expected. | n/a (No Problem Encountered) |
| Manufacturer: HP Intel Core i5 4200M 2.5Ghz 4GB DDR3 RAM @ 1600Mhz Intel HD Graphics 4600 500GB Hitachi HDD 7200RPM MICROSOFT WINDOWS 8 – HOME PREMIUM | Program executed but failed/crashed after startup. | Crashed due to the main hard disk partition label being “G:\” on the computer instead of “C:\” The program tries to create the required file directories under the “C:\” drive. The lack of this drive label causes program to crash. |