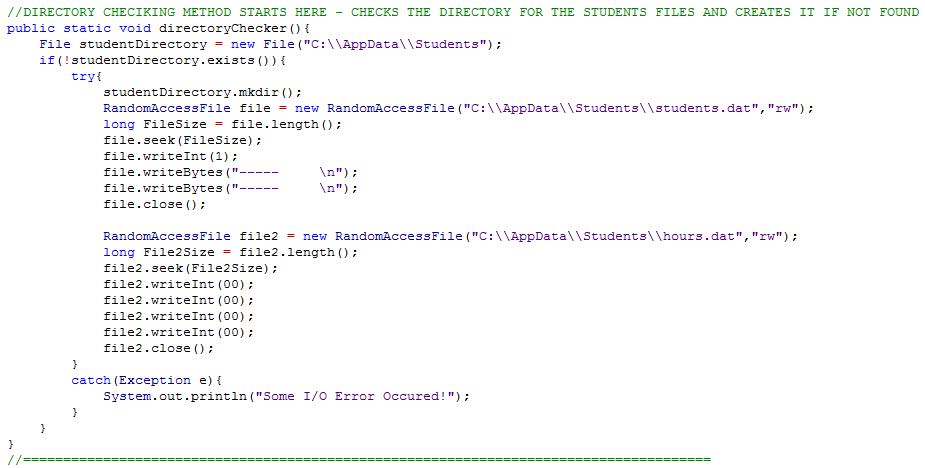
Criterion C: Development

**C. Techniques Used**

The following programming techniques have been used to develop the product:

* Random/Direct File Accessing
* Binary Searching
* Bubble Sorting
* Conditional Decision Making

**C1. Random/Direct File Accessing**

For the program I have used a random/direct access file over a sequential access file to store the data on the hard disk.

Files being initialized here.

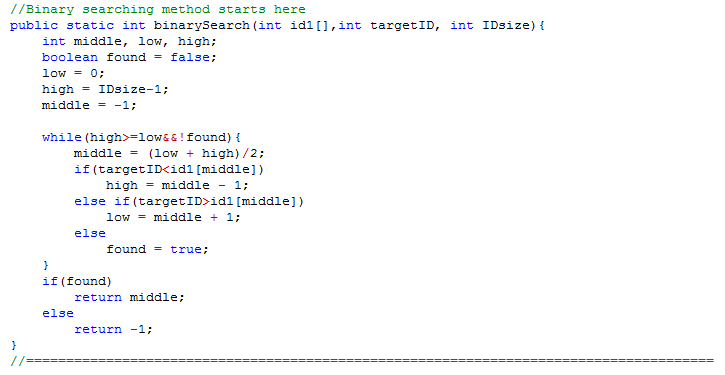
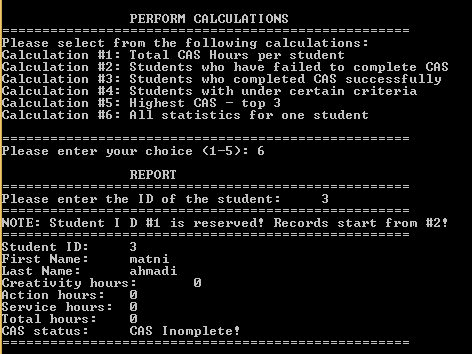
Random Access File being used

**Figure 1Evidence of using Random/Direct Access**

Random/Direct file access is a method of accessing data on file without having to read the whole file from start to end in a sequence. Using this type of file storage and access, allows us to move the file pointer to wherever required within the file to read the data. Reasons for using this type of file access:

* This type of file access/storage was preferred because in both files, the length of records was fixed (int values were 4 bytes and string values were limited to 11 bytes). This made it convenient to use this type of file system.
* When using random access the entire student (or hours) file will not have to be read from start to finish. Instead only the required portion of the file can be read by specifying the location/position of the record. This tends to be much faster than reading the whole file and speed up execution especially if we need to access a record at the end of the file.
* Lastly any errors while writing/reading using random access can only cause that specific record to be lost. However any errors during writing/reading using sequential access can cause all the data to be lost.

**C2. Binary Searching**

For the program I have used binary search to carry out searching operations when a student needs to be found using his student ID because the students are already arranged in ascending order on file according to their student ID. Thus there will be no additional processing involved in binary searching while as at the same time it will be faster for a students’ data base which can potentially be large (benefit of doubt). Should the database have been a small one – then a linear search would be optimal due to lesser records. The following method is called when we user wants the statistics of a particular student to be displayed. Therefore the following binary search method is called to see if the user exists in the file or not. Since the file contents are pre-read into arrays, the method returns the index value of the student if found or returns -1 if the student is not found.

Program asking user to input the ID to be searched

Figure 2Code evidence of implementing Binary Search

**C3. Bubble Sorting**

One of the reports/calculation involves displaying 3 students with highest CAS hours. In order to show the three highest students with CAS hours the program first arranges the students in ascending order. **This sorting is done through a bubble sort**. The bubble sort runs through all the parallel arrays of IDs, first names, last names and totals and arranges them in ascending order according to the total values. It does this by comparing each value to the next value in the array and rearranges them if necessary. After it reaches the end of the array it continues again from the start until the entire array is sorted. After the bubble sort is complete, another loop runs in the reverse order (**descending**) to display the 3 students with highest CAS hours.

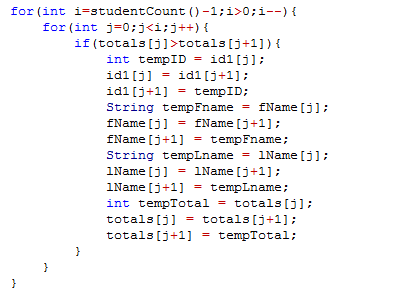


Figure 3Evidence of using Bubble Sort

The main reasons why I felt bubble sort would be adequate are: Firstly bubble sort tends to be very easy to code for programmers in general thus takes less time during the development phase. Furthermore it is easy to understand making dry running the algorithm easier and also helps to debug the program faster.

Since the student database will not be bigger than 100 students per cohort/year, the fact that bubble sort is not as efficient as other sorting techniques (e.g. insertion sort or selection sort) will not make much of the difference. This is because with small databases, such as the one this program is intend for, the performance and speed benefit or selection/insertion sort over bubble sort will not be significant to make it worthwhile for the programmer or user.

**C1(d). Conditional Decision Making**

The program includes numerous instances where the control needed to be transferred based on certain conditions. Thus two of the decision making techniques used in this program are:

* The *“If-Else If-Else”* technique
* The *“Switch-Case”* technique

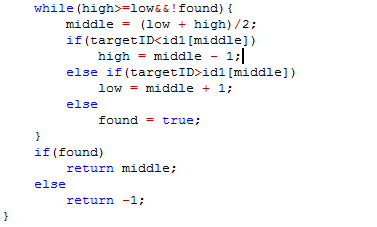
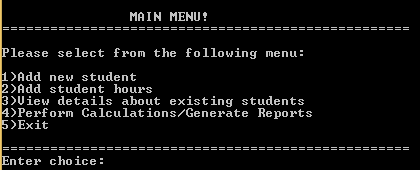


Figure 4Evidence of using conditional decision making using If-Else If-Else

Each of the above techniques was used depending on the situation. I preferred to use *“If-Else If-Else”* technique for usual decision making and transferring control around the various methods in the program due to the fact that it is very simple to use and easy to debug.

On the other hand, I preferred to use the “*Switch-Case”* technique in places where I needed to transfer control in a menu. Both the “Main Menu” and “Calculations” submenu utilize the switch case technique. I felt the “*Switch-Case”* technique was more adequate for menu driven interfaces because of how the code is written for it. Although in general the *“If-Else If-Else”* technique tends to be very easy to program, it is not suitable for long menus with lots of options.

Switch-Case used for menus

[Total Word Count: 894 Words]