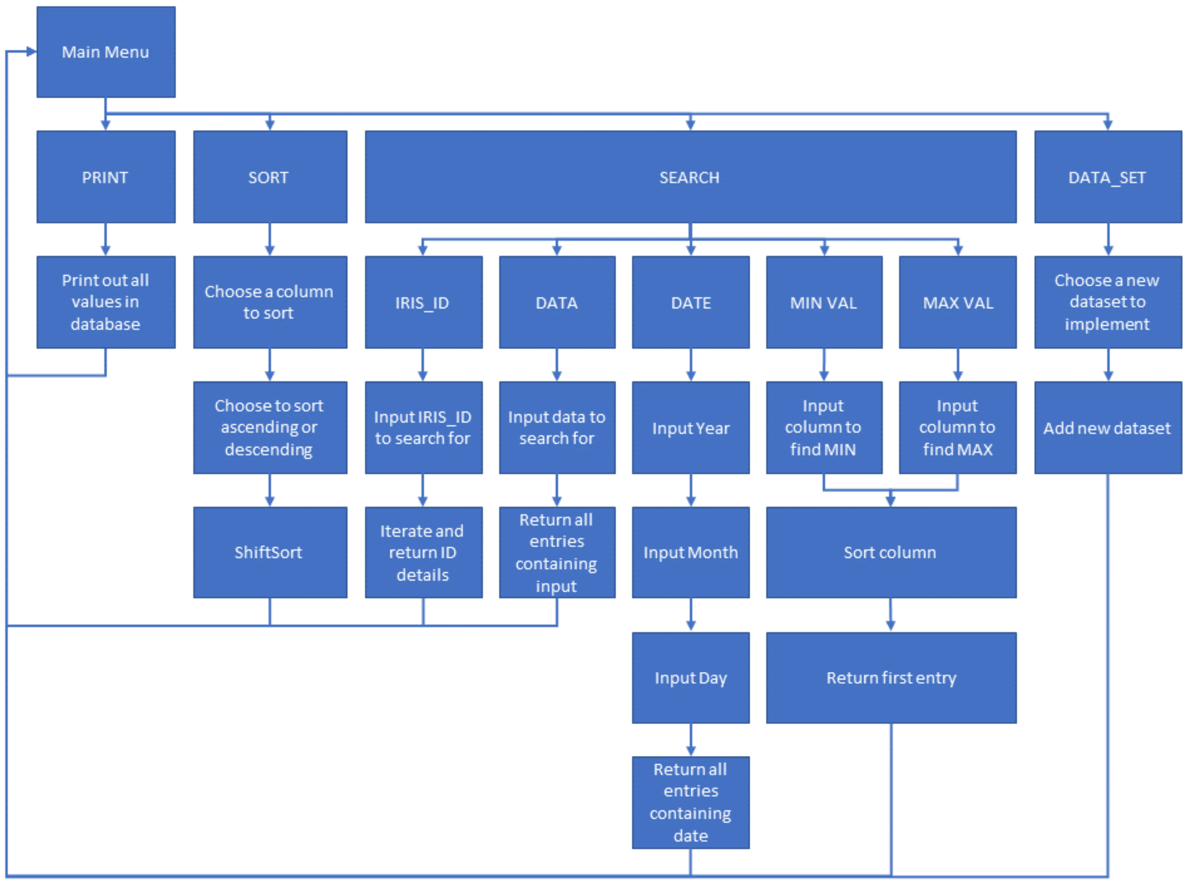
Application Description:

Contents

Application:

The program is broken down into a simple option hierarchy giving the user 4 options at the start, PRINT, SORT, SEARCH, and SELECT_DATABASE.
The PRINT option, simply iterates through and prints out the loaded database (the default one being the first dataset).
The SORT option allows the user a second option telling them to select a column to sort by, giving them the option of [1-11]. Once a column has been selected which is in the range provided, they are given the option to select whether they want to sort the data ascending or descending. Once the data is sorted, and the user is sent back to the first menu.
The SEARCH option takes the user to a choice of 5 sub-options which use search functionality. These are IRIS_ID, DATA, DATE, MIN VAL, and MAX VAL. Selecting IRIS_ID allows the user to enter an ID number to search for, which searches the first column only. DATA allows the user to search all columns for a matching piece of data. MIN VAL and MAX VAL each sort the respective row chosen, and return the first value (the only difference between them being whether they sort the data in an ascending patter, or a descending pattern). The DATE option takes the user to a screen where they must enter in turn, the search criteria for the Year, followed by the Month, followed by the Day. This then searches the Year column, then the results are searched for any which have the Month specified, then of the ones which remain, the Day is searched for. Returning a result if the date entered does not exist. Once the data for any of the searches has been found, it is displayed clearly to the user and they are returned to the main menu.
The SELECT_DATABASE option gives the user an input screen to type the name of another dataset to use. This then reads txt file which is placed in the filed for the program, takes the individual txt files for each column within it, and puts that data into the data table in the same way that the default data is entered.



Algorithms:

Used within this program is an algorithm for searching, and an algorithm for sorting the database; the database of which is created from the file inputs selected.

The algorithm for searching the database if simple and is used to search by IRIS ID, for specific entries, and to search by date; with slight adaptions for each. The algorithm works simply by iterating through the whole array, row by row, checking the specific columns for the IRIS ID and date and returning a result if they match the inputs, but checking every column in each row when the search is for a specific entry in the database.

This method, despite having a worst-case of O(N) for the full search, this is the only method which could be used as any others require the data to be sorted, which would require a sorting algorithm to be used first. These would be highly inefficient given the run-time increase and the memory required also where a full search does not require anything but time. Searching for the minimum and maximum uses the inefficient but effective method of sorting the data and returning the first value. This takes time; but is simple to implement and works effectively.

The sorting algorithm included in the program was created by adapting two existing sorting algorithms the bubble sort, and the quick-sort; customised to work more efficiently with the given information. The adaptations allowed this algorithm to surpass the other algorithms in run-time, with a O(N) best-case and O(N^2) for both the average-case and worst case, making its worst-case equal to that of the bubble-sort, and quick-sort algorithms.

Of the four common issues associated with algorithm development, the quality of solution is irrelevant as there is only one solution, the time to program is irrelevant as the time it took to design and implement compared to writing the adaptions for a conventional algorithm to work with text, was too short to have an impact. The memory limitations are the only part of the algorithm to be flawed in comparison to conventional algorithms, where unlike those which rely heavily on swapping 2 values and only requiring a single string extra memory O(1). Whereas, the adapted algorithm requires an additional O(N^2) of memory to hold the duplicate data set while resorting the rest of the columns.

It works in brief, by utilising the comparison method made popular by the bubble sort, between the first index of element in position 1 of the array the data is being sorted into and the first index of the value in the original array being sorted (SEE Pseudocode below).

```
//get item X of input (current[0])
    //get item N (first elem in output) (sorted[N])
        //get char a of item N (sorted[N][a])
            //see if current[0][a] comes before sorted[N][a]
            //if it does,
                //move all elem from end of sorted to N, +1
            //if it does not,
                //move to next N
            //if they equal,
                //move to next a
```

Using a recursive function the algorithm iterates through the index of the string values being tested against each other. When the two values are identical, the latter is placed before of the one already placed. The values are sorted using an alphanumeric array containing all single characters which appear in the data sets, allowing the same algorithm to sort both numerical (integer, double and float) values and textual string values by reading all values in the array as strings.

The only exception being when a date is sorted, for example months are sorted by their position in the year rather than by an alphanumeric approach. For this, the same algorithm was used, but the alphanumeric array was replaced with an array containing each of the months, and the algorithm was slightly adapted to test positions based on the whole string rather than the first then subsequent characters of the words.

For the unsorted data to be arranged based on the positions of the sorted column, an array is kept of the original positions of the sorted list in the new sorted positions. This list is iterated through once the data sorting is complete and each of the pieces of data (including the one being sorted) are assigned to the row position denoted by the position array. The sorted array is also resorted like this to ensure that all pieces of data are with their respective column details.

Pseudocode:

```
//function shiftSort(input_data, sort_column, sort_data)

    //create array to hold current data              (current)
    //create array to hold sorted data               (sorted)
    //create array to hold sorted data positions     (output)
    //create array to hold data to sort              (input)

    //for each element in column (sort_column)
        //add to input array
    //for each element in input (X)
        //save item X to current          (current[0])
        //save value of X                 (current[1])
        //unsorted = true
        //int N = 0
        //int a = 0
        //while unsorted
            //if position N == null
                //put item X in position N
                //unsorted = true
            //else
                //get item N (first elem in output) (sorted[N])
                //get char a of item N (sorted[N][a])

                //if position of current[0][a] comes before position of sorted[N][a] in sort_data
                    //insert_shift(current, sorted, N)
                    //unsorted = true
                //if position of current[0][a] comes after position of sorted[N][a] in sort_data
                    //N++
                //if current[0][a] == sorted[N][a]
                    //a++
                    //if length of current[a] = a
                        //insert_shift(current, sorted, N)
                        //unsorted = true
        //function insert_shift(current, sorted, N)
        //for each position from final
            //if position not empty
                //save position value
                //break loop
        //for each element from saved position to first position
            //move element N to position, N+1
        //put item X in position N
```

Video URL:

http://youtu.be/ODK1Q9SGNxE?hd=1