University of Wollongong

School Computing & Information Systems

CSCI251/851 Advanced Programming Autumn 2017

Assignment 1 (Due: Week 4, Tuesday 21 March) 8 marks

Aim:

- Design a C++ solution to a problem from a partially complete framework.
- Gain some experience writing C++ database applications.
- Use file I-O in C++ programs.

Instructions:

You are given the task of developing a database that can facilitate the administration of student records. Each record in the database should contain the following information:

```
Student No.    // 8 digit number
Family Name    // up to 20 chars
First Name     // up to 20 chars
Status         // FT / PT
Subjects       // 4 subject codes max
Results        // marks for the above subjects
```

The program is to be implemented in 3 files: **main.cpp** contains the text-menu based user interface. **ass1.h** should contain function prototypes and shared constants (if any)accessible by the **main.cpp** file. **ass1.cpp** contains the function definitions of the database. Test data can be found in **students.txt**. (**Note:** you should put the struct for representing the above information in **ass1.cpp** as this does not get accessed by **main.cpp**)

When the program runs, it will display the following menu:

```
*********************************
*    Student Records Database   *
*     (r)ead data file          *
*     (d)isplay records         *
*     (a)dd record to DB        *
*     (s)earch records          *
*     (q)uit                    *
*********************************
Command:
```

Step 1 involves understanding the menu and declaring an appropriate **`StudentRec`** struct for containing the above information. When deciding what type and array size to make the data fields, you should look at the test data in **students.txt** to confirm it will fit in all the fields. You can assume this data is typical. Although the student number could be stored as an integer, it will be better stored in a character array for search purposes. Also, declare a global array of **`StudentRec`** structs and a global integer for storing the data and counting the number of records in the array. (Note: for testing the database you can assume no more than 100 records are to be stored in the array. The array size can be easily changed by altering a constant and recompiling the source code).

Step 2 involves implementing a **`ReadFile()`** function for loading the array from the data file: `r` reads records from the file **students.txt** into the array of records. For example:

```
Command: r
There are 50 records in the student records database
```

You may assume that all data in the file is correct. Make sure you can handle students with less than 4 subjects, especially the last student in the file.

Step 3 involves implementing a **`DisplayRecs()`** function for displaying records on the screen: Example:

```
Command: d
Student No    91711912
First Name    Mary
Family Name   Cook
Status        FT
Subjects      MATH112 ELEC113 CSCI102 IACT114
Results        PC 46   P  53   D  79   PC 48
Display next record (y/n): y

Student No    16171829
First Name    Mavis
Family Name   Hendley
Status        FT
Subjects      IACT114 MATH112 CSCI102 CSCI114
Results        P  57   P  54   P  54   C  70
Display next record (y/n): n
```

Note: Make sure you strictly comply with the above format, including the addition of the grade to the mark output.

Step 4 involves implementing an **`AddRecord()`** function for adding a new record to the data array: Example:

```
Command: a
++++++++++++++++++++++++++++++++++++++++++++++
            Add Record to Database
++++++++++++++++++++++++++++++++++++++++++++++
Student Number: 21123456
Family Name: White
First Name: John
Status (FT/PT): FT
Enter 4 Subjects and the Results:
CSCI102 77
CSCI103 77
CSCI114 66
MATH112 53
++++++++++++++++++++++++++++++++++++++++++++++
         51 records are in the database
++++++++++++++++++++++++++++++++++++++++++++++
```

You may assume whatever characters appear in the names are acceptable, but always store the first letter as upper case and the rest as lower case. The student number should be checked for length. For the subjects, you can assume FT students always do 4 subjects and PT students 2. Make sure you convert any lower case letters to upper case and check for the correct length of 7 characters. The mark should be in the range 0 to 100. If any input is unacceptable, report the problem to the user, and repeat the input request until it is acceptable. After the record has been successfully added to the database array, implement the WriteFile() function and call it here to update the database file with the new record.

Step 5 involves implementing a `Search()` function. This should request a student number for searching the database. A matching record should be displayed in a similar fashion to the `d(display)` main menu option above. (Hint: a function for displaying one record would be useful.)

Step 6 (for CSCI851 Students Only) This additional step involves modifying the search in step 5 so that wildcard characters can be entered on the searches. (e.g. 61?????? should match any student number starting with 61, while ??2????? would match any number with a third digit 2.) Because there may be more than one match, use a similar "continue" request, as was done for the d(isplay) option.

For example:

```
Enter Student Number: 61??????

Student No    61832845
First Name    Paul
Family Name   Smith
Status        FT
Subjects      CSCI102 MATH112 ELEC113 CSCI114
Results        P  55   P  56   C  67   C  65
Display next record (y/n): y

Student No    61492373
First Name    Harry
Family Name   Gray
Status        PT
Subjects      CSCI114 MATH112
Results        C  67   P  56
Display next record (y/n): n
```

Note: Do not alter the menu options or the input data requirements or your program may not run when we test it on Linux. Avoid inappropriate global variables. It is ok to make the record array and record counters global. Ask if you are unsure. Marks will be awarded for each step you complete.

Implementation:

Appropriately place your code in the three files: **main.cpp**, **ass1.cpp** and **ass1.h**. Ensure each file is appropriately commented (don't forget your name, etc). Make sure your code continues to work after each step before proceeding onto the next step. If you are unsure of exactly what you are to do in each step do not hesitate to ask your lab supervisor or the lecturer. Failure to comply with any of the above requirements may result in marks being deducted. (The onus is on you to resolve anything you find unclear or ambiguous).

Marking:

Steps 1-5 are worth 1.6 marks each (1.6 x 5 = 8 marks). CSCI851 students who fail to complete Step 6 will receive a 1 mark deduction for failing to meet the post-grad requirements. <u>You must demonstrate your program during week 4 or 5. Failure to demo on time will result in a 1 mark deduction for each week late.</u>

Submit:

Before submitting your files, test your program with the input given on the last page and save the output to a file named "output.txt". Submit your files using the submit facility on UNIX as shown below:

```
$ submit –u login –c CSCI251 –a1 main.cpp ass1.h ass1.cpp output.txt
```

where 'login' is your UNIX login ID (Note: CSCI851 should also submit to –c CSCI251).

Deductions will be made for untidy work or for failing to comply with the submission instructions. Requests for alternative submission arrangements will only be considered before the due date. An extension of time for the assignment submission may be granted in certain circumstances. Any request for an extension of the submission deadline must be made to the Subject Coordinator before the submission deadline. Supporting documentation must accompany the request for any extension. Late assignment submissions without granted extension will be marked but the points awarded will be reduced by 5 points for each day late. Assignments will not be accepted if more than three days late.

Before you submit your code, run your program and enter the following input in response to the user prompts. Save your screen output in a file named "output.txt". Do not edit your screen output. Marks will be deducted if your program fails to run properly or is poorly formatted. Also, copy and paste the test input (below) into "input.txt" and run your program with the input redirection on. Make sure your program runs correctly on this test data and demo this to the Tutor when you demonstrate your program.

```
r
d
y
y
n
s
21123456
A
21123456
White
John
FT
CSCI102 77
CSCI103 77
CSCI114 66
MATH112 53
s
21123456
q
```