COMP 3512

Assignment 1

In this assignment, you are asked to implement a program to handle student records. Our purpose is to practise using the STL containers & algorithms.

The program basically reads student records from a file & allows the user to search for matching records. To reduce the amount of validation the program needs to perform, we assume that the records in the file are all valid.

1 Student Records

Information about a student consists of an ID, a name (first name & last name) & a number of grades. A grade in turn consists of a course & a score. The following shows 2 sample student records in the data file:

```
a22222222
ned flanders
2
comp1510 90
comp2510 85
a11111111
homer simpson
3
comp2510 25
comp2525 60
comp3512 45
```

In the above, the first line contains the ID & the second line contains the name (which consists of a first name followed by a last name). This is then followed a line that contains the grade count — a non-negative integer that indicates the number of grades that the student has. For example, the above shows that ned flanders has two grades (for 2 different courses) whereas homer simpson has three. The grades then follow the grade count, each on a separate line & each consisting of a course (e.g., comp1510) & a score (e.g., 90).

The grade count is necessary in order for the program to tell how many grades to read when it reads a student record.

As mentioned, we assume that the records in the data file are all valid. Furthermore, we assume that

1. there are no duplicate student IDs, i.e., every record has a unique student ID; and
2. a student cannot have multiple grades for the same course.

2 Data Structures

We will refer to the information about a student as a student record, or simply, as a record. We will store such information in a "student" object. This means that we need to implement a `Student` class. Since we will only be dealing with valid records, we'll use an STL pair for a name. We will also use a map to store the grades of a student — it maps a course to the corresponding score.

```
typedef std::pair<std::string, std::string>  Name;   // first  name, last name
typedef std::map<std::string, int>           Grades; // map courses to scores

class Student {
public:
  void display(std::ostream& os) const;
  // friend declarations & additional function declarations if necessary
private:
  std::string  id_;      // e.g. "a22222222"
  Name         name_;    // e.g. {"ned", "flanders"}
  Grades       grades_;  // e.g. {{"comp1510", 90}, {"comp2510", 85}}
};
```

We need to overload the input operator (`operator>>`) to read a student. By calling it twice, we should be able to read back the 2 sample student records shown in section 1. The implementation of this operator may assume that any data it is able to read is valid.

We won't be saving student records to files in this assignment but we'll be displaying them. This is the purpose of the `display` method in the `Student` class. For the record with ID a11111111 in section 1, the output of `display` should be:

```
a11111111
homer simpson
comp2510 25
comp2525 60
comp3512 45
```

Note that the grade count is not displayed.

3 The Program

The program must be invoked with the name of a student record file as a command-line argument. It reads in all the records & stores them in an STL vector. Note that the records in the data file are not sorted in any way.

After reading & storing the records, the program goes into a loop where it displays a prompt, reads a command from the user & executes the command. This repeats until the user presses the end-of-file key, whereupon the program exits.

The prompt must be printed to standard error so that we can separate it from regular output.

Commands are used to search records. All valid commands must start with the word `show`, `-show`, `showid` or `-showid`. Both `show` & `-show` display all information about matching records using the `display` method. The difference is that `show` displays matching records in ascending order of IDs whereas `-show` displays them in descending order. Both `showid` & `-showid` only display the IDs of matching records, the former in ascending order & the latter in descending order. *These commands all display matches to standard output.*

If `show`, `-show`, `showid` or `-showid` is not followed by any words, all records are displayed in the appropriate format & order.

Otherwise, the word that follows `show`/`-show`/`showid`/`-showid` is the subcommand & specifies whether we want to search by ID, student name or grade:

- if the subcommand is `id`, it must be followed by the ID we want to look for;
- if it is `name`, it must be followed by the first name & then the last name we want to look for. The asterisk (`*`) may be used for the first name & for the last name — it means match any first name & any last name respectively. (We assume that valid first & last names cannot be just an asterisk); and
- if it is `grade`, it must be followed by the name of a course & then by one or two integers: if there is one integer, we are looking for students whose score in the specified course equals that integer & if there are two integers, we are looking for students whose score in the specified course is between the first & the second integer inclusive.

Any other subcommand is invalid.

Note that the 3 valid subcommands expect different "arguments". Extra words after those arguments are simply ignored.

If a command is valid & there are matching records, they are displayed to standard output. After displaying the records, a blank line is printed, again to standard output. If there are no matching records or if a command is invalid, a blank line is simply printed to standard output — no other message should be printed. If the user presses the end-of-file key when prompted for a command, the program exits, in this case without printing a blank line.

Here are some examples of valid commands with explanations:

- `show` : display all records in ascending order
- `show id a11111111` : display record with ID `a11111111`
- `-show name homer simpson` : display records with name `homer simpson` in descending order of IDs
- `show name homer *` : display records with first name `homer` in ascending order of IDs
- `-showid name * simpson` : display, in descending order, the IDs of records with last name `simpson`
- `show name * *` : this basically displays all records in ascending order of IDs
- `showid grade comp3512 100` : display, in ascending order, the IDs of records where the score in `comp3512` is `100`
- `show grade comp2510 0 50` : display records where the score in `comp2510` is between 0 & 50 inclusive in ascending order of IDs

4 Additional Requirements & Information

Do not use global variables. Put the implementations related to `Name`, `Grade` & `Student` (see section 2) in a header file. The rest of the program can be in one or more files.

Note that all matching records (including the blank lines) are displayed to standard output. All other output should be printed to standard error.

Use generic algorithms, either ones provided by the STL or ones that you implement yourself. Also you must implement & use at least one class of function objects.

Sample data, input & output files will be provided. Make sure that your program generates exactly the same outputs as those provided.

5 Submission and Grading

This assignment is due at 11pm, Wednesday, March 2, 2016. Submit a zip file to `In` in the directory:

```
\COMP\3512\a1\set<X>\
```

where `<X>` is your set. Your zip file should be named `<name_id>.zip`, where `<name_id>` is your name & student ID separated by an underscore, e.g., `SimpsonHomer_a12345678.zip`. Do not use spaces in your zip file name. Your zip file must unzip directly to your source files without creating any directories. We'll basically compile your files using

```
g++ -std=c++11 -W -Wall -pedantic *.cpp
```

after unzipping.

Do not submit rar files. They will not be accepted.

If you need to submit more than one version, name the zip file of each later version with a version number after your ID, e.g., `SimpsonHomer_a12345678_v2.zip`. If more than one version is submitted, we'll only mark the version with the highest version number. (If it is not clear to us which version to mark, you may fail to get credit for the assignment.)

Your program must compile without errors or warnings using the command shown above. If that is not the case, you may receive a score of 0 for the assignment. Otherwise, the grade breakdown for this assignment is *approximately* as follows:

Design & Coding style	10%
Displaying all records	20%
Search by ID	15%
Search by name	25%
Search by grade	30%

Note that in order to get any credit for the above, your program must at least be able to read the data file, create `Student` objects & store them in a vector.