COMP 3512

Assignment 2

In this assignment, we'll be using STL containers & algorithms. We'll be performing searches on student records.

1 The Classes

We'll need a `Student` class. We'll also be dealing with student grades. So we'll also need a `Grade` class.

The `Grade` class is very simple. A grade consists of 2 pieces of information — a course title & a score. The `Grade` class is basically as follows:

```
class Grade {
public:
  // ctor(s) & additional functions if necessary
  friend std::ostream& operator<<(std::ostream& os, const Grade& g);
                                        // sample output: comp3512 90
  friend std::istream& operator>>(std::istream& is, Grade& g);
  // additional friend declarations if necessary
private:
  std::string  course_;  // e.g. comp3512
  int          score_;   // e.g. 90
};
```

At a minimun, we'll need a (default) constructor, `operator<<` & `operator>>`. The intention is that `operator>>` is used to read grades previously written out by `operator<<`. Hence it may assume that if it is able to read data, the data read is valid (i.e. both course & score are valid).

A student has an ID, a name & a number of grades. The `Name` & `Student` classes are basically as follows:

```
class Name {
public:
  // ctor(s) & additional functions if necessary
  friend std::ostream& operator<<(std::ostream& os, const Name& n);
                                        // sample output: homer simpson
  friend std::istream& operator>>(std::istream& is, Name& n);
  // additional friend declarations if necessary
private:
  std::string  first_;  // e.g. homer
  std::string  last_;   // e.g. simpson
};

class Student {
public:
  // ctor(s) & additional functions if necessary
  void display(std::ostream& os) const;
```

```
    friend std::istream& operator>>(std::istream& is, Student& s);
    // additional friend declarations if necessary
  private:
    std::string         id_;          // e.g. a11111111
    Name                name_;
    // an STL container of Grade objects
  };
```

Again, for both classes, `operator>>` is used to read data (stored in a file) previously written out by `operator<<`. Hence it may be assumed that any data `operator>>` is able to read is valid. The following shows 2 sample student records in a data file (written using `operator<<`):

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

Note the two numbers 3 & 2 above. They indicate the number of grades that follow. So in the above, homer simpson has grades for 3 courses while ned flanders only has grades for 2 courses.

you may need to add additional member functions to the 3 classes. Also, you may need other non-member functions

It is assumed that grades for each student is sorted in increasing order of course titles & each student has no duplicate grades (i.e. grades with the same course title). *Any STL container that you use to store the grades must maintain their order.* (This is to standardize the output of the program to facilitate testing.) Clearly, any of the sequence containers (if used appropriately) will work. But given that grades are sorted & that there are no duplicates, it is also possible to use a set or a map.

We won't be saving student records to files in this assignment. So we don't really need to implement `operator<<`. However, we need a way to display student records. Hence the `display` method. For homer simpson above, this should display:

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

Note the omission of the line containing the number of grades in this output.

For the 3 classes mentioned, you may need to implement additional member & non-member functions to those listed. For example, you may need get methods or functions to compare students, names, or grades.

2 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 container.

We assume that the student records within the data file are already sorted in increasing order of student IDs & that there are no duplicate IDs. *Any STL container that you use to store them must maintain this order.* As with grades in a student, it is possible to use a set, a map or any of the sequence containers. Furthermore, if you so choose, you may store pointers to `Student` objects (rather than the objects themselves) in the container. However, in this case, you may need to specify an ordering if you use a set or map.

After reading & storing the records, the program goes into a loop where it displays a prompt, reads a command from the user, then 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 `showall` or `showid`. `showall` means show all information about matching records (using the `display` method) whereas `showid` only displays the IDs of matching records. *Matching records are always displayed to standard output.*

If `showall` or `showid` is not followed by any words, all records are displayed (fully or just their IDs).

Otherwise, the word that follows `showall/showid` specifies whether we want to search by ID, student name or grade:

- if the word 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 are assuming that valid first & last names cannot be asterisks.)
- 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 two integers, we are looking for students whose score in the specified course is between the first & the second integers inclusive.

In all cases, any extra words are ignored. If a command is not valid, it is skipped — no error message is necessary.

After executing a (valid) command, the matching records are displayed. If there are no matching records, no output is necessary.

Here are some examples of valid commands with explanation:

- `showall` : show all records fully (i.e. not just IDs)
- `showall id a11111111` : show student with ID a11111111
- `showall name homer simpson` : show students with name homer simpson
- `showall name homer *` : show students whose first name is homer
- `showall name * simpson` : show students whose last name is simpson

- `showall name * *` : this basically displays all records
- `showid grade comp3512 100` : show the IDs of students whose score in comp3512 equals 100
- `showall grade comp2510 0 50` : show the students whose score in comp2510 is between 0 & 50 inclusive

3 Additional Requirements & Information

You are not allowed to use global variables.

Note that all matching records are displayed to standard output. All other output should be printed to standard error.

Try to use generic algorithms, either ones provided by the STL or ones that you implement yourself. Some useful STL algorithms are `for_each`, `find` & `find_if`. You may also find function objects as well as function pointers useful for this assignment. Note also that maps already provide lookups & sets have a `find` method.

4 Submission and Grading

This assignment is due at 10pm, Saturday, November 15, 2008. Submit your source file(s) to `In` in the directory:

```
\COMP\3512\a2\set<X>\<id_name>\
```

where `<X>` is your set and `<id_name>` is your student ID & name separated by an underscore (e.g. `a00000001_SimpsonHomer`). Do not use spaces to separate your name. We'll basically go into each `id_name` directory & compile your files using something like

```
g++ *.cpp
```

but with additional switches. So make sure that you put your actual source files in that directory & not, for example, just a zip file. If you ever need to submit more than one version, name the folder of each later version with a version number after your name, e.g., `a00000001_SimpsonHomer_v2`.

If your program does not compile, you may receive zero 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 be able to read in the data file, create `Student` objects & store them (or their addresses) in an STL container. This means that you won't get any credit for displaying all records if you simply print back what's in the data file (skipping some lines).