

Advanced Programming in C++

Afshin Sepehri - Summer 2002

Project 2 – Student Registry

Purpose

To practice using C++ input and output streams and also working with files. You will also get familiar to design a very simple relational database system.

1. Overview

In this project you design a simple database system, which performs the student registry operations of a school. You register students, courses and the grades of the students on the corresponding files on disk and then retrieve data and get reports from them.

2. Program Definition

2.1. File Structure

There are two data files in this project storing information of *students* and their *registered courses*. These files are named *students.dat* and *courses.dat* respectively. Look at the contents of each file in the following:

2.1.1. students.dat

This is a *text file* containing the information of the students including *student id* and *student name* in this order. A *space (blank)* character separates these two fields of data. Each student data is located in one single line in the file. The length of the lines is variable based upon the length of the name and id. *Student name* is a *variable length alphabetic* string. *Student id* is a *positive integer* number in the valid range. (e.g. 123456)

2.1.2. courses.dat

This is a *binary* file, which includes the registry information of the students. The contents of this file are *student id* (referring to a student in file *students.dat*), *course number* (an *alphanumeric* string with size 5, e.g. EE620), *grade* (a floating point number) and a flag (character) showing whether the course is *graded* for the student or not. You may choose the two characters 'Y' and 'N' for this purpose. The following structure can be used for reading or writing the *records* of this file:

```
struct TCourse {
    int studentId;
    char courseNo[6]; /*one character for NULL*/
    char graded;
    float grade;
};
```

Note that records in this file are in a *fixed length format*. This makes *random access* to the records possible, which may not be used in this project, of course. In addition, you can read or write one *complete* record at once and by using one single read or write command.

2.2. Type of Input and Output Devices

Inputs and outputs can be sent to and received from the program in two different ways: Using standard input and output consoles (*keyboard* and *screen*) or through input and output *files*. This can be determined in the beginning of the program by asking a question from the user:

Do you want to use the standard input and output (Y/N)?

The user enters *Y* (*Yes*) or *N* (*No*) in response. If the answer is *Y*, all the commands are received from *keyboard* (standard input) and outputs are reported on *screen* (standard output). In case of *N* as the answer, the name of the input and output files are asked from the user:

Enter the name of the input file:

Enter the name of the output file:

From then on, all the commands are read from the mentioned input file and all the report messages are sent to the output file.

2.3. Input Commands

There are *five* commands in this project, which should be handled by the program. The number and order of the commands is up to the user. The end of the commands can be detected by a character *end of file*, which is automatically located in the input file or is entered by the user (*<ctrl-d>*) in case of using *standard* input. Here is description of the commands:

2.3.1. Register

By this command, user introduces a new student to the system. The format of this command is “*register <student_id> <student_name>*”, which *student_id* is the id and *student_name* is the name of the student to be registered. You can assume that the entered student id is always new to the system, which means it does not exist in the file. The student information is *appended* to the file *student.dat* and NO message is reported.

2.3.2. Add

This command adds a number of courses for a student. The format of this command is “*add <student_id> <course_no> [<course_no> . . .]*”, which *student_id* is the id of the student and the *course_no* is the course number to be added. As it is shown in the syntax of the command, there may be more than one course added by a single command. Look at the primary input for an example. This command *appends* the entered data to the file *courses.dat*. You can assume that the entered course numbers are *new* to the file for this student. Also assume, the requested student exists in file *student.dat*. No message is reported.

2.3.3. Drop

This command drops a course for a student. The format of this command is “*drop* <student_id> <course_no>”, which *student_id* is the id of the student and the *course_no* is course number to be dropped. This command removes the entered data from the file *courses.dat* ONLY in case that it is a non-graded course. Otherwise, it ignores the command. Obviously, a student cannot drop a course, which is already graded. No message is generated in any case.

2.3.4. Grade

The format of this command is “*grade* <student_id> <course_no> <grade>”. By this command, the program registers the *grade* (a real number) for the student *student_id* in course *course_no*. If the course is already graded, the new grade *replaces* the old one. Assume the mentioned course is already registered (added) for the desired student. No message is generated in response.

2.3.5. Report

The format of this command is “*report* <student_id>”. By this command, the program reports the student specified by *student_id* and his/her registered courses in the format of the following *BNF*:

```
<report> ::=
  <double_line><student_info><single_line><courses_info><single_line><summary>
<student_info> ::=
  Student<b>ID:<b><student_id><b>Name:<b><student_name><endl>
<student_id> ::= <integer_no>
<student_name> ::= <student_name><alpha> | <alpha>
<courses_info> ::= | <courses_info><course>
<course> ::= <course_no>:<b><grade><endl>
<course_no> ::= <alpha><alpha><digit><digit><digit>
<grade> ::= <real_no> | No Grade
<summary> ::= Number of Courses:<b><integer_no><b>GPA:<b><real_no><endl>
<integer_no> ::= <integer_no><digit> | <digit>
<real_no> ::= <integer_no>.<digit><digit>
<alpha> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
W | X | Y | Z
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<double_line> ::=
  <ddashes><ddashes><ddashes><ddashes><ddashes><ddashes><endl>
<single_line> ::= <dashes><dashes><dashes><dashes><dashes><dashes><endl>
<ddashes> ::= <ddash><ddash><ddash><ddash><ddash>
<dashes> ::= <dash><dash><dash><dash><dash>
<ddash> ::= =
<dash> ::= -
<b> ::= blank character ( ‘ ’ )
<endl> ::= end of line character ( ‘\n’ )
```

In the summary part of the report, you need to count all the courses related to the student (graded or non-graded) and also calculate his/her *GPA*. For calculating *GPA* just consider the graded courses. The *GPA* should be reported as a *fixed-point real number* with 2 digits after decimal point. *GPA* can be calculated by dividing the summation of all graded courses for the desired student to the number of graded courses. If *no* course is graded for the student, *GPA* will be *0.00*.

For sample reports, refer to the *primary_output* coming in a later section.

2.4. Hints and Requirements

1. The size of the file *courses.dat* should be kept as *minimum* as possible. On the other hand, when a course is dropped, it should be *physically* removed from the file and all the following records should be shifted up. This causes the size of the file to be shrunk by the size of one record.
2. You do NOT have to replicate your functions to be able to work with both standard input and outputs and the files. You can pass the *handle* of the input or output stream to the functions as discussed in class.
3. This program is *case insensitive*. That is all the inputs can be entered in *lower case* or *upper case* or a *combination* of both. For instance, there is no difference between *afshin*, *Afshin*, or *AFSHIN* in the program. As it is shown by the *report BNF*, all the student information should be reported in *upper case* in the output.
4. All the input commands are single line commands. It means character *end of line* can be detected for the end of a command.
5. The only output generated by the program is in response to the command *report* and no other command causes program to report a message. However, you may add some *temporary* messages to track the steps done by the program. Do NOT forget to remove these temporary messages before submission.
6. The main purpose of this project is working with files. It requires you to store all the activities (*register*, *add*, *drop*, *grade*) on the relevant files. This causes your program to have a *permanent state*, which can be retrieved in next runs of the program. Therefore, do NOT remove any file at the end of the program. Your project will be evaluated by the result of a number of consecutive executions.
7. File *courses.dat* is a *binary* file, so the contents of this file may NOT be *readable*. On the other hand, data are stored in this file in a way stored in memory. Then do NOT look for the numeric values (integer or floating points) as strings in the file. However, you may work on them to find out how data are stored in memory or to find out the *Endianness* (*little* or *big*) of the machine you are working with. This is an arbitrary research, which is NOT required for the sake of this project.

8. Do NOT worry about the *range* of the *grades*. This can be chosen by the user and consequently, the GPA will be in the same range.

9. In command *report* note that there may be no course added for the student. In this case the *courses_info* part of the report is empty as shown in *BNF*.

10. Courses should be reported in *the order of insertion*, which is same as their orders in the file *courses.dat*.

3. Primary Input File

If your program works properly, it should be able to handle ANY combination of inputs that I can come up with, within the bounds of this project description. In fact, your project will be tested with a wide variety of inputs.

Note that the input and output of this program are different from the previous projects. In fact, you choose the type of them by answering a question in the beginning. Then do NOT use the *Unix redirection commands*. To get you started, I have provided a file called "*primary_input*" that you may use as input to test your program. To use the contents of this file as your input instead of typing from the keyboard, enter its name as the input name while you are working with files (answer *N* to the first question).

Look over the contents of this file to be sure that you understand different types of command that your program must be able to handle. Here are the contents of the primary input file. It can also be accessed through class web page:

```
register 123 Afshin
register 65415 ARASH
add 123 EE646
add 65415 EE721 EE620 EE621
report 65415
grade 65415 ee621 91.4
register 3 babak
add 123 ee759
drop 65415 ee620
register 53123 shabnam
add 53123 EE620
grade 123 ee646 85.23
report 123
report 3
grade 65415 ee721 95
grade 53123 EE620 90.1
report 65415
reort 53123
```

4. Primary Output File

Below are the contents of the output that should be produced by your program when it is fed the *primary_input* file that I have given you.

You can easily check (using the Unix *diff* command) if your output matches the *primary output* file when the program is fed the *primar_input* file:

```
diff -bwi output primary_output
```

If you receive NO message from this command, it shows that your generated output is identical to the *primary_output* and this is desired. Switch *-bwi* is used to ignore any extra *blank* or *white space* or *case sensitivity mismatch*.

Here are the contents of the *primary_output* file:

```
=====
Student ID: 65415 Name: ARASH
-----
EE721: No Grade
EE620: No Grade
EE621: No Grade
-----
Number of Courses: 3 GPA: 0.00
=====
Student ID: 123 Name: AFSHIN
-----
EE646: 85.23
EE759: No Grade
-----
Number of Courses: 2 GPA: 85.23
=====
Student ID: 3 Name: BABAK
-----
-----
Number of Courses: 0 GPA: 0.00
=====
Student ID: 65415 Name: ARASH
-----
EE721: 95.00
EE621: 91.40
-----
Number of Courses: 2 GPA: 93.20
=====
Student ID: 53123 Name: SHABNAM
-----
EE620: 90.10
-----
Number of Courses: 1 GPA: 90.10
```

5. General Requirements

1. There is no assumption on the order of inputs.

2. You are NOT allowed to use any *global* variable in this project. That is all your variables should be *local* or passed to functions as *formal* parameters.

3. Use C++ specific input and output stream commands and work with C++ *file streams*. You are NOT allowed to use any traditional C input and output or file commands such as *printf*, *scanf*, *fread*, *fwrite*, *fscanf*, *fprintf*, *fputc*, *fgetc* and so on.

4. All the variables in this project are allocated statically. You are NOT allowed to use *new* or *delete* operators. Moreover, all the required information should be retrieved from the files. You are NOT allowed to use any array of structures keeping a huge amount of data for using in later commands.

6. Compilation Process

This project is a C++ program. Then consider *.cc* as the extension of your file (*proj2.cc*). Use the *g++* compiler. The syntax is:

```
g++ -Wall proj2.cc -o proj2
```

In this way an executable file named *proj2* is created. (If you get no compiler error!) Then you can run *proj2* and test it. Switch *-Wall* is to show all warnings. Pay attention to all of the warnings. You are supposed to submit a project with NO error and NO warning.

7. Before Submission

1. Test your project thoroughly on a variety of input cases. I have provided the primary input and output to get you started. Your project will be tested (and graded) on a variety of different input cases, so it is important that your program can handle all possible cases.

2. IMPORTANT: At a bare minimum, your program MUST produce the correct output when given the *primary_input file*. To test your program on these files, enter *N* as the answer to the first question and enter *primary_input* and an arbitrary name (other than *primary_output*) as names of input and output files. Then use the *diff* command as described earlier. If your program does not produce the correct output file when given the primary input file, you will receive a grade of *zero* on this project.

8. Submitting your Project

1. You will submit ONLY your source code, which will be a file called "*proj2.cc*". No other files will be accepted. Do NOT try to submit your executable. Do NOT try to submit your output or created files.

2. To submit the project, attach your source file to an email and send it to me in address *afshin@cs.umd.edu*. Use the expression *project_submission* as the subject of your email. Your programs are to be graded automatically. Hence mistyping even a letter causes problem. Hence, I cannot accept any submission missing this subject. If you receive no reply from the auto-grader in 5 minutes, it is probably because *auto-grader* is down for any unpredictable reason. Please resubmit your project after about one hour.

3. You may submit the project more than once, but ONLY the LAST submission will be considered. Then, you are encouraged to submit it as much as you want to get feedback from the auto-grader. However, if your last submission is after the due date, it will be considered as your submission date.

9. Documentation and Style Requirements

1. At the top of your `.cc` file, please specify (using comments) your name.
2. You must *indent* properly and obey all the styles to be discussed in class.
3. You must choose proper variable names.