Data Structures Homework #4

GRADE BOOK

Write a program that automates an instructor's grade book. This program processes commands to add student names, record and alter grades, calculate a final semester grade and print out the student information in different orders. The instructor (the user) interacts with the program through a simple command-based user interface.

Input

1. The retained student information, generated in previous executions of this program, is input from file "Grades.dat" at the beginning of each execution of the program.
2. The commands from the instructor are input (in response to "user-friendly" prompts from the program) as single letters. The commands are discussed in detail in the Processing instructions below.

Output

1. Responses to user commands are to be written to the screen, as described in the Processing instructions below.
2. A summary of each command's transaction must be written to a text file called "Grades.trn". You may determine the format of this file; it should be labeled and formatted clearly.
3. The output from command O (Output) should be printed to file "Grades.out", as described in the Processing instructions below.
4. The updated grade information must be saved to file "Grades.dat".

Processing

The user commands are printed to the screen as a menu, and the user responds with a single-letter command. Some commands require the program to prompt the user to enter additional information. After each command is processed, the menu should be redisplayed. The commands are to be processed as described in the following table:

COMMAND INSTRUCTIONS

S	Set up for new semester. The program must prompt the user for the number of programming assignments (range 0 .. 6), the number of tests (range 0 .. 4), and final exams (range 0 .. 1). It must also prompt the user for the relative weights of programs, tests, and final exam in determining the students' grades. The relative weights are expressed as percentages of the semester grade, and must add up to 100%.
A	Add a student. The program must prompt the user for the student's name (last name and first name, strings of at most 20 characters), and student number (integer in the range 1 - 9999).
P	Record programming assignment grade for all students. The program must prompt the user to ask which program number is to be recorded. Upon receiving a valid program number, the program must prompt the user with each student's name (processing them alphabetically), at which time the user inputs the program grade for that student. A valid program number is one that is less than `numPrograms`, and that has not been previously recorded. If the program number is not valid, print an appropriate error message.

T	Record test grade for all students. The program must prompt the user to ask which test number is to be recorded. Upon receiving a valid test number, the program must prompt the user with each student's name (processing them alphabetically), at which time the user inputs the test grade for that student. A valid test number is one that is less than `numTests`, and that has not been previously recorded. If the test number is not valid, print an appropriate error message.
F	Record Final exam grade for all students. The program must prompt the user with each student's name (processing them alphabetically), at which time the user inputs the final exam grade for that student. If final exam grades have been previously recorded, print a message.
C	Change a grade for a particular student. The program must prompt the user to get the student number, the new grade, and the type of grade to change (P, T, or F, as described above).
G	Calculate final grade. Add the program grades for each student and store the average in the student's record. Add the test grades for each student and store the average in the student's record.
O	Output the grade data, ordered alphabetically by name (last name/first name) or by student number (in increasing order). The name, student number, and grade book data must be printed out in the order indicated, to file "Grades.out". Format the data with appropriate labels.
Q	Quit. Save all the student record data to the "Grades.dat" file, and terminate the program.

NOTE: The S (Setup) command must be made (once per semester) before any of the other commands can be executed. If this command is made in a particular execution of the program, the "Grades.dat" input file should be ignored, and the grades data structure should be initialized to the empty state.

Program Enhancement

Modify the O (Output grades) command to prompt the user to indicate in which order the grade data must be printed: alphabetically by name (last name/first name) or by student number (in increasing order).

Data Structures

The design of the grade book data structure is an important part of this program. You may use any of the ADTs that we have discussed so far this semester. The "application" parts of your program may only access the data structure through the set of operations that are specified for the ADTs that you use. The more effective your use of abstract data types, the better your program (and your grade) will be.

Schedule

This is a large program, which you should implement and test in more than one development iteration.

1. In the first iteration, you should build your program "shell" -- the outer control structure, leaving stubs in place of the code for all of the command processing.
2. In the second iteration, you should replace the stubs for the A (Add student), O (Output grade data), and Q (Quit) commands with the actual processing procedures.
3. In the third iteration, you should replace the stubs for the P, T, and F commands with the actual processing procedures. To support the range checking in these procedures, the stub for the S command should set the number and weights of tests, programs, and final exam as follows:

Item	Number	Weight (%)
Programs	4	50
Tests	2	20
Final exam	1	30

4. In the fourth iteration, you should replace the stubs for the S, C, and G commands with the actual processing procedures.

Testing

The testing of each of the development iterations may be performed using test data of your choice. You must develop and implement a testplan at every iteration of program development. You need to execute the program more than one time. Make a hard copy of each of the output files ("Grades.trn" and "Grades.out") after each test run, as the program rewrites these files each time it is executed.

Deliverables

- Tested source program listing for each iteration of program development.
- Hard copies of files "Grades.dat" and "Grades.out", from each of the final test executions.
- Implemented test plans and test drivers if appropriate.