Programming Languages CS 181
Winter of 2005

Programming Assignment #2
Class Records Manager in Prolog
Due: Sunday, March 13, 2005, 11:59 pm

Skeleton Program:

```
class_records :-   getCommand(Command),
                   executeCommand(Command, Status),
                   displayReport(Command, Status),
                   fail.

class_records :-   write_ln('Good Bye!').
```

Note:

- The predicate "**`getCommand`**" sequentially retrieves individual commands to be processed by the **class_records** manager. I will provide this predicate during grading. **DO NOT** include it in your submitted programs.
- The predicate "**`executeCommand`**" attempts to faithfully execute the given command and returns a status report.
- The predicate "**`displayReport`**" must output a clear and complete message to the screen providing the detailed status of the execution of the last command. The message must include:
  - Listing of the actual command
  - English description of the meaning of the command
  - Any output required by the command itself.
  - Actions that have been performed
  - Any problems encountered, including any command syntax problems.
- Initially there is no data in the database.
- You can use dynamic predicates **`assert`** and **`retract`** only to store class data.

Types of Commands:

There are several types of commands. Required arguments are listed in **red** – all the others are optional.

In the following commands ID is an integer **key** and thus it **must be always unique**, First, Middle, Last, Name and Major are atoms, Active is either yes or no, Grade is integer 0-100.

The meaning of most of the following record maintenance commands is self-explanatory.

Additions:

(Note: You can only add items which do not yet exist)

a) `add_student( id(ID), name(last(Last), first(First), middle(Middle)), major(Major), active(Active)).`
`(Active=yes is the default)`

b) `add_exam( id(ID), name(Name), weight(Weight), max(Max), active(Active) ).`
`(max means that exam scores must be between 0 and Max. Max=100 and Active=yes is the default)`

c) `add_grade( student_id(SID), exam_id(EID), grade(Grade), active(Active)).`
`(Active=yes is the default)`

Updates:

(Note: If an argument is instantiated then it updates the previous value. Otherwise the value is unchanged)

d) `update_student( id(ID), name(last(Last), first(First), middle(Middle)),major(Major), active(Active)).`

e) `update_exam( id(ID), name(Name), weight(Weight), max(Max), active(Active) ).`

f) `update_grade( student_id(SID), exam_id(EID), grade(Grade), active(Active) ).`

Queries:

(Note: In all queries - except for find queries - all inactive entries must be ignored)

g) `student_grades(student_id(ID), Grade_List).`

`Returns a list of all active grades of a given student as a list: [ (Exam_ID, Grade), (Exam_ID, Grade), … ].`

h) `exam_grades(exam_id(ID), Grade_List).`

`Returns a list of all active grades for a given exam as a list: [ (Student_ID, Grade), (Student_ID, Grade), … ].`

i) `total_grade(student_id(ID), Total_Grade).`

`Returns the weighted percent average of all active grades of a given student. Total_Grade is an integer. Remember that e.g. a score of 30 on an exam with Max=40 is equivalent to a score of 75/100.`

j) `mean_grade(exam_id(ID), Mean_Grade).`

`Returns the average of all active exam grades (in percents). Mean_Grade is an integer 0-100. For example, a score of 30 on an exam with Max=40 is equivalent to 75/100.`

k) `find_students (id(ID),name(last(Last),first(First), middle(Middle)),major(Major), active(Active), Student_ID_List). (Finds all matching students and returns a list Student_ID_List of IDs of matching students.`

Good Luck!