

University of Toronto at Scarborough
CSC C24
Scheme Programming Project

Due Wednesday, February 14, 2001
12:30pm electronically and 1:00pm in the dropbox

This assignment tests your knowledge of functional programming in Scheme. It is worth 10% of your final grade.

1 Description of Assignment

In this assignment you will extract different sorts of information from a database of courses. For each course, the database describes which course (if any) logically follows it, who teaches the course, and which courses (if any) are prerequisites of the course.

We will encode the database in Scheme as a list of entries, with one entry for each course in an imaginary Computer Science Department. Each entry will be a list with the following form:

```
(<course #> <next course> <prof> <prereq1>...<prereqN>)
```

where the number of prerequisites for the course is N. If the course has no prerequisites, the list will have the form:

```
(<course name> <next course> <prof>)
```

If the course has alternative prerequisites, they are grouped in a list. For example:

```
(cs301 cs302 Smith (cs102 ee220) cs201)
```

means that cs301 requires cs102 OR ee220, and also requires cs201.

When the course has no logical successor, <next course> will be designated by the symbol NONE.

We will use a `define` statement to associate a name (a Scheme symbol) with the representation of the database. The Scheme code for encoding a *sample* database is:

```
(define COURSES
'((cs101 cs102 Petty)
  (cs102 NONE Petty cs101 math101)
  (cs201 cs202 Vega (cs102 ee105) math101)
  (cs202 cs330 Gabriel cs201)
  (cs230 cs330 Vega cs201 math102)
  (cs330 NONE Vega (cs202 cs230) math102)
  (cs340 cs341 Gabriel cs202)
  (cs341 NONE Gabriel cs340)
  (cs380 NONE Gabriel cs102)
  (math102 NONE Lee math101))
)
```

After loading this `define` statement, you can treat COURSES as a global variable whose value is the database list.

IMPORTANT NOTES:

- You may not use iteration (e.g., `do` or `loop`), or assignment statements (e.g., no functions that end with `!`; all *define* statements must define functions, with the exception of the course database definition). In short, you must follow pure functional programming style. Furthermore, you may not use any of the `let` expressions either without my explicit permission; if you think you need them to write good code, you should speak to me about it.
- A sample database definition is given in the `define` statement above, and posted on the course web page. You may use this in developing and testing your code, but **the code you turn in should not include a define statement for the database**. This example is for your use during code development; we will run your submitted code on a different database, but (obviously) one with the same encoding as described above.
- With the description of each function below are some sample calls to the function to illustrate what is intended by the textual description. These are not intended to be a complete test suite. You should thoroughly test your code by devising your own tests.

YOUR TASKS:

1. Write the following functions:
 - (a) `SEQ`: takes a single argument (a course name) and returns the next logical course following that one.
 - (b) `PROF`: takes a single argument (a course name) and returns the name of the professor that teaches that course.
 - (c) `PREREQS`: takes a single argument (a course name) and returns a list of prerequisites for that course.

For example, given the sample database:

```
> (SEQ 'cs101) ⇒ cs102
> (SEQ 'cs102) ⇒ NONE
> (SEQ 'ee105) ⇒ NONE
> (PROF 'cs202) ⇒ Gabriel
> (PROF 'ee105) ⇒ NONE
> (PREREQS 'cs101) ⇒ ()
> (PREREQS 'cs102) ⇒ (cs101 math101)
> (PREREQS 'cs201) ⇒ ((cs102 ee105) math101)
> (PREREQS 'ee105) ⇒ ()
```

HINT: Each of these functions takes a single argument, but the functions need to step through the `COURSES` database using recursion. This means that you will need a helper function to recursively step through the `COURSES` database.

2. Write the function `TEACHES` that takes a single argument (a prof name) and returns a list of courses that the prof teaches.

- > (TEACHES 'Lee) ⇒ (math102)
- > (TEACHES 'Vega) ⇒ (cs201 cs230 cs330)
- > (TEACHES 'Lester) ⇒ ()

3. Write PREREQ-OF, a function that takes a course name and returns a list of courses that it is a prerequisite of. The list should include courses for which the course is an alternative prerequisite (note the call to (PREREQ-OF 'cs202) below).

- > (PREREQ-OF 'cs101) ⇒ (cs102)
- > (PREREQ-OF 'cs202) ⇒ (cs330 cs340)
- > (PREREQ-OF 'cs380) ⇒ ()
- > (PREREQ-OF 'ee201) ⇒ ()

4. Write ALLOWED?, a boolean function which takes two arguments—a single course name and a list of courses—and returns true (#t) if the list of courses includes the prerequisites for the single course, and false (#f) otherwise.

The intention is that the input argument that is a course name represents a course the student wants to take, and the input list argument represents courses that the student has taken. The result thus indicates whether the single course is allowed to be registered for.

- > (ALLOWED? 'cs101 '()) ⇒ #t
- > (ALLOWED? 'cs101 '(cs102 cs201 math102)) ⇒ #t
- > (ALLOWED? 'cs330 '(cs230 math102)) ⇒ #t
- > (ALLOWED? 'cs330 '(cs202 cs230)) ⇒ #f
- > (ALLOWED? 'ee105 '(cs101 cs102)) ⇒ #f

5. Write ALL-ALLOWED, a function that takes a single argument—a list of course names—and returns a list of courses for which the courses in the input argument list include the prerequisites. The returned list must not contain courses that are in the input argument list. The input list represents courses that a student has taken, and the output list courses that they are allowed to take.

- > (ALL-ALLOWED '()) ⇒ (cs101)
- > (ALL-ALLOWED '(cs101 math101)) ⇒ (cs102 math102)
- > (ALL-ALLOWED '(ee105 math101)) ⇒ (cs101 cs201 math102)
- > (ALL-ALLOWED '(cs101 cs102 cs201)) ⇒ (cs202 cs380)
- > (ALL-ALLOWED '(ee201)) ⇒ (cs101)

6. Write PREREQS-REC, a function that takes one argument, a course name, and returns a list that contains all of its prerequisites, as well as all of their prerequisites, and the prerequisites of those, etc.

PREREQS-REC can be defined recursively as: C_i is in PREREQS-REC of C_j if C_i is a prerequisite of C_j , or C_i is a prerequisite of one of the PREREQS-REC of C_j . The only exception is that if a list of alternative prerequisites is added to the list, none of their prerequisites should be recursively added. Furthermore, the returned list should contain no duplicates.

- > (PREREQS-REC 'cs101) ⇒ ()
- > (PREREQS-REC 'cs102) ⇒ (cs101 math101)
- > (PREREQS-REC 'cs201) ⇒ ((cs102 ee105) math101)
- > (PREREQS-REC 'cs202) ⇒ (cs201 (cs102 ee105) math101)
- > (PREREQS-REC 'cs230) ⇒ (cs201 math102 (cs102 ee105) math101)
- > (PREREQS-REC 'cs330) ⇒ ((cs202 cs230) math102 math101)
- > (PREREQS-REC 'cs341) ⇒ (cs340 cs202 cs201 (cs102 ee105) math101)
- > (PREREQS-REC 'ee105) ⇒ ()

7. The following functions can easily be written using calls to the set of functions you have already written above. Your code will be graded on the use of higher-order functions to accomplish these tasks. You may also need additional “helper” functions to “clean up” the list to return.

NOTE: No list returned by any of these functions should contain duplicate elements.

- (a) CLASSES-PROFS: takes a single argument (a list of profs) and returns a list of all classes that they are teaching.

- > (CLASSES-PROFS '(Vega Lee)) ⇒ (cs201 cs230 cs330 math102)
- > (CLASSES-PROFS '(Vega Who)) ⇒ (cs201 cs230 cs330)

- (b) ALLOWED-PROFS: takes a single argument (a list of course names representing courses that the student has taken) and returns a list of the names of all profs that are teaching courses that the student would be allowed to register for (based on the courses they've taken).

- > (ALLOWED-PROFS '()) ⇒ (Petty)
- > (ALLOWED-PROFS '(ee105)) ⇒ (Petty)
- > (ALLOWED-PROFS '(cs101 math101)) ⇒ (Petty Lee)

- (c) ALL-SEQ: takes a single argument (a list of course names representing courses that the student has taken) and returns all the logical successors of the courses (ie, suggested courses to take). Thus, the return list should not contain any courses in the input argument list. Furthermore, the return list of suggested courses should only contain valid course names, and not the atom NONE.

- > (ALL-SEQ '(cs101 math101)) ⇒ (cs102)
- > (ALL-SEQ '(cs101 cs102 cs201 math101 math102)) ⇒ (cs202)
- > (ALL-SEQ '()) ⇒ ()
- > (ALL-SEQ '(ee105)) ⇒ ()

- (d) NEEDED: takes two list arguments (a list of course names representing courses the student has taken, and a list of course names representing courses the student wants to take in the future), and returns a list of additional courses that the student must take before all the courses of the “wanted” list can be registered for. Note that the returned list of “needed” courses may contain alternatives, as in the specification of prerequisites.

- > (NEEDED '(cs101 math101) '(cs102)) ⇒ ()
- > (NEEDED '(cs101 cs102) '(cs340 cs380)) ⇒ (cs202)

- > (NEEDED '(cs101 cs102 cs201 math101) '(cs230 cs330)) ⇒ ((cs202 cs230) math102)
- > (NEEDED '() '(cs101 math101)) ⇒ ()
- > (NEEDED '(cs101) '(ee105)) ⇒ ()
- > (NEEDED '(ee105) '(cs201 cs202)) ⇒ (math101 cs201)
- > (NEEDED '(cs101) '()) ⇒ ()

2 To Be Handed In

2.1 On paper

Staple (no paperclips, no folded corners!) the following together with the cover sheet (available on the course web page) on top:

- A printout of all of your code.
- A printout of one or more script files containing test runs for each function. (See `man script` if you are unsure of how to create a script file.)
- A written report on your testing strategy.

2.2 Electronically

In addition to your paper submission, you must submit your code electronically.

- Put all definitions of the functions above, as well as any helper functions, in a file called `cl.scm`.
- **Do not include** the `define` statement for the COURSES database in the file you submit. If you do so, you will lose marks on the project.
- Submit the file `cl.scm` using the following command on `fissure.scar`:

```
submit -N p1 csc24s cl.scm
```

See `man submit` if you want more information about how the `submit` command works. Note that if you have already submitted a file and you wish to replace it with a different version, you can submit it again. But you must use the `-f` option to warn `submit` that you really mean to overwrite the old version.

3 How the Assignment Will Be Marked

It is worth emphasizing that your assignment will be marked not only for correctness, but for functional programming style, comments, and your testing strategy as well.

Furthermore, we will test the code that you submit electronically on our own test cases, using a new COURSES database. For that reason, you **must** use the exact function names and argument lists that we have specified, and use the exact file name and submit instructions given above.