UNIVERSITY OF TORONTO
Faculty of Applied Science and Engineering

FINAL EXAMINATION, DECEMBER 2006

Third Year — Computer Engineering, Electrical Engineering, Engineering Science Computer Option

CSC 326H1 F — Programming Languages

Exam Type: D

Aids allowed: one two-sided aid sheet on standard letter paper.

Examiner — J. Clarke

Check that this test paper has 10 pages, including this cover page.

Answer all questions in the space provided in this paper. When the response requires you to write a program, your marks will depend on the style of your program as well as its correctness. Comments are generally not necessary. Helper functions, methods or predicates are always allowed.

PLEASE COMPLETE THIS SECTION.

NAME: Family name ____________________

Given names ____________________

STUDENT NUMBER ____________________

for use in marking:		
	1.	/15
	2.	/5
	3.	/20
	4.	/15
	5.	/20
	Total	**/75**

1. [15 marks]

This question is to be answered using Python. You may use any modules or functions from standard Python that you wish to use.

Here are a program and its output:

PROGRAM

```
from classes import Student, TA

s1 = Student('Jim')
s1.addmark(67)
s1.addmark(92)
s1.addmark(53)
print s1

s2 = Student('Mary')
print s2

ta = TA('Josh')
ta.add_stu('Jim')
ta.add_stu('Mary')
ta.add_stu('Alice')
ta.add_stu('Bob')
ta.add_stu('Clara')
ta.add_stu('David')

ta.mark('Jim', 56)
ta.mark('Bob', 62)
ta.mark('Clara', 89)
ta.mark('David', 75)
ta.mark('Alice', 42)

ta.mark('Jim', 73)
ta.mark('Clara', 99)
ta.mark('David', 65)

print ta.marks_list()

print ta['Jim']
```

OUTPUT

```
Student: Jim: 70.6666666667
Student: Mary: no marks available
TA: Josh -- marks list:
  Student: Jim: 64.5
  Student: Clara: 94.0
  Student: Alice: 42.0
  Student: David: 70.0
  Student: Bob: 62.0
  Student: Mary: no marks available

Student: Jim: 64.5
```

The file `classes.py` contains the classes `Student` and `TA`.

Write the required classes, `Student` and `TA`. Your mark will be better if your code is clear and brief.

No error checking is required, but it is not an error for a `Student` to have no marks.

1. (continued)

2. [5 marks]

<u>This question is to be answered using Scheme.</u> You may use these standard Scheme functions and special forms: `car`, `cdr`; the variants `caar`, `cadr`, `cdar`, …, `cdddddr`; `cons`, `length`; `define` (but you may define only functions, not variables); `lambda`, `quote`, `eval`, `cond`, `if`, `and`, `or`, `not`; `let`, `let*`, `letrec`, named `let`; `map`, `apply`, `eq?`, `eqv?`, `equal?`, `null?`; arithmetic comparisons and operations; `number?`, `symbol?`, `list?`, `zero?`, `string->list`. This list of permissions is the same as in question 3.

Standard functions that are not allowed include `list`, `member?`, `append`, `reverse` and `set!`. If you require those or other standard functions, you must define them yourself.

Write a function `inorder` that takes two parameters, a string `p` and a string `s`. The return value is a true if all the characters of p appear in s in their proper order, and false otherwise. It is allowed for other characters of p to appear between the characters of p.

For example, `(inorder "abc" "abracadabra")` returns true, and `(inorder "hi" "there")` returns false.

3. [20 marks = 5 + 5 + 10]

<u>This question is to be answered using Scheme.</u> You may use these standard Scheme functions and special forms: `car`, `cdr`; the variants `caar`, `cadr`, `cdar`, …, `cdddddr`; `cons`, `length`; `define` (but you may define only functions, not variables); `lambda`, `quote`, `eval`, `cond`, `if`, `and`, `or`, `not`; `let`, `let*`, `letrec`, named `let`; `map`, `apply`, `eq?`, `eqv?`, `equal?`, `null?`; arithmetic comparisons and operations; `number?`, `symbol?`, `list?`, `zero?`, `string->list`. This list of permissions is the same as in question 2.

Standard functions that are not allowed include `list`, `member?`, `append`, `reverse` and `set!`. If you require those or other standard functions, you must define them yourself.

This question concerns gift-giving. The kinds of things involved in gift-giving are *recipients*, *gifts*, and *categories*. All are atoms. For example, `jim` might be a recipient, `sports-stuff` might be a category, and `basketball` might be a gift.

There are also three kinds of lists:

- A *wish-list* is a list of pairs. The first element of each pair is a recipient, and the second element is a list of wished-for categories. Here is an example wish-list:

```
((jim (book footwear)) (mary (video-game sports-stuff clothing)))
```

- A *category-list* is also a list of pairs. The first element of each pair is a gift, and the second element is a category. Essentially, a category-list gives the types of gifts. Here is an example category-list:

```
((basketball sports-stuff) (x-box video-game) (keds footwear)
   (python-in-a-nutshell book) (helmet sports-stuff))
```

- A *stock-list* is a list of available gifts. Here is an example stock-list:

```
(keds helmet x-box python-in-a-nutshell)
```

You may assume that there are no errors in the lists, and that all gifts in `stock-list` are included in `category-list`. However, all functions should handle any empty-list condition sensibly.

(a) Write a function `getwishes` that returns a list containing a recipient's wishes.

That is, `(getwishes who wish-list)` returns the second element of the pair in `wish-list` that has `who` as the first element. If `who` is not a recipient in `wish-list`, return the empty list.

Using the example lists above, `(getwishes 'jim wish-list)` should return `(book footwear)`.

3. (continued)

(b) Write a function `countwishes` that returns a list of the number of wishes for each recipient.

That is, `(countwishes wish-list)` returns a list of pairs that is the same as `wish-list` except with the second element of each pair replaced by the number of elements in the corresponding second element in `wish-list`.

Using the example lists above, `(countwishes wish-list)` should return `((jim 2) (mary 3))`.

For this part, you may *not* define helper functions (but as always you may use lambda expressions).

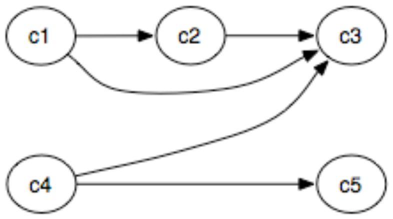
(c) Write a function `giftchoices` that returns a list of the available gifts that will satisfy a recipient's wishes. More formally, `(giftchoices who wish-list category-list stock-list)` returns a list of the gifts that appear in `stock-list` and that according to `category-list` belong to categories requested by recipient `who` in `wish-list`. You may call the functions you defined in parts (a) and (b), if you wish.

Using the example lists above, `(giftchoices 'jim wish-list category-list stock-list)` should return `(python-in-a-nutshell keds)`. The order of the items in the returned list is unimportant.

4. [15 marks = 6 + 5 + 4]

This question is to be answered using Prolog. You may use these standard Prolog operations: list head and tail extraction with `[ | ]`; `not`; cut; `fail`. If you require other standard predicates, you must define them yourself. This list of permissions is the same as in question 5.

This question concerns student enrolments in courses, and the prerequisite relationships between courses. Students and courses are atoms such as `jim` and `mary` (who are students), and `c1` and `c2` (which are courses). Here is a diagram of the particular courses we will be interested in here:



$$\begin{tikzcd} c1 \arrow[r] \arrow[rr, bend right] & c2 \arrow[r] & c3 \\ c4 \arrow[urr] \arrow[rr] & & c5 \end{tikzcd}$$

The arrows indicate prerequisite relationships. For example, the arrow from course `c1` to course `c2` indicates that `c1` is a prerequisite for `c2`. We will use a different definition of "prerequisite" from the one common at the University of Toronto. Here, we will say that a student has satisfied the prerequisite for a course if she or he has completed *any* of the prerequisite courses (rather than *all* of them as is more common). In our example, a student can satisfy the prerequisite requirement for `c3` by taking any of `c1`, `c2` or `c4`.

(a) We have two students: `jim` and `mary`:

- `jim` has completed `c1` and is enrolled in `c3` and `c5`.
- `mary` has completed `c4` and is enrolled in `c3` and `c5`.

Write Prolog facts that state what we know about prerequisites, completed courses and student enrolments. Call the predicates `prq`, `done` and `enrol`. Choose the order of the parameters as you like.

4. (continued)

(b) Write a predicate `noprq(Stu, Crs)` that succeeds if `Stu` is enrolled in `Crs` but has not completed any pre-requisite for `Crs`. In our example, `noprq(jim, c5)` should succeed, and `noprq(mary, c3)` should fail.

(c) Are there any limitations on the use of your predicate `noprq`? If so, what are the limitations? If there are none, explain why not.

5. [20 marks = 5 + 5 + 5 + 5]

<u>This question is to be answered using Prolog.</u> You may use these standard Prolog operations: list head and tail extraction with `[ | ]`; `not`; cut; `fail`. If you require other standard predicates, you must define them yourself. This list of permissions is the same as in question 4.

In this question, `L1` and `L2` are lists.

(a) Write a predicate `rmevens(L1, L2)` that succeeds if `L2` is the same as `L1` except that the even-numbered element of `L1` do not appear in `L2`. (The first element is numbered 1.) For example, `rmevens([1, 2, 3, 4], [1, 3])` succeeds.

(b) Write a predicate `rmsec(L1, L2)` that succeeds if `L2` is the same as `L1` except that the second-last element of `L1` does not appear in `L2`. For example, `rmsec([1, 2, 3, 4], [1, 2, 4])` succeeds. If `L1` has fewer than two elements, `rmsec` should fail.

5. (continued)

(c) Write a predicate `rev(L1, L2)` that succeeds if `L2` is the same as `L1` except that the order of its elements is reversed. For example, `rev([1, 2, 3], [3, 2, 1])` succeeds. In addition to the predicates generally allowed in this question, you may use `append` — and in fact you *must* use it.

(d) Repeat part (c), but this time without using `append`. Instead, you must use an accumulator. (An accumulator is a data structure that is built as the program works toward a solution.)