UNIVERSITY OF TORONTO

Faculty of Applied Science and Engineering

FINAL EXAMINATION, DECEMBER 2005

Third Year — Computer 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.	_____/5
	2.	_____/10
	3.	_____/10
	4.	_____/15
	5.	_____/10
	6.	_____/10
	7.	_____/10
	8.	_____/10
	Total	**_____/80**

1. [5 marks] **This question is to be answered using Python.**

Write a function `everyother` that takes one parameter, a list, and returns a list containing every other element of the parameter, starting with the first element.

For example,

```
everyother([1, 2, 'buckle', 'my', 'shoe'])
```

should return

```
[1, 'buckle', 'shoe']
```

You may assume the parameter is in fact a list.

2. [10 marks] **This question is to be answered using Python.**

The regular-expression module `re` in Python returns "match objects" that have a `group()` method. This method takes one parameter, a string or an integer, and returns the part of the match that has the corresponding name or sequence number. In this question you will do something similar in an artificial class.

Write a class `NamedList`. The constructor (the `__init__` method) takes two parameters besides `self`: `names` and `vals`. The `names` parameter is a list of strings, and `vals` is a list of values of any type. The two lists (`names` and `vals`) are of the same length. The parameters `names` and `vals` are to be saved as instance variables, also called "`names`" and "`vals`".

A `NamedList` also has a method `lookup()` that takes one parameter `p`. If `p` is an int, `lookup(p)` returns the p^{th} item in `vals`. If `p` is a string, `lookup(p)` returns the k^{th} item in `vals`, where `names[`k`]` is `p`.

Write the entire class `NamedList`, including the constructor and `lookup()`. No other methods are required.

You must not use a dictionary. You do not need to worry about what happens if `p` is out of range (if it is an integer) or does not appear in `names` (if it is a string). You may assume that `names` and `vals` really are the same length.

3. [10 marks] **This question is to be answered using Scheme.**

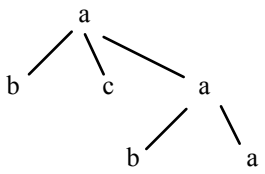
Write a function `cflist` that takes one parameter, a string `S`. The return value is a list of functions, with one function for each character in `S`. Each function takes a single character as parameter, and returns true or false. The *k*th function in the list returns true if its parameter is the same as the *k*th character of `S`, and false otherwise.

For example, if `S` is "`hi`", then there are two functions in the list returned. The first function returns true if its parameter is the character '`h`', and the second returns true if its parameter is '`i`'.

4. [15 marks = 12 + 3] **This question is to be answered using Prolog.**

In this question we deal with trees that are not binary and are not search trees. "Not binary" means that each tree node may have any number of children, from 0 upwards. "Not search trees" means that there is no particular relation between the location in the tree of any one node label and any other node label.

Here is an example of a non-binary non-search tree:



We will represent the children of a node as a list of subtrees, and each tree will be a structure `node(Label, Children)`. Thus, our example tree in the picture would be

```
node(a, [ node(b, []), node(c, []), node(a, [node(b, []), node(a, [])]) ])
```

To clarify that expression, the part corresponding to the lowest-level nodes is underlined.

(a) Write a predicate `occurrences(X, T, N)` that succeeds if `N` is the number of times the element `X` occurs in the tree `T`. For example, if `T` is the tree in the picture above, `occurrences(a, T, N)` would find `N = 3` as an answer, while `occurrences(b, T, N)` would find `N = 2`. You may assume that `T` is not empty.

(The space below may not be quite sufficient. Please use the back of the previous page if necessary.)

(b) In part (a) you assumed that the tree was not empty. Partly that is because the node structure as defined here cannot represent an empty tree. How would you modify the representation to allow empty trees?

5. [10 marks = 3 + 3 + 4] **This question is to be answered using Prolog.**

In this question, `L` is a list and `X`, `Y` and `A` are possible elements of the list.

(a) Write a predicate `third(L, X)` that succeeds if `X` is the third element of `L`. For example, if `L` is [3, 4, 5, 6], then `X` should be 5.

(b) Write a predicate `nth(L, X, N)` that succeeds if `X` is the N^{th} element of `L`. As in part (a), elements are counted starting from 1, not from 0. You may assume that `N` is instantiated.

(c) Write a predicate `after(L, A, X, Y)` that succeeds if `A` is an element of `L`, `X` appears in `L` immediately after `A`, and `Y` appears immediately after `X`.

6. [10 marks = 6 + 4]

(a) Here is a complete Prolog program.

```
eval(Num) :-
    print(Num),
    nl,
    fail.

eval(Num) :-
    Num > 0,
    N1 is Num - 1,
    eval(N1).

do(Num) :-
    eval(Num).

do(Num) :-
    print('exit after call with parameter = '),
    print(Num),
    nl.

go :-
    do(5).

:- go.    % This line is the first query called.
:- halt.  % This line is the second and last query called.
```

What is the output when this program is run? (Hint: There is more than one line of output.)

(b) Prolog has three central features: unification, resolution and backtracking. Which one would you say is most obviously illustrated by the program in part (a)? Explain your answer in a sentence or two.

7. [10 marks] = 1 + 1 + 1 + 2 + 2 + 3]

All three languages — Python, Scheme and Prolog — have a built-in function or predicate that can be used to find the length of a list. Here, we assume that no such function or predicate exists, and you write one of your own.

(a) In Python, write a function `length(lis)` that returns the length of its list parameter `lis`.

(b) In Scheme, write a function `(len lis)` that returns the length of its list parameter `lis`.

(c) In Prolog, write a predicate `len(Lis, N)` that succeeds if `N` is the length of its list parameter `Lis`.

7. (continued)

In a hypothetical language not too different from some actual languages, here is how you might write a length facility:

```
def length([]) = 0
def length([H | T]) = 1 + length(T)
```

(d) How does this language resemble Prolog?

(e) How does this language resemble Scheme?

(f) Would you classify this hypothetical language as a functional language or a logic programming language? Explain your choice in a sentence or two.

8. [10 marks = 5 + 5]

Several modern languages, including Java, C++ and Python, include the concept of an *iterator* — an object that embodies the idea of processing each element of a list, one at a time.

In Java, you might iterate over a list using an iterator like this:

```
for (Iterator it = list.iterators(); it.hasNext(); ) {
    item = it.next();
    ...
}
```

In C++, an iterator might be used like this:

```
for (iterator it = list.begin(); it != list.end(); it++) {
    item = *it;
    ...
}
```

In Python, iterators are usually called implicitly when you say something like

```
for item in list:
```

but they can also be used explicitly.

(a) In Scheme, what code that you might write, or operation or function that you might use interactively, has a role similar to that of an iterator? Answer in a sentence or two, perhaps illustrated with a code fragment.

(b) In Prolog, what code that you might write, or operation or query that you might use interactively, has a role similar to that of an iterator? Answer in a sentence or two, perhaps illustrated with a code fragment.