UNIVERSITY OF TORONTO
Faculty of Arts and Science

December Examinations 1999

CSC324: Principles of Programming Languages

Duration: 3 hours

Aids allowed: None.

- Make sure your examination booklet has 12 pages (not including this one).
- When asked to write code, comments are not necessary.
- All questions that ask you to circle the one best answer are worth 2 marks, and have a **one-mark penalty for wrong answers**.

Family Name: ______________________ Given Names: ______________________

Student #: ______________________

Tutor (circle one): Diana Ziau Michal Karczmarek Peeter Piegaze Anastasia Bezerianos

1. ________	/ 4	9. ________	/ 6
2. ________	/ 6	10. ________	/ 5
3. ________	/ 6	11. ________	/ 4
4. ________	/ 8	12. ________	/ 8
5. ________	/ 9	13. ________	/ 10
6. ________	/ 10	14. ________	/ 9
7. ________	/ 6	15. ________	/ 4
8. ________	/ 4	16. ________	/ 6
		Total ________	/ 105

Good Luck!

Question 1

[4 marks in total]

(a) [2 marks]
Why are java applets downloaded as bytecode rather than machine code? Circle the one best answer.

1. Because bytecode is faster to download than machine code.
2. Because bytecode is faster to execute than machine code.
3. Because machine code is machine-specific.
4. Because bytecode is machine-specific.
5. There is no reason; it is just the accepted standard.

(b) [2 marks]
Suppose you wrote two versions of the same program — one using only subprograms, and the other using only macros. Which one would run faster? Circle one.

subprograms only   macros only

For which one would the translated code require less space? Circle one.

subprograms only   macros only

Question 2

[6 marks in total]

Write three different Lisp expressions, each of which returns whether or not integer `a` is greater than integer `b`.

Expressions using `apply`:	`(apply '> (list a b))`
Expressions using `funcall`:	`(funcall '> a b)`
Expressions using `eval`:	``(eval `(> a b))``

Question 3

[6 marks in total]

Write a Lisp function called `atoms` that takes a list `L` and returns a list containing all the atoms at the top level of `L`, in order. For example,

```
(atoms ‘(F (A B) F G H () (C (D)) I))
```

should return

```
(F F G H NIL I).
```

ANSWER:

```
(defun atoms (L)

    (cond ( (null L) nil )

          ( (atom (car L)) (cons (car L) (atoms (cdr L))) )

          ( t (atoms (cdr L)) )
    )
)
```

Question 4

[8 marks in total]

(a) [5 marks]
Complete the following Lisp function. You may use the Lisp function `and`, but use it only with exactly two arguments.

```
; Returns a Lisp expression, ie some Lisp code,
; representing the conjunction of all ‘tests’ applied to ‘arg’.
; Precondition: ‘tests’ is a list of Lisp functions.  Each of them is a
;               function of one argument, that can be applied to ‘arg’.

(defun conjunction (tests arg)

     (cond ( (eq (length tests) 1)
                      (list (car tests) (list ‘quote arg)) )
            ( t (list ‘and
                      (list (car tests) (list ‘quote arg))
                      (conjunction (cdr tests) arg)
                 )
            )
     )
  )
```

(b) [3 marks]
Suppose you have a list called `myList`. Use `conjunction` to print “yes” if `myList` has at least 5 elements and the first one is a list.

```
(cond ( (eval (conjunction (list (lambda(x) (> (length x) 5))
                                 (lambda(x) (list (car x)))
                           )
                           myList
                )
         )

         ; code to print "yes" would go here.
       )
)
```

Question 5

[9 marks in total]

For parts (a) to (d), circle the one best answer.

(a) Suppose we wrote a program in which we declared all variables to be static. This is not as bad as making them global. Why not?

1. Global variables have unlimited scope, whereas static variables have limited scope.
2. Global variables have unlimited lifetime, whereas static variables have limited lifetime.
3. Global variables require time to be allocated during subprogram calls and deallocated during returns, whereas static variables do not.
4. Global variables cannot be reused during program execution, whereas static variables can.

(b) What advantage is there to making all variables static as opposed to dynamic?

1. Dynamic variables have unlimited scope, whereas static variables have limited scope.
2. Dynamic variables have unlimited lifetime, whereas static variables have limited lifetime.
3. Dynamic variables require time to be allocated during subprogram calls and deallocated during returns, whereas static variables do not.
4. Dynamic variables cannot be reused during program execution, whereas static variables can.

(c) What is the major disadvantage to making all variables static as opposed to dynamic?

1. Static variables have unlimited scope, whereas dynamic variables have limited scope.
2. Static variables have unlimited lifetime, whereas dynamic variables have limited lifetime.
3. Static variables require time to be allocated during subprogram calls and deallocated during returns, whereas dynamic variables do not.
4. Static variables cannot be reused during program execution, whereas dynamic variables can.

(d) What is the scope of a variable in a prolog rule?

1. Local to the rule, and all instances of the variable within the rule must unify.
2. Local to the rule, and the instances of the variable within the rule need not unify.
3. Global.
4. It depends on whether the variable begins with an upper-case or lower-case letter.

(e) [1 mark]
What is another name for static scope? `lexical scope`

Question 6

[10 marks]

The code on the right comes below the code on the left in a source file.

```
int w = 1;
int x = 2;
String y = "achoo";
String z = "ohm";

function p () {
   int w = 13;
   int x = -1;
   String z = "hello";
   print w, x, y, z;
   q();
}
function q () {
   int x = 99;
   String y = "goodbye";
   print w, x, y, z;
   r();
   s();
}
```

```
function r () {
   int x = 101;
   String z = "googoo";
   print w, x, y, z;
}
function s () {
   int x = -555;
   print w, x, y, z;
   t();
}
function t () {
   print w, x, y, z;
}
main() {
   p();
}
```

Fill in the table below to show what the output from this program would be if the language used dynamic scope.

Output from function p:	13	-1	achoo	hello
Output from function q:	13	99	goodbye	hello
Output from function r:	13	101	goodbye	googoo
Output from function s:	13	-555	goodbye	hello
Output from function t:	13	-555	goodbye	hello

Question 7

[6 marks in total]

Write a Prolog predicate `atoms(L, Ans)` that succeeds iff `L` is a list, and `Ans` is a list containing all the atoms at the top level of `L`, in order. For example,

```
atoms( [f, [a, b], f, g, h, [], [c, [d]], i], [f, f, g, h, [], i] ).
```

should succeed.

```
atoms([],[]).

atoms([H|R], [H|New]) :- atom(H), !, atoms(R,New).

atoms([_|R], New) :- atoms(R,New).
```

Question 8

[4 marks in total]

Write a predicate `compare( List, Comparison, Value )` that succeeds iff `List` is a list whose first element compares to `Value` as indicated by `Comparison`. For example

```
compare([12,1,49,5], >, 11).
```

should succeed because 12 is greater than 11.

```
compare( [First|Rest], Comparison, Value ) :-

        X =.. [Comparison, First, Value],

        call(X).
```

Question 9

[6 marks in total]

The `mystery` predicate below is intended to work with database tables represented as in Assignment 4. That is, by a list containing: first, a list of the column names for the table, then a sequence of lists that give values for each row in the table.

Yes, the predicate and variable names are horrific. :)

```
mystery( [A|B], [A|C] ) :- helper( B, C ).

helper( [X|Y], Q ) :- helper( Y, Z ), stick( Z, X, Q ).
helper( [X], [X] ).

stick( [], N, [N] ).
stick( [H|R], N, [H|O] ) :- stick( R, N, O ).
```

Complete the following sentences to explain what `stick` and `mystery` do:

`stick(X, Y, Z)` succeeds iff ...

```
X is a list and Z is the same list but with N appended to the end.
```

`mystery(X, Y)` succeeds iff X and Y are database tables and ...

```
Y is the same table as X, except that the rows are reversed
(the headline still stays in place, however).
```

You may want to do some rough work below this line.

Question 10

[5 marks in total]

The predicate `numOccurrences(Key, List, N)` should succeed iff `Key` occurs at the top level in `List` exactly N times. It doesn't work.

```
numOccurrences( _, [], 0 ).
numOccurrences( Key, [Key|Rest], N ) :- numOccurrences( Key, Rest, R ),
                                        N is R+1.
numOccurrences( Key, [Head|Rest], N ) :- numOccurrences( Key, Rest, N ).
```

(a) [3 marks]
Give an example query that will lead to the wrong answer on the first attempt to satisfy it (*i.e.*, before using ";"), and state the answer that would be given.

Query: `numOccurrences(x,[x,y,z],0).`

Answer: `0`

(b) [2 marks]
Fix the predicate. Make your changes directly on the code above.

```
% A cut is needed here to prevent the last rule from being applied
% (which would not count this occurrence of Key.
numOccurrences( Key, [Key|Rest], N ) :- !, numOccurrences( Key, Rest, R ),
                                        N is R+1.
```

Question 11

[4 marks in total]

The predicate `sumto(N, X)` should succeed iff `X` is an integer which is the sum of the first `N` positive integers. For example `sumto(3, 6)` should succeed because $1 + 2 + 3 = 6$. The predicate doesn't work.

```
% Precondition: N is instantiated to an integer >= 0.
sumto( 0, 0 ).
sumto( 1, 1 ).
sumto( N, A+N ) :- sumto( N-1, A ).
```

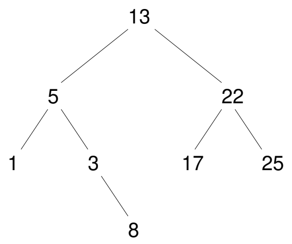
Fix the predicate so that it meets its specifications. Make your changes directly on the code above.

```
% This clause has to change as shown:
sumto( N, Ans ) :- Less is N-1, sumto( Less, A ), Ans is A+N.
```

Question 12

[8 marks in total]

Suppose we represent a tree as in assignment 4. For example, this tree:



is represented by `[13, [5, [1], [3, [], [8]]], [22, [17], [25]]]`.

Write a predicate `insert(key, T, newT)` that succeeds iff `newT` is the binary search tree one would have if one inserted `key` into `T`. You may assume that `T` is a binary search tree,

```
insert(Key, [], [Key]).
insert(Key, [Key|Rest], [Key|Rest]) :- !.

insert(Key, [Root], [Root|[[Key]]]) :- Key < Root, !.
insert(Key, [Root], [Root|[[], [Key]]]).

insert(Key, [Root|[L|R]], [Root|[NewL|R]]) :-
        Key < Root, !,
        insert(Key, L, NewL).

insert(Key, [Root|[L|[R]]], [Root|[L|NewR]]) :-
        insert(Key, R, NewR).
```

A couple of comments about this question:

- Getting the brackets exactly right is a pain; minor errors in bracketing were ignored when marking the question.
- The solution is awkward because of the particular tree representation, which we copied from assignment 4 because it was familiar to students. For example: right children who are nil are not represented in the tree. The last clause in my "solution" won't match a tree if the right child of its root is nil. To fix this requires adding more special cases. (The solution does work fine if the null right child is explicitly represented.)

Question 13

[10 marks in total; 2 marks each]

With an ambiguous grammar, how many parse trees are there for any given sentence that is in the language? Circle the one best answer:

0 exactly 1 more than 1 $\boxed{\text{1 or more}}$

With an ambiguous grammar, how many parse trees are there for any given sentence *not* in the language? Circle the one best answer:

$\boxed{0}$ exactly 1 more than 1 1 or more

With an *unambiguous* grammar, how many parse trees are there for any given sentence in the language? Circle the one best answer:

0 $\boxed{\text{exactly 1}}$ more than 1 1 or more

With an *unambiguous* grammar, how many parse trees are there for any given sentence *not* in the language? Circle the one best answer:

$\boxed{0}$ exactly 1 more than 1 1 or more

Suppose you have a grammar that gives two different derivations for the same sentence. Is that grammar ambiguous? Circle the one best answer:

Definitely yes Definitely no $\boxed{\text{There is not enough information here to tell}}$

Question 14

[9 marks in total]

For each of the following languages, write a regular expression for it, or write "cannot" if it cannot be described with a regular expression.

(a) All strings of 0s and 1s that have at least 3 leading 0s.

$$000(0+1)^*$$

(b) All strings of 0s and 1s that are palindromes, i.e., are the same whether read forward or backwards.

`Cannot`

(c) All strings made from the letters a, b, and c, where no a occurs after any b.

$$(a+c)^*b(b+c)^* + (a+c)^*$$

$(a+c)^*b^*(b+c)^*$ also works, as does $(a+c)^*(b+c)^*$.

Question 15

[4 marks in total]

Write a context-free grammar for the following language: All strings of 0s and 1s that are palindromes, i.e., are the same whether read forward or backwards.

```
<pal> ::= 0 <pal> 0 | 1 <pal> 1 | 0 | 1 | <empty>
```

Question 16

[6 marks in total]

Consider the following program:

```
subprogram swap (int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main() {
    int temp = 7;
    int other = 2;
    swap(temp, other);
    print temp, other;
}
```

Fill in the table below to show what the output from this program would be if the language used pass-by-value, if it used pass-by-reference, and if it used pass-by-name.

pass-by-value:	7	2
pass-by-reference:	2	7
pass-by-name:	2	7

END OF EXAM; Total pages = (12)