University of Toronto at Scarborough
cscC24—Principles of Programming Languages, Winter 2001

Midterm Test

March 3, 2001

Duration: 90 minutes

Aids allowed: None

Weight: 20% of your course grade

This exam contains a total of 9 pages (including this one). Write your answers clearly in the spaces provided. Use the back pages for your rough work.

Surname: _____

First name: _____

Student #: _____

Tutorial Sct.: _____

0: _____/ 2
1: _____/ 5
2: _____/ 4
3: _____/ 6
4: _____/ 6
5: _____/ 9
6: _____/22
7: _____/ 5
8: _____/ 6
9: _____/10

TOTAL: _____/75

Good Luck!

Question 0. [2 MARKS]

Write your name (or your initials if your name is long) and student number legibly at the top of every page of this test.

Question 1. List Manipulation [5 MARKS]

For each of the following expressions, circle "E" if it would cause an error in Scheme, or "N" if it would not. If it would not cause an error, show the value that Scheme would return.

Expression	Causes an error?	Value (if not an error)
`(list '(a b) '(c d))`	E N	
`(car '((() a b) c))`	E N	
`(cons 'a '(cdr (a b c)))`	E N	
`(append (car '(b c)) (cdr '(b d)))`	E N	
`(cdr '((a b c)))`	E N	

Question 2. Memory Representation of Lists [4 MARKS]

Draw the memory representation of the following list:

```
(a (b (c d) e) f)
```

Question 3. Function Calling Mechanisms in Scheme [6 MARKS]

For each of the expression below, write what it will output if it is correct, or briefly say why it is wrong.

`(let ((a 1)) (apply + '(a 3)))`	
`(let ((a 1)) (eval '(+ a 3) ()))`	
`(let ((a 1)) (+ '(a 3)))`	
`(let ((a 1)) (apply + a 3))`	
`(let ((a 1)) (eval '(+ (a 3)) ()))`	
`(let ((a 1)) (+ a 3))`	

Question 4. Writing Scheme functions [6 MARKS]

Write a function `(deep-count x lst)` which counts how many times `x` occurs in `lst`. For example,

```
(deep-count 'a '(a b c b a))          returns  2
(deep-count 'a '())                   returns  0
(deep-count 'a '(a b (a a) ((a)) b))  returns  4
```

Question 5. Writing HOFs [9 MARKS]

Part (a) [6 MARKS]

Write a higher-order function `(count lst test)` which takes a predicate (a function that returns `#t` or `#f`) `test` and a list `lst` and returns a number indicating how many elements of `lst` pass `test`. (An element `x` passes `test` if `(test x)` returns `#t`.)

Part (b) [3 MARKS]

Complete the following calls, using either a built-in function or a lambda expression, so that they return the values specified.

```
; Returns the length of lst
(count lst                                                    )


; Returns the number of sublists in lst
(count lst                                                    )
```

Question 6. Syntax [22 MARKS]

Part (a) [4 MARKS]

Describe in English the language that is generated by the following BNF grammar.

```
<statement>  -->  B <statement> | Z <plunger>
<plunger>    -->  Z <plunger> | F <pretzel>
<pretzel>    -->  S <pretzel> | S
```

Part (b) [4 MARKS]

Prove that this grammar is ambiguous:

```
<S>  -->  <S><S> | 0 | 1
```

Part (c) [4 MARKS]

Transform the grammar from question (b) into a grammar that is unambiguous but that accepts the same strings. Use BNF notation.

Question 6. Syntax (CONTINUED)

Part (d) [6 MARKS]

Consider the following BNF grammar:

```
<pop>  --> + <bop> , <pop> = | <bop>
<bop>  --> <boop> | (<pop>)
<boop> --> x | y | z
```

Which of the strings below is a `<pop>`? Circle yes or no as appropriate.
Do not guess; correct answers are worth 1 mark, and incorrect answers are worth -1 mark.

	Is it a `<pop>`?	
`z`	Yes	No
`(x)`	Yes	No
`+y=`	Yes	No
`(+y=)`	Yes	No
`+(x),y=`	Yes	No
`+(x),+y,x==`	Yes	No

Part (e) [4 MARKS]

Give an unambiguous context-free grammar in BNF that can generate the language

$$a^n b^m a^{n-m},\ n \geq m \geq 0.$$

Question 7. Type Checking [5 MARKS]

Assume the assignment statements below are syntactically correct in some imperative language. The necessary type definitions are given as "type-name = description" and variable declarations are given as "type: var-name". Full array, structure, and string assignments are allowed in this language.

```
type VOWELS = (a, e, i, o, u);
     WORDS = array[VOWELS] of string;
     LENGTH = 1..26;
     PHRASE = struct { 1..26: subject;
                       1..26: verb;
                       WORDS: rest; };

var  array[1..100] of WORDS: sentences;
     array[a..i] of string: letters;
     VOWELS: v;
     LENGTH: n;
     integer: i;
     PHRASE: p;
```

For each statement below, write C (for compile time) next to those that are statically type checkable, and R (for run time) next to those that are only able to be type checked dynamically.

1. `read(i);`
2. `letters[v] := "letter";`
3. `sentences[n][v] := "sent";`
4. `p.subject := i;`
5. `v = succ(v);`

Question 8. Type Equivalence [6 MARKS]

Assume some hypothetical imperative language uses the following type equivalence rules:

1. A type name is equivalent to itself.
2. Two types are equivalent if they are formed by applying the same type constructor to equivalent types.

Below are several type and variable declarations in this language:

```
type intarray = array[0..9] of integer;
     i = integer;

var  a: intarray;
     b: array[0..9] of integer;
     c: array[0..9] of integer;
     d: array[0..9] of i;
     e: integer;
     f: i;
```

Part (a) [3 MARKS]

Indicate which of these variables are equivalent under the type equivalence rules of the hypothetical imperative language mentioned above.

Part (b) [3 MARKS]

Indicate which of these variables are equivalent under structural equivalence.

Question 9. Short Questions [10 MARKS]

(a) [2 MARKS] What characterizes functional programming?

(b) [2 MARKS] Give two properties of good programming languages.

(c) [2 MARKS] What does a data type specify beyond the data representation?

(d) [2 MARKS] What is a type-safe program?

(e) [2 MARKS] What is a strongly typed programming language?

Total Marks = 75