

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)))	4
(let ((a 1)) (eval '(+ a 3) ()))	a not bound
(let ((a 1)) (+ '(a 3)))	wrong type of arg for +
(let ((a 1)) (apply + a 3))	too many args for apply
(let ((a 1)) (eval '(+ (a 3)) ()))	a unbound or wrong type of arg for +
(let ((a 1)) (+ a 3))	4

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
 Solution:

```
(define (deep-count x lst)
  (cond ((null? lst) 0)
        ((list? (car lst))
         (+ (deep-count x (car lst)) (deep-count x (cdr lst))))
        ((eq? x (car lst)) (+ 1 (deep-count x (cdr lst))))
        (else (deep-count x (cdr lst))))
  )
)
or, using map:
(define (deep-count x lst)
  (apply + (map (lambda (y) (cond ((list? y) (deep-count x y))
                                ((eq? x y) 1)
                                (else 0)))
             lst))
)
```

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.)

Solution:

```
(define (count lst test)
  (cond ((null? lst) 0)
        ((test (car lst)) (+ 1 (count (cdr lst) test)))
        (else (count (cdr lst) test))
  )
)
or:
(define (count lst test)
  (cond ((null? lst) 0)
        (else (+ (if (test (car lst)) 1 0)
                  (count (cdr lst) test)))
  )
)
(define (count lst test)
  (apply + (map (lambda (x) (if (test x) 1 0)) lst))
)
```

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 (lambda (x) #t))
 ; Returns the number of sublists in lst
 (count lst list?)

Question 6. Syntax [22 MARKS]
Part (a) [4 MARKS]
 Describe in English the language that is generated by the following BNF grammar.

`<statement>` \rightarrow B `<statement>` | Z `<plunger>`
`<plunger>` \rightarrow Z `<plunger>` | F `<pretzel>`
`<pretzel>` \rightarrow S `<pretzel>` | S

Solution: All strings containing 0 or more B's, followed by 1 or more Z's, then one F, and then one or more S's. Can also be stated with the regular expression $B^*Z^+FS^+$.

Part (b) [4 MARKS]
 Prove that this grammar is ambiguous:
 $\langle S \rangle \rightarrow \langle S \rangle \langle S \rangle \mid 0 \mid 1$

Solution: Show two parse trees for any string of length ≥ 3 , and conclude that the grammar is ambiguous.

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.

Solution:
 $\langle T \rangle \rightarrow \langle T \rangle \mid 0 \mid 1$

Question 6. Syntax (CONTINUED)
Part (d) [6 MARKS]
 Consider the following BNF grammar:

`<pop>` \rightarrow + `<bop>` , `<pop>` = | `<bop>`
`<bop>` \rightarrow <bop> | `<pop>`
`<bop>` \rightarrow 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	<input type="checkbox"/> Yes	No
(x)	<input type="checkbox"/> Yes	No
+y=	Yes	<input type="checkbox"/> No
(+y=)	Yes	<input type="checkbox"/> No
+(x),y=	<input type="checkbox"/> Yes	No
+(x),*y,x==	<input type="checkbox"/> Yes	No

Part (e) [4 MARKS]
 Give an unambiguous context-free grammar in BNF that can generate the language $a^n b^m q^{n-m}$, $n \geq m \geq 0$.

`<S>` \rightarrow a `<S>` a | `<P>`
`<P>` \rightarrow a `<P>` b | `<empty>`

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

- Solution:
Only the following variables have equivalent types:
- b, c.

Part (b) [3 MARKS]

Indicate which of these variables are equivalent under structural equivalence.

- Solution:
The following variables have structurally equivalent types:
- a, b, c, d;
 - e, f.

Solutions /

Name/Initials: _____

escC24 Midterm Test

March 3, 2000

Student No: _____

Question 9. Short Questions [10 MARKS]

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

Functional programming is characterized by recursively-defined functions. Each function, and the whole program, is a mathematical function of the inputs, and has no state or side effect.

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

Any two of simplicity, orthogonality, ease of learning, ease of reading/writing, etc.

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

The operations that can be performed on the data.

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

A program that executes without type errors on all possible inputs.

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

A language whose definition forces all type-checked programs to be type safe.

Total Marks = 75

Total Pages = 9

Page 9

END OF MIDTERM SOLUTIONS