UNIVERSITY OF TORONTO
St. George Campus

CSC 324

Principles of Programming Languages

Duration – 50 minutes

Examination Aids: No Exam Aids Allowed

Student #: _____

Last Name:_____ **First Name:**_____

Do **not** turn this page until you have received the signal to start. In the meantime, please fill out the identification section above, and read the instructions below carefully.

This term test consists of 3 questions on 8 pages (including this one), printed on both sides of the paper. When you receive the signal to start, please make sure that your copy of the examination is complete. Answer each question directly on the examination paper, in the space provided.

- Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0.0) so write legibly.
- Make sure to use meaningful variable/predicate/function/parameter names.
- Document your code to gain better partial marks if your answer is wrong. You will not lose marks for not documenting your code though.
- Do not use imperative features in Prolog or Scheme questions.

#1: Short Answers	**_____/15**
#2: Prolog	**_____/25**
#3: Scheme	**_____/20**
***Total*:**	**_____/60**

Question 1: Short Answers [*10 marks total*]

(a) [*2 marks*] DrRacket is an interactive program development environment for which programming language(s)? (*circle all the correct answers*).

1. Squash
2. Spitbol
3. Zizi-Zaza
4. Racket
5. Scheme

Answer:

Scheme, Racket ➔ **1 each**

(b) [*2 marks*] What is the Von Neumann bottleneck?

Answer:

All of the data, the names (locations) of the data, the operations to be performed on the data, must travel between memory and CPU a word at a time ➔ **2 marks if student mentions memory and CPU separation**

(c) [*2 marks*] What characterizes functional programming?

Answer:

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 effects. ➔ **Any 2 of the underlined deserves a mark**

(d) [*2 marks*] You are asked to write a program to drive a robot for delivering mail from the mail room to CS and ECE professors' office in Bahen. The robot is already equipped with optical character recognition system to read text and a map of the building. Which language would you use to develop such module and why?

Answer:

Prolog ➔ **1 mark**

Because the searching and backtracking mechanisms simplifies the coding. ➔ **each of underlined is 0.5 marks**

(e) [*2 marks*] If we were to load the following Prolog database:

```
mother(petra,katie).
mother(katie,diane).
mother(diane,william).
grandmother(katie,william) :- mother(katie,X), mother(X,william).
```

and then ask grandmother(petra,diane) Prolog would say no. Why?
(*circle the correct answer*).

1. No fact declares grandmother(petra,diane) to be true
2. ✓ The facts and rules don't allow grandmother(petra,diane) to be inferred
3. The grandmother rule cannot be properly expressed in first order logic.
4. The grandmother rule cannot be properly expressed in Prolog
5. petra should begin with an upper case letter

(f) [*5 marks*] For each of the following Scheme expressions, circle E if it would cause an error in Scheme or N if would not. If it would not cause an error, show the value Scheme would return

E N `(car '((() a b) c))`

Answer: `'` (() a b)

E N `(cons 'a '(cdr (a b c)))`

Answer: (a cdr (a b c))

E N `(append (car '(b c)) (cdr '(b d)))`

E N `(cdr '((a b c)))`

Answer: `'()`

E N `(let ((a 1)) (+ a 3))`

Answer: 4

Question 2: Prolog [25 *marks total*]

(a) [*10 marks*] Write a predicate **getslice(+List,+Index1,+Index2,-Slice),** where **List** is a list, **Index1** and **Index2** are integers between 1 and the length of **List**, and **Slice** is a sublist of **List**, which contains all the elements with indices between **Index1** and **Index2**. *Tip:* you might find the is operator and append relation useful depending on how you approach the solution. *Sample invocation:*

```
?-  getslice([a,b,c,d],2,3 ,Slice)
Slice = [b,c]
```

Answer:

```
get_slice(List,Index1,Index2,Slice) :-
        append(Piece1,Rest,List),
        append(Slice,_,Rest),
         length(Piece1,Len1), Len1 is Index1 - 1,
         length(Slice,LenS), LenS is Index2 - Index1 + 1.
```

Marking:

Decrement both indices, moving through list without adding to slice until Index1 reaches 1 (not 0) ➔ +2
Decrement Index2 adding to slice until Index2 reaches 0 ➔ +2
Base case for recursion ➔ +1
Put everything together correctly ➔ +5

Each logical error ➔ -1 or -2

(b) [*10 marks*] Write a predicate **compress(+L1,?L2)** where list **L2** is obtained from the list **L1** by compressing repeated occurrences of elements into a single copy of the element. *Sample invocation:*

```
?- compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
X = [a,b,c,a,d,e]
```

Answer:

```
compress([],[]).
compress([X],[X]).
compress([X,X|Xs],Zs) :- compress([X|Xs],Zs).
compress([X,Y|Ys],[X|Zs]) :- X \= Y, compress([Y|Ys],Zs).
```

Marking:

Correct set of predicates ➔ +2
Splitting list correctly ➔ +2
Base case for recursion ➔ +2
Put everything together correctly ➔ +4

Each logical error ➔ -1 or -2

(c) [5 *marks*] The following relation classifies numbers into three clauses: positive, zero and negative:

```
class(Number, positive):-Number > 0.
class(0,zero).
class(Number,negative):- Number < 0.
```

Re-define this procedure in a more efficient way using cuts. Specify whether your cut is green or red !

***Answer*:**

```
%-----------------valid---------------------------------
class(Number, positive) :- Number > 0,!.
class(0,zero) :- !.
class(Number,negative).
%-----------------valid---------------------------------
class(Number,positive) :- Number > 0.
class(0,zero).
class(Number,negative):- class(Number,positive),!,fail.
class(Number,negative):- class(Number,zero),!,fail.
class(Number,negative).
%-----------------output no for class(0,X)-------------
class(Number,negative):-Number<0,!.
class(0,zero):-!,fail.
class(Number,positive).
%-----------------output 2 answers for class(0,X)------
class(0,zero).
class(Number,positive):-Number>0,!.
class(Number,negative).
```

***Marking*:**

4 for code

-1 for each wrong output

1 for green/red

Question 3: Scheme Program [*20 marks total*]

(a) [*10 marks*] Write a function (**count lst test**) which takes a predicate **test** (a function that returns #t or #f) test and a list **lst** and returns a number representing how many elements of **lst** pass test. An element x passes **test** if (**test** x) returns #t. Assume that each of **lst** members is a valid input for the predicate **test**. *Sample invocation:*

```
]=> (count '(1 -1 3 -2 99 -4) positive?)
3
```

***Answer*:**

```
(define (count lst test)
        (cond ((null? lst) 0)
        (else (+ (if (test (car lst)) 1 0)
                        (count (cdr lst) test)))))
```

or

```
(define (count lst test)
                (apply + (map (lambda (x) (if (test x) 1 0)) lst)))
```

***Marking*:**

Base case ➔ +2
Recursive call ➔ +3
Calling (test (car lst)) ➔ +2
Combining all correctly ➔ +3

Logical errors ➔ -2

(b) [*10 marks*] Describe the purpose of the Scheme procedure **quest** provided below. What is the result of this procedure call? (quest '((a b) c (d e f))).

```
(define (quest lst)
        (cond
          ((null? lst)
                        '())
              ((and (eq? 1 (length lst)) (not (list? (car lst))))
                    (car lst))
              ((eq? 1 (length lst))
                        (quest (car lst)))
              (else
                        (quest (cdr lst))))
```

***Answer*:**

If we scan the list right-to-left and ignore all internal brackets, the first atom encountered is the return value. ➔ 6 marks

The call (quest '((a b) c (d e f))) returns the value f. ➔ 4 marks

END OF MIDTERM