Family name _____SOLUTION_____
Given name(s) _____
Student number _____

York University
Faculty Science & Engineering / Faculty of Arts
Department of Computer Science & Engineering

Class Test 1

AK/AS/SC – CSE 3401 3.0
Functional & Logic Programming
2008 June 11

Instructions

1. The test time is approximately 75 minutes.

2. This is a closed book examination. No examination aids are permitted.

3. All questions are to be attempted.

4. All questions are of equal value.

5. Using annotated diagrams, examples, complete sentences and paragraphs will increase the effectiveness of your answer.

6. All programming is to include comments. When using functions like apply, mapcar, append, etc. clearly indicate what is the effect you want using diagrams and/or symbolic notation.

7. If a question is ambiguous or unclear then please write your assumptions and proceed to answer the question.

Ques	Max	Mark
1	**8**	_____
2	**8**	_____
3	**8**	_____
4	**8**	_____
5	**8**	_____
Total	**40**	_____
Letter grade		_____

Question 1 (1 x 8 = 8 points)

Assume the following forms have been typed into the interpreter and evaluated in the given order.

```
( defun a ( y ) ( list 'list y ) )

( setq a 'b )

( setq b '9 )

( setq c 'd )

(setq d '11)

( set c '8 )

( setq f '( ( 111 ) 222 333 ) )

( defun b ( x ) ( maplist #'list x ) )

( setq e ( ( lambda ( x  y ) (+ x y) ) b b ) )
```

What will the following forms evaluate to? State **error** if the form does not evaluate correctly

1. (a a) _____(LIST B)_____

2. (funcall 'a (list c b d)) _____(LIST (D 9 8))_____

3. (apply 'a '(a b c)) _____**error**_____

4. (eval e) _____18_____

5. (eval (eval (eval '''a))) _____B_____

6. (eval (a a)) _____(9)_____

7. (b f) ((((111) 222 333)) ((222 333)) ((333)))

8. (mapcar 'b '((1 2 3) (2 3) (3)))

((((1 2 3)) ((2 3)) ((3))) (((2 3)) ((3))) (((3))))

Question 2 (5 + 3 = 8 points)

A Define a recursive Lisp function **longestRun** that takes a list of atoms and returns the length of the longest run of consecutive occurrences of the same atom in the list. For example,

```
(longestRun '( b a b a b b b a ) )  → 3
(longestRun '( a b a a c b ) )      → 2
(longestRun '(c) )                  → 1
(longestRun '( ) )                  → 0
```

You may use **ONLY** the Lisp functions defun, cond, car, cdr, cons, append, null, <, >, equal, max, +. You may use combinations of car and cdr such as caddr. You may define a helper function.

```
(defun longestRun (aList) ;; You provide the rest
```

ANSWER

```
; longest run takes one argument, a list, and returns the length of
; the longest run of consecutive occurrences of the same atom in the
; list
; precondition: list is a valid list

(defun longestRun (list)
      (cond ( ( null list ) 0 )
            ( t (longestHelp (car list ) (cdr list ) 1 0 ) )
      )
)

; longesthelp takes 4 arguments, the current element being looked at,
; the list, the current count for element elem and the maximum count so
; far
; helper function, all the work is done here

(defun longestHelp (elem list c1 c2)
      (cond
       ; empty list return either current max or max so far
       (( null list )(max c1 c2 ) )
       ; next element is same as elem
       ((equal elem (car list))
            (longestHelp elem (cdr list) (+ 1 c1) c2 ) )
       ; next element is diff than elem
       ( t ( longestHelp (car list) (cdr list) 1 (max c1 c2)))
      )
)
```

B Explain how you can create a static variable in LISP.

ANSWER

In Lisp static variables are associated with a function as a lambda-closure. To create a static variable, you need to create closure of a function by supplying the function `function' with a lambda expression as argument – if the free symbols of that lambda expression are bound to the values of an enclosing function's parameters, `function' will return a "snapshot" of the function denoted by the lambda expression, and we'll get a static variable associated with this free symbol.

e.g.

```
(defun egen (*seed*)
      (function
            (lambda() (setq *seed* (+ *seed* 2)))
      )
)
```

Here, *seed* will become a static variable after we create a closure of egen (see Lecture 4, slides 16-21 for more).

Question 3 (3 + 5 = 8 points)

A What is functional programming? What are the prime attributes of functional programs?

ANSWER

Functional programming consists of writing functions that have functions as input and frequently as output. That is writing functions that themselves create new functions. Use of generalized functions that abstract control flow patterns -- e.g. mapcar and reduce.

Functional programs have no explicit loops (recursion), have no sequencing at a low level, have no local variables. Frequently input is a single list of parameters.

B Define a functional program, with no explicit recursion, that produces the following output for a given integer **max**. Do not use lambda-functions.

```
(fun-list 0) → NIL
(fun-list 1) → (1)
(fun-list 2) → ((1 2) 2)
(fun-list 3) → (((1 2) 3) (2 3) 3)
(fun-list 5) → (((((1 2) 3) 4) 5) (((2 3) 4) 5) ((3 4) 5) (4 5) 5)
```

Hints: Consider the top level sub-lists of the output; note that they are nothing but the reduction (**reduce**) of some **range** using **list**.

```
(defun fun-list (max) ;; You supply the rest
```

ANSWER

```
(defun fun-list (max)
      (maplist (bu 'reduce 'list)(range 1 max)))
```

OR

```
(defun fun-list (max)
      (mapcar (bu 'reduce 'list) (genlist (range 1 max) 'cdr max))
)
```

Question 4 (3 + 3 + 2 = 8 points)

A What are macros? State the advantages of using macros?

ANSWER

Macros are Lisp functions that when invoked with appropriate parameters create as output Lisp program text. Macros are used to create custom and more understandable syntax. Macros are often used in place of functions to remove function call execution time overhead. Many apparent functions in Lisp are actually macros.

Advantages: 1) makes program more readable
2) insulates program from low level implementation decisions

B Write a macro definition
(select all elements from aList but the aPosition)
that produces a program, which when evaluated, returns **aList** after removing the element in position **aPosition**. Here, **aPosition** is one of the following.

aPosition: first, second, third

For all other values of **aPosition** return **nil**.

Example

```
( select all elements from '(1 2 3) but the first )   →    ( 2 3 )
( select all elements from '(1 2 3) but the third )   →    ( 1 2 )
```

Complete the macro definition, without using backquote.

```
(defmacro select ;; complete the parameters and body
```

ANSWER

```
(defmacro select (all elements from list but the pos)
      (cond ((equal pos 'first)(list 'cdr list))
            ((equal pos 'second)
             (list 'cons (list 'car list) (list 'cddr list)))
            ((equal pos 'third)
             (list 'cons (list 'car list)
                         (list 'cons (list 'second list)
                                     (list 'cdddr list))))
            (t 'nil)
      )
)
```

C Complete a macro definition **select**, as in Part B but this time use backquote.

```
(defmacro select ;; complete the parameters and body
```

ANSWER

```
(defmacro select (all elements from list but the pos)
      (cond ((equal pos 'first)`(cdr ,list))
            ((equal pos 'second)`(cons (car ,list) (cddr ,list)))
            ((equal pos 'third)`(cons (car ,list)
                                      (cons (second ,list) (cdddr ,list))))
            (t 'nil)
      )
)
```

Question 5 (8 points)

Trace and annotate (i.e. make notes to explain what you are doing) the evaluation of the following lambda expression.

```
{λ A,B . B[A+1] + B[A]}[ {λ D . {λ C . C * D}[1 + D]} [ 2 ]
                       , {λ C . {λ D . D * C}} [ 3 ] ]
```

ANSWER

Have two arguments
1 – { λ D . { λ C . C * D}[1 + D]} [2] – that is passed to A
2 – { λ C . { λ D . D * C}} [3] – that is passed to B
1.1 Evaluate expression (1) – substitute 2 for D (argument is already evaluated)
{ λ C . C * 2}[1 + 2]
1.1.1 Evaluate the argument [1 + 2] ==> 3
1.1.2 Substitute C = 3 ==> 3 *2 ==> 6
2.1 Evaluate expression (2) – substitute 3 for C (argument is already evaluated).
{ λ D . D * 3}
No further evaluation is possible

At the outer level we now have
{ λ A,B . B[A+1] + B[A]}[6 , { λ D . D * 3}]

Substitute the arguments for A and B
{ λ D . D * 3}[6+1] + { λ D . D * 3}[6]
Evaluate the left-hand and right-hand expression, then do addition
Argument first 6+1 ==> 7
Substitute D = 7 ==> 7*3 + 6*3 ==> 21+18 ==> 39.

For remarking you need to write a note stating clearly and exactly where you believe your grade should be increased or decreased. Note that the grade is a qualitative one. You need to explain why you believe the quality of your answer should, for example, if you think the grade should go up, be good (B) and not competent (C+), or, if you think the grade should go down, very good (B+) and not excellent (A).

The entire test will be reevaluated. Your grade may go up, it may stay the same, or it may go down. I will look over the entire test and see if the grades good, excellent, minimal, etc are applicable to the work as a whole independent of the points assigned to the parts.