Family name ___SOLUTION_____

Given name(s) _____

Student number _____

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

Class Test 2
AK/AS/SC/COSC3401.03
Functional & Logic Programming

2006 July 20

Instructions

1. The test time is approximately 120 minutes.
2. This is a closed book examination. No examination aids are permitted.
3. All questions are to be attempted.
4. All programming is to include comments.
5. If a question is ambiguous or unclear then please write your assumptions and proceed to answer the question.

Ques	Max	Mark
1	6	_____
2	6	_____
3	2	_____
4	6	_____
5	4	_____
6	4	_____
7	6	_____
8	7	_____
9	7	_____
Total		_____
Letter grade		_____

Question 1 [6 marks]

A. Consider the following Lisp program.

```
( defun g (X  Y)
          (cond  ((null Y)  nil )
                 ((null (cdr Y))  nil)
            ((eq  X  (car  Y) ) (cons  (cadr  Y)  (g  X  (cdr  Y))))
                 (t  (g  X  (cdr  Y)))
          )
)
```

What will be returned after executing the following expressions?

1. (g ‘d ‘(a d f d g))

   (F G)

2. (g ‘d ‘(q w e r))

   NIL

B. What will be returned when the following Lisp expression is evaluated?

(mapcar #‘(lambda (x) (cons (cons (car x) nil) (cdr x))) ‘((a b) (c d)))

(((A) B) ((C) D))

Question 2 [6 marks]

Define a Lisp **macro** `select` that takes 4 arguments: a list, a function, a predicate and a value. If executed `select` returns the sublist of items from the list such that if you apply the given function to an item in the list then the item is included in the sublist if when applying the given predicate to the result and the given value, the predicate is true.

Examples showing the result of executing the macro.

(select item from '(10 20 25 15 30 12 23 5) if 1+ (item) > 20) (20 25 30 23)
(select item from '(10 20 25 15 30 12 23 5) if 1+ (item) > 21) (25 30 23)
(select item from '(10 20 25 15 30 12 23 5) if 1- (item) < 20) (10 20 15 12 5)
(select item from '(10 20 25 15 30 12 23 5) if 1- (item) < 19) (10 15 12 5)

Use the back quote style to write your macro. Your answer should have no explicit recursion. You may use a lambda function but no helper function.

```
;; must include noise words in the argument list so the macro can
;; be called as per the examples given above

 (defmacro select (item from list if func (item) relation value)

      ;; need apply ‘append to get rid of nils
    `(apply 'append

        ;; go thru the list one element at a time
       (mapcar

          ;; use lambda function to check condition (don’t need

          ;; else since the default is nil
         (function (lambda (x)
                (cond ((,relation (,func x) ,value)

                       ;; use (list x) as result since we are
                       ;; using apply append trick
                      (list x))))) , list))
)
```

Question 3 [2 marks]

Consider the following Prolog program.

```
p([], Q, Q).
p([A|L], S, SL):-
        A > 5,
        !,
        p(L, [A|S], SL).
p([A|L], S, SL):- p(L, S, SL).
```

Show all the answers generated by Prolog for the following goal.

?- p([2, 5, 7, 9, 6], [], Result).

Result = [6, 9, 7];
No

Question 4 [6 marks]

Consider the following Prolog program.

```
r(e, L).
r(c(X, L1), c(X, L2)) :- r(L1, L2).
q(L1, L2) :- r(L1, L2).
q(L1, c(Y, L2)) :- q(L1, L2).
```

For each goal below, show the result of Prolog execution. In the case that the goal is proved, you should also show the bindings for the variables in the goal. You only need to show the first answer generated by Prolog.

?- q(c(d, e), c(a, c(b, W))).

W = c(d, _G249)
Yes

?- q(c(u, W), c(U, c(b, d))).

W = e
U = u
Yes

?- q(c(a, b), c(A, c(B, e))).

No

Question 5 (4 marks)

Although Prolog's name comes from Programming in Logic, there are several "non-logical" features built-in to Prolog. **Briefly**, describe three of these.

Negation by failure is different from logical negation. In the case of Prolog the answer no means not provable NOT false.

As shown in class \+ \+ X can produce different results from X.

All built-in predicates with side-effects are non-logical. For example, !, write(X), read(X), etc.

Question 6 [4 marks]

Define a Prolog predicate ordered(List) that asserts that List is an ordered list of numbers.

For example:

ordered([1, 5, 6, 6, 9, 12]).	Returns	yes
ordered([1, 6, 5, 6, 9, 12]).	Returns	no

```
% an empty list is ordered
ordered( [ ] ).

% a list of 1 element is ordered
ordered( [X] ).

% make sure first 2 elements are ordered and recursively check rest
ordered( [ X, Y | Rest ] ) :-
        X  =< Y,
        ordered( [ Y | Rest ] ).
```

Question 7 [6 marks]

Assuming that A is a list of integers, the predicate blah(A, B) is supposed to sum up all the positive values in A. It doesn't work.

```
blah( [ ], 0 ).

blah( [ X | Y ], XAns ) :- X > 0,  blah( Y, YAns ),  XAns  =  YAns + X.

blah( [ _ | Y ], Ans ) :- blah( Y, Ans ).
```

A What is the first result of the query blah([5, 1, -3, 8], Sum).

Sum = 0 + 8 + 1 + 5

B Fix the predicate by making changes directly on the code above. Do **not** rewrite the predicate.

Put ! after X > 0

Change = to is

C Write an appropriate header for your revised predicate (including preconditions).

```
blah(A, B) asserts that A is a list of integers and B is the sum
of all the positive integers in A.

Precondition:  A must be instantiated and all elements of A must
be instantiated
```

Question 8 [7 marks]

Define a Prolog predicate sort(X, Y) that asserts that X is a list of integers and Y is the same list, but sorted in ascending order. Your algorithm MUST be the following

Repeatedly choose the smallest remaining element from X and add it to Y.

Hint: use a helper predicate called smallest.

```
% recursive case: First is the smallest
smallest( [ First | Rest ], First, Rest ) :-
            smallest( Rest, Other, _ ), First < Other, !.

% recursive case: First is not the smallest
smallest( [ First | Rest ], Ans, [ First | Rem ] ) :-
            smallest( Rest, Ans, Rem ).

% base case: one element list
smallest( [ X ], X, [ ] ).

% base case: empty list
mysort( [ ], [ ] ).

% recursive case: find smallest, remove it from list, sort the
% rest
mysort( List, [X | Y] ) :- smallest( List, X, Rem ),
                           mysort( Rem, Y ).
```

NOTE: many students found the smallest element and then called sort again on the Tail of the list instead of removing the smallest element from the list. The **most** they were given is 4 out of 7.

Question 9 [7 marks]

A. Consider the following Prolog grammar.

```
a(0)  -->  [ x ].

a(X)  -->  b( _ ),  a(X).

a(X)  -->  b(G),  c(H),  a(X1),  { X is G  *  H  +  X1 }.

b(1) -->  [ r ].

b(2)  -->  [ w ].

c(X)  -->  d(X).

c(X)  -->  d(X1),  c(X2),  {X  is X1  + X2}.

d(2)   -->  [ y ].

d(-3)   -->  [ z ].
```

What do the following Prolog queries return?

```
?- a( X, [ r,  w,  y,  y,  x,  z,  r ], Y ).
```

X = 8
Y = [z, r]

```
?- a( X, [ w,  r,  y,  w,  z,  x ], Y ).
```

X = -4
Y = [ ]

B. Translate the **first 3** clauses in the DCG given in part A above to regular Prolog clauses.

```
a(0, [x | Rest], Rest).

a(X, List, Rest) :-
      b(_, List, List1),
      a(X, List1, Rest).

a(X, List, Rest):-
      b(G, List, List1),
      c(H, List1, List2),
      a(X1, List2, Rest),
      X is G*H+X1.
```