Question 1 (8 points)

```
(defun removeContig (theList)
      (cond ((atom theList) theList) ;; Null list is also an atom.
                  ;; Have a list with at least one item from here on.
                  ;; If only one item in the list then check within the
                  ;; item. Need the cons as we are removing from inside
                  ;; a list.
            ((null (cdr theList))
                  (cons (removeContig (car theList)) nil))
                  ;; At least two items are in the list. Check for
                  ;; contiguous identical items.
            ((equal (car theList)(cadr theList))
                  ;; First two items are identical. Forget the first
                  ;; one and continue processing the rest of the list.
            (removeContig (cdr theList)))
                  ;; First item is different from second item. Process
                  ;; the first item, then process the rest of the list.
            (t (cons (removeContig (car theList))
                  (removeContig (cdr theList))))
))
```

No need for support functions. The recursion is straightforward. Remove equal items before recursion -- no need to remove from within items that will be removed.

Items could be lists. (reduceContig '((a) (a) (b))) → ((a) (b))

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

A The problem is that arguments to functions are evaluated before the function body executes. In the first call the first argument increments a and 5 is passed as the value of thenPhrase and the second argument decrements a and 4 is the value of elsePhrase. The condition prints 5 which is expected but then a is still 4, due to the side effect, which is not what is intended. In the second call, the same values are passed and 4 is the result when a 3 is expected, and again the value of a remains as 4 and not 3 as is intended.

B The arguments should not be evaluated. The only way to do this is use a macro and delay execution of the arguments until the value is needed. In this case backquote can be used as there is no need to compute the text as had to be done in Question 2. Explanations are required to get more than a minimal grade for such questions.

```
(defmacro new-if (condition thenPhrase elsePhrase)
      `(cond (,condition ,thenPhrase)
            (t ,elsePhrase)))
```

C See http://www.cse.yorku.ca/course/3401/lecture2.pdf slides 19-20.

Question 3 (3 + 5 = 8 points)

A See http://www.cse.yorku.ca/course/3401/lecture6.pdf slides 15-18.

B

```
(mapcar '/ ;; Combine the lists

   ;; Create the numerator list
      (genlist n
      #'(lambda(prev_term) (* -1 prev_term x x))
      x)
   ;; Create the denominator list
      (mapcar (comp 'factorial #'(lambda(i) (1- (* 2 i))))
              (range 1 n))))
```

Alternate for denominator expression:

```
      (mapcar 'factorial (filter 'oddp (range 1 (* 2 n)))))
```

The parameter next in genlist is a function, hence need a lambda function.

Question 4 (1 + 4 + 3 = 8 points)

A To program is to create a database of facts and rules. Running a program is to make a query on the database.

B Data structures, also called compound terms, in Prolog consist of a name, which is called a functor, followed by a parenthesized list of components. Each component can be a constant, variable or, recursively, a compound term.

Functors, other than those that represent standard arithmetic and Boolean operators begin with lower case letters and contain letters, numbers and underscore.

Prolog has syntactic sugar that permits lists being written using square brackets, [], to designate a list and use comma to separate the items in the lists, for example, [a,b,c]. This is in place of using the functor dot with two components; for example .(a, .(b, .(c, []). [] is the empty list.

Constants are compound terms with no components, so the parentheses are not required. Variable names begin with either underscore, or an upper case alphabetic letter followed by underscores, alphabetic letters and digits. A variable name consisting of just an underscore is called the anonymous variable.

C A fact is a compound term followed by a period. It represents a relationship among its components called a predicate.

A rule is of the following form, including the ending period.

```
      head :- rule body .
```

The head is a predicate. The rule body is a logical combination of predicates separated with ',' to mean and, ';' to mean or, and

parenthesis to combine sub expressions. :- represents "if". Thus a rule is interpreted as "head is true if rule body is true".

Question 5 **(4 + 1 + 3 = 8 points)**

A

```
% empty list - return empty list
remove2nd( [ ], [ ] ).

% one element list - return same list
remove2nd( [ X ], [ X ] ).

% list has more than 1 element remove 2nd and return
remove2nd( [ X, _ | Z], [ X | Z ] ).
```

B The first rule requires that the arguments to a and b unify the second rule does not.

C This is problematic due to left-recursion. E.g.: consider the query person(P); Prolog will return P = eve for this query. If you initiate backtracking by typing a ;, Prolog will return P = habel. However, if you type ; again, the query will loop indefinitely.