Answer Question 1 and any TWO other questions.

You may find the following definitions useful throughout this paper:

```
fun const x y = x

fun lay ([] : string list)     = "" : string
|   lay (a::x)                 = a ^ "\n" ^ (lay x)

fun foldr f def [] = def
|   foldr f def (front :: rest) = f front (foldr f def rest)

fun map f []      = []
|   map f (x::xs) = (f x) :: (map f xs)

fun filter p []     = []
|   filter p (x::xs)
    = if (p x) then (x :: (filter p xs))
                else (filter p xs)

fun takewhile f []     = []
|   takewhile f (x::xs)
    = if (f x) then (x :: (takewhile f xs)) else []
```

1. (a) Write a function that will take a number and a list of names and will "shuffle" the names by interleaving them the given number of times.

   To interleave a list of names, the list should be cut into two halves and then put together by taking elements from each half alternately. For example:

   ```
   shuffle 1 ["Andy","Brijesh","Catherine","Doris","Erica","Fred"]
   ```

   shuffles the names once to give:

   ```
   ["Doris","Andy","Erica","Brijesh","Fred","Catherine"]
   ```

   [13]

   (b) Write an SML program (with types and comments) which takes a list of items and returns a list of lists which gives all the different permutations of the input list. For example, if the main function of your program is called "perms" then a sample interaction with the SML system would be as follows:

   ```
   -  perms [1, 2, 3];

   val it =
   [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]: int list list
   ```

   Note that the order in which the permutations are given is not important. [10]

[Question 1 cont. over page]
[TURN OVER]

[Question 1 cont.]

(c) Write a function which takes two numbers and generates a result using the following rules:

(i) Multiply the two numbers together

(ii) Then add all the digits of the result

(iii) If the sum of the digits has itself only one digit then return it as the result of the function, otherwise repeat from (ii)

For example, if your function is called "f", a sample interaction with the SML system would be:

```
- f 3 4;
val it = 3 : int
- f 7 7;
val it = 4 : int
- f 30 19;
val it = 3 : int
```

The last of the above examples is calculated as follows:

*30 * 19 is 570;*

5 + 7 + 0 is 12 (which has 2 digits);

1 + 2 is 3 (which has 1 digit);

the result is 3.

[9]

(d) Explain, with examples, what is meant by the following terms:

abstract type

partial function

datatype

[8]
[Total 40]

[CONTINUED]

2. Given the following extract from an SML program:

```
(* Quicksort program.
An unsorted list of integers is sorted by recursively
subdividing the list. *)

fun qsort ([] : int list) = ([] : int list)
|   qsort (front :: rest)
        = let
          fun lessthan x y = x > y
          val left  = filter (lessthan front)  rest
          val right = filter (not o (lessthan front)) rest
          in
          (qsort left) @ [front] @ (qsort right)
          end
```

(a) Give SML's response for each of the following expressions assuming it is typed at the SML prompt:

```
qsort [3,2,1];

(hd o qsort) [1,2,3];

qsort [qsort,qsort];
```

[6]

(b) Write a function, with types and comments, which implements *insertion sort* (where elements of an unsorted list are inserted, one at a time, into a sorted list). Your function should take an unsorted list of numbers and return a list of numbers sorted in ascending order. [8]

(c) Generalize `qsort` to create a new *curried* function `gsort`, which can sort a list of items of any type and where the ordering function is passed as an extra parameter. Give both the type declaration and function definition of `gsort`.

[8]

(d) Provide specialized versions of `gsort`:

(i) To sort a list of 2-tuples in descending order. For this function (2,3) is considered less than (1,7), because (2+3) is less than (1+7).

(ii) To sort into ascending order a list of items of type "dice" where "dice" is defined as:

```
datatype dice = One | Two | Three | Four | Five | Six
```

[8]
[Total 30]

[TURN OVER]

3. Compare and contrast the implementation of reference counting garbage collection with two other common garbage collection mechanisms. Your discussion should make special reference to the following points:

   — memory overheads

   — limited-width reference counts

   — full-width reference counts

   — Hughes's scheme for the collection of cycles [Total 30]

4. Write an SML program to implement the following simple game. Do **not** write a complex user interface. All that is required is a main function which can take as arguments the starting information and the list of moves; this function then evaluates the moves and returns either an error message (game over) or a printout of the game board.

   The game board is a 2-dimensional square array with 100 elements. The user is asked to give a starting position and then a succession of single-step movement commands (North, South, East and West). The user does not see the board during the game and must therefore navigate "blind". The starting position is set to be an asterisk character "*" and thereafter every position that is visited is also set to be an asterisk. The board is initialised so that all positions start as the underscore character "_".

   If the user moves off the board, the game is over. If the user revisits a position that has already been visited, the game is over. When the game is over, the values ("*" or "_") of all the positions of the board array are printed to the screen, so that the user can see which positions have been visited - the aim of the game is to try to get as many stars as possible on the board.

   Note that because there is a limit to the number of positions on the board then there is also a limit to the number of moves that the user can make - when the limit is reached, the game is over.

   You should pay attention to the data structures that will be required, with appropriate use of datatypes and/or abstract types. [Total 30]

[CONTINUED]

5. (a) Given the combinator definitions:

```
fun S f g x = f x (g x)

fun B f g x = f (g x)

fun W h x = h x x

val E = S (B W B) (B W B)
```

and given that the Y fixpoint combinator is defined axiomatically as

```
Y f = f (Y f)
```

show that E is equivalent to Y by illustrating that successive transformations of the graph representation of `(E f)`, for any `f`, produces a graph equivalent to `(f (E f))`. At each stage, indicate which of the nodes in the graph are the same as the previous stage, which have been updated, and which are new. [12]

(b) In fact the combinator E given above is not a valid SML definition. Explain why the definition of E gives rise to an error in SML. [6]

(c) What is the most general type of each of the following functions:

```
fun fun1 (a::b) f x = a ((x f b) o f)

fun fun2 []         j         acc = acc
|   fun2 f          []        acc = acc
|   fun2 (x :: xs) (y :: ys) acc = y   (fun2 xs ys (x y acc))


fun mysterious [] y z = z
|   mysterious (front :: rest) y z
    = (front z) andalso mysterious rest y ((y o front o y) z)
```

[12]

[Total 30]

[END OF PAPER]