1. Give IR translations for:

   (a) $[\text{break}]_{\text{L}}$

   (b) $[\text{x}]_{\text{Lt,Lf}}$ (where x is a variable)

   (c) $[\text{e}_1||\text{e}_2]_{\text{Lt,Lf}}$ (using short-circuit evaluation for e_1 and e_2)

   (d) Generate code for the repeat-until statement: "repeat S until e" executes S and tests e, and repeats until e becomes true. Thus, it is equivalent to "S; while !e do S".

2. Write APL expressions for the following calculations.

   (a) the average of the numbers from 1 to n

   (b) the sum of the squares of the elements of a vector V

   (c) the product of all positive elements of a vector V

   (d) a matrix with the numbers 1, 2, ..., n on the diagonal and 0 everywhere else. You may use the function idmat(x) to produce the identity matrix of size x.

3. (a) Name the two parts of a compiler's front end.

   (b) Name the two parts of a compiler's back end.

(c) What are the two outputs of the front end?

4. (a) Give two advantages of the copying garbage collection algorithm over the non-copying (mark-and-sweep) algorithm.

(b) Give two advantages of the non-copying (mark-and-sweep) garbage collection algorithm over the copying algorithm.

(c) Reference counting is not a popular algorithm. What is its major drawback?

5. (a) What is the type of the following function? fun f -> fun g -> fun x -> f (g x)

(b) Write an OCaml function that reverses a list, using fold_right instead of explicit recursion.

(c) Use map to write a function map_first f l which applies f to the first element of each item in l, assuming that l is a list of pairs.

(d) Write a function *curry* that converts a function f on pairs to curried form. In other words, if f is defined by let f (x,y) = e for some expression e, curry f should return the function g defined by let g x y = e.

(e) Using fold_right and no explicit recursion, define a function that concatenates the elements of a string list.

6. Recall that sets can be defined by `type 'a set = 'a -> bool`. For the following problems, you may use any previously defined functions on sets, and any library functions from the List library.

   (a) Write an OCaml function add_list such that add_list lst s returns a set that contains all the elements of s, and also all the elements in lst.

   (b) Write an OCaml function has_list such that has_list lst s returns true if every element of lst is in s, and false otherwise.

   (c) Write an OCaml function image such that image f lst returns the set of values produced by applying f to the elements of lst. You may use your solutions from the previous parts.

7. Write a function object for case_map (see the OCaml definition below). For the sake of simplicity, we assume that f : int -> bool, g,h : int -> int.

   let case_map f g h lis = map (fun x -> if (f x) then (g x) else (h x)) lis;;

   Your answer:

```
interface BoolFun{
  boolean apply(int n);
}
interface IntFun{
  int apply(int n);
}

class Map{
  static int[] map(IntFun f, int lis[]){
    int lis2[] = new int[lis.length];
    for(int i = 0; i < lis.length; i++)
      lis2[i] = f.apply(lis[i]);
    return lis2;
  }
}

class Case_Map{
  static int[] case_map(BoolFun f, IntFun g, IntFun h, int lis[]){
//complete this method









  }
}
```