1. Give IR translations for:

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

   JUMP L

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

   CJUMP x, Lt, Lf

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

   let L = getlabel() in
   $[\text{e}_1]_{\text{Lt,L}}$
   L: $[\text{e}_2]_{\text{Lt,Lf}}$

   (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".

   let L1, L2 = getlabel(), getlabel() in
   L1: $[\text{S}]_{\text{L2}}$
   $[\text{e}]_{\text{L2,L1}}$
   L2:

2. Write APL expressions for the following calculations. You may use either real APL syntax or OCaml APL syntax.

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

   $(+/(\iota n)) \div n$

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

   $+/(P * P)$

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

   $*/((P > 0)/P)$

   (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.

   $(\iota n) * \text{idmat}(n)$

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

   lexer, parser

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

   optimization, code generation

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

   AST, symbol table

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

   It does not need to traverse the entire heap, only the reachable objects. All free memory is gathered into one block.

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

It allows the entire memory to be used, instead of reserving half for the free area. Reachable data is never moved or copied.

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

It cannot handle circular heap structures.

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

   (β -> γ) -> (α -> β) -> α -> γ

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

   let rev l = fold_right (fun x -> fun y -> y @ [x]) l []

   (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.

   let map_first f l = map (fun x -> f (fst x)) l

   (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.

   let curry f = fun x -> fun y -> f (x,y)

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

   let concat_list l = fold_right (^) l ""

6. Recall that sets can be defined by `type 'a set = 'a -> bool`. For the following problems, you may use 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, plus all the elements in lst.

   let add_list lst s a = if List.mem a l then true else s a

   (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.

   let has_list lst s = List.for_all s l

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

   let image f lst = add_list (List.map f lst) emptyset

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$_m$ap 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
    IntFun fgh = new IntFun(){
      int apply (int n){
        return f.apply(n) ? g.apply(n) : h.apply(n);
      }
    }
    return Map.map(fgh, lis);
  }
}
```