```
3 + (5 - 4)
 (+ 3 (- 5 4))
 3 5 4 - +
                                   function
 int sqr(int n) {
   return n * n;
                            int
 }
                                    sqr
                                             arg
                                                              body
 (int sqr (int n)
   (return (* n n)))
                                                           return
                                          int
                                                   n
                                                                        n
                                                                                n
type 'a lst<sub>₹</sub>
                                             type 'a tree =
  | Empty
                                               Leaf
  | Link of 'a * 'a lst
                                               | Node of 'a * 'a tree * 'a tree
                                             let rec tf (t : 'a lst) ... : ... =
let rec lf (1 : 'a lst) ... : ... =
                                               match t with
  match 1 with
    | Empty \ -> \dots \ base \ case \dots
                                                 | Leaf \-> ... base case ...
                                                 | Node(val, left, right) ->
    | Link(fst, rest) ->
                                                   ... fst\...
      ... fst\ ...
                                                   ... (tf\ left ...) ...
      ... (lf rest ...) ...
                                                   ... (tf 'right ...) ...
type expr←
                                      let rec compile (e : expr) : string list =
  | ENum of int
                                         match e with
  | EOp1 of string * expr
                                           | ENum(n) ->
                                             ["mov eax, " ^ (string_of_int n)]
                                           | EOp1(op, arg) ->
let rec ef (e : expr) ... : ... =
  match e with
                                             let argis = compile arg in
    | ENum(n) -> ... base case ...
                                             if op == "add1" then
                                               argis @ ["add eax, 1"]
    | EOp1(op, arg) ->
                                             else if op == "sub1" then
      ... op \...
      ... (ef arg ...) ...
                                               argis @ ["sub eax, 1"]
```

```
type 'a lst<mark></mark>=
                                            type 'a tree =
  Empty
                                              Leaf
  | Link of 'a * 'a lst
                                              Node of 'a * 'a tree * 'a tree
                                            let rec tf (t : 'a lst) ... : ... =
let rec lf (1 : 'a lst) ... : ... =
  match 1 with
                                              match t with
                                                | Leaf -> ... base case ...
    | Empty\-> ... base case ...
                                                | Node(val, left, right) ->
    | Link(fst, rest) ->
      ... fst ...
                                                  ... fst\...
                                                  ... (tf left ...) ...
      ... (lf rest ...) ...
                                                  ... (tf \right ...) ...
type expr←
  | ENum of int
  | EOp1 of string * expr
let rec ef (e : expr) ... : ... =
 match e with
    | ENum(n) -> ... base case ...
    | E0p1(op, arg) ->
      ... op \...
      ... (ef arg ...) ...
                                  +
3 + (5 - 4)
                              3
(+ 3 (- 5 4))
354 - +
                                  5
                                  function
int sqr(int n) {
  return n * n;
                          int
}
                                  sqr
                                                            body
                                            arg
(int sqr (int n)
  (return (* n n)))
                                                                          *
                                                         return
                                        int
                                                 n
                                                                     n
                                                                              n
```