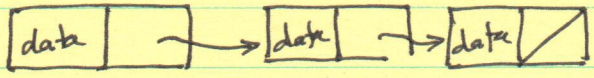
How to lists as records & pointers

pointers = refs



How do we express something that may be a pointer or may be nil?

Recall option types

A → type

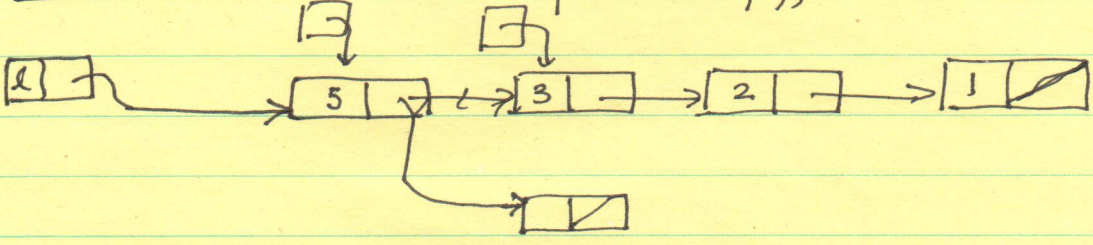
A option → type containing two sorts of values

a : A

Some a : A option

None : A option

```
type cell = { data : int ; next : +list }
and +list = cell option ref;;
```



```
let reverse (lst : +list) =
    let rec helper ((l: +list), (acc : +list)) =
        match !l with
          | None -> acc
          | Some c when !(c.next) = None ->
                (c.next := !acc; acc := Some c; acc)
          | Some c ->
(l := !(c.next); c.next := !acc; acc := Some c;
    helper (l, acc))
  in
    helper (lst, ref None)
```

—— x ——

Exceptions → Modern form of Go to

$$a x^2 + b x + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{(Bhramagupta)}$$

```
let solve (a, b, c) =
    let disc = (b *. b -. 4.*. a *. c) in
    in  disc < 0.0 || a = 0.0 then
        failwith " - - - - "
    else - - - - - -

exception Complex
exception Linear
```

```
let solve_quad (a, b, c) =
    let disc = - - -  in
      if a=0.0 then raise Linear
      else if disc < 0.0 then raise Complex
      else - ——— - -
```

exception handling

```
let solve_catch (a, b, c) =
    try
        solve_text (a, b, c)
    with
      | Linear -> "Degenerate linear equation"
      | Complex -> "The roots are complex"
```

```
let solve_robust (a, b, c) =
    try
      let disc = - - -  in
      if a = 0.0 then raise Linear
      else if disc < 0.0 then raise Complex
      else (- - . . )
    with
      | Linear -> (print_string "This is not a quadratic
                                 you idiot!");
                  (-.c/.b, -.c/.b)
      | Complex -> (print_string "The roots are complex.
                    The real & imaginary parts are:"
```

[1; 3; 2; 5] → [1; 4; 6; 11]

4

```
let psum lst =
  let rec helper l a =
    match l with
    | [ ] → [a]
    | x::xs → a::(helper xs (x+a))
  in
  match lst with
  | [] → [0]
  | x::xs → helper xs x;;
```

[[1;3;2]; [5;1;3]; [2;0]] ↦ [1;3;2;5;1;3;2;0]

```
let smash ll = List.fold_left (@) [] ll
```

```
let rec inter item lst =
    match lst with
  | [ ] → [[item]]
  | x::xs → (item::lst)::(List.map (fun u→ x::u)
            (inter item xs));;
```

o [1;2;3] ↦ [[0;1;2;3]; [1;0;2;3]; [1;2;0;3]; [1;2;3;0]]

```
let rec perms l =
    match l with
  | [ ] → [[ ]]
  | x::xs →
  smash (List.map (fun u→ inter x u) (perms xs))
```