Daniel Leeds, 15-212 R10, October 31, 2007

Modules

A **signature** describes an interface

```
signature PB =
sig
  type Phonebook
  val empty : Phonebook
  val get : Phonebook -> string -> int option
  val put : ((string * int) * Phonebook) -> Phonebook
  val delete : Phonebook -> string -> Phonebook
end
```

A **structure** contains an implementation

```
structure PhonebookList : PB =
struct
  type Phonebook = (string * int) list
  val empty = nil
  fun get nil s = NONE
    | get ((a,b)::xs) s = if a = s then SOME(b) else get xs s
  fun put (entry,P) = entry::P
  fun delete nil s = nil
    | delete ((a,b)::xs) s = if a = s then (a,b)::(delete xs s) else delete xs s
end
```

```
PhonebookList.empty
open PhonebookList
empty
```

Imperative Programming – coding with side effects

Assignment: val x = ref 2 : int ref
Re-assignment: x := 5
De-referencing: !x
Evaluating multiple expressions:
(y := 12; x := !y+4; y := 9)
Loops: while E do (E1; E2; …; EN)

```
fun impFact n =
   let val resultp = ref 1
       and ip = ref 0
   in while !ip < n do (ip := !ip + 1;
                              resultp := !resultp * !ip) ;
      !resultp
   end

fun irev l =
   let val resultp = ref []
       and lp = ref l
   in while not (null (!lp)) do
          (resultp := hd(!lp) :: !resultp
           lp := tl(!lp));
      !resultp
end



signature COMPLEX =
sig
  type t
  val empty : t
  val complement : t -> t
  val sum : t * t -> t
  val prod : t * t -> t
  val diff : t * t -> t
end

structure Comp : COMPLEX =
struct
  type t = real*real
  val empty = (0.0,0.0)
  fun complement (x,y) = (x,~1.0*y)
  fun sum ((a,b),(c,d)) = (a+c,b+d) : (real*real)
  fun prod ((a,b),(c,d)) = (a*c-b*d,a*d+b*c) : (real*real)
  fun diff ((a,b),(c,d)) = (a-c,b-d) : (real*real)
end
```