LET AND LOCAL BLOCKS
(local declarations)

```
let
  ⟨ local declarations ⟩
in
  ⟨ expression ⟩
end
```

(this is an expression)

```
local
  ⟨ local declarations ⟩
in
  ⟨ declarations ⟩
end
```

(this is a list of declarations)

(local declarations)

Example

```
val x = 4;
let val x = 2
in
   x + 4
end
```

Example

```
val x = 4;
let val x = 1
in
   (let val x = 2 in x-1 end)  +  (let val y = x+2 in y+2 end)
end
```

(local declarations)

Example

```
let fun f x = x+1
    val y = 4
in
    f y
end + 1
```

Example

```
fun f y =
    let
        val x = y - 1
    in
        fn z => x + y + z
    end
val g = f 3;
g 1;
```

(local declarations)

Example

```
val x = 4
fun f y =
    let
        val x = y - 1
    in
        fn z => x + y + z
    end
val y = 2
val g = f 3
val z = g 1
```

(local declarations)

```
fun fact n =
    let fun facth i r =
        if i>n then r  else facth (i+1) (i*r)
    in
        facth 1 1
    end

local
    fun facth 0 r = r
      |  facth n r = facth (n-1) (r*n)
in
    fun fact n = facth n 1
end;

- fact 10;
val it = 3628800 : int
- facth 10;
stdIn:1.1-1.6 Error: unbound variable or constructor: facth
```

```
- fun find (x,  []) = false
= | find (x, (y::l)) = x=y orelse find (x, l);
val find = fn : ''a * ''a list -> bool
- find (3, [1,2,3,4]);
val it = true : bool
- find (5, [1,2,3,4]);
val it = false : bool

- fun find (x,l) =
=   let fun f [] = false
=         | f (y::l) = x=y orelse f l
=   in f l
= end;
val find = fn : ''a * ''a list -> bool

- List.exists;
val it = fn : ('a -> bool) -> 'a list -> bool
- fun find (x,l) = List.exists (fn y => x=y) l;
val find = fn : ''a * ''a list -> bool
```

```
- fun checkSorted cmp l =
=   let fun f (x::y::l) = cmp (x,y) andalso f (y::l)
=         | f _ = true
=   in f l
= end;
val checkSorted = fn : ('a * 'a -> bool) -> 'a list -> bool

- checkSorted (op <) [1,2,3,3,4,5,5];
val it = false : bool
- checkSorted (op <=) [1,2,3,3,4,5,5];
val it = true : bool

- fun checkSorted cmp =
=   let fun f (x :: (l as y :: _)) = cmp (x,y) andalso f l
=       | f _                      = true
=   in f
= end;
```

```
- fun f x = ();
val f = fn : 'a -> unit
- f 2;
val it = () : unit
- fun f () = 3;
val f = fn : unit -> int
- f ();
val it = 3 : int
- print;
val it = fn : string -> unit
- print "foo\n";
foo
val it = () : unit
- fun printList [] = ()
= | printList (x::t) =
= ( print (x^"\n");
=   printList t
= );
val printList = fn : string list -> unit
- printList ["foo", "bar"];
foo
bar
val it = () : unit
- fun printList [] = ()
= | printList (x::t) =
= let val s = x^"\n" in
=   print s;
=   printList t
= end;
val printList = fn : string list -> unit
```

```
- map;
val it = fn : ('a -> 'b) -> 'a list -> 'b list
- fun incr x = x+1;
val incr = fn : int -> int
- map incr [1,2,3];
val it = [2,3,4] : int list
- map incr;
val it = fn : int list -> int list
- map (fn x => 2*x) [1,2,3];
val it = [2,4,6] : int list
- fun map _ [] = []
= | map f (x::t) = f x :: map f t;
val map = fn : ('a -> 'b) -> 'a list -> 'b list
- fun map f l =
= let fun mapf [] = []
=       | mapf (x::t) = f x :: mapf t
= in
= mapf l
= end;
val map = fn : ('a -> 'b) -> 'a list -> 'b list
- fun map f =
= let fun mapf [] = []
=       | mapf (x::t) = f x :: mapf t
= in
= mapf
= end;
val map = fn : ('a -> 'b) -> 'a list -> 'b list
- map incr [3,4,5];
val it = [4,5,6] : int list
- map length [[1,2], [1,2,3], [], [1]];
val it = [2,3,0,1] : int list
```

```
- app;
val it = fn : ('a -> unit) -> 'a list -> unit
- app (fn s => print (s^"\n")) ["foo", "bar"];
foo
bar
val it = () : unit
- foldl;
val it = fn : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
- fun f(x,y) = "f(" ^ x ^ "," ^ y ^ ")";
val f = fn : string * string -> string
- f ("x","y");
val it = "f(x,y)" : string
- foldl f "x" ["a", "b", "c"];
val it = "f(c,f(b,f(a,x)))" : string
- foldr f "x" ["a", "b", "c"];
val it = "f(a,f(b,f(c,x)))" : string
- foldl (op +) 0 [1,2,3,4,5];
val it = 15 : int
- foldl (op * ) 1 [1,2,3,4,5];
val it = 120 : int

- fun fold (f, acc, xs) =
=    case xs of
=   [] => acc
=    | x::xs => fold(f, f(x,acc), xs);
val fold = fn : ('a * 'b -> 'b) * 'b * 'a list -> 'b
-  fold (fn (x,y) => "f(" ^ x ^ "," ^ y ^ ")", "x", ["a","b","c"]);
val it = "f(c,f(b,f(a,x)))" : string
```

```
val len = fn : 'a list -> int
- fun rev l = foldl (op ::) [] l;
val rev = fn : 'a list -> 'a list
- rev [1,2,3];
val it = [3,2,1] : int list


- foldl (fn (_,x) => x+1) 0 ["foo", "bar", "foobar"];
val it = 3 : int
- fun len l = foldl (fn (_,x) => x+1) 0 l;
val len = fn : 'a list -> int

-fun len (l, lo, hi) =  foldl (fn (x,acc) => acc + (if x >= lo andalso x <= hi then 1 else 0))
                              0 l;
val len = fn : int list * int * int -> int
- len ([1,2,3,4,5,6], 2, 4);
val it = 3 : int


fun out [] = 0 | out (h :: t) = let
                                    fun check (x, (y,acc)) = (x, if x < y then acc+1 else acc)
                                in
                                    #2 (foldl check (h,0) t)
                                end
```

```
- List.tabulate;
val it = fn : int * (int -> 'a) -> 'a list
- List.tabulate (10, Int.toString);
val it = ["0","1","2","3","4","5","6","7","8","9"] : string list
- List.tabulate (4,fn x => x*x);
val it = [0,1,4,9] : int list
- List.tabulate (20, fn x => x mod 3);
val it = [0,1,2,0,1,2,0,1,2,0,1,2,...] : int list
- List.tabulate (5, fn x => "f("^(Int.toString x)^")");
val it = ["f(0)","f(1)","f(2)","f(3)","f(4)"] : string list
- List.find even [1,2,3];
val it = SOME 2 : int option
- List.exists even [1,2,3];
val it = true : bool
- List.all (not o even) [1,3,5,7];
val it = true : bool
- List.filter (not o even) [1,2,3];
val it = [1,3] : int list
```

Note on foldl and foldr: ('a * 'b -> 'b) -> 'b -> 'a list -> 'b

```
List.foldl operator initValue list
```

performs the computation of method compute below:

```
abstract class Fold<A,B> {

  public B compute (List<A> list, B initValue) {
    B value = initValue;
    for (A current : list)
      value = operator(current, value);
    return value;
  }

  public abstract B operator (A left, B right);
}
```