Lecture 5:

- Let
- Booleans
- Induction

Declarations

```
val x:int = 5
val y:int = x+2

fun f(x) = 2*x+3
```

Expressions

~~(val x:int = 5) + 2~~

let

```
let
    val x:int = 5
    val y = x + 1
in
    x + y
end
```

these variables exist in this expression

(local variables)

```
↦ let val x = 5
      val y = 5 + 1
  in
     5 + y
  end

↦ let val x = 5
      val y = 6
  in
     5 + y
  end

↦ 5 + 6 ↦ 11
```

```
val z = let val x = 5
            val y = 6
        in
            x + y
        end

val w = z + x    error: no x in scope
```

"nearest enclosing binder"

let val $x : A = e_1 : A$
in
$e_2 : T$ Stand for expressions
end

$: T$

---

let val $x = 5$
in
(let val $y = 7$
in
$x + y$
end) $+ x$
end

To run let val x = e_1 in e_2 end

1. run e_1 to produce a value

2. substitute that value in for x in e_2

(let val x = 5 in x+x end) $\mapsto$ 5+5

(let val x = 2+3 in x+x end) $\mapsto$ let val x = 5 in x+x end $\mapsto$

```
fun f(x) =
  let val y = x + y
  in
     - - - - - -
  end
```

"multiple Statements"

```
let val x : int = 4
in
  Int.toString(x) : string
end                        : String
```

$\longmapsto$ "4"

```
let val x = 4
in
    let val y = x+7
    in
        x+y
    end
end
```

Same as

```
let
  val x = 4
  val y = x+7
in
  x+y
end
```

```
val x = 4
val y = let val z = x+1 in .... end
```

```
fun f((x,y) : int * int) : int =
    2 * x + y
```

f(3,2) $\mapsto$ 2*3+2

---

same as

```
fun f(p : int * int) : int =
    let val (x,y) = p
    in
        (2 * x) + y
    end
```

pair pattern in val

f(3,2)
$\mapsto$ let val (x,y) = (2,3)
in 2*x+3 end

SML is "garbage-collected"

let val x=5 in x+x end

memory for x gets freed

$\mapsto$ 5+5

Representing a time

12-hour clock

3:16 pm

```
type time12 = int * int * string
(* (h, m, p) where
   1 ≤ h ≤ 12
   0 ≤ m < 60
   p = "am" or p = "pm" *)
```

24-hour clock

15:16

```
type time24 =
  int * int
(* 0 ≤ h < 24
   0 ≤ m < 60
*)
```

```
(* Purpose: write a function to
     convert a 12-hour time to
     a 24-hour time

   E.g.   to24(3, 20, "pm") = (15, 20)
          to24(3, 20, "am") = (3, 20)
          to24(12, 0, "pm") = (12, 0)
          to24(12, 0, "am") = (0, 0)
*)
```

```
fun to24( (h,m,p) :time12): time24 =
(case p of
   "am" => (case h of
              12 => (0, m)
            | any => (h, m))

 | "pm" => (case h of
              12 => ( 12 , m )
            | any => ( h+12 m )))
```

parens around

let val x = 5

in

blah

end

$\boxed{\text{Booleans}}$

a boolean is either

- true, or
- false

⟶ and that's it!

Operation: case analysis

case $b^{:\,\text{bool}}$ of

true $\Rightarrow e_1$

| false $\Rightarrow e_2$

---

case true of

true $\Rightarrow 5$ $\quad\longmapsto 5$

| false $\Rightarrow 6$

---

case false of

true $\Rightarrow 5$ $\quad\longmapsto 6$

| false $\Rightarrow 6$

$x = y$ $\quad x, y : int$

$x < y$ $\quad x, y : int$

$x > y$ $\quad x, y : int$

b_1 orelse b_2 :=
case b_1 of
true => true
| false => b_2

&&

$(b_1^{:bool}\ \underline{\text{andalso}}\ b_2^{:bool}) : bool :=$

case b_1 of
false => false
| true => b_2

opening helper

```
(* Purpose: check if one 12-hour time
            is earlier than another *)

fun earlier(t1: time12, t2: time12): bool =
  let val (h1, m1) = to24(t1)
      val (h2, m2) = to24(t2)
  in
     (h1 < h2) orelse (h1 = h2
                        andalso
                       m1 < m2)
  end
```

"using to24 as a helper function"

<= (beside h1 < h2)

<= (beside m1 < m2)

(* Example: 12:00 am is earlier than 12:00 pm

7:00 am is earlier than 6:00 pm *)

7:15 am earlier than 7:16 am

```
earlier((7,0,"am"), (6,0,"pm"))

↦ let val (h1,m1) = to24(7,0,"am")
      val (h2,m2) = to24(6,0,"pm")
  in
     ----
  end

↦ let val (h1,m1) = (7,0)
      val (h2,m2) = (18,0)
  in
     h1<h2 orelse ----
  end

↦ 7 < 18 orelse ....   ↦ true orelse ...
                        ↦ true
```

```
Val x = 5
```

```
case b of
  true => . . . . .
```

~ 17

$5 - 17$

Laughs

$$\text{laughs}(n) = \text{"}\underbrace{\text{hahahaha}\ldots\text{ha}}_{n \text{ letters}}\text{"}$$

$$\text{laughs}(n) = \text{"ahaha" if } n \text{ is odd}$$

```
fun laughs(n: int): string =
  case n of
     0 => ""
   | 1 => "a"
   | any => laughs(n-2) ^ "ha"
```

$\text{laughs}(5)$

$\mapsto \text{laughs}(3) \mathbin{\char`\^} \text{"ha"}$

$\mapsto (\text{laughs}(1) \mathbin{\char`\^} \text{ha}) \mathbin{\char`\^} \text{"ha"}$

$\mapsto (\text{"a"} \mathbin{\char`\^} \text{"ha"}) \mathbin{\char`\^} \text{"ha"}$

$\mapsto \text{"ahaha"}$

```
fun altlaughs(n:int):string =
    case n of
        0 => ""
      | any =>
            (case evenP(n) of
                true => "h" ^ altlaughs(n-1)
              | false => "a" ^ altlaughs(n-1))
```

altlaughs(4) = "haha"

5 = "ahaha"

altlaughs(5)

$\mapsto$ case evenP(5) of false => "a" ^ altlaughs(5-1)

$\mapsto$ "a" ^ altlaughs(4)

$\mapsto$ "a" ^ case evenP(4) of true => "h" ^ altlaughs(3)

$\mapsto$ "a" ^ "h" ^ altlaughs(3)

$\mapsto$ "ah" ^ altlaughs(3)

$\mapsto$ . . . . —

$\boxed{\text{Theorem}}$ For all nats n,

$$\text{laughs}(n) = \text{altlaughs}(n)$$

Helper Theorem:

If n is odd,

then $\overbrace{\text{laughs}(n)\ \hat{}\ \text{"ha"}}^{\text{laughs}} = \overbrace{\text{"ah"}\ \hat{}\ \text{laughs}(n)}^{\text{altlaughs}}$

$\underbrace{\text{"ahaha ....ha"}}_{n \text{ letters}}\ \hat{}\ \text{"ha"}$ $\qquad$ $\text{"ah"}\ \hat{}\ \text{"ahaha....ha"}$

Proof case for 1:

To show laughs(1) ^ ha = ah ^ laughs(1)

But laughs(1) = "a"

Both = "aha" ✓

Case for k+2, where k is odd

Inductive hypothesis: laughs(k) ^ ha = ah ^ laughs(k)

To show laughs(k+2) ^ ha = ah ^ laughs(k+2)

$$\boxed{\text{laughs}(k+2)\text{ ^ ha}}$$

$\mapsto$ (laughs(k) ^ ha) ^ ha

= (ah ^ laughs(k)) ^ ha   by inductive hypothesis

= ah ^ (laughs(k) ^ ha)   associativity

$\Leftarrow\!\!\dashv$ $\boxed{\text{ah ^ (laughs(k+2))}}$

$$\boxed{\text{using IH}}$$ laughs(k) ^ ha

= ah ^ laughs(k)