Lecture 1
The ML programming language

- Two widely use descendents of the original ML
  - Standard ML and Caml

- Standard ML used in this course
  - Available on Thor

- Caml is a lightweight language
  - From INRIA in France
  - Better suited than Standard ML for small machines
  - Still evolving
  - Public domain, runs on puny PCs, Macs etc
  - Pretty similar to Standard ML
Interacting with ML

- ML is an interactive language
- A common way to run it is inside a shell window from emacs
- The main things one does in ML are:
  - evaluate expressions
  - perform declarations
Expressions

- The ML prompt is “–”
  - As ML reads a phrase it prompts with “=”
  - until a complete expression or declaration is found

hammer.thor.cam.ac.uk% /group/clteach/acn/ml/unix/cml
FAM /group/clteach/acn/ml/unix/fam started on 02-Jan-1996 16:03:07
  (version 4.2.01 of Jan 25 1995)
Image file /group/clteach/acn/ml/unix/cml.exp
  (written on 25-Jan-1995 15:42:47 by FAM version 4.2.01)
[Loading Generic Heap...resexing...relocating by efff1ff8 (bytes)]

Edinburgh ML for DOS/Win32s/Unix
(C) Edinburgh University & A C Norman

- 2+3;
> 5 : int

- 2
= +
= 3
= ;
> 5 : int

- it;
> 5 : int

- Prompts will (usually) not be shown
- As above, output will be flagged with >
Declarations

- Declaration `val x=e`
  - evaluates `e`
  - binds the resulting value to `x`

```ml
val x=2*3;
> val x = 6 : int

it=x;
> val it = false : bool
```

- Declarations do not affect `it`
- `e;` at top level is treated as `let it = e;`
- ML initially binds `it` to a special value `()`
  - the only value of the one-element type `unit`
Multiple declarations

- To bind the variables \( x_1, \ldots, x_n \) simultaneously to the values of the expressions \( e_1, \ldots, e_n \)
  - \( \text{val } x_1 = e_1 \) and \( x_2 = e_2 \ldots \) and \( x_n = e_n \)
  - \( \text{val } (x_1, x_2, \ldots, x_n) = (e_1, e_2, \ldots, e_n) \).

- These two declarations are equivalent

```ml
val y = 10 and z = x;
> val y = 10 : int
> val z = 6 : int

val (x, y) = (y, x);
> val x = 10 : int
> val y = 6 : int
```

- \textit{let } \( d \text{ in } e \text{ end} \) makes \( d \) local to \( e \)

```ml
let val x = 2 in x * y end;
> val it = 12 : int

x;
> val it = 10 : int
```
Comments

- Comments start with (* and end with *)
  - nest like parentheses
  - can extend over many lines
  - can be inserted wherever spaces are allowed

```
tr(* comments can’t go in the middle of names *)ue;
> Error: unbound variable or constructor: tr
> Error: unbound variable or constructor: ue

1 (* this comment is ignored *) < 2;
> val it = true : bool

(* Inside this comment (* another one is nested *) ! *)
```
Functions

- To define function $f$ with formal parameter $x$ and body $e$ perform the declaration:
  
  - `fun $f$ $x$ = $e$

- To apply $f$ to $e$ evaluate $f$ $e$

```
fun f x = 2*x;
> val f = fn : int -> int

f 4;
> val it = 8 : int
```

- Functions are printed as
  
  - `fn` in SML/NJ
  - `Fn` in Edinburgh ML
  - Function values are not printable

- Functions are printed as `fn` here

- The type of the function is also printed
Typechecking errors

- Applying a function to an argument of the wrong type results in a typechecking error
  - Error messages are system dependent

- In SML/NJ

```
-f true;
std_in:12.1-12.6
Error: operator and operand don’t agree
  operator domain: int
  operand: bool
  in expression:
    f true
```

- In Edinburgh ML

```
-f true;
Type clash in: (f true)
Looking for a: int
I have found a: bool
```
Binding power of function application

- Function application binds tightly

- Consider: $f \ 3 \ + \ 4$
  - means $(f \ 3)+4$
  - not $f(3+4)$
Functions of several arguments

fun add (x:int) (y:int) = x+y;
> val add = fn : int -> int -> int

add 3 4;
> val it = 7 : int

val f = add 3;
> val f = fn : int -> int

f 4;
> val it = 7 : int

- Application associates to the left
  - add 3 4 means (add 3)4

- In add 3
  - add is applied to 3
  - the result has type int -> int
  - which adds 3 to its argument
  - add takes its arguments ‘one at a time’
Overloading

• ML needs help to tell whether:
  • + is addition of integers
  • or addition of reals

• + is overloaded

- fun add x y = x+y;
Type checking error in: (syntactic context unknown)
Unresolvable overloaded identifier: +
Definition cannot be found for the type: ('a * 'a) -> 'a

• Only built-in operators are overloaded

  • users cannot overload their operators