Functional Programming

Functional programming supports a form of declarative programming.

The building blocks are *true* functions, i.e., mathematical functions.

Functional Programming in ML

```
- 2 + 2;
val it = 4 : int

- it;
val it = 4 : int

- val b = exp(0.5);
val b = 1.64872127070013 : real

- val bb = ln(b);
val bb = 0.5 : real
```

Functional Programming in ML

```
- fn ((x:real),(y:real)) => x*x+y*y;
val it = fn: real * real -> real
```

`it` is $\lambda p.\ \text{add}\ (\text{sqr}\ (\text{first}\ p))\ (\text{sqr}\ (\text{second}\ p))$.

```
- fn ((x:real),(y:real)) => x*x+y*y (5.0,2.0);
val it = 29.0 : real
```

`it` is $(\lambda p.\ \text{add}\ (\text{sqr}\ (\text{first}\ p))\ (\text{sqr}\ (\text{second}\ p)))\ (5.0,2.0)$

Functional Programming in ML

```
- fn((x:real),(y:real)) => x*x+y*y (5.0,2.0);
val it = 29.0 : real
```

`it` is $(\lambda p.\ \text{add}\ (\text{sqr}\ (\text{first}\ p))\ (\text{sqr}\ (\text{second}\ p)))\ (5.0,2.0)$

```
- val sumsq = fn((x:real),(y:real)) => x*x+y*y;
val sumsq= fn: real * real -> real
```

$sumsq = (\lambda p.\ \text{add}\ (\text{sqr}\ (\text{first}\ p))\ (\text{sqr}\ (\text{second}\ p)))$

```
- sumsq(5.0,2.0);
val it = 29.0 : real
```

$sumsq\ (5.0,2.0)$

Functional Programming in ML

```
- 2 + 2;
val it = 4 : int

- 1*1.0;
ERROR
```

ML is a strongly typed language.

```
- (real 1)*1.0;
val it = 1.0 : real

- real;
val it = fn : int -> real

- fun sqr x = x * x;
ERROR
```

Functional Programming in ML

```
- fun sqr(x:int) = x*x;
val sqr = fn: int -> int

- fun sqr x = (x:int)*x;
val sqr = fn: int -> int

- fun sqr x = x*(x:int);
val sqr = fn: int -> int

- fun sqr x = (x*x):int;
val sqr = fn: int -> int
```

$$sqr = \lambda x.\ x \cdot x$$

```
- fun abs x = if x >= 0 then x else ~x;
val abs = fn : int -> int
- abs(4-7);
val it = 3 : int
```

$abs = \lambda x.\ \text{if } x \geq 0 \text{ then } x \text{ else } -x$

$$abs = \lambda x. \begin{cases} x & x \geq 0 \\ -x & \text{otherwise} \end{cases}$$

```
- fun negative x = x < 0;
val negative = fn : int -> bool
- negative(~3);
val it = true : bool
```

$negative = \lambda x.\ \ x < 0$

```
- fun nonneg x = not(negative x);
val nonneg = fn : int -> bool
- nonneg(0);
val it = true : bool
```

$nonneg = \lambda x.\ \ \neg(negative\ x)$

RECURSION

```
- fun fact n = if n <= 0 then 1
=     else n*fact(n-1);
val fact = fn: int-> int
- fact(4);
val it = 24 : int
```

TUPLE

```
- val origin = (0.0,0.0,0.0);
val origin = (0.0,0.0,0.0) : real * real * real

- fun length (x,y,z) = sqrt(x*x+y*y+z*z);
val length = fn : real * real * real -> real
- fun first (x,_,_) = x;
val first = fn : 'a * 'b * 'c -> 'a
- fun second (_,y,_) = y;
val second = fn : 'a * 'b * 'c -> 'b
- fun third (_,_,z) = z;
val second = fn : 'a * 'b * 'c -> 'c
```

Local Declarations in ML

```
- fun area (a,b,c) = let val p = (a+b+c)/2.0 in
=     sqrt(p*(p-a)*(p-b)*(p-c))
= end;
val area = fn : real * real * real -> real
```

```
- val sides1 = (1.0,2.0,3.0);
val sides1 = (1.0,2.0,3.0) : real * real * real
- val sides2 = (3.0,4.0,5.0);
val sides2 = (3.0,4.0,5.0) : real * real * real
- area sides1;
val it = 0.0 : real
- area sides2;
val it = 6.0 : real
```

Local Declarations in ML

```
- (let fun divides(x,y) = y mod x = 0 in
=  fn age => divides(10,age) orelse divides(25,age)
= end) 10;
val it = true : bool
```

$divides = \lambda p.\ (\text{second } p) \bmod (\text{first } p) = 0$

$(\lambda\ age.\ \ (divides\ (10, age) \vee divides\ (25, age)))\ 10$

$(\lambda\ age.\ \ (\vee\ divides\ (10, age)\ divides\ (25, age)))\ 10$

Note: There are **`andalso`** and **`orelse`** in ML.

User Defined Types in ML

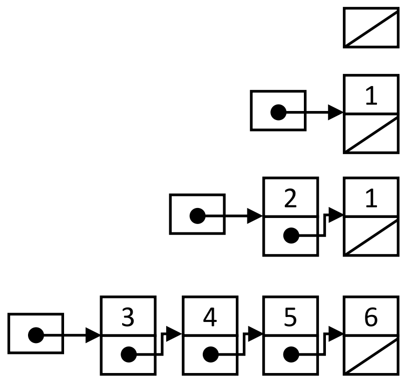
```
- datatype DIRECTION = North | East | South | West;
datatype DIRECTION
  con East : DIRECTION
  con North : DIRECTION
  con South : DIRECTION
  con West : DIRECTION

- val dir = East;
val dir = East : DIRECTION
```

Case Expressions in ML

```
- (fn dir =>
=   case dir of North => 0
=     |         East  => 90
=     |         South => 180
=     |         West  => 270
= ) East;
val it = 90 : int
```

Lists in ML

```
[]              []
[1]             1::[]
[2,1]           2::[1] = 2::(1::[])
[3,4,5,6]       3::[4,5,6]
                = 3::(4::[5,6])
                = 3::(4::(5::[6]))
                = 3::(4::(5::(6::[])))
```

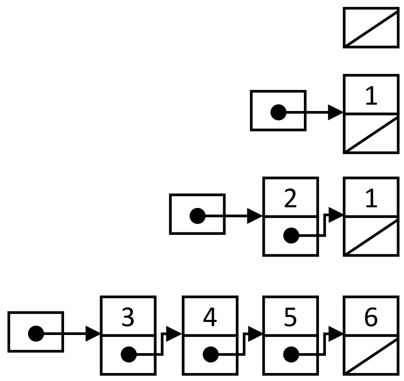



```
- [1,2,3];
val it = [1,2,3] : int list
- ["jim","mary"];
val it = ["jim","mary"] : string list
- [[1,2,3],[4,5,6]];
val it = [[1,2,3],[4,5,6]] : int list list
- [1,true];
ERROR
```

Lists in ML must be homogeneous.

Lists in ML

```
[]              []
[1]             1::[]
[2,1]           2::[1] = 2::(1::[])
[3,4,5,6]       3::(4::(5::(6::[])))
```




```
- hd([3,4,5,6]);
val it = 3 : int
- tl([3,4,5,6]);
val it = [4,5,6] : int list
- hd(tl(tl([3,4,5,6])));
val it = 5 : int
- tl([1]);
val it = [] : int list
- tl([]);
```

ERROR

Lists in ML

```
[]            []
[1]           1::[]
[2,1]         2::[1] ➔ 2::(1::[])
[3,4,5,6]     3::(4::(5::(6::[])))
```

```
- hd([3,4,5,6]);
val it = 3 : int
- tl([3,4,5,6]);
val it = [4,5,6] : int list
- hd(tl(tl([3,4,5,6])));
val it = 5 : int
- tl([1]);
val it = [] : int list
- tl([]);
ERROR
```

Lists in ML

```
- fun sumList list = if null list then 0
=          else (hd list) + sumList(tl list);
val sumList = fn : int list -> int
```

```
- fun sumList [] = 0
=   | sumList(a::list) = a + sumList list;
val sumList = fn : int list -> int
```

```
- sumList [1,2,3,4]
val it = 10 : int
```

Higher-Order Functions in ML

```
- fun double f = fn x => 2 * f(x);
val double = fn : ('a -> int) -> 'a -> int
```

$double = \lambda f\,.\,(\lambda x\,.\,\mathrm{mul}\ 2\ f(x))$

```
- fun inc x = x + 1;
val inc = fn : int -> int
```

$inc = \lambda x\,.\,\mathrm{add}\ x\ 1$

```
- double inc 3;
val it = 8 : int
```

$$
\begin{aligned}
double\ inc\ 3 \quad & \Rightarrow (\lambda f\,.\,(\lambda x\,.\,\mathrm{mul}\ 2\ f(x)))\ (\lambda x\,.\,\mathrm{add}\ x\ 1)\ 3 \\
& \Rightarrow (\lambda x\,.\,\mathrm{mul}\ 2\ ((\lambda w\,.\,\mathrm{add}\ w\ 1)\ x))\ 3 \\
& \Rightarrow \mathrm{mul}\ 2\ ((\lambda w\,.\,\mathrm{add}\ w\ 1)\ 3) \\
& \Rightarrow \mathrm{mul}\ 2\ (\mathrm{add}\ 3\ 1) \\
& \Rightarrow 8
\end{aligned}
$$

Higher-Order Functions in ML

```
- fun double f = fn x => 2 * f(x);
val double = fn : ('a -> int) -> 'a -> int
```

$double = \lambda f\ .\ (\lambda x\ .\ \mathrm{mul}\ 2\ f(x))$

```
- fun inc x = x + 1;
val inc = fn : int -> int
```

$inc = \lambda x\ .\ \mathrm{add}\ x\ 1$

```
- double inc;
val it = fn : int-> int
```

$$
\begin{aligned}
double\ inc \quad &\Rightarrow (\lambda f\ .\ (\lambda x\ .\ \mathrm{mul}\ 2\ f(x)))\ (\lambda x\ .\ \mathrm{add}\ x\ 1) \\
&\Rightarrow \lambda x\ .\ \mathrm{mul}\ 2\ ((\lambda w\ .\ \mathrm{add}\ w\ 1)\ x) \\
&\Rightarrow \lambda x\ .\ \mathrm{mul}\ 2\ (\mathrm{add}\ x\ 1)
\end{aligned}
$$