Structured Data Types

- Tuples - ordered collection of values
- Records - collection of named values
- Lists - list of values of homogeneous types
Tuples

- Syntax: ( exp-list )
  - ( 1, 2, 3);
  val it = (1,2,3) : int * int * int
  - (pi,r,area);
  val it = (3.14159,2.0,fn) : real * real * (real -> real)
Tuples

• Access by pattern matching or by label
  - `val (a, b) = (2.3, "zippy");`
  val a = 2.3 : real
  val b = "zippy" : string
  - `#3 (a, b, pi);`
  val it = 3.14159 : real
Multi-argument functions

- Argument of a function can be a tuple
  - fun mult (x,y) = x*y;
  val mult = fn : int * int -> int
  - fun mult (t : int*int) = #1 t * #2 t; (* ugly! *)
  val mult = fn : int * int -> int
Curried Functions

- Functions with two arguments would look like

  - \( \text{fun } \textit{power}(m,n) : \text{int } = \)
  
  \( = \text{if } n = 0 \text{ then } 1 \)
  
  \( = \text{else } m \times \text{power}(m,n-1); \)
  
  \( \text{val power } = \text{fn : int } \times \text{int } \rightarrow \text{int} \)
Curried Functions

• Equivalent function
  
  - fun `cpower` m n : int =
  
  = if n = 0 then 1
  
  = else m * cpower m (n-1);

  val cpower = fn : int -> int -> int
Curried Functions

- `power` and `cpower` are different implementations of an exponentiation function

```
- power(2,3);
val it = 8 : int
- cpower 2 3;
val it = 8 : int
```

Function `cpower` is “Curried” (Haskell Curry)
Curried Functions

• Can define new functions by partial evaluation

  - val power_of_two = cpower 2;
  val power_of_two = fn : int -> int
  - power_of_two 3;
  val it = 8 : int
Records

• A collection of labeled data items

- val ex = { name = "george", userid = 12 };

val ex =
{name="george",userid=12} :
{name:string, userid:int}
Records

- Access elements by pattern matching or label
  - `#name ex;`
  - `val it = "george" : string`
  - `val {name=username, ...} = ex;`
  - `val username = "george" : string`
Lists

- All elements must be of the same type in a list
  - [2, 6, 4, 9];
  val it = [2,6,4,9] : int list
  - ["a", "b", "c"];
  val it = ["a","b","c"] : string list
Lists

- [ 1, "a" ];

... Error: operator and operand don't agree

operator domain: int * int list
operand: int * string list
in expression:
1 :: "a" :: nil
List constructors

- `[]`, nil - empty list (all types)
- `::` cons operator (for “construct”) combines an element and a list into larger list
  
  - `1 :: [];

  ```
  val it = [1] : int list
  ```

  - `1 :: (2 :: [2]);`

  ```
  val it = [1,2,2] : int list
  ```
Functions on Lists

• length

• Head and tail
  - hd [ 3, 4 ];
  val it = 3 : int
  - tl [3, 4, 5 ];
  val it = [4, 5] : int list
Functions on Lists

- Concatenation
  
  - \([1, 2] @ [3, 4]\);
  
  val it = \([1,2,3,4] : \text{int list}\)

- rev – Reverse list
Exercise

• Given a list of integers, compute the sum of its elements

1. Add the head to the sum of elements in the tail
2. The base case is that we output 0 for an empty list, (the identity element for addition)

   
   - fun sum(L) = if L=[] then 0 else
     hd(L)+sum(tl(L));

   val sum = fn : int list -> int
Map function

- **map** applies another function to all elements of a list
  
  - fun sqr x = x * x;
  
  val sqr = fn : int -> int

  - map sqr [2,3,4,5];

  val it = [4,9,16,25] : int list