

```
fun append (xs,ys) =  
  if xs=[]  
  then ys  
  else (hd xs)::append(tl xs,ys)  
  
fun map (f,xs) =  
  case xs of  
    [] => []  
  | x::xs' => (f x)::(map(f,xs'))  
  
val a = map (increment, [4,8,12,16])  
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

# Programming Languages

Dan Grossman  
2013

*Pairs and Other Tuples*

# *Tuples and lists*

So far: numbers, booleans, conditionals, variables, functions

- Now ways to build up data with multiple parts
- This is essential
- Java examples: classes with fields, arrays

Now:

- *Tuples*: fixed “number of pieces” that may have different types

Coming soon:

- *Lists*: any “number of pieces” that all have the same type

Later:

- Other more general ways to create compound data

# *Pairs (2-tuples)*

Need a way to *build* pairs and a way to *access* the pieces

*Build:*

- Syntax:  **$(e1, e2)$**
- Evaluation: Evaluate  **$e1$**  to  **$v1$**  and  **$e2$**  to  **$v2$** ; result is  **$(v1, v2)$** 
  - A pair of values is a value
- Type-checking: If  **$e1$**  has type  **$ta$**  and  **$e2$**  has type  **$tb$** , then the pair expression has type  **$ta * tb$** 
  - A new kind of type

# *Pairs (2-tuples)*

Need a way to *build* pairs and a way to *access* the pieces

Access:

- Syntax: **#1 e** and **#2 e**
- Evaluation: Evaluate **e** to a pair of values and return first or second piece
  - Example: If **e** is a variable **x**, then look up **x** in environment
- Type-checking: If **e** has type **ta \* tb**, then **#1 e** has type **ta** and **#2 e** has type **tb**

# Examples

Functions can take and return pairs

```
fun swap (pr : int*bool) =  
  (#2 pr, #1 pr)
```

```
fun sum_two_pairs (pr1 : int*int, pr2 : int*int) =  
  (#1 pr1) + (#2 pr1) + (#1 pr2) + (#2 pr2)
```

```
fun div_mod (x : int, y : int) =  
  (x div y, x mod y)
```

```
fun sort_pair (pr : int*int) =  
  if (#1 pr) < (#2 pr)  
  then pr  
  else (#2 pr, #1 pr)
```

# Tuples

Actually, you can have *tuples* with more than two parts

- A new feature: a generalization of pairs

- `(e1, e2, ..., en)`
- `ta * tb * ... * tn`
- `#1 e, #2 e, #3 e, ...`

Homework 1 uses triples of type `int*int*int` a lot

# Nesting

Pairs and tuples can be nested however you want

- Not a new feature: implied by the syntax and semantics

```
val x1 = (7, (true, 9)) (* int * (bool*int) *)  
  
val x2 = #1 (#2 x1)      (* bool *)  
  
val x3 = (#2 x1)         (* bool*int *)  
  
val x4 = ((3, 5), ((4, 8), (0, 0)))  
          (* (int*int)*((int*int)*(int*int)) *)
```