lists and tuples; more functions; mutable state
Ullman 2.4.1, 2.4.3; 3 - 3.2; 2.3

slides created by Marty Stepp
http://www.cs.washington.edu/341/
(* Computes n!, or 1*2*3*...*n-1*n. *
   precondition: n >= 0 *)

fun factorial(n) =
    if n = 0 then 1
    else n * factorial(n - 1);
Running an ML program (1.2)

• an ML program can be thought of as a series of bindings between names (variables, functions, etc.) and values

• from your operating system's terminal / console:
  
  sml filename.sml
  ▪ preferred; gives a cleaner environment

• running a program from within ML interpreter:
  
  use "filename.sml";
  ▪ drawback: any previous definitions still exist (a "dirty environment")
Lists (2.4.3)

\[ expr1, expr2, \ldots, exprN \]

- **list**: contains 0 or more values of the same type
  
  ```
  val lst = [42, ~7, 19];
  val lst = [42,~7,19] : int list
  ```

- The empty list is written as [ ] or nil

- You do **not** access a list's elements by indexes. Instead:
  
  ```
  hd(list)    returns the list's first element
  tl(list)    returns the list of all elements except the first
  ```

- Does \( \text{tl} \) copy the list, or use a reference? Does it matter?
Concat and cons: growing lists

• concatenate two lists: 
  \[10, 20\] @ \[30, 40\];
  \text{val it = [10,20,30,40] : int list}

\text{list1 @ list2}

"concat"

• attach an element onto a list: 
  10 :: \[20, 30\];
  \text{val it = [10,20,30] : int list}

\text{element :: list}

"cons"

▪ How would we attach an element to the end of a list?

\text{equivalent to [element] @ list}
More about lists

• find out a list's length with the length function:
  \( \text{length}(["ab","cd","e"]) \rightarrow 3 \)

• strings can be converted to/from lists
  \( \text{explode}("CSE") \rightarrow ["C", "S", "E"] \)
  \( \text{implode}(["H", "i"])) \rightarrow "hi" \)
  \( \text{concat}(["ab","cd","e"])) \rightarrow "abcde" \)

• ML has a List structure with many other functions such as List.find, List.rev, and List.partition
Exercise

• Define a function named `sum` that takes a list of integers as a parameter and computes the sum of its elements. A list that contains no elements has a sum of 0.
  ▪ example: `sum([1, 7, ~2, 15])` should produce 21

• Define a function named `last` that takes a list of integers as a parameter and produces the last element of the list. You may assume that the list is non-empty.
  ▪ example: `last([1, 7, ~2, 15])` should produce 15
fun sum(lst) =  
    if lst = [] then 0  
    else hd(lst) + sum(tl(lst));  

fun last(lst) =  
    if length(lst) = 1 then hd(lst)  
    else last(tl(lst));
Parametric polymorphism

• What are the types of `hd` and `tl`? (and `length`?)
  - `hd`;
    `val it = fn : 'a list -> 'a`
  - `tl`;
    `val it = fn : 'a list -> 'a list`

• **parametric polymorphism**: ability of a function to handle values identically without depending on their type
  ▪ language is more expressive; still handles types properly
  ▪ similar to generics in Java (e.g. `ArrayList<String>`)  
  ▪ can we write a function using parametric polymorphism?
Tuples (2.4.1)

\((expr_1, expr_2, \ldots, expr_N)\)

- **tuple**: contains 1 or more values of *any* type
  
  ```ml
  val t = (42, 19, 4.6, "hi");
  val t = (42,19,4.6,"hi") : int * int * real * string
  ```

- You can access a tuple's elements by 1-based indexes:
  
  ```ml
  #2(t);
  val it = 19 : int
  ```
More about tuples

• The type of a tuple is written as its element types with *
  ▪ The type of (1, 2.7) is int * real
  ▪ What is the type of (true, ~1, 7)?

• lists and tuples can be nested
  [[4, 3], [~7], [55, 99, 1]]
  (42, 19.6, ("hi", "bye", true), ~7, "ok")
• A tuple can be passed as a parameter list to a function:

- fun max(a, b) = if a > b then a else b;
  val max = fn : int * int -> int

- val nums = (7, 24);
  val nums = (7,24) : int * int

- max(nums);
  val it = 24 : int
• Define a function named `convertNames` that accepts a list of("first-name", "last-name") tuples and produces a list of "last-name, first-name" strings. For the list:

```scala
val names = ["Hillary", "Clinton"),
             ("Barack", "Obama"),
             ("Joseph", "Biden")];
```

The call of `convertNames(names);` should produce:

```
["Clinton, Hillary", "Obama, Barack", "Biden, Joseph"];```
Approaching a problem

• One strategy: think procedurally and write pseudo-code:
  – create new result list = [].
  – for each element \( e \) of list:
    – convert \( e \) into "last, first" format.
    – append \( e \) onto result list.
  – return result list.

• How do we express these computations recursively?
• Can we simplify the problem? Can we break it down?
• Write a *helper function* to solve part of the problem:

  – create new result list = [].
  – for each element e of list:
    – *convert e into "last, first" format.*
    – append e onto result list.
  – return result list.

```haskell
fun lastFirst(name : string*string) =
    #2(name) ^ "", " ^ #1(name);
```

or, expand the tuple in the definition:

```haskell
fun lastFirst(first, last) =
    last ^ ", " ^ first;
```
Thinking recursively

• Useful questions to ask:
  ▪ What is the base case?
  ▪ How would I handle a case that is "one-above" the base? (That is, one iteration/call away from being a base case?)
  ▪ How do I target a small part of the problem and solve it?
  ▪ What recursive call(s) will solve the rest of the problem?
fun lastFirst(first, last) = last ^ ", " ^ first;

fun convertNames(lst) = 
    if lst = [] then []
    else lastFirst(hd(lst)) ::
        convertNames(tl(lst));
**Mutable state**

- **mutable state**: Ability for data to be modified after creation / declaration.
  - Example:
    ```java
    int x = 3;  
    ...  
    x = 7;
    ```

- Mutable state is good, right? Do we ever *not* have it?
  - constants (public static final ...)
  - Strings (s.toLowerCase();)
  - objects with only get methods, no set ("immutable")
Why are Strings immutable?

• Why was Java designed with immutable strings?

```java
public Employee(String name, ...) {
    this.name = name;
    ....
} // how could this code
    // be abused if Strings
    // were mutable in Java?

public String getName() {
    return name;
}
```


• But what if I want a mutable string?

  • StringBuilder, StringBuffer
• **Q:** Is it always okay to replace the expression:

\[ f(x) + f(x) \]  \quad \text{with:} \quad 2 \times f(x) \, ?

- maybe; \( f \) might do something besides return its value
  - might produce output, e.g. `System.out.println`
  - might increment a global counter, change a field value, etc.

• **side effect:** When a function, in addition to producing a value, modifies state or has an external interaction.
  - referential transparency: if call can always be replaced with result value
  - idempotent: if it always returns the same result for the same input
Minimizing side effects

• ML (like many func.langs.) tries to minimize side effects
  ▪ (almost) everything is immutable
  ▪ variables' values cannot be changed (only re-defined)
  ▪ functions' behavior depends only on their inputs

• Benefits of this philosophy?
  ▪ the compiler/interpreter can heavily optimize the code
  ▪ much easier to understand/predict behavior of code; code can be more thoroughly verified for correctness
  ▪ robust; hard for one chunk of code to damage another
  ▪ lack of side effects reduces dependency between code – allows code to be more easily parallelized
val x = [2, 4];
val y = [5, 3, 0];
val z = x @ y;

• Does z have a copy of y? Or refer to the same list?
  ▪ in Java: it's important to know what is shared
    – if somebody changes z, it might change x or y, too
  ▪ in ML: doesn't matter; data is immutable