CSE 341
Lecture 12

structures

slides created by Marty Stepp

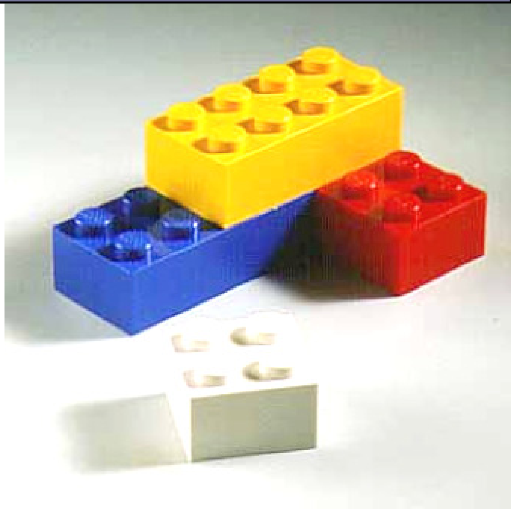
http://www.cs.washington.edu/341/

Modules

- **module**: A separate, self-contained, reusable, interchangeable software component.
  - basis of the idea of *modular programming*

- ML's module system includes:
  - **structures** (like classes)
  - **signatures** (like interfaces)
  - **functors** (like parameterized class factories)

Why modules?



- **organization**: puts related code together
- **decomposition**: break down a problem
- **information hiding / encapsulation**: protect data from damage by other code
- group identifiers into **namespaces**; reduce # of globals
- provide a layer of **abstraction**; allows re-implementation
- ability to rigidly enforce data **invariants**
- provides a discrete unit for **testing**

Structure syntax

```
structure name =
struct
    definitions
end;
```

a structure can contain:

- function definitions
- `val` declarations (variables; class constants)
- exceptions
- type definitions and datatypes

Structure example

```
(* Functions and data types for binary search trees of integers. *)
structure IntTree = struct
    datatype intTree = Empty | Node of int * intTree * intTree;

    (* Adds the given value to the tree in order.
       Produces/returns the new state of the tree node after the add. *)
    fun add(Empty, value) = Node(value, Empty, Empty)
    |   add(n as Node(data, left, right), value) =
              if value < data then Node(data, add(left, value), right)
         else if value > data then Node(data, left, add(right, value))
         else n;   (* duplicate; no change *)

    (* Produces the height of the given tree.
       An Empty tree has a height of 0. *)
    fun height(Empty) = 0
    |   height(Node(_, left, right)) =
            1 + Int.max(height(left), height(right));

    (* Produces the smallest value in the tree, if the tree has any data. *)
    fun min(Node(data, Empty, right)) = SOME data
    |   min(Node(data, left, right))  = min(left)
    |   min(Empty) = NONE;
end;
```

Using a structure

structure.member

```
val t1 = IntTree.add(IntTree.Empty, 42);
val t2 = IntTree.add(t1, 27);
val mn = IntTree.min(t2);
```

- structure members such as `add` and `Empty` are no longer part of the global namespace

Importing a structure's contents

```
open structure;
```

```
open IntTree;
val t1 = add(Empty, 42);
val t2 = add(t1, 27);
val mn = min(t2);
```

- if you `open` a structure, its members are brought into the global namespace and can be used without a prefix
  - +: shorter client code
  - -: namespace pollution / confusion (e.g. with `Int.min`)

ML's built-in structures

struct	**members** (partial)
`Int`	int minInt maxInt abs min max toString +-*
`Real`	real precision +-*/ abs min max compare floor ceil trunc round toString fromString
`Char`	char ord chr isAscii isDigit toLower toUpper isSpace
`String`	string size sub concat explode tokens compare ^
`Bool`	bool not toString fromString
`List`	@ :: hd tl null length nth take getItem rev concat append map find filter partition foldl foldr exists all

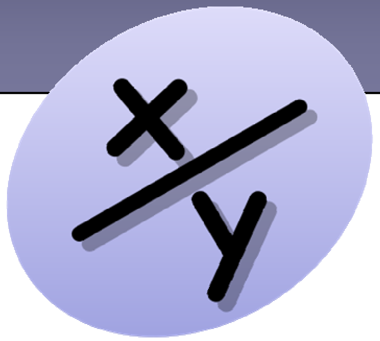
http://www.standardml.org/Basis/

More built-in structures

struct	**members** (partial)
Option	option isSome valOf getOpt compose join
General	unit exn (exceptions) order ! := o before ignore
Math	pi e sqrt sin cos tan asin acos atan pow ln log10
IntInf	divMod pow log2 orb xorb andb notb << ~>>
TextIO	openIn openOut print inputLine stdIn stdOut stdErr
OS.Process	status success failure exit getEnv sleep
others	Date Time Timer Array Vector Socket CommandLine

http://www.standardml.org/Basis/

Structure exercise



- Define a structure `Rational` to represent rational numbers, i.e., fractions.
  - It can be a whole number, or a numerator/denominator.
  - Define an `add` function to add two rational numbers.
  - Define a `toString` method to produce a rational string.
  - Don't worry (yet) about the notion of reducing fractions.

Structure solution

```
(* initial version of Rational structure that shows how to group
   a datatype, constructors, and functions into a single unit. *)

structure Rational = struct
  datatype rational = Whole of int | Fraction of int * int;

  fun add(Whole i, Whole j) = Whole(i + j)
  |   add(Whole i, Fraction(j, k)) = Fraction(j + k * i, k)
  |   add(Fraction(j, k), Whole i) = Fraction(j + k * i, k)
  |   add(Fraction(a, b), Fraction(c, d)) = Fraction(a*d + b*c, b*d);

  fun toString(Whole i) = Int.toString(i)
  |   toString(Fraction(a, b)) = Int.toString(a) ^ "/"
                               ^ Int.toString(b);
end;
```

Structure exercise 2

- Improve the Rational structure by adding features:
  - Prohibit rational numbers that have a denominator of 0.
  - Represent all rational numbers in *reduced* form.
    - e.g. instead of 4/12, store 1/3.
    - make use of Euclid's formula for greatest common divisors:

```
fun gcd(a, 0) = abs(a)
|   gcd(a, b) = gcd(b, a mod b)
```

Structure solution 2

```
(* Includes gcd/reduce and 'new' function to guarantee invariants *)
structure Rational = struct
    datatype rational = Whole of int | Fraction of int * int;
    exception Undefined;

    fun gcd(a, 0) = abs(a)
    |   gcd(a, b) = gcd(b, a mod b);

    fun reduce(Whole(i)) = Whole(i)
    |   reduce(Fraction(a, b)) =
            let val d = gcd(a, b)
            in  if b = d then Whole(a div d)
                else Fraction(a div d, b div d)
            end;

    fun new(a, 0) = raise Undefined      (* constructs a fraction *)
    |   new(a, b) = reduce(Fraction(a, b));

    fun add(Whole(i), Whole(j)) = Whole(i + j)
    |   add(Whole(i), Fraction(c, d)) = Fraction(i*d + c, d)
    |   add(Fraction(a, b), Whole(j)) = Fraction(a + j*b, b)
    |   add(Fraction(a, b), Fraction(c, d)) =
            reduce(Fraction(a*d + c*b, b*d));

    (* toString unchanged *)
end;
```

The order datatype

```
datatype order = LESS | EQUAL | GREATER;
```

- part of ML standard basis library
- used to indicate whether one value is <, =, > than other
  - can be used when defining *natural orderings* for types

- many structures (Int, Real, String, etc.) define a `compare` method that returns a value of type `order`
  - some also implement <, <=, >, >= operators based on it, but overloaded operators don't work well on structures

Order example

```
(* Includes gcd/reduce and 'new' function to guarantee invariants *)
structure Rational = struct
    datatype rational = Whole of int | Fraction of int * int;

    ...

    fun compare(Whole(a), Whole(b)) = Int.compare(a, b)
    |   compare(Fraction(a, b), Whole(c)) = Int.compare(a, c*b)
    |   compare(Whole(c), Fraction(a, b)) = Int.compare(a, c*b)
    |   compare(Fraction(a,b), Fraction(c,d)) = Int.compare(a*d, c*b)
end;
```