

Introduction to Functional Programming and ML

Features of ML

- A pure functional language
 - serious programs can be written without using variables
- Widely accepted
 - reasonable performance (claimed)

Functional Programming

- *Function evaluation* is the basic concept for a programming paradigm that has been implemented in *functional programming languages*.

SML

- The language ML (“Meta Language”) was originally introduced in the 1970’s as part of a theorem proving system, and was intended for describing and implementing proof strategies in the Logic for Computable Functions (LCF) theorem prover.
- Standard ML of New Jersey (SML) is an implementation of ML.
- The basic mode of computation in SML is the use of the definition and application of functions.

Install Standard ML

- Download from:
 - <http://www.smlnj.org>
- Start Standard ML:
 - Type **sm1** from the shell (run command line in Windows)
- Exit Standard ML:
 - **Ctrl-Z** under Windows
 - **Ctrl-D** under Unix/Mac
 - Mac: `/usr/local/smlnj/bin`

Standard ML

- The basic cycle of SML activity has three parts:
 - read input from the user,
 - evaluate it,
 - print the computed value (or an error message).

Hello, world in SML

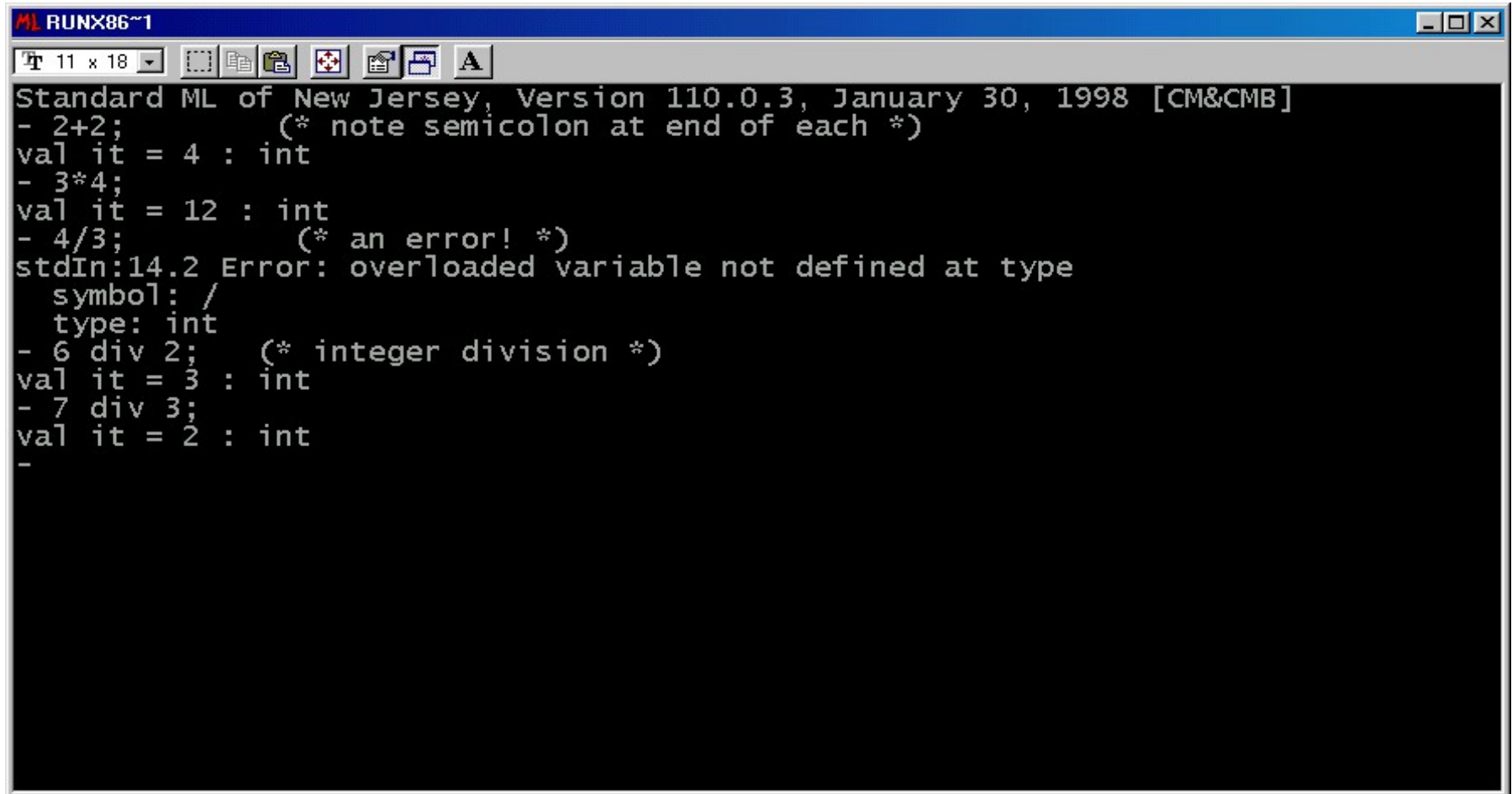
```
Standard ML of New Jersey,  
- print("Hello world\n");  
Hello world  
val it = () : unit  
-
```

Arithmetic in ML

- Copy and paste the following text into a Standard ML window

```
2+2;          (* note semicolon at end*)
3*4;
4/3;          (* an error! *)
6 div 2;      (* integer division *)
7 div 3;
```


It should look like this



```
ML RUNX86~1
Standard ML of New Jersey, Version 110.0.3, January 30, 1998 [CM&CMB]
- 2+2;          (* note semicolon at end of each *)
val it = 4 : int
- 3*4;
val it = 12 : int
- 4/3;          (* an error! *)
stdIn:14.2 Error: overloaded variable not defined at type
  symbol: /
  type: int
- 6 div 2;      (* integer division *)
val it = 3 : int
- 7 div 3;
val it = 2 : int
-
```

Declaring Variables

- SML variables do not vary (!)
 - once a variable is bound to a value, it is bound for life
 - they can be redefined, but existing uses of that constant (e.g. in functions) aren't affected by such redefinition

```
val freezingFahr = 32;
```

Variable Scope

- The scope of a variable is limited by using let expression and local declarations
- let
 val m:int = 3
 val n:int = m*m
in
 m*n
end;

What is the value of m after the following evaluation?

- ```
val m:int = 2
val r:int =
 let
 val m:int=3
 val n:int=m*m
 in
 m*n
 end * m
```

# Declaring Functions

- A function takes an input value and returns an output value
- ML will figure out the types

```
fun fahrToCelsius f = (f -freezingFahr) * 5 div 9;
fun celsiusToFahr c = c * 9 div 5 + freezingFahr;
```

```
ML RUNX86~1
T 11 x 18
type: int
- 6 div 2; (* integer division *)
val it = 3 : int
- 7 div 3;
val it = 2 : int
- val freezingFahr = 32;
val freezingFahr = 32 : int
- fun fahrToCelsius f = (f -freezingFahr) * 5 div 9;
val fahrToCelsius = fn : int -> int
- fun celsiusToFahr c = c * 9 div 5 + freezingFahr;
val celsiusToFahr = fn : int -> int
- fahrToCelsius 0;
val it = ~18 : int
- fahrToCelsius 32;
val it = 0 : int
- GC #0.0.0.0.1.5: (0 ms)
fahrToCelsius 212;
val it = 100 : int
- celsiusToFahr 0;
val it = 32 : int
- celsiusToFahr 100;
val it = 212 : int
- celsiusToFahr 30;
val it = 86 : int
-
_
```

# Notes

- ML is picky about not mixing types, such as int and real, in expressions
- The value of “it” is always the last value computed
- Function arguments don’t always need parentheses, but it doesn’t hurt to use them

# Types of arguments and results

- ML figures out the input and/or output types for simple expressions, constant declarations, and function declarations
- If the default isn't what you want, you can specify the input and output types, e.g.

```
fun divBy2 x:int = x div 2 : int;
fun divideBy2 (y : real) = y / 2.0;
divBy2 (5);
divideBy2 (5.0);
```



# Two similar divide functions

```
- fun divBy2 x:int = x div 2 : int;
val divBy2 = fn : int -> int
```

```
- fun divideBy2 (y : real) = y / 2.0;
val divideBy2 = fn : real -> real
```

```
- divBy2 (5);
val it = 2 : int
```

```
- divideBy2 (5.0);
val it = 2.5 : real
-
```

# Ints and Reals

- Note `~` is unary minus
- `min` and `max` take just two input arguments, but that can be fixed!
- `Real` converts ints to real
- Parens can sometimes be omitted

```
Int.abs ~3;
Int.sign ~3;
Int.max (4, 7);
Int.min (~2, 2);
real(freezingFahr);
Math.sqrt(real(2));
Math.sqrt(real 3);
```

```
- Int.abs ~3;
val it = 3 : int
- Int.sign ~3;
val it = ~1 : int
- Int.max (4, 7);
val it = 7 : int
- Int.min (~2, 2);
val it = ~2 : int
- real(freezingFahr);
val it = 32.0 : real
- Math.sqrt(real(2));
val it = 1.41421356237 : real
- Math.sqrt(real 3);
val it = 1.73205080757 : real
```

# Strings

- Delimited by double quotes
- the caret mark ^ is used for string concatenation, e.g. “house”^”cat”
- \n is used for newline, as in C and C++

# Lists in ML

- Objects in a list must be of the same type
  - `[1,2,3]`;
  - `[“dog”, “cat”, “moose”]`;
- The empty list is written `[]` or `nil`

# Making Lists

- The `@` operator is used to concatenate two lists of the same type
- The functions `hd` and `tl` give the first element of the list, and the rest of the list, respectively

# List Operations

```
- val list1 = [1,2,3];
val list1 = [1,2,3] : int list
- val list2 = [3,4,5];
val list2 = [3,4,5] : int list
- list1@list2;
val it = [1,2,3,3,4,5] : int list
- hd list1;
val it = 1 : int
- tl list2;
val it = [4,5] : int list
```

# Strings and Lists

- The explode function converts a string into a list of characters
- The implode function converts a list of characters into a string
- Examples:
  - `explode("foo");`  
`val it = [#"f",#"o",#"o"] : char list`
  - `implode [#"c",#"a",#"t"];`  
`val it = "cat" : string`
  -



# Heads and Tails

- The cons operator  $::$  takes an element and prepends it to a list of that same type.
- For example, the expression  $1::[2,3]$  results in the list  $[1,2,3]$
- What's the value of  $[1,2]::[[3,4], [5,6]]$  ?

# Functions and Patterns

- Recall that min and max take just two arguments
- However, using the fact that, for example,
  - $\min(a, b, c) = \min(a, \min(b, c))$