Lecture 06: Programming Languages [CMPU-235]
September 19, 2017

Sameer S Pradhan
Vassar College

Programming in the Small—Revisited (Solutions and Discussion)

Learning SML by Interactive Examples

```
> sml
Standard ML of New Jersey v110.78 [built: Wed Jan 14 12:52:09 2015]

- 1 + 2;
val it = 3 : int

- 3+4;
val it = 7 : int

- 5+6
= ;
val it = 11 : int

- 7
= +
= 8;
val it = 15 : int
```

Naming Values

```
- val a = 2 + 3;
val a =  : int

- a * a;
val it = 25 : int

- it + a;
val it = 30 : int
```

Negative Quirks

```
- 2 - 5;
val it = ~3 : int

- -17;
stdIn:60.1 Error: expression or pattern begins with infix
identifier "-"
stdIn:60.1-60.4 Error: operator and operand don't agree
[literal]
  operator domain: 'Z * 'Z
  operand:         int
  in expression:
    - 17

- ~17;
val it = ~17 : int

- 3 * ~1;
val it = ~3 : int
```

Division Quirks

```
- 7 / 2;
stdIn:1.1-1.6 Error: operator and operand don't agree
[literal]
  operator domain: real * real
  operand:         int * int
  in expression:
    7 / 2

- 7.0 / 2.0;
val it = 3.5 : real

- 7 div 2; (* integer division *)
val it = 3 : int
```

Simple Functions

```
- val inc = fn x => x + 1;
val inc = fn : int -> int (* SML figures out type! *)

- inc a;
val it = 6 : int

- fun dbl y = y * 2;
  (* Syntactic sugar for val dbl = fn y => y * 2 *)
val dbl = fn : int -> int

- dbl 5;
val it = 10 : int

- (fn x => x * 3) 10; (* Don't need to name function to use it *)
val it = 30 : int
```

When Parentheses Matter

```
- dbl(5); (* parens are optional here *)
val it = 10 : int

- (dbl 5); (* parens are optional here *)
val it = 10 : int

- inc (dbl 5); (* parens for argument subexpressions are required! *)
val it = 11 : int

- (inc dbl) 5;
stdIn:1.2-2.2 Error: operator and operand don't agree [tycon mismatch
  operator domain: int
  operand:         int -> int
  in expression:
    inc dbl

- inc dbl 5; (* default left associativity for application *)
stdIn:22.1-22.10 Error: operator and operand don't agree [tycon
mismatch]
  operator domain: int
  operand:         int -> int
  in expression:
    inc dbl
```

Function Composition

```
- (inc o dbl) 10; (* SML builtin infix function composition *)
val it = 21 : int

- (dbl o inc) 10;
val it = 22 : int

- fun id x = x; (* we can define our own identity fcn *)
val id = fn : 'a -> 'a (* polymorphic type; compare to
  Java's public static <T> T id (T x) {return x;} *)

- (inc o id) 10;
val it = 11 : int

- (id o dbl) 10;
val it = 20 : int

- (inc o inc o inc o inc) 10;
val it = 14 : int
```

Functions as Arguments

```
- fun app5 f = f 5;
val app5 = fn : (int -> 'a) -> 'a

- app5 inc;
val it = 6 : int

- app5 dbl;
val it = 10 : int

- app5 (fn z => z - 2);
val it = 3 : int
```

We'll see later that functions can also be returned as results from other functions and stored in data structures, so funtions are first-class in SML just as in Racket.

Scope of Top-Level Names

```
- val b = a * 2; (* recall a is 5 from before *)
val b = 10 : int

- fun adda x = x + a; (* a is still 5 from before *)
val adda = fn : int -> int

- adda 7;
val it = 12 : int

- adda b;
val it = 15 : int

- val a = 42; (* this is a different a from the previous one *)
val a = 42 : int

- b; (* ML values are immutable; nothing can change b's value *
val it = 10 : int

- adda 7;
val it = 12 : int (* still uses the a where adda was defined *)
```

Booleans

```
- 1 = 1;
val it = true : bool

- 1 > 2;
val it = false : bool

- (1 = 1) andalso (1 > 2);
val it = false : bool

- (1 = 1) orelse (1 = 2);
val it = true : bool

- (3 = 4) andalso (5 = (6 div 0)); (* short-circuit evaluation *)
val it = false : bool

- fun isEven n = (n mod 2) = 0;
val isEven = fn : int -> bool (* SML figures out type! *)

- isEven 17;
val it = false : bool

- isEven 6;
val it = true : bool
```

Conditionals

```
- fun f n = if n > 10 then 2 * n else n * n;
val f = fn : int -> int

- f 20;
val it = 40 : int

- f 5;
val it = 25 : int
```

Recursion

```
- fun fact n =
=   if n = 0 then
=     1
=   else
=     n * (fact (n - 1)); (* fun names have recursive scope *)
val fact = fn : int -> int
  (* simpler than Java definition b/c no explicit types! *)

- fact 5;
val it = 120 : int

- fact 12;
val it = 479001600 : int

- fact 13;
uncaught exception Overflow [overflow]
  raised at: <file stdIn>
  (* SML ints have limited size ☹ *)
```

Local Naming via `let`

`let` is used to define local names. Any such names "shadow" existing definitions from the surrounding scope.

```
- let val a = 27    (* 1st let binding *)
=     val b = 3   (* 2nd  binding *)
=     fun fact x = x + 2 (* 3rd binding *)
=  in fact (a div b) (* let body *)
= end; (* end terminates the let *)
val it = 11 : int
```

`let`-bound names are only visible in the body of the `let`.

```
- fact (a div b); (* these are global names *)
val it = 24 : int
```

Easier to Put Your Code in a File

```
(* This is the contents of the file
   ~gdome/cs251/sml/mydefns.sml.
   (* By the way, comments nest properly in SML! *)
   It defines integers a and b and the fact function. *)

val a = 2 + 3

val b = 2 * a

fun fact n = (* a recursive factorial function *)
  if n = 0 then
    1
  else
    n * (fact (n - 1))
```

- File is a sequence of value/function definitions.
- Definitions are **not** followed by semi-colons in files!
- There are **no equal signs** for multiple-line definitions.

Using Code From a File

```
- Posix.FileSys.getcwd(); (* current working directory *)
val it = "/students/gdome" : string

- Posix.FileSys.chdir("/students/gdome/cs251/sml");
  (* change working directory *)
val it = () : unit

- Posix.FileSys.getcwd();
val it = "/students/gdome/cs251/sml" : string

- use "mydefns.sml";  (* load defns from file as if *)
[opening mydefns.sml] (* they were typed manually *)
val a = 5 : int
val b = 10 : int
val fact = fn : int -> int
val it = () : unit

- fact a
val it = 120 : int
```

Another File Example

```
(* This is the contents of the file test-fact.sml *)
val fact_3 = fact 3
val fact_a = fact a
```

```
- use "test-fact.sml";
[opening test-fact.sml]
val fact_3 = 6 : int
val fact_a = 120 : int
val it = () : unit
```

Nested File Uses

```
(* The contents of the file load-fact.sml *)
use "mydefns.sml"; (* semi-colons are required here *)
use "test-fact.sml";
```

```
- use "load-fact.sml";
[opening load-fact.sml]
[opening mydefns.sml]
val a = 5 : int
val b = 10 : int
val fact = fn : int -> int
val it = () : unit
[opening test-fact.sml]
val fact_3 = 6 : int
val fact_a = 120 : int
val it = () : unit
val it = () : unit
```

Tuples

```
- val tpl = (1 + 2, 3 < 4, 5 * 6, 7 = 8);
val tpl = (3,true,30,false) : int * bool * int * bool

- #1 tpl;
val it = 3: int

- #2 tpl;
val it = true : bool

(* In practice, always use pattern matching (below)
    rather than #1, #2, etc. *)
- ((#1 tpl) + (#3 tpl), (#2 tpl) orelse (#4 tpl));
val it = (33,true) : int * bool

(* Can "deconstruct" tuples via pattern matching *)
- let val (i1, b1, i2, b2) = tpl
=  in (i1 + i2, b1 orelse b2)
= end;
val it = (33,true) : int * bool
```

Strings

```
- "foobar";
val it = "foobar" : string

- "foo" ^ "bar" ^ "baz";
val it = "foobarbaz" : string

- print ("baz" ^ "quux");
bazquuxval it = () : unit

- print ("baz" ^ "quux\n"); (* parens are essential here! *)
bazquux
val it = () : unit

- print "baz" ^ "quux\n";
stdIn:1.1-1.23 Error: operator and operand don't agree
[tycon mismatch]
  operator domain: string * string
  operand:         unit * string
  in expression:
    print "baz" ^ "quux\n"
```

Other String Operations

```
- String.size ("foo" ^ "bar");
val it = 6 : int

- String.substring ("abcdefg", 2, 3); (* string, start index, len *)
val it =       : string

("bar" < "foo", "bar" <= "foo", "bar" = "foo", "bar" > "foo");
val it = (true,true,false,false) : bool * bool * bool * bool

-(String.compare("bar", "foo"), String.compare("foo", "foo"),
= String.compare("foo", "bar"));
val it = (LESS,EQUAL,GREATER) : order * order * order

- String.size;
val it = fn : string -> int

- String.substring;
val it = fn : string * int * int -> string

- String.compare;
val it = fn : string * string -> order

(* An API for all SMLNJ String operations can be found at:
http://www.standardml.org/Basis/string.html *)
```

Characters

```
- #"a";
val it = #"a" : char

- String.sub ("foobar",0);
val it = #"f" : char

- String.sub ("foobar",5);
val it = #"r" : char

- String.sub ("foobar",6);
uncaught exception Subscript [subscript out of bounds]
  raised at: stdIn:17.1-17.11

- String.str #"a"; (* convert a char to a string *)
val it = "a" : string

- (String.str (String.sub ("ABCD",2))) ^ "S"
= ^ (Int.toString (112 + 123));
val it = "CS235" : string

- (1+2, 3=4, "foo" ^ "bar", String.sub("baz",2));
val it = (3,false, "foobar“ , #"z“ ) : int * bool * string * char
```

Pattern-matching Function Arguments

```
- fun swap (x,y) = (y, x);
val swap = fn : 'a * 'b -> 'b * 'a (* infers polymorphic type *)

- swap (1+2, 3=4);
val it = (false,3) : bool * int

- swap (swap (1+2, 3=4));
val it = (3,false) : int * bool

- swap ((1+2, 3=4), ("foo" ^ "bar", String.sub("baz",2)));
val it = (("foobar",#"z"),(3,false)) : (string * char) * (int *
bool)
```

How to Pass Multiple Arguments

```
- fun avg1 (x, y) = (x + y) div 2; (* Approach 1: use pairs *)
val avg1 = fn : int * int -> int

- avg1 (10,20);
val it = 15 : int

- fun avg2 x = (fn y => (x + y) div 2); (* Approach 2: currying *)
val avg2 = fn : int -> int -> int

- avg2 10 20;
val it = 15 : int

- fun avg3 x y = (x + y) div 2; (* Syntactic sugar for currying *)
val avg3 = fn : int -> int -> int

- avg3 10 20;
val it = 15 : int

- app5 (avg3 15);
val it = 10 : int

- app5 (fn i => avg1(15,i));
val it = 10 : int
```

Iterating via Tail Recursion

```
(* This is the contents of the file step.sml *)

fun step (a,b) = (a+b, a*b)

fun stepUntil ((a,b), limit) = (* no looping constructs in ML; *)
  if a >= limit then           (* use tail recursion instead! *)
    (a,b)
  else
    stepUntil (step(a,b), limit)
```

```
- use ("step.sml");
[opening step.sml]
val step = fn : int * int -> int * int
val stepUntil = fn : (int * int) * int -> int * int
val it = () : unit

- step (1,2);
val it = (3,2) : int * int

- step (step (1,2));
val it = (5,6) : int * int

- let val (x,y) = step (step (1,2)) in x*y end;
val it = 30 : int

- stepUntil ((1,2), 100);
val it = (371,13530) : int * int
```

Adding print statements

```
(* This is the contents of the file step-more.sml *)

fun printPair (a,b) =
  print ("(" ^ (Int.toString a) ^ ","
          ^ (Int.toString b) ^ ")\n")

fun stepUntilPrint ((a,b), limit) =
  if a >= limit then
    (a,b)
  else
    (printPair (a,b); (* here, semicolon sequences expressions *)
     stepUntilPrint (step(a,b), limit))
```

```
- use ("step-more.sml");
[opening step-more.sml]
val printPair = fn : int * int -> unit
val stepUntilPrint = fn : (int * int) * int -> int * int
val it = () : unit

- stepUntilPrint ((1,2),100);
(1,2)
(3,2)
(5,6)
(11,30)
(41,330)
val it = (371,13530) : int * int
```

How to exit SML interpreter?

```
Standard ML of New Jersey v110.78
[built: Wed Jan 14 12:52:09 2015]

- 1 + 2;
val it = 3 : int

-
```

Type Control-d at the SML prompt