Tutorial 5

Reference: Standard ML of New Jersey website: www.smlnj.org
Reference: Concept in Programming Lang, by John C. Mitchell
1 ML Environment

1. How do I start SML?
   On CDF just type `sml` and you will enter the SML environment:

   ```
   werewolf:\$ sml
   Standard ML of New Jersey, Version 110.0.7,
    September 28, 2000 [CM; autoload enabled]
   ```

   The `>` is the SML prompt. Now you can type in ML code and the interpreter immediately evaluates it and produces an output.

2. How do I quit SML?
   simply press Ctrl-D  (under Windows press Ctrl-Z)

3. How do I load my code from a file?
   The function `use` is defined at top level and will load a file containing SML source code.
   For example, loading your file `myfile.sml`:

   ```
   use "myfile.sml";
   [opening myfile.sml]
   val it = () : unit
   ```

   What is `val it = () : unit`? The function `use` returns `()` of type `unit`. We will see what this means later.

   If you are are loading a file that is in a directory different from the one you started `sml` in, then you need to provide a full path to the file.

   Note the semicolon at the end: to ML, it means “the user is done typing, now I should start working”.

4. How do I determine the current directory or change it?
   The function `OS.FileSys.getDir()` returns the current directory.
   The function `OS.FileSys.chDir("path")` sets the current directory to `path`.

5. What do I use for comments?
   Use `(* comments are here *)`. The comment can be multi-line.

6. SML is case sensitive:
   So you must match the case exactly (e.g. `getdir` would be an error!)
7. SML Interactive input:

Input to the top level interpreter (i.e., declarations and expressions) must be terminated by a semicolon (and carriage return) before the system will evaluate it. The system then prints out a response indicating the effect of the evaluation. Expressions are treated as implicit declarations of a standard variable it. For example,

- 3;  
  (* user input after prompt *)
val it = 3 : int  
  (* system response *)

This means that the value of the last top level expression evaluated can be referred to using the variable it. For example,

- 2.5;
val it = 2.5 : real
- it;
val it = 2.5 : real
-

8. Multi-line expressions:

If you type a long expression, you can enter it in several lines. In this case the prompt for subsequent lines changes to =. Note that you still need to end the expression with a semicolon.

- if 10 > 5
  = then 1
  = else 2;
val it = 1 : int

9. Interrupting compilation or execution:

Typing Ctrl-C should interrupt the interpreter and return you to top level (useful to interrupt infinite recursion).

10. Error messages:

When compiling files, the error messages include line numbers and character positions within the line. For example:

- if true
  = then 5 true
  = else 6;
std_in:7.6-7.11 Error: operator is not a function [literal]
  operator: int
  in expression:
    5 true
This means: there is an error that starts at line 7, character 6 and ends at line 7 character 11. And the error is: I am trying to apply 5 to true, and I can’t.

11. More Error Messages:

There are a number of different forms of type error message, and it may require some practice before you become adept at interpreting them. The most common form indicates a mismatch between the type of a function (or operator) and its argument (or operand). A representation of the offending expression is usually included, but this is an image of the internal abstract syntax for the expression and may differ significantly from the original source code.

12. A very useful reference for a list of all error messages.

All the error messages produced by SML/NJ are documented in the SML/NJ Error and Warning Messages page:

http://www.smlnj.org/doc/errors.html

13. Debugging in SML:

There is no debugger installed with SML/NJ. SML is a strongly typed language and its type inference system is powerful enough to capture a wide variety of errors and ambiguities. Furthermore, there is run-time checking so there’s no way you can crash the system (except for an infinite loop).
2 ML Quick Review: basics data types

1. **unit**: this type has only one element ()

   ```ml
   - ();
   val it = () : unit
   ```

2. **bool**: this type has two elements: true and false
   Operations on bools: not, andalso, orelse.
   For example:

   ```ml
   - if (not true andalso false) orelse true
     = then "foo"
     = else "bar";
   val it = "foo" : string
   ```

3. **integer**: {... -2, -1, 0, 1, 2, ... }
   Operations: +, -, *, div, mod, <, >, =, <=, >=, <>
   For example:

   ```ml
   - 5 + 6 * 2 - 3 div 2;
   val it = 16 : int
   - 5 mod 2 >= 6 mod 2;
   val it = true : bool
   ```

Note the use of `¬` for negation. Here's why: "-" is a binary operator while "¬" is a unary operator. For example:

```ml
- -5;
stdin:27.1 Error: expression or pattern begins with infix identifier "-"
stdin:27.1-27.3 Error: operator and operand don't agree [literal]
   operator domain: 'Z * 'Z
   operand:        int
   in expression:
   - 5
   - '¬5;
   val it = '¬5 : int
```

In the first case, SML complains that "¬" is a binary operator and we only provided one argument.
4. **real**: 1.0, 3.14159, 11.7, ...
   
   Operations: +, -, *, <, >, =, <=, >=, <>

   *Note*: You **cannot** mix reals and integers in one expression. For example:

   ```
   - 2 - 1.5;
   stdIn:43.1-43.8 Error: operator and operand don’t agree [literal]
   operator domain: int * int
   operand: int * real
   in expression: 2 - 1.5
   ```

   This means: there is an error between line 43, character 1 and line 43, character 8. The error is: operator and operand don’t agree, meaning the type of the operand provided to an operator is not what the operator expects. In this case: operator domain is two integers, but we gave it an integer and a real.

   Now, but wasn’t there an operator ",-" defined for reals as well? Yes, that's right. But for mathematical operators, if the type is not specified (or inferred), then type integer is assumed. For example:

   ```
   - 2;
   val it = 2 : int

   - 2.0;
   val it = 2.0 : real

   - real(2);
   val it = 2.0 : real

   - 2.0 - 1.5;
   val it = 0.5 : real

   - real(2) - 1.5;
   val it = 0.5 : real
   ```

5. **string**: "Hello, world", "Wow!", "CSC324 is fun!"

   Operations: ^ for concatenation

   For example:

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

   - "Hello" ^ " " ^ "Jim";
   val it = "Hello Jim" : string
   ```
6. **list**: [1, 5, 2, 3] [] is called **nil**, it is the empty list

Operations: ;, tl, hd

For example:

- [];
  val it = [] : 'a list

- [1, 2, ¬3];
  val it = [1, 2, ¬3] : int list

- "foo"::["bar"];
  val it = ["foo", "bar"] : string list

- hd [1,2];
  val it = 1 : int

- tl [1,2];
  val it = [2] : int list

Very important: all elements of the list **must** be of the same type. Consider the following:

- [1, 2, "Hello"];

  stdIn:17.1-17.16 Error: operator and operand don’t agree [literal]
  operator domain: int * int list
  operand: int * string list
  in expression:
    2 :: "Hello" :: nil

The error message means that there is an error between line 17, character 1 and line 17, character 16. Then it says that operator and operand don’t agree. But did we have any operators??? Yes!!!!

When we typed [1,2,"Hello"], we actually did 1::2::"Hello"::nil. And the error is: cannot do 2::"Hello"::nil, because 2 is an integer, and ["Hello"] is a list of strings!
7. **tuple**: (1, "Test", 2.5)  note that unit is an empty tuple
   Operations: accessing component N of a tuple Tuple: #N Tuple
   - (1, "Test", 2.5);
     val it = (1,"Test",2.5) : int * string * real
   - #2(1, "Test", 2.5);
     val it = "Test" : string

8. **record**: `{lastname="Smith", ID=200, Age=59}`
   Note: records are similar to tuple but their components have names, and the order
does not matter! In fact, a tuple is a special case of a record, where the names of
the fields are 1, 2, 3, ...
   Operations: accessing component Name of record Record: #Name Record
   - `{lastname="Smith", ID=200, Age=59};
     val it = {Age=59,ID=200.lastname="Smith"} : {Age:int, ID:int, lastname:string}
   - #lastname {lastname="Smith", ID=200, Age=59};
     val it = "Smith" : string

But what if I want my own variable? Not just this it that keeps changing? You can
declare your own:

   - val aNumber = 4*5;
     val aNumber = 20 : int
   - val aBiggerNumber = aNumber + 10;
     val aBiggerNumber = 30 : int