

CS 3721: Programming Languages Lab

Lab #06: Programming in ML

Due date: March 4, 3:30pm. At the beginning of the next recitation.

Goals of this lab:

- Getting used to ML and ML programming environment

ML is a statically typed (that is, all types of variables are predetermined at compile time) functional programming language. It is similar to Scheme/Lisp in supporting recursive functions and supporting function abstractions as first-class objects. At the same time, it also support statements and a wider variety of compound data structures including lists, tuples, records(similar to C struct), functions, and user-defined data types. In summary, ML supports both concepts from Lisp and concepts from C/Pascal.

Standard ML of New Jersey compiles into native code one function at a time, similar to a JIT compiler. To run a ML program interactively, you can start ML environment by typing

```
> sml
```

After the ML interactive environment starts, you can then type in ML statements and expressions. The expressions are compiled by the interactive environment as they are inputted. After you are done using ML, type “Ctrl-D” to exit the environment. If you have a ML program in a file, say myprogram.ml, you can also load the file and run your program using the *use* built-in function.

```
> use “myprogram.ml”;
```

The command above loads the file “myprogram.ml”, processes all the statements and expressions in the file, and then exits.

The syntax of ML is closer to C than to Scheme. In ML, expressions are written in conventional mathematical notations, and statements (definition to variables) terminate with semicolons (;). Note that ML interprets one statement at a time. So ML will not start interpreting your expression unless you place a ; at the end of an expression (which makes it a statement).

1. Following is an example ML program.

```
fun quick []      = []
  | quick [x]     = [x]
  | quick (a::bs) = (* This is a comment. The pivot is the head "a".*)
    let fun partition (left, right, []) : real list =
          (quick left) @ (a :: quick right)
        | partition (left, right, x::xs)           =
          if x<=a then partition (x::left, right, xs)
            else partition (left, x::right, xs)
        in partition([], [], bs) end;
```

Run this program in the ML environment with the following input:

```
val inpList = [8.7, 5.4, 9.4, 2.4, 1.2, 7.3, 0.3];
```

To call the function, type the following:

```
quick(inpList);
```

Write down the output, and make some comments about what the program is doing.

Solution:

Output:

```
val it = [0.3,1.2,2.4,5.4,7.3,8.7,9.4] : real list
```

Comment:

The quick function takes a real list data type as its argument, and it sorts the elements in the list in ascending order using quicksort algorithm. Note that the quick function uses the local *partition* function as a helper function.