CMSC 22610 | **Implementation of** | **Lab 0**
Winter 2009 | **Computer Languages - I** | **January 7, 2009**

Lab 0: A Constructive Introduction to ML Programming

Due: ∅

Welcome! By working through this lab you will gain experience programming in ML. If you have written a lot of ML already, this lab might not be worth your time, in which case, so be it. If, on the other hand, you have not yet produced any ML code, working through this exercise will give you good foundational experience.

1 Set up your subversion repository.

We will use a system called *gforge* to manage a subversion repository for each of you. Your *subversion repository* will contain your coursework (and store sequential versions of your work over time).

Your assignments will be automatically collected from your subversion repository by a script. Please note that the assignment-collection scripts, as is their nature, are cold, mechanical, heartless, somewhat obtuse entities. They can hear no excuses for lateness, nor can they tolerate any deviation from the path names we ask you to use. More concretely, if we ask you to save Lab 0 work in a directory called `username-proj/labs/lab0`, make sure your directory name is exactly that. Consider yourself forewarned.

If you haven't yet activated your gforge account, do so now by checking your uchicago email, clicking the appropriate link, and logging in with password `compilers`.

Please change your password right away.

Matthew or Adam will need to add you as a developer to your gforge project after you're active.

Somewhere in your CS account, create a directory to hold your CS 226 coursework. Navigate there. Check out your subversion repository with the following instruction:

```
svn co https://cs22610.cs.uchicago.edu/svn/username-proj
```

(Substitute your username for `username`, of course.) This will create a directory with the name `username-proj`.

You will create the following directory for this lab:

```
username-proj/labs/lab0
```

Note: You do this with `svn mkdir labs` from inside `username-proj`, then `svn mkdir lab0` inside `labs`.

2 Preliminary coding tips.

You are now ready to write some code. Here are some suggestions rooted in bitter experience.

- Raise exceptions where there is yet-to-be-written code. Raise expressions always pass the typechecker, so types never get in the way here.

In other words, from now on, you like this:

```
fun isMersennePrime n = raise Fail "todo"
```

and you dislike this:

```
fun isMersennePrime n = false
```

The former is just as easy as the latter, and almost incalculably better.

- Ensure your functions work individually on simple cases, including seemingly trivial ones, before using them in aggregate.
- Insert type ascriptions to track down bugs. More on this is the weeks to come.

3 The exercise: a multitude of sorts.

We will use *sorting* as our first problem area since it consists of a body of nontrivial algorithms already known to you.

3.1 Write a signature to specify your software.

Save the following code under the filename `sort.sig`.

```
signature SORT = sig
  type 'a order = ('a * 'a) -> bool
  val sort : 'a order -> 'a list -> 'a list
end
```

3.2 Write a CM file to manage compilation.

Save the following under the filename `sources.cm`.

```
Group
  signature SORT
is
  $/basis.cm
  sort.sig
```

Fire up an interactive session with sml and type

```
- CM.make "sources.cm";
```

to make sure your ducks are in a row. as it were.

3.3 Implement three sorts.

Implement three popular comparison sorting algorithms. We suggest insertion sort, mergesort, and quicksort. If you have other favorites, you may implement those instead, but assistance in the implementation of exotic sorting algorithms is not guaranteed!

Each sort should look more or less like this:

```
structure Mergesort : SORT = struct

  type 'a order = ('a * 'a) -> bool

  fun sort ...

end
```

Each sort stucture should live in its own file, with names like `mergesort.sml`. It is important to ascribe each sorting structure with the signature `SORT`. That ascription is done in the first line above, where `Mergesort` is followed by a colon and the signature name `SORT`. Having made this signature ascription, the compiler will check that the structure provides whatever the signature says it must. (Signatures are similar to Java's interfaces.)

Note the convention among ML programs is to name signatures in all capitals (`SORT`), and to capitalize structure names (`Mergesort`).

3.4 Write a functor to drive your sorts.

A functor is a structure that is paramaterized over other structures. It is in some ways a higher-order structure. This may be more clearly understood by making one than by reading a description of one.

Create a file `try-fn.sml` and type this text into it:

```
functor TryFn (S : SORT) = struct

  val data = [5,0,4,2,1,3]

  fun main () = let
    val up   = raise Fail "todo" (* code to sort data ascending  *)
    val down = raise Fail "todo" (* code to sort data descending *)
    in
      (up, down)
    end

end
```

Replace the two exception-raisings with working code. Inside the functor, you can call the `sort` function of the structure `S` by using the qualified name `S.sort`.

(By convention, functor names often end with `Fn`.)

3.5 Use your functor.

If you haven't added all your files to the `sources.cm` file, do so now. Then interact with sml as follows:

```
- CM.make "sources.cm";
  ... (* you may have some type errors to deal with here! *)
- structure I = TryFn(InsertionSort);
  ...
- structure M = TryFn(Mergesort);
  ...
- structure Q = TryFn(Quicksort);
  ...
```

You can now try any sort, say `Q`, by typing `Q.main ()`.

4 You’re done.

Mazel tov! You’re finished. For the sake of practice, you should add all your files to your subversion repository, and check them all in.

To do so, type

```
> svn add sources.cm
> svn add sort.sig
  ...
```

and so on. Shell hackers might write a for loop at the prompt to do this.

Having added each file to the repository, you can check them in using

```
> svn commit
```

or `svn ci` for short.

(Google "subversion manual" for Subversion’s documentation, which is good. You may want to bookmark it for future reference.)

5 Errata

Revision 1 (Jan 8)

- `svn co https://svn.cs22610.cs.uchicago.edu/svn/username-proj` ⇒ `svn co https://cs22610.cs.uchicago.edu/svn/username-proj`
- `TryFn{Mergesort}` ⇒ `TryFn(Mergesort)`
- `svn checkin` ⇒ `svn commit`