COMS W4115
Programming Languages and Translators
Homework Assignment 1

Prof. Stephen A. Edwards
Columbia University

Due February 23rd, 2011
at 11:59 PM

On-campus students: submit your solutions on paper, not electronically (e.g., print them out).

CVN students: FAX the solutions to CVN.

Include your name and your Columbia ID (e.g., se2007).

Do this assignment alone. You may consult the instructor or a TA, but not other students.

All the problems ask you to use O'CAML. You may download the compiler from caml.inria.fr.

1. In O'Caml, write a function "uniq" that takes a list and returns the same list with adjacent duplicate entries removed. Show that for the list `[1;1;1;3;4;1;1]` your function returns the list `[1;3;4;1]`. Hint: my solution is a six-line, three-way case split.

2. Write a word frequency counter. Here is a starting point: an ocamllex program (wordcount.mll) that gathers in a list of strings all the words in a file, then prints them.

```
{ type token = EOF | Word of string }

rule token = parse
  | eof { EOF }
  | ['a'-'z' 'A'-'Z']+ as word { Word(word) }
  | _ { token lexbuf }

{
 let lexbuf = Lexing.from_channel stdin in
 let wordlist =
   let rec next l =
     match token lexbuf with
         EOF -> l
     | Word(s) -> next (s :: l)
   in next []
 in
 List.iter print_endline wordlist
}
```

   Instead of List.iter, write code that scans through the list and builds a string map whose keys are words and whose values are the number of times a string was found, then uses StringMap.fold to convert this to a list of (count, word) tuples, sorts them using List.sort, and prints them with List.iter.

   Sort the list of (count, word) pairs using

```
 let wordcounts =
   List.sort (fun (c1, _) (c2, _) ->
                Pervasives.compare c2 c1)
     wordcounts in
```

   Compiling and running my (20-more-line) solution:

```
$ ocamllex wordcount.mll
4 states, 315 transitions, table size 1284 bytes

$ ocamlc -o wordcount wordcount.ml

$ ./wordcount < wordcount.mll

9 word
7 map
7 let
7 StringMap
6 in
...
```

3. Extend the three-slide "calculator" example shown at the end of the Introduction to O'Caml slides (the source is also available on the class website) to accept the variables named $0 through $9, assignment to those variables, and sequencing using the "," operator. For example,

```
$1 = 3, $2 = 6, $1 * $2 + 2
```

   should print "20"

   Use an array of length 10 initialized to all zeros to store the values of the variables. You'll need to add tokens to the parser and scanner for representing assignment, sequencing, and variable names.

   The ocamllex rule for the variable names, which converts the numerals 0–9 into the corresponding literals, is

```
| '$'['0'-'9'] as lit
  { VARIABLE(int_of_char lit.[1] - 48) }
```

   The new ast.mli file is

```
type operator = Add | Sub | Mul | Div
type expr =
    Binop of expr * operator * expr
  | Lit of int
  | Seq of expr * expr
  | Asn of int * expr
  | Var of int
```

   My solution required adding just 20 lines of code across the four files.