COMS W4115
Programming Languages and Translators
Homework Assignment 2

Prof. Stephen A. Edwards
Columbia University

Due October 27th, 2017
at 12:00 PM

Submit your assignment on paper (e.g., printouts) at the beginning of class. **Include a demonstration of your code working on some examples.** Hybrid section students may slip their assignments under my door.

Do this assignment alone. You may consult the instructor or a TA, but not other students. All the problems ask you to use OCaml. You may download the compiler from ocaml.org.

1. Write an OCaml function `all` that, given a function and a list, returns *true* if applying the function to every element of the list returns *true* and *false* otherwise.

```
all (fun x -> x) [] = true
all (fun x -> false) [1] = false
all (fun x -> x mod 2 == 0) [2;4;6] = true
all (fun x -> x mod 2 == 0) [2;5;6] = false
```

2. Write a word frequency counter. Start from the following 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
}
```

Replace the List.iter call with code that scans through the list and builds a string map whose keys are words and whose values count the number of apearances of each word. Then, use StringMap.fold to convert this to a list of (count, word) tuples; sort them using List.sort; and print 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 OCaml slides (the source is also available on the class website) to accept the variables named a through z, assignment to those variables, and sequencing using the ";" operator. For example,

```
a = 3; c = b = 6; a * b + c
```

should print "24"

Use an array of length 26 initialized to all zeros to store the values of the variables. Add tokens to the parser and scanner for representing assignment, sequencing, and variable names.

The ocamllex rule for the variable names, which converts the letters a–z into the corresponding literals, is

```
| ['a'-'z'] as lit
  { VARIABLE(int_of_char lit - 97) }
```

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.

Make sure your code compiles without warnings