COMS W4115
Programming Languages and Translators
Homework Assignment 1

Prof. Stephen A. Edwards
Columbia University

Due October 1st, 2018
at 4:00 PM

Submit your assignment as a single PDF file on Courseworks. **Include a demonstration of your code working on examples**, e.g., by including a screenshot of your code compiling and working.

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 `maxrun` that reports the length of the longest contiguous run of equal values in a list. E.g.,

```
val maxrun : 'a list -> int = <fun>
# maxrun [];;
- : int = 0
# maxrun [1];;
- : int = 1
# maxrun [1;1];;
- : int = 2
# maxrun [1;1;2;2;2;1;3;3];;
- : int = 3
```

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 in the OCaml slides (the source is also available on the class website) to accept variables named with identifiers consisting of lowercase letters, assignment to those variables, and sequencing using the ";" operator. For example,

```
foo = 3; bar = baz = 6; foo * bar + baz
```

should print "24"

Use a string-to-integer Map to track variable 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 id { VARIABLE(id) }
```

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 string * expr
  | Var of string
```

Make sure your code compiles without warnings