CS 241 — Winter 2013 — Assignment 6 — The WLPP Programming Language and Scanning

This assignment is the first in a series of five that involve learning WLPP (short for *WaterLanguage Plus Pointers*) and writing a compiler that translates WLPP to MIPS assembly language.

Please consult WLPP Programming Language Tutorial for an informal explanation of WLPP; The WLPP Language Specification should be consulted for the definitive specification of the WLPP language.

All solutions must be submitted to Marmoset. Your submissions for A6P1 and for A6P2 should each be a file containing a complete WLPP program. Your submission for A6P3 should be a DFA in the format used in A5. Your submission for A6P4 should be a Scheme, C++, or Java program that implements a scanner for WLPP.

Due date: Thursday, February 28, 8:00 p.m.

The following WLPP program computes the sum of two integers, a and b.

```
//
// WLPP program with two integer parameters, a and b
//   returns the sum of a and b
//
   int wain(int a, int b) {
      return a + b;   // unhelpful comment about summing a and b
   }
```

You may test this program on the student.cs environment by placing it in a file named `test.wlpp` and entering the following commands, which compile it to MIPS machine language and run it with the familiar `mips.twoints` command from A1 and A2:

```
java cs241.wlppc < test.wlpp > test.mips
java mips.twoints test.mips
```

A6P1 through A6P3 familiarize you with the WLPP programming language. Beginning with A6P4 and through the next several assignments, you will be writing a compiler that translates WLPP into MIPS assembly language.

Problem A6P1 (10 marks of 60) (filename: `gcd.wlpp`)

Write a WLPP program that takes two parameters, x and y, and returns the greatest common divisor gcd(x,y). You may assume that $0 < x,y < 2^{31}$. Hint: http://www.google.ca/search?q=euclidean+algorithm

Problem A6P2 (10 marks of 60) (filename: `sum.wlpp`)

Write a WLPP program that takes two parameters a and n, where a is an array of integers, and n is the number of elements in a. Your program should produce the sum of the integers in a.

Problem A6P3 (20 of 60 marks) (filename: `wlpp.dfa`)

Using the DFA description file format described in assignment 5, create a deterministic finite automaton that recognizes any valid WLPP token, from the list given in the WLPP specification. The alphabet consists of every character that may appear in any token; this does **not** include white space characters.

[While a *lexical scanner* for WLPP must allow WLPP tokens to be separated by white space so that it may distinguish between two consequtive tokens (eg 42 and 17) whose concatenation would constitute a single token (eg 4217), a DFA that accepts only input comprising a *single* token has no such need.]

Hint: WLPP tokens include keywords (such as int, wain, return, etc.), as well as identifiers. When scanning WLPP, two approaches are equally valid. One approach is to distinguish between keywords within the DFA, using separate states for each keyword. Another approach is to design a DFA that just recognizes identifiers and keywords the same way, and write additional code outside of the DFA to determine whether the token is a specific keyword or an identifier. For this problem and for Problem 4, you may choose either approach.

Problem A6P4 (20 of 60 marks) (filename: `wlppscan.ss` or `wlppscan.cc` or `WLPPScan.java`)

Write a scanner (lexical analyzer) for the WLPP programming language. Your scanner should read an entire WLPP program and identify the tokens, in the order they appear in the input. For each token, your scanner should compute the name of the token and the lexeme (the string of characters making up the token), and print it to standard output, one line per token.

For example, the correct output for the example WLPP program above is:

```
INT int
WAIN wain
LPAREN (
INT int
ID a
COMMA ,
INT int
ID b
RPAREN )
LBRACE {
RETURN return
ID a
PLUS +
ID b
SEMI ;
RBRACE }
```

If the input cannot be scanned as a sequence of valid tokens (possibly separated by white space as detailed in the WLPP language specification), your program should produce exactly one line of output to standard error containing the string `ERROR`, in upper case. We recommend that you try to make the error message informative.

A reference implementation of the scanner called cs241.WLPPScan is available after running `source /u/cs241/setup`. When testing your program, you can compare its output to cs241.WLPPScan's to make sure it is correct. Run it as follows:

```
java cs241.WLPPScan < input.wlpp > tokens.txt
```

You may wish to modify asm.ss, asm.cc, or Asm.java to solve this problem. These are the scanners that were given to you for assignment 3. Note that the lexer in asm.ss, asm.cc, and Asm.java is based on a DFA that repeatedly recognizes the longest prefix of a string that is a token. You should be able to replace the DFA (which recognizes MIPS assembly language tokens) by one which recognizes WLPP tokens (see Problem 3 above).

For this problem only, all of the Marmoset tests are public tests. There are no release tests or blind tests.