SIMDC12 Lexer

February 7, 2012

SIMDC12 is a small dialect of the C programming language created specifically for this Spring 2012 course. It is a simple C dialect using SIMD (Single Instruction Stream, Multiple Data Stream) semantics and incorporating a few SIMD-oriented extensions. In flavor, it is very similar to MPL, the MasPar Programming Language used for the MP1 and MP2 SIMD supercomputers. However, you will be compiling this code for a different target.

1 Lexemes

SIMDC12 is based on C. Consequently, it follows C's syntax and semantics fairly closely. This means that the rules for handling of white space, characters in an identier, etc. are the same except as noted here. An identifier begins with an alphabetic character, and is optionally followed by any number of alphabetic or numeric characters; the underscore character (_) is considered to be alphabetic. Case is signicant; a is not thesame as A. It is allowable to ignore characters after the first 7 in an identier. The SIMDC keywords all would be identiers if they were not keywords.

The SIMDC keywords all would be identiers if they were not keywords. They are:

```
mono
poly
if
where
else
while
return
IPROC
NPROC
```

All keywords except the last two are lowercase.

Your lexer also must recognize decimal numbers, which look like a sequence of one or more digits. In C, a leading 0 digit is used to indicate an octal value, 0 is the octal value zero. so, technically, However, SIMDC doesn't have octal constants, so leading 0s do not change the base to octal. There are not too

many special symbols treated as tokens in SIMDC, and none of them is more than one character long. The special symbols you must recognize are:

```
[ ] ( ) , ; : { } & | ^ < > + - * / % = ! ~ .
```

Note that end-of-file, EOF, also behaves somewhat like a special symbol.

2 Structure Of This Project

Dierent kinds of lexical analyzers have been discussed; this project will involve writing a simple lexical analyzer and minimal symbol table interface. The solution must be in the form of a C program which you will submit directly via a WWW form (without a paper copy)

It must be your own work, and you are not permitted to use tools such as lex, yacc, or PCCTS to construct the C code. Your symbol table may use linear search or whatever method you wish, but again, it must be your own code.

Your program is to be written in a style that facilitates reuse of the lexer in a parser. Toward this goal, the `main()` must call a function `lex()` to return each token's type until an EOF token is returned. The `main()` should print, on a separate line for each lexical item, the type and lexeme of the token just read. The formatting should be:

- For any identifier, output the lexeme followed by the number of previous times this lexeme has been seen; this number should be recorded in the symbol table entry created by the lexer
- For any of the keywords listed, simply output the keyword's lexeme
- For any decimal number, output the value of that number using fprintf(stderr, "%d", ...)
- For any sepcial symbol, simply output the symbol; the sole exception is the end-of-file, which should output "EOF"

The lexer should not return a token for any other characters; lex() should essentially ignore the other symbols except in the sense of treating them as "edges" between tokens – like whitespace. It is generally easiest to do this by having the lexer advance past other characters before recognizing the current token.

3 Submission

See the course WWW site, `http://aggregate.org/STCH/`, for details.