CS210 - Programming Languages
Homework #3 - Fall 2020

Modern language translation systems often employ tools that automatically generate scanners. One such tool is `lex`. Lex is sometimes known as `flex` in certain Linux distributions. Lex can automatically generate the C code for a scanner when given a proper *lex specification*. By default, lex places the generated scanner in a file called `lex.yy.c`.

A lex specification is a set of rules. Each rule in a lex specification is composed of a *pattern* and an *action*. The pattern in a rule uses UNIX-style *regular expressions* to specify the set of lexemes that match a given rule. The action in a rule specifies what is to be done as a result of matching the pattern of a particular rule. When you write a lex specification, you create a set of patterns that lex matches against the input to the generated scanner. Each time one of the patterns matches, the scanner generated by lex invokes an action associated with a specific rule contained in the lex specification.

The scanner generated by lex has as its interface a single function called `yylex`. In order to use the scanner generated by lex, a call to the `yylex()` function must be made. The scanner generated by lex gets its input from a stream called `yyin`. If you examine the `lex.yy.c` file generated by lex you will find that the declaration for this stream is of the form

```
extern FILE * yyin;
```

and typically this stream is attached to a given input file or *standard input* (keyboard.)

For this homework assignment you are to use lex to build a scanner that satisfies the same requirements that you were given for homework 2. In homework 2 you were required to build the scanner in C "by hand." For this assignment you are to build the same scanner using lex. Your scanner must satisfactorily handle the same input files that were given in homework 2. The lexeme classification specified in homework 2 is the same classification that is to be used for this homework.

Getting Started

On the course website you will find a link to a file called `ccx_start.l`. This file contains the beginnings of a lex specification that will satisfy the requirements for this homework. Please download this file, examine it, and feel free to use it as a starting point for your lex specification.

Although the `ccx_start.l` file is in nascent form, you can build an executable from the file as-is by using the commands

```
-bash-4.1$ lex ccx_start.l
-bash-4.1$ gcc lex.yy.c -o ccx_start
```

which will create an executable called `ccx_start`. You can test this executable by providing it input in a file, or just by typing input from the keyboard.

The ccx_start.l file contains comments designed to familiarize the reader with the structure and meaning of a lex specification, but you will want to seek out other references describing lex. One good reference is found at http://dinosaur.compilertools.net and there are many, many others. If you prefer textbooks, "Lex & Yacc" by Levine, Mason, and Brown (O'Reilly & Associates, Inc., 1995) is widely held in high regard.

Regular Expressions

As mentioned previously, a lex specification uses UNIX-style regular expressions to specify patterns that are used to match input to the generated scanner. A regular expression is a pattern description using a "meta-language." The characters used in this meta-language are part of the standard ASCII character set, which may lead to confusion. The characters that are used in regular expressions are:

```
.   Matches any single character except newline (\n).

*   Matches zero or more instances of the preceding expression.

[]  A character class that matches any character within the brackets.  If the
    first character is a circumflex (^) it changes the meaning to match any
    character except the ones within the brackets.  A dash inside the brackets
    indicates a character range. For instance, "[0-9]" means the same thing as
    "[0123456789]". C-style escape sequences starting with \ are recognized in
    character classes.

^   Matches the beginning of a line as the first character of a regular
    expression.  Also used for negation within character classes.

$   Matches the end of a line as the last character of a regular expression.

{}  Indicates how many times the previous pattern is allowed to match when
    containing one or two numbers. For instance, A{1,3} matches one to three
    occurrences of the letter A.

\   Used to escape meta-characters, and as part of the usual C-style escape
    sequences. For instance, "\n" is a newline, and "\*" is a literal
    asterisk.

+   Matches one or more instances of the preceding regular expression. For
    instance "[0-9]+" matches "1", "22", "19", or "12345", but not the empty
    string.

?   Matches zero or one instance of the preceding regular expression. For
    example, "-?[0-9]+" matches a signed integer including an optional
    leading minus (negation.)

|   Matches either the preceding regular expression or the regular expression
    that follows. For example, "cow | pig | sheep" matches any of the 3 words.
```

```
"..." Interprets everything within the quotation marks literally. Meta-
    characters other than C-style escape sequences lose their meaning.

/   Matches the preceding regular expression but only if followed by the
    trailing regular expression. For example, "0/1" matches 0 in the string
    "01" but will not match the 0 in the string "02".

()  Groups a series of regular expressions together to form a new one.
```

There are many, many references on regular expressions available. Any good lex reference will typically discuss regular expressions since they are used to build a lex specification.

Submitting Your Homework

Your homework must be submitted using the `checkin` program. The deliverable for this homework is a single file called "ccx.l" that contains the lex specification that you created to automatically generate your scanner. You must turn in **only** your lex specification, and it must be in a single file named as specified above.