CPEG 421/621 - Homework 0
Issue Date: Tuesday, 14 February, 2012
Due Date: Tuesday, 6 March, 2012

The goal of this homework is to acquaint yourself with two very powerful tools for the generation of compilers: *lex* and *yacc*. When a program is presented to a compiler, the compiler has to divide the program into meaningful units, called *tokens*, and then discover the relationship among these units. The algorithm that divides the program into units is called a *lexical analyzer*, *scanner*, or *lexer*.

Once the tokens are identified, the compiler needs to find the expressions, statements, declarations, blocks, and procedures in the program. This task is the function of a *parser*. The parser requires a list of rules that define the relationships among the tokens. This list of rules is called a *grammar*.

Lex is a program that takes a set of descriptions of possible tokens and produces a C routine that implements a scanner. *Yacc* takes a concise description of a grammar and produces a C routine that implements a parser for the grammar.

Lex and yacc are tools which can be used to solve a variety of interesting problems, even outside the realm of compilers. We are sure that once you understand how they function, you will find that the use of lex and yacc will save you many programming hours. It is strongly recommend that you read pages 1-17 of the Lex & Yacc Tutorial by Tom Nieman to benefit the most from this exercise.

Exercise 0:

For this assignment you can use any Linux machine that has lex and yacc (or flex and bison) installed. Some servers that you may use are as follows:

- mlb.acad.ece.udel.edu
- strauss.udel.edu

You may use a Windows machine provided that you install *MobaXterm* along with the *Gcc, G++ and development tools* plugin for it. See:

- `http://mobaxterm.mobatek.net`
- `http://mobaxterm.mobatek.net/plugins.html`.

You may use a Mac OS X machine provided that you install *Xcode*. See:

- `https://developer.apple.com/xcode/`

Tasks

All the files needed for this assignment can be found at the following URL:

- `http://www.capsl.udel.edu/courses/cpeg421/2012/homework/hw0.tar`

In this tarball, there is a lex specification *scan.l*. You can build a standalone scanner using this file to see how the scanner works. Follow these instructions to build the scanner:

1. Extract files from the tarball.

   $ tar -xvf hw0.tar

2. Invoke this command to convert a lex specification file into a C source file.

   $ lex scan.l

3. Now you must compile this C source file into an executable program using the following command:

   $ gcc -o scanner lex.yy.c

4. Now you can run the executable scanner and play with it by typing in text and checking if it is recognized as a token.

   $ ./scanner

We recommend that you consult the scan.l file as you play with the scanner to understand how the tokens are specified. Make sure you try operators, words, parenthesis, etc. The scanner will output the type of symbol that it regonizes for the input that you type. You can use CTRL-D to terminate the scanner.

Turn in: You do not have to turn in anything specifically for this exercise. However, this exercise has to be done in order to do the next exercise.

Exercise 1:

For this exercise the files *prefix.l* and *prefix.y* will be used. The file prefix.l is the lex file written to match the patterns that are required to generate the tokens needed to implement a simple calculator. The file prefix.y described the rules of the grammar that implement the calculator. First you should experiment playing with the calculator. Make sure you run the *make clean* command to remove older versions of the files then type *make*:

$ make clean
$ make

You now have a prefix calculator. You invoke the calculator by running the program *"./prefix"*. If you type "+ 3 2" the calculator answers with "= 5"; if you type "x = 4" followed by "* 3 x" the calculator responds with "= 12". Here is a demonstration:

$ ./prefix
+ 3 2
= 5
x = 4
*** 3 x**
= 12

Tasks

Now your should play with the calculator and check which operations execute and which do not. You should be able to do addition, subtraction, multiplication, division, as well as utilize variable names. We suggest you look at the prefix.l and prefix.y files to try to understand how the calculator was specified.

Your task is to create your own yacc file, based on the prefix.y file provided that makes the calculator operate in infix mode. I.e. instead of typing:

+ 2 2

in a infix calculator you type:

2 + 2

You must also create a yacc file to make the calculator operate in postfix mode. I.e. instead of typing:

+ 2 2

in a postfix calculator you type:

2 2 +

Turn in: You do not have to turn in anything specifically for this exercise. However, this exercise has to be done in order to do the next exercise.

Exercise 2

Now you should have three yacc files (prefix.y, infix.y, and postfix.y) that implement three working calculators. Modify the three calculators to each support the operations listed below. Make sure the operators follow the rules explained above for each calculator type. I.e. for the prefix calculator the operators should precede the operands; for the postfix calculator the operators should follow the operands; and for the infix calculator the operators should be inbetween the operands.

- Binary bitwise operators AND, OR, and XOR. Use the following symbols: **&**, **|**, and **^** respectively for the operators.
- Unary bitwise operator NOT. Use the following symbol: **~**. For postfix notation, the operator should follow the operand. E.g. "**5~**". For prefix and infix notation, the operator should precede the operand.
- Left logical shift and right logical shift. Use the following symbols: **<<**, **>>** respectively for the operators.
- The power operator (e.g 2 to the power of 4). Use the following symbol: ******.

Note: The parser should not recognize symbols consisting of two characters with spaces inbetween them as valid (e.g. < <).

Extra Credit: What would you need to do to support a negation symbol (e.g. negative numbers)? Note: for this question, please just write your answer in words.

Turn in: Your lex and yacc files (and extra credit) to `aron+ta@udel.edu` and `szuckerm@eecis.udel.edu`. Please verify that you haven't forgotten any files.