CSC-235 Problem Set 3

Assigned: Mar 1 2023

Due: Mar 15 2023

In this homework, you'll get some experience using `flex` and `bison` to generate lexers and parsers (syntax analyzers). `flex` takes your regular expressions and creates the rest of the C program that scans an input stream and tokenizes it (as we'll see, `flex` can do a lot of other stuff while it's tokenizing).

Once the stream is tokenized, the parser or syntax analyzer takes over. Just like `flex` generates a lexical scanner or lexer based on our regexes, `bison` generates the parser based on a grammar we give it.

Part 1

In this folder, you will find several files, including `scan.l`. The scanner it generates can be run interactively or in batch mode. In interactive mode, you type sentences from the language and watch it tokenize. In batch mode, it will tokenize the contents of a file instead.

1. Make a terminal in the PS03 folder.
2. Go `flex scan.l`
3. Type `ls` to see the new file `flex` created, named `lex.yy.c`. You can look in there, but it may not be too enlightening. Don't change anything.
4. Now you must compile the generated C code:
5. `gcc -o scanner lex.yy.c`
6. Run the scanner: `./scanner`
7. To exit the scanner hit Ctrl+D (even on Mac)

You should consult the `scan.l` file as you play with the scanner to see how the tokens are specified. Try numbers, letters, operators, parenthesis, white space, and so on.

You don't have to turn anything in for Part 1.

Part 2

For this part, you will use the files `prefix.l` and `prefix.y`. The `prefix.l` file is the lex file for the patterns needed by a simple calculator. The grammar rules are in `prefix.y`. To play with the calculator, you will use the Makefile provided to do the compilation. Go

1. `make clean`
2. `make`

to compile the calculator. Run it with `./prefix`

Now if you type `+ 3 2` you should get `= 5`. If you type `x = 4` followed by `* 3 x` you should get `= 12`.

Now play with the calculator and check which operators are defined. You should at least have addition, subtraction, multiplication, and division. Look at the `prefix.l` and prefix.y files to see how things are specified.

Tasks

1. Create files `infix.y` that make the calculator operate in infix mode. You will need to modify the Makefile to get them included in the compilation. I want you to experiment and try to figure it out, but I will definitely help if you get too stuck. There's way more here than we could possibly cover in detail.
2. Same as the last, but create `postfix.y` to implement a postfix calculator.

Make copies of your files named `postfix.y.part2` and `infix.y.part2` to turn in, since you will be modifying all three further in part 3.

Part 3

For this part you need all three calculators from Part 2. Implement the following operations:

1. Binary bitwise operators AND, OR, XOR (use the symbols &, |, ^)
2. Unary bitwise NOT (use ~). For postfix, this operator follows the operand. For infix and prefix, it precedes it.
3. Left logical shift and right logical shift (ask a classmate or google them if you aren't sure). Use »for right shift and «for left.
4. The power operator. Use the symbol **.

The parser should NOT recognize symbols with a space between as valid.

Leave your modified calculator files in cocalc to turn them in.