Programming Systems | 2020 Fall

HW 2 | Due on 10/30/2020 11:59 PM

The goal of this homework is to acquaint yourself with two very powerful tools for the generation of compilers: Lex and Yacc. Lex (or the gnu version which is called flex) 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. In this homework, you are expected to design a scanner and parse for a calculator.

# Warm-up

* 1. Setup your development environment. This homework is designed for Linux operating systems. You can develop in either a physical Linux distribution (it could be your own PC or the Apollo servers) or a virtual machine.
  2. Run a test example. You can find test.l and test.y in the attached zip file. Please use the following commands to test your development environment.

$ flex test.l  
$ bison -d -y test.y  
$ cc lex.yy.c y.tab.c -o test  
$ ./test

The attached test files are designed to detect expressions like “1+2”. If you input a valid expression, it returns “EXPRESSION”. Otherwise, it returns “syntax error” since the grammar is not defined.

Please note that there are warnings when you compile the generated scanner/parser using “cc lex.yy.c y.tab.c -o test”. It is normal.

# Design a scanner/parser using flex and bison

We will implement a calculator whose input is an arithmetic expression and output is the evaluation result of this expression.

To simplify the settings, files are provided, and you are expected to add some snippets of code to assemble a compiler. **The calculator is designed to only support integer and four operators (+, -, \*, /).**

* 1. Add regular expressions in the flex file. (10 pts)

The flex file tokenizes the input using regular expression to identify patterns and execute actions. Please open calc.l using a text editor, and add a regular expression (to replace REGULAR\_EXPRESSION\_HERE) in Line 8 that **match integers**. The actions are already filled in the braces. Please enter your regular expression into calc.l and write again in the following blank. Please make sure your answers are consistent.

* 1. Add grammars in the bison file.

The bison file describes the rules of a grammar in which way the expressions are executed. There are plenty of implementations of a specific task. In this homework, the grammar is given, and you are expected to translate the grammar into bison rules. The grammar is shown as below:

|  |  |
| --- | --- |
| EXPR: | EXPR“+”EXPR |
|  | |EXPR“-”EXPR |
|  | |EXPR“\*”EXPR |
|  | |EXPR“/”EXPR  |INTEGER |

Please translate the grammar into bison rules and fill the blanks for RULE\_1/2/3/4 in Line 11-14. For the ease of grading, please paste your answers in the following blanks. Please note that the corresponding actions are already filled. Please note the whitespaces among strings when you design rules.

# Generate the scanner and parser.

flex and bison can generate the scanner and parser for calculator according to our specification. The generated files are in the format of C. So you are expected to compile the files and obtain executables.

The generate commands are:

$ flex calc.l  
$ bison -d -y calc.y

After generation, you would find several C files, e.g. lex.yy.c and y.tab.c, which are the scanner and parser.

The compilation command is:

$ cc lex.yy.c y.tab.c -o calc

The executable command is

$ ./calc

Then you can type expressions and obtain the result immediately.

# Test Examples

If the regular expression and rules are correctly specified, the executable can pass the following test examples. Please paste the screenshots when you test your executable. (In the following test examples, “$” represents the beginning of commands. You should enter the strings after “$” when you test.)

* 1. $ 10+2
  2. $ 10-2
  3. $ 10\*2
  4. $ 10/2

# Grading criteria

* 1. Pack and compress your report (this file), calc.l, and calc.y as a zip file and rename as HW2\_Firstname\_Lastname.zip, e.g. HW2\_Alice\_Brown.zip. (10 pts)
  2. The submission can be successfully uncompressed. (5 pts)
  3. The files can be complied without errors (warnings are acceptable). (25 pts )
  4. The regular express (for integer) in calc.l is correct in this report (based on your answers in ) and the lex/bison files. (5 pts)
  5. The rules in calc.y are correct. (15 pts)
  6. The executable can correctly calculate four types of expression (based on the screenshots in Section 4). (10 pts for each)