Lex and Yacc Tutorial

Ming-Hwa Wang, Ph.D
COEN 259 Compilers
Department of Computer Engineering
Santa Clara University

Lex

Lex is a scanner generator tool for lexical analysis, which is based on finite state machine (FSM). The input is a set of regular expressions, and the output is the code to implement the scanner according to the input rules.

To implement a scanner for calculator, we can write the file "cal1.l" as below:

```
/* this is only for scanner, not link with parser yet */

%%

\(          { printf("(\n"); }
\)          { printf(")\n"); }
\+          { printf("+\n"); }
\*          { printf("*\n"); }
[ \t\n]+    ;
[0-9]+      { printf("%s\n", yytext); }

%%

int yywrap() {
    return 1;
}

int main () {
    yylex();
    return 1;
}
```

Here is the Makefile used to build the scanner:

```
p1: lex.yy.o
    gcc -o p1 lex.yy.o

lex.yy.o: cal1.l
    lex cal1.l; gcc -c lex.yy.c

clean:
    rm -f p1 *.o lex.yy.c
```

Note: for more complex lex input file, you might get an error message like `"parse tree too big, try %a num (or %e num)"`

Then you need to define %e <num>. You should put it between macro and %start symbol. The other options are %a, %o, %n, %p, etc.

Yacc

Yacc is a LALR(1) parser generator tool for syntax analysis, which is based on pushdown automata (PDA). The input is a set of context-free grammar (CFG) rules, and the output is the code to implement the parser according to the input rules.

To implement a parser for calculator, we can write the file "cal.y" as below:

```
%token NUMBER
%token '(' ')'
%left '+'
%left '*'
%union {
    int val;
    int line;
}

%start cal

%%

cal
  : exp
    { printf("The result is %d\n", $1.val); }
  ;

exp
  : exp '+' factor
    { $$.val = $1.val + $3.val; }
  | factor
    { $$.val = $1.val; }
  ;

factor
  : factor '*' term
    { $$.val = $1.val * $3.val; }
  | term
    { $$.val = $1.val; }
  ;

term
  : NUMBER
    { $$.val = $1.val; }
  | '(' exp ')'
    { $$.val = $2.val; }
  ;

%%
```

```
#include <stdio.h>
#include <ctype.h>
int lineNum = 1;

yyerror(char *ps) {     /* need this to avoid link problem */
   printf("%s\n", ps);
}

int main() {
   yyparse();
   return 0;
}
```

To integrate both the scanner and parser, we need to modify the scanner input file "cal1.l" and save it as "cal.l" as below:

```
%{
#include <stdlib.h> /* for atoi call */
#define NUMBER 257  /* copy this from y.tab.c */
#define DEBUG /* for debuging: print tokens & line numbers */
typedef union {     /* copy this from y.tab.c */
   int val;
   int line;
} YYSTYPE;
extern YYSTYPE yylval; /* for passing value to parser */
extern int lineNum; /* line number from y.tab.c */
%}


%%

[ \t]+ {}
[\n]   { lineNum++; }
"(" {
#ifdef DEBUG
       printf("token '(' at line %d\n", lineNum);
#endif
       return '(';
    }
")" {
#ifdef DEBUG
       printf("token ')' at line %d\n", lineNum);
#endif
       return ')';
    }
"+" {
#ifdef DEBUG
       printf("token '+' at line %d\n", lineNum);
#endif
       return '+';
    }
"*" {
#ifdef DEBUG
       printf("token '*' at line %d\n", lineNum);
#endif
       return '*';
    }
[0-9]+ {
#ifdef DEBUG
   printf("token %s at line %d\n", yytext, lineNum);
#endif
   yylval.val = atoi(yytext);
   return NUMBER;
}

%%

int yywrap() {      /* need this to avoid link problem */
   return 1;
}
```

Here is the Makefile used to build the scanner and parser:

```
p2: lex.yy.o y.tab.o
   gcc -o p2 lex.yy.o y.tab.o

lex.yy.o: cal.l
   lex cal.l; gcc -c lex.yy.c

y.tab.o: cal.y
   yacc cal.y; gcc -c y.tab.c

clean:
   rm -f p2 y.output *.o y.tab.c lex.yy.c
```