Jeffrey Lansford
CS 370
Lab 2.2
January 25, 2019

We used both a lex rotuine and a yacc routine to simulate a calculator. The Input is was the user types into for the lex rountine to use, and the output is what yacc prints based on what lex gave it.

Code :

YACC:

```
%{

/*
 *              **** CALC ****
 *
 * This routine will function like a desk calculator
 * There are 26 integer registers, named 'a' thru 'z'
 *
 */

/* This calculator depends on a LEX description which outputs either
VARIABLE or INTEGER.
   The return type via yylval is integer

   When we need to make yylval more complicated, we need to define a
pointer type for yylval
   and to instruct YACC to use a new type so that we can pass back
better values

   The registers are based on 0, so we substract 'a' from each single
letter we get.

   based on context, we have YACC do the correct memmory look up or
the storage depending
   on position

   Shaun Cooper
    January 2015

   problems  fix unary minus, fix parenthesis, add multiplication
   problems  make it so that verbose is on and off with an input
argument instead of compiled in
*/
```

```
/*
Jeffrey Lansford
CS 370
Lab 2.2
January 25, 2019

yacc routine to simulate a calculator. Fixed parentheses,
multiplication,
and UMINUS by adding a multiplication expr, fixing UMINUS with
removing the
first expr, and adding parentheses in the lex file. Input is the
tokens from
the lex routine that the user types. Output is the calculations from
the
different operations that the user provides.
*/

     /* begin specs */
#include <stdio.h>
#include <ctype.h>
#include "lex.yy.c"

int regs[26];
int base, debugsw;

void yyerror (s)  /* Called by yyparse on error */
     char *s;
{
  printf ("%s\n", s);
}


%}
/*  defines the start symbol, what values come back from LEX and how
the operators are associated  */

%start list

%token INTEGER
%token  VARIABLE

%left '|'
%left '&'
%left '+' '-'
```

```
%left '*' '/' '%'
%left UMINUS


%%   /* end specs, begin rules */

list :    /* empty */
     |    list stat '\n'
     |    list error '\n'
               { yyerrok; }
     ;

stat :    expr
               { fprintf(stderr,"the anwser is %d\n", $1); }
     |    VARIABLE '=' expr
               { regs[$1] = $3; }
     ;

expr :    '(' expr ')'
               { $$ = $2; }
     |    expr '-' expr
               { $$ = $1 - $3; }
     |    expr '+' expr
               { $$ = $1 + $3; }
     |    expr '/' expr
               { $$ = $1 / $3; }
    |   expr '*' expr
              { $$ = $1 * $3; /* added multiplication */}
     |    expr '%' expr
               { $$ = $1 % $3; }
     |    expr '&' expr
               { $$ = $1 & $3; }
     |    expr '|' expr
               { $$ = $1 | $3; }
     |    '-' expr  %prec UMINUS
               { $$ = -$2; /* fixed UMINUS to have it work correctly
*/}
     |    VARIABLE
               { $$ = regs[$1]; fprintf(stderr,"found a variable
value =%d\n",$1); }
     |    INTEGER {$$=$1; fprintf(stderr,"found an integer\n");}
     ;
```

```
%%   /* end of rules, start of program */

main()
{ yyparse();
}
```

LEX:

```
/*                  Small LEX routine which returns two formal tokens
(INTEGER and VARIABLE)
                    along with single string elements like '+'.

                     This LEX definition is the companion to the
docalc.y YACC routine which
                    is a simple calculator

                     Shaun Cooper
                     January 2015
*/

/*
Jeffrey Lansford
CS 370
Lab 2.2
January 25, 2019

lex program to create tokens and send them to to yacc routine. Added
parentheses so lex knows what to do with parentheses. The Lex rotine
gets its input from the user as they type. The Output is the differnt
tokens that are sent to the yacc routine to process.
*/
%{

int mydebug=1;
#include "y.tab.h"
%}


%%

[a-z]          {if (mydebug) fprintf(stderr,"Letter found\n");
```

```
                        yylval=*yytext-'a'; return(VARIABLE);}
[0-9][0-9]*    {if (mydebug) fprintf(stderr,"Digit found\n");
                        yylval=atoi((const char *)yytext);
return(INTEGER);}
[ \t]          {if (mydebug) fprintf(stderr,"Whitespace found\n");}
[=\-+*/%&|()]  { if (mydebug) fprintf(stderr,"return a token %c\
n",*yytext);
                        return (*yytext);/* added parentheses to be
read */}
\n        { if (mydebug) fprintf(stderr,"cariage return %c\
n",*yytext);
                        return (*yytext);}

%%

int yywrap(void)
{ return 1;}
```