Lab2_2
Santiago Flores
2/4/2020

MAKEFILE:

```
#Santiago Flores
#Lab2.2 Makefile
#2/4/2020
#
#YACC created files: y.tab.c  y.tab.h
#LEX created files: lex.yy.c
#Yacc takes the tokens that were found through the lex program which are shared
through y.tab.h so compiling of lex.yy.c does not have all the data.

all: lab2_2

lab2_2: y.tab.c
      gcc y.tab.c -o lab2_2

y.tab.c: lab2docalc.l
      lex lab2docalc.l

lab2docalc.l: lab2docalc.y
      yacc -d lab2docalc.y
```

FILE: lab2docalc.l

```
/*                  Lab2.2 lab2docalc.l
                    Santiago Flores
                    2/4/2020


                    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 lab2docalc.y YACC
routine which
                    is a simple calculator

*/
%{

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);  //
parenthasis added to list
                        return (*yytext);}
\n          { if (mydebug) fprintf(stderr,"cariage return %c\n",*yytext);
                        return (*yytext);}

%%

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

FILE: lab2docalc.y

```
%{

/*Lab2.2  lab2docalc.y
  Santiago Flores
  2/4/2020
  Takes input from user and does calculations
  outputs the solutions to user input calculations
  Example: -(3 * 9) will output -27
*/

/*
 *                **** 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 correc   Shaun Cooper
    January 2015t memmory look up or the storage depending
   on position

   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
*/


      /* 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             //Added multiplication
            { $$ = $1 * $3; }
      |     expr '%' expr
                  { $$ = $1 % $3; }
      |     expr '&' expr
                  { $$ = $1 & $3; }
      |     expr '|' expr
                  { $$ = $1 | $3; }
      |   '-' expr      %prec UMINUS  //removed extra expression
                  { $$ = -$2; }
```

```
      |     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 */

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