Collin Gros
02/03/2020

CS-370 LAB 2.2

Fix unary minus, parenthesis, and multiplication in the given yacc (and lex) code.

We don't directly compile lex output because yacc takes the lex file as input to create its own c-code. Lex is simply a lexicographic analyzer. Yacc takes the tokens provided by lex, and generates c-code that we can compile.

MODIFIED LEX FILE

```
%{
/*
        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


        collin gros
        02/03/2020

        changed line 30 to include parenthesis.
*/

int mydebug=0;
#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);}
```

```
\n                    { if (mydebug) fprintf(stderr,"cariage return %c\n",*yytext);
                      return (*yytext);}

%%

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

MODIFIED YACC FILE

```
%{

/*
 *                    **** 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


        collin gros
        01/31/2020

        i changed the lex file to pass the symbols, '(' and ')' as tokens to
        yacc. this solved the paranthesis issue.

        i added an expression for handling the '*' token. this solved the
        multiplication issue.

        i deleted the 'expr' right before the '-' token under the unary minus
        rule. this solved the unary minus issue.
*/


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

int regs[26];
int base, debugsw = 0;

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
```

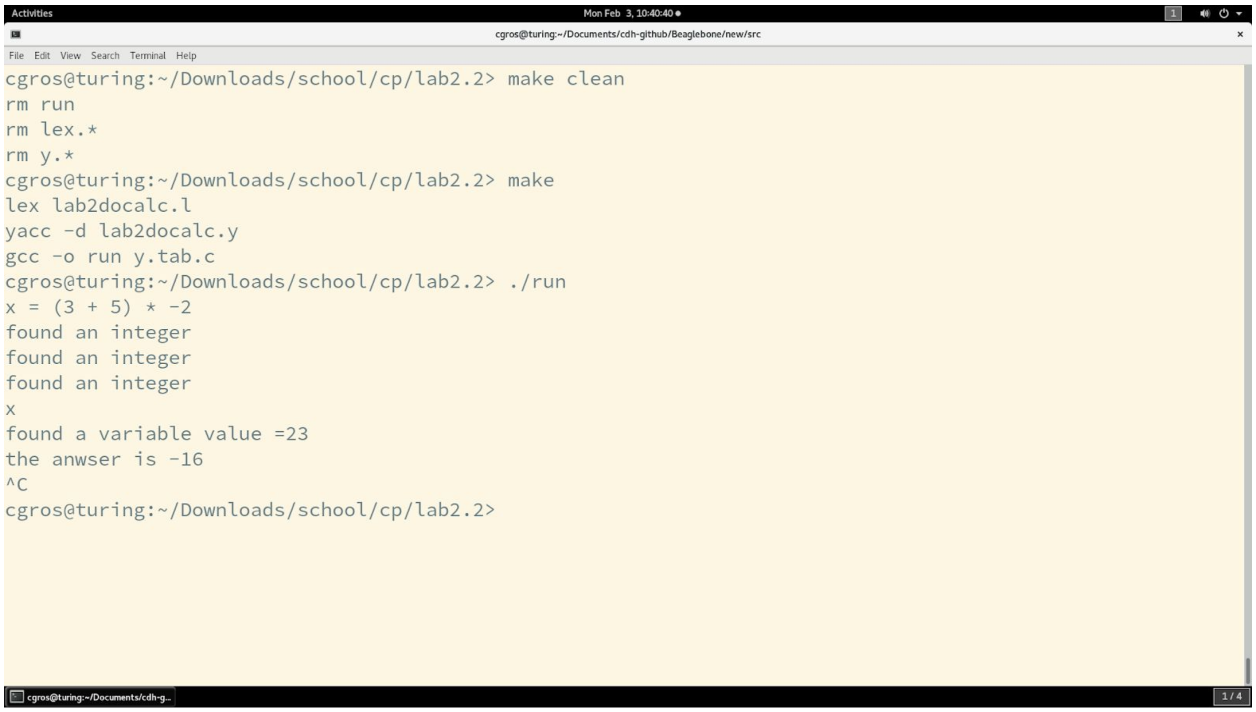
```
%%      /* end specs, begin rules */

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 ')'
                { $$ = $2; }
        |       '-' expr            %prec UMINUS
                { $$ = -$2; }
        |       expr '-' expr
                { $$ = $1 - $3; }
        |       expr '+' expr
                { $$ = $1 + $3; }
        |       expr '*' expr
                { $$ = $1 * $3; }
        |       expr '/' expr
                { $$ = $1 / $3; }
        |       expr '%' expr
                { $$ = $1 % $3; }
        |       expr '&' expr
                { $$ = $1 & $3; }
        |       expr '|' expr
                { $$ = $1 | $3; }
        |       VARIABLE
                { $$ = 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(int argc, char* argv)
{
	yyparse();
}
```

MAKEFILE

```
# collin gros
# 02/03/2020
#
# makefile for lab2.2
#
# yacc creates y.tab.c, y.tab.h
# lex creates lex.yy.c

# we don't directly compile lex output because yacc takes the lex file
# as input to create its own c-code. lex is simply a lexographic analyzer.
# yacc takes the tokens provided by lex, and generates c-code

all: lab2.2

# compile lex-generated code after running lex
lab2.2: lex.yy.c y.tab.c
	gcc -o run y.tab.c

# feed lex the lex file
lex.yy.c: lab2docalc.l
	lex lab2docalc.l

# feed yacc the yacc file
y.tab.c: lab2docalc.y
	yacc -d lab2docalc.y

# get rid of all the generated crap
clean:
	rm run
	rm lex.*
	rm y.*
```

OUTPUT


Activities
Mon Feb 3, 10:40:40
cgros@turing:~/Documents/cdh-github/Beaglebone/new/src
File Edit View Search Terminal Help
cgros@turing:~/Documents/cdh-g...
1 / 4

```
cgros@turing:~/Downloads/school/cp/lab2.2> make clean
rm run
rm lex.*
rm y.*
cgros@turing:~/Downloads/school/cp/lab2.2> make
lex lab2docalc.l
yacc -d lab2docalc.y
gcc -o run y.tab.c
cgros@turing:~/Downloads/school/cp/lab2.2> ./run
x = (3 + 5) * -2
found an integer
found an integer
found an integer
x
found a variable value =23
the anwser is -16
^C
cgros@turing:~/Downloads/school/cp/lab2.2>
```