Lab 2.2 -- Using YACC

Kay Sweebe
CS370
January 28, 2018

Problem

A lex and yacc program were given that had 3 problems: 1) unary minus does not work, 2) parenthesis does not work, 3) multiplication does not work. The goal of the program was to create a simple calculator that supported addition, subtraction, division, and the three missing components listed above.

The changes I made to the Lex file include adding parenthesis, (,), to the regular expression that contains +, -, = and other chars. Without this, whenever a parenthesis was found, lex did nothing with it. Once we added the parenthesis, Lex was then able to pass the parenthesis as tokens to Yacc.

The changes I made to the yacc file include editing the unary minus declaration and adding a declaration for multiplication. Unary minus was represented as a binary operation. It was changed from the form "expr '-' expr" to "'-' expr". Multiplication was missing so I added "expr '*' expr" followed by "{ $$ = $1 * $3; }" so that the first ($1) and third ($3) tokens are multiplied together. The result is stored as an expression.

lab2docalc.l

```
/*    Kay Sweebe
    CS370
    January 25, 2019
    Lab2.2

    The changes made to this lab include:
          - Adding parenthesis (, ) to the regular expression that contains +, -, = and other
chars.
                Without this, whenever a parenthesis was found, lex did nothing with it. Once
we added
           the parenthesis, Lex was then able to pass the parenthesis as tokens to Yacc.
```

```
*/



/*              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
*/
%{

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

%%

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

lab2docalc.y

```
%{
/*    Kay Sweebe
 *    CS370
 *    January 25, 2019
 *    Lab2.2
 *
 *    The changes made to this lab include:
 *          - Unary minus was represented as a binary operation. It was changed from the form
 *            "expr '-' expr" to "'-' expr". Unary minus is left associative.
 *          - Multiplication was missing so I added "expr '*' expr" followed by "{ $$ = $1 * $3; }"
 *            so that the first ($1) and third ($3) tokens are multiplied together. The result is
 *            stored as an expression.
 */

/*
 *                    **** 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
*/
```

```
    /* 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 // this was added
                    { $$ = $1 * $3; } // this was added
    |    expr '%' expr
                    { $$ = $1 % $3; }
    |    expr '&' expr
                    { $$ = $1 & $3; }
    |    expr '|' expr
                    { $$ = $1 | $3; }
    |    '-' expr    %prec UMINUS // this was modified
                    { $$ = -$2; }
    |    VARIABLE
                    { $$ = 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 */

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

Makefile

```
# Kay Sweebe
# CS370
# Lab 2.2
#
# Yacc outputs y.tab.h, y.tab.c
# Lex  outputs lex.yy.c
# Lex output is not directly compiled because it is missing a portion of the C script to make
#  a complete program
all:
    yacc -d lab2docalc.y
```

```
lex lab2docalc.l
gcc y.tab.c -o lab2docalc
```

Output

```
ksweebe@godel:~/Workspace/cs370/Lab2.2> ./lab2docalc
-(3*9)
return a token -
return a token (
Digit found
found an integer
return a token *
Digit found
found an integer
return a token )
cariage return

the anwser is-27
```