**一、实验目的：**

运用YACC，针对给定的文法，构造一个语法分析器。给出实验方案，实施并描述结果。

**二、实验预习提示**

1、表达：针对5.5节中的calculator文法，设计输入和输出  
给定的.lex文件定义了TINY语言的词法规则，包括数字、运算符和分隔符的识别。.y文件定义了TINY语言的语法规则，包括表达式的解析和计算。主函数main.c中，打开一个名为"test.c"的文件作为输入，并调用yyparse()函数进行语法分析。输出结果为解析后的表达式的值。

2、观察：观察parsing table，解析parsing table.  
在.y文件中，定义了语法规则、终结符和非终结符。YACC会根据这些信息生成一个parsing table，用于分析输入的词法单元流并生成语法树。通过观察parsing table，可以了解语法分析器的行为和决策过程。

3、模块间的衔接，如何同时使用lex 和 yacc  
在.lex文件中，使用了%{}部分来引入头文件，并声明了一些变量和函数。这些变量和函数可以在.lex文件中使用，以及在.y文件和主函数main.c中使用。通过这种方式，可以同时使用lex和yacc来构建一个完整的语法分析器。

4、能力：是否有能力用工具完成一个分析器  
这个实验的目的之一是检验是否有能力使用工具（YACC）完成一个分析器。YACC提供了强大的语法分析功能，可以根据给定的语法规则自动生成分析器代码，大大简化了分析器的开发过程。

5、比较：用工具和手工编写程序的比较  
实验中还提到了用工具（YACC）和手工编写程序的比较。使用YACC可以更快速地生成分析器代码，并且保证了语法规则的正确性。相比手工编写程序，使用工具可以节省大量时间和精力，并降低出错的可能性。  
  
总体来说，这个实验通过使用YACC构建了一个TINY语言的语法分析器。在预习阶段，进行了表达设计、parsing table的解析和模块间的衔接的工作。在实施阶段，根据给定的.lex和.y文件，通过编译主函数main.c，将所有模块组合起来，最终得到了一个可以解析和计算TINY语言表达式的分析器。

**针对TINY语言**

**.l文件的代码**

%{

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#define true 1 // Changed 'ture' to 'true'

#define false 0 // Changed 'false' to 'false'

#include "yacc.tab.h"

extern int lexverbose;

extern int linecount;

%}

digit [0-9]

letter [a-zA-Z]

%%

{digit}+ {

yylval.real = atof(yytext); // Corrected the spelling of 'atof'

if (lexverbose)

printf("real:%g\n", yylval.real);

return(number);

}

\+ {

yylval.chr = yytext[0];

if (lexverbose)

printf("operator:%c\n", yylval.chr);

return('+');

}

\- {

yylval.chr = yytext[0];

if (lexverbose)

printf("operator:%c\n", yylval.chr);

return('-');

}

\\* {

yylval.chr = yytext[0];

if (lexverbose)

printf("operator:%c\n", yylval.chr);

return('\*');

}

\/ {

yylval.chr = yytext[0];

if (lexverbose)

printf("operator:%c\n", yylval.chr);

return('/');

}

\( {

yylval.chr = yytext[0];

if (lexverbose)

printf("separator:%c\n", yylval.chr);

return('(');

}

\) {

yylval.chr = yytext[0];

if (lexverbose)

printf("separator:%c\n", yylval.chr);

return(')');

}

\; {

return(';');

}

\n {

printf("line %d\n", linecount);

/\* linecount++; \*/ // Commented out the increment statement

return('\n');

}

[ \t]+ {

printf("lexical analyzer error\n");

}

quit {

printf("Bye!\n");

exit(0);

}

%%

int yywrap() {

return 1;

}

**.y文件的代码**

%{

#include <ctype.h>

#include <stdio.h>

#define MSDOS

int linecount;

extern int yylex();

extern int yyerror();

%}

%union {

char chr;

char\* str;

int integer;

float real;

double dbl;

}

%token number

%type <integer> expr

%left '+'

%left '-'

%left '\*'

%left '/'

%%

lines: lines expr '\n'

{

printf("line %d:%g\n", linecount++, $2);

}

| lines '\n'

{

linecount++;

}

| ;

expr: expr '+' expr

{

$$ = $1 + $3;

}

| expr '-' expr

{

$$ = $1 - $3;

}

| expr '\*' expr

{

$$ = $1 \* $3;

}

| expr '/' expr

{

$$ = $1 / $3;

}

| '(' expr ')'

{

$$ = $2;

}

| '-' expr %prec uminus

{

$$ = -$2;

}

| number

{

$$ = $1;

};

%%

int yyerror(char \*s)

{

fprintf(stderr, "syntactic error:%s\n", s);

return 0;

}

**编译主函数main.c**

#include "lex.yy.c"

#include "yacc.tab.c"

#include <stdlib.h>

#include <stdio.h>

int lexverbose = 1;

extern int yyparse();

int main(int argc, char\* argv[])

{

extern FILE\* yyin;

printf("Compiling...!\n");

if ((yyin = fopen("test.c", "rt")) == NULL) {

perror("can not open file test.c\n");

exit(1);

}

if (yyparse() == 1) {

fprintf(stderr, "parser error\n");

exit(1);

}

printf("yyparse() completed successfully!\n");

return 0;

}