语法分析程序的设计与实现(YACC 实现)

班级：2022211304
学号：2022211119
姓名：赵宇鹏

实验内容及要求

利用 YACC 自动生成语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生。

```
E→ E+T | E-T | T
T→ T*F | T/F | F
F→ (E) | num
```

实验要求

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

实现方法要求

根据给定文法，编写 YACC 说明文件，调用 LEX 生成的词法分析程序。

程序设计说明

本项目由`parser.y`以及`lexer.l`组成，用于实现对用户输入的数学表达式进行词法分析再进行语法分析，判断识别表达式的语法是否有效，若有效则输出计算结果，否则输出错误信息。

词法分析部分

词法分析通过 Lex 实现，用于识别输入字符串中的数字、运算符及空白字符等基本组成部分，代码如下：

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}

%%
[0-9]+(\.[0-9]+)?   { yylval = atoi(yytext); return NUM; }  // 识别整数和小数
[ \t]               ;  // 忽略空格
\n                  { return '\n'; }
"+"                 { return '+'; }
"-"                 { return '-'; }
"*"                 { return '*'; }
"/"                 { return '/'; }
"("                 { return '('; }
")"                 { return ')'; }
.                   { return yytext[0]; }  // 其他
```

```
%%

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

主要功能如下：

- **整数和小数识别：**

  通过使用正则表达式 `[0-9]+(\.[0-9]+)?` 来识别整数和小数，并将识别到的数值赋给 `yylval` 并返回 `NUM` 标记给 `parser.y` 文件。

- **运算符识别**：

  可以识别加号（`+`）、减号（`-`）、乘号（`*`）、除号（`/`）、括号（`(` 和 `)`）等符号，并返回相应的字符标记。

- **空白符和换行处理：**
  忽略空格和制表符（`[ \t]`），将换行符（`\n`）返回给 Yacc，表示一行输入的结束。

- **其他字符处理：**
  通过通配符 `.` 处理不符合预定义规则的其他字符，直接返回该字符的 ASCII 码，便于后续错误处理。

语法分析部分(YACC)

语法分析部分通过 Yacc 实现，根据表达式的文法规则分析输入的结构，代码如下：

```
%{
#include <stdio.h>
#include <stdlib.h>
#define YYLEX yylex
void yyerror(const char *s);
int yylex();
%}

%token NUM
%left '+' '-'
%left '*' '/'
%right UMINUS

%%
line:
    expr '\n' { printf("Result: %d\nExpression is valid.\n", $1); }
;

expr:
    NUM { $$ = $1; }
    | expr '+' expr { $$ = $1 + $3; }
    | expr '-' expr { $$ = $1 - $3; }
    | expr '*' expr { $$ = $1 * $3; }
    | expr '/' expr { $$ = $1 / $3; }
    | '(' expr ')' { $$ = $2; }
;
%%
```

```c
int main() {
    printf("Enter an expression: ");
    if (yyparse() == 0) {
    } else {
        printf("Expression is invalid.\n");
    }
    return 0;
}

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
```

语法分析部分主要参考了课本上的示例代码，其主要功能如下：

- **优先级和结合性设置：**
  使用 `%left` 和 `%right` 定义运算符的优先级和结合性。加减法和乘除法设置为左结合，一元负号 `UMINUS` 设置为右结合，用于处理负数。
- **语法规则定义：**
  Yacc 语法规则用于匹配和计算基本的四则运算和带括号的表达式，包括：
  - `NUM`：匹配数字，直接返回其值。
  - `expr '+' expr`、`expr '-' expr`：计算加法或减法。
  - `expr '*' expr`、`expr '/' expr`：计算乘法或除法。
  - `'(' expr ')'`：优先计算括号中的表达式。
- **错误处理**
  定义 `yyerror()` 函数用于输出错误提示。当解析出现错误时，Yacc 会调用该函数并输出相关的错误信息，提示用户表达式无效。
- **主函数**
  主函数中使用 `yyparse()` 调用 Yacc 的语法解析器，判断输入表达式的有效性。若解析成功，输出计算结果和提示；否则，输出错误信息。

执行步骤

1. 在命令行输入命令，使用 Bison 生成 `y.tab.c` 和 `y.tab.h` 文件：

   ```
   yacc -d parser.y  
   ```

2. 使用 Flex 生成 `lex.yy.c` 文件：

   ```
   lex lexer.l
   ```

3. 编译并链接：

   ```
   gcc lex.yy.c y.tab.c -o parser -ll
   ```

4. 最后即可生成parser可执行文件，输入命令：

   ```
   ./parser
   ```

即可运行程序。

运行结果及分析

测试一

输入

```
5+7-6*(7-8+2/2*(4-7)+66)
```

输出

```
Enter an expression: 5+7-6*(7-8+2/2*(4-7)+66)
Result: -360
Expression is valid.
```

分析

本输入为一个正确的表达式，可以看到程序正确输出了结果，并给出了判断，程序运行正常。

测试二

输入

```
3+*5
```

输出

```
3+*5
Error: syntax error
Expression is invalid.
```

分析

可以看到程序输出错误，正确检测到了输入语句存在的语法错误，程序运行正常。

实验总结

本次实验我通过Lex和Yacc实现了一个能够识别题目所给文法的语法分析工具，能够正确识别相应表达式并输出计算结果。相较于手工编写的语法解析程序，Lex 和 Yacc 提供了强大的自动化支持，极大地简化了编程过程，减少了代码编写量和调试工作量。通过 Lex 的词法分析功能，可以轻松实现对数字、运算符、空白字符等基本元素的识别，并忽略不必要的空白字符，保证了输入的整洁。Yacc 则通过定义文法规则，实现了四则运算和括号表达式的解析，自动处理了运算符的优先级和结合性，使表达式的解析更为准确。