YACC语法分析器——实验报告

作者：郭栩源

一、实验题目要求

本次实验要求使用YACC工具自动生成语法分析程序，实现对算数表达式进行语法分析。
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
本人使用开发环境为macOS 13.0。

二、工具介绍

本实验主要涉及到的工具有lex和yacc，下面对其进行介绍：

1. Lex (词法分析器生成工具)

- 作用：Lex 是一个用于生成词法分析器（或称为扫描器）的工具。词法分析器的任务是将输入的字符流转换为一系列的词法单元或标记。
- 输入：Lex 通常需要一个定义了输入字符模式的规范文件。
- 输出：Lex 生成的输出是一个C语言源代码文件，该文件可以编译和链接以生成实际的词法分析器程序。

2. Yacc (Yet Another Compiler Compiler)

- 作用：Yacc 是一个生成语法分析器的工具。与词法分析器不同，语法分析器处理的是词法单元，并且根据某种给定的语法结构组织它们。
- 输入：Yacc 通常需要一个定义了语法规则的规范文件。
- 输出：与 Lex 类似，Yacc 的输出是一个C语言源代码文件。这个文件可以被编译和链接，以生成实际的语法分析器程序。

在编译器的前端，Lex 和 Yacc 经常一起使用。Lex 负责词法分析，产生一系列的词法单元或标记；而 Yacc 则处理这些标记，并根据给定的语法规则组织它们。这两个工具合起来，可以帮助开发者构建一个完整的编译器或解释器的前端部分。
在将lex与yacc结合使用时，需在lex.l中包含头文件"y.tab.h"，在yacc.y中显式地声明在lex中定义的yylex（lex词法分析函数），yyin和yyout（文件指针）。

三、lex程序设计说明

由于此次试验文法较简单，包含的非终结符只含有+，-，*，/，(，)，num，故不使用词法分析实验中编写的针对C语言的词法分析器，而是直接重新写一个简单的词法分析器，便于后续调试。

词法分析程序中涉及的记号介绍

- number（数字常量）
  包括十进制整数，如：123，
  十进制浮点数，如12.34，
  十进制指数，如2.5E3，
  八进制整数，如017
  和十六进制整数，如0x4A。
- operator（操作符）
  包括：+、-、*、/
- delimiter（分隔符）
  包括：(、)。

程序中定义的正则式

正则式	说明
digit	十进制数字字符
oct_digit	八进制数字字符
hex_digit	十六进制数字字符
oct_num	八进制整数
hex_num	十六进制整数
num	十进制整数、浮点数和指数

错误检测

在lex中，我们定义默认规则或特定的不合法规则以实现错误检测和错误恢复。
lex中的默认规则是指，当没有其他规则匹配时，这个默认规则会匹配，这可以用于捕获和报告词法错误。
特定的不合法规则是指，检测特定的正则式以识别词法错误，如识别以数字开头的非法标识符可定义正则式{digit}({letter}|{digit})*。
在该程序中，由于输入较简单，故只使用默认规则，即当没有其他规则匹配时，使用默认规则报出错误“Unexpected character”。

四、yacc程序设计说明

翻译规则介绍

由于在完成yacc自动生成语法分析器实验前我已经完成了手工编写LR语法分析器实验，在手工编写LR语法分析器实验中我已经证明了该实验中的文法是SLR(1)文法，而SLR(1)文法都是无二义性的，所以该实验中编写yacc分析程序时并不需要考虑使用优先级方法来处理文法二义性的问题。
该程序中使用的翻译规则如下，使用每条翻译规则时都将打印对应产生式：

```
E : E '+' T     {fprintf(yyout,"E->E+T\n");}
  | E '-' T     {fprintf(yyout,"E->E-T\n");}
  | T           {fprintf(yyout,"E->T\n");}
  ;

T : T '*' F     {fprintf(yyout,"T->T*F\n");}
  | T '/' F     {fprintf(yyout,"T->T/F\n");}
  | F           {fprintf(yyout,"T->F\n");}
  ;

F : '(' E ')'   {fprintf(yyout,"F->(E)\n");}
  | NUM         {fprintf(yyout,"F->NUM\n");}
  ;
```

错误检测

yacc提供了一个特殊的令牌error，可以在文法规则中使用。当yacc在输入中遇到一个不符合任何文法规则的令牌时，它会尝试与error令牌匹配。你可以在包含error的文法规则中定义恢复策略。
在该程序中，由于语法较为简单，故不适用error令牌，在遇到任何语法错误时直接打印“error:syntax error”。

五、测试报告

测试样例1

输入：

```
2.3 + 2E3 * 3.9
```

输入说明：在该测试用例中，词法分析器应将2.3，2E3，3.9都以标记NUM返回，即语法分析器以词法分析器的返回结果“NUM + NUM * NUM”为输入。该测试用例旨在测试词法分析程序和语法分析程序的基础分析功能。

输出：

```
F->NUM
T->F
E->T
F->NUM
T->F
F->NUM
T->T*F
E->E+T
```

测试结果：对比手工编写的语法分析程序，两者分析结果相同，证明两者都有基础的语法分析功能。

测试样例2

输入：

```
0xA + (2E3 * 067)
```

输入说明：对不同类型的输入进行进一步的测试。

输出：

```
F->NUM
T->F
E->T
F->NUM
T->F
F->NUM
T->T*F
E->T
F->(E)
T->F
E->E+T
```

测试结果：词法分析器能够处理入如八进制整数、十六进制整数等不同的输入；语法分析器能够处理更复杂的运算式。

测试样例3

输入：

```
1 + 2 * a
```

输入说明：测试词法分析器的错误检测能力。

输出：

```
Unexpected character a
```

测试结果：词法分析器具有基本的错误检测能力。

测试样例4

输入：

```
1 + 2 * 3 + + 4 5
```

输入说明：测试语法分析器的错误检测能力。

输出：

```
error:syntax error
```

测试结果：语法分析器具有基本的错误检测能力。

六、使用说明

在macOS和Linux环境下使用如下命令对lex说明文件和yacc说说明文件进行编译：

```
lex lex.l
yacc translate.y -d
gcc -o main y.tab.c lex.yy.c
```

使用如下命令运行：

```
./main in out
```

其中in和out指定了输入输出文件，若省略则默认为控制台输入输出。
需要注意的是，lex和yacc是unix下的词法生成器和语法生成器生成工具，在Windows系统下需要额外安装lex和bison环境，此处不再赘述。

七、改进与完善

该程序实现了简单的语法分析功能，但还有许多可以改进和完善的方面：

1. 语义动作：目前的文法只识别输入是否符合文法规则，但并不执行任何操作。
   我们可以加入语义动作以实现实际的算数运算。例如，在E : E '+' T规则中，可以返回 E 和 T 的值的和。
2. 错误恢复：增加更强大的错误恢复功能。当前的yyerror函数只打印一个简单的错误消息。
   一个更好的方式是提供更详细的错误信息，例如错误的位置或预期的令牌。