编译原理与技术实验二：语法分析程序的设计与实现（YACC 实现）
实验报告

张梓良　2021212484

日期：2023 年 11 月 15 日

1　概述

1.1　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。

$$E \to E + T \mid E - T \mid T$$

$$T \to T * F \mid T/F \mid F$$

$$F \to (E) \mid num$$

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。

1.2　实验环境

- flex version 2.5.4
- bison version 2.4.1
- gcc version 8.1.0
- Visual Studio Code 1.82.2
- OS: Windows_NT x64 10.0.22621

1.3　实验目的

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。

2　程序设计说明

该部分具体介绍了 LEX 说明文件和 YACC 说明文件的设计。

2.1　LEX 说明文件

`lexer.l`中实现了以下功能：

- 通过正则表达式规则匹配输入流中的数字，并将其识别为 NUM 标记返回给语法分析器。
- 通过正则表达式规则匹配输入流中的运算符和括号字符，并将其作为相应的字符标记返回给语法分析器。
- 通过正则表达式规则匹配输入流中的换行符，并将其识别为换行符标记返回给语法分析器。
- 忽略输入流中的空格和制表符。
- 对于无法匹配前面定义的任何规则的字符，输出错误消息并报告为无效输入。

具体实现代码如下：

```
%{
#include "y.tab.h"
%}

%%
[1-9][0-9]* {return NUM;}
[-+*/()]    {return yytext[0];}
"\n"    {return '\n';}
[ \t]+      {}
.           {fprintf(stderr, "Invalid input: %s\n\n", yytext);}
%%

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

2.2 YACC 说明文件

syntax.y中实现了以下功能：

- 定义了词法分析器返回的标记类型。
- 定义了 line 规则，用于表示输入的一行，包括空行、换行符和错误处理。
- 定义了 expr 规则，用于表示表达式，包括加法、减法和只有项的情况。
- 定义了 term 规则，用于表示项，包括乘法、除法和只有因子的情况。
- 定义了 factor 规则，用于表示因子，包括括号中的表达式和数字。
- 实现了 yyerror 函数，用于输出语法错误信息。
- 实现了 main 函数，调用 yyparse 函数开始解析输入。

具体实现代码如下：

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

%token NUM
```

```
8
9  %%
10 line:line expr '\n'  {printf("ACCEPT\n\n");}
11     |               {}
12     |line '\n'       {}
13     |error '\n'      {yyerror("Please retry!\n");yyerrok;}
14 ;
15
16 expr:expr '+'term    {printf("E->E+T\n");}
17     |expr '-'term    {printf("E->E-F\n");}
18     |term            {printf("E->T\n");}
19     ;
20
21 term:term '*'factor  {printf("T->T*F\n");}
22     |term '/'factor  {printf("T->T/F\n");}
23     |factor          {printf("T->F\n");}
24     ;
25
26 factor:'('expr')'    {printf("F->(E)\n");}
27     |NUM             {printf("F->num\n");}
28     ;
29 %%
30
31 void yyerror(const char* s)
32 {
33     printf("%s\n", s);
34 }
35
36 int main() {
37     yyparse();
38     return 0;
39 }
```

3 用户手册

通过以下指令编译得到可执行程序Syntax.exe：

```
bison --yacc -d syntax.y
flex lexer.l
gcc lex.yy.c y.tab.c -o Syntax
```

编译前确保已经安装flex和bison并加入到了系统的环境变量中。

运行可执行程序Syntax.exe，可循环输入待分析的符号串，程序会输出相应语法分析。

按下ctrl+z以结束程序。

4 程序测试

4.1 测试集 1

此测试集为一个简单的算术表达式，用于测试程序是否能够正常运行。

4.1.1 测试内容

```
(1+(2*3))/4
```

4.1.2 测试结果

```
F->num
T->F
E->T
F->num
T->F
F->num
T->T*F
E->T
F->(E)
T->F
E->E+T
F->(E)
T->F
F->num
T->T/F
E->T
ACCEPT
```

4.1.3 结果分析

经过验证，测试结果为算数表达式 `(1+(2*3))/4` 的最右推导的逆序，说明程序可以对简单的算数表达式正确分析。

4.2 测试集 2

此测试集为一个较为复杂的算术表达式，用于测试程序的鲁棒性。

4.2.1 测试内容

```
((1+(2*(3/4+5)-(6+7)/(8-9))*(10+11+12*13))/(14*(15+16)))
```

4.2.2 测试结果

```
F->num
T->F
E->T
F->num
T->F
F->num
T->F
F->num
T->T/F
E->T
F->num
T->F
E->E+T
F->(E)
T->T*F
E->T
F->num
T->F
E->T
F->num
T->F
E->E+T
F->(E)
T->F
F->num
T->F
E->T
F->num
T->F
E->E-F
F->(E)
T->T/F
E->E-F
```

```
F->(E)
T->F
F->num
T->F
E->T
F->num
T->F
E->E+T
F->num
T->F
F->num
T->T*F
E->E+T
F->(E)
T->T*F
E->E+T
F->(E)
T->F
F->num
T->F
F->num
T->F
E->T
F->num
T->F
E->E+T
F->(E)
T->T*F
E->T
F->(E)
T->T/F
E->T
F->(E)
T->F
E->T
ACCEPT
```

4.2.3 结果分析

经过验证,测试结果为算数表达式((1+(2*(3/4+5)-(6+7)/(8-9))*(10+11+12*13))/(14*(15+16))) 的最右推导的逆序，说明程序可以对较为复杂的算数表达式正确分析。

4.3 测试集 3

此测试集为一个缺少右括号的算术表达式，用于测试程序是否能够发现该错误。

4.3.1 测试内容

```
(1+2
```

4.3.2 测试结果

```
F->num
T->F
E->T
F->num
T->F
E->E+T
syntax error
Please retry!
```

4.3.3 结果分析

程序能够发现算数表达式缺少右括号的错误。

4.4 测试集 4

此测试集为一个缺少左括号的算术表达式，用于测试程序是否能够发现该错误。

4.4.1 测试内容

```
1+2)
```

4.4.2 测试结果

```
F->num
T->F
E->T
F->num
```

```
T->F
E->E+T
syntax error
Please retry!
```

4.4.3 结果分析

程序能够发现算数表达式缺少左括号的错误。

4.5 测试集 5

此测试集为一个缺少运算符的算术表达式，用于测试程序是否能够发现该错误。

4.5.1 测试内容

```
1(2+3)
```

4.5.2 测试结果

```
F->num
T->F
E->T
syntax error
Please retry!
```

4.5.3 结果分析

程序能够发现算数表达式缺少运算符的错误。

4.6 测试集 6

此测试集为一个缺少运算对象的算术表达式，用于测试程序是否能够发现该错误。

4.6.1 测试内容

```
*(2+3)
```

4.6.2 测试结果

```
syntax error
Please retry!
```

4.6.3 结果分析

程序能够发现算数表达式缺少运算对象的错误。

5 实验总结

本次实验是关于语法分析程序设计与实现，使用 YACC 来自动生成语法分析程序，并实现对算术表达式的语法分析。在完成实验的过程中，我总结了以下几点体会。

首先，通过编写 YACC 说明文件以及调用 LEX 生成的词法分析程序，我们可以快速构建一个语法分析器。YACC 提供了一种简洁的语法规则定义方式，能够自动生成语法分析程序的代码，大大简化了语法分析器的开发过程。

其次，设计和定义文法是实现成功的关键。在本次实验中，根据给定的文法，我定义了相应的语法规则，包括表达式、项和因子等。合理的文法设计可以确保语法分析器能够正确地解析输入的算术表达式，并输出相应的产生式。

在实验过程中，我遵循了 YACC 的规范，按照需要的顺序编写了 YACC 说明文件，并在其中定义了词法分析器返回的标记类型以及相应的语法规则。通过调用 YACC 和 LEX 工具，我成功生成了语法分析程序，并进行了测试。

最后，通过实验，我深刻理解了语法分析的重要性和作用。语法分析是编译器中的重要阶段，它负责将输入的源代码转化为抽象语法树，为后续的语义分析和代码生成提供基础。掌握 YACC 工具的使用，能够更高效地实现语法分析器，并加深对语法规则以及产生式的理解。

总的来说，本次实验让我学习了 YACC 的使用方法，并实践了通过 YACC 生成语法分析程序的过程。通过设计和定义文法，我成功实现了对算术表达式的语法分析，并输出了相应的产生式。这次实验不仅加深了我的编译原理理论知识，也增强了我的实践能力。