编译原理

实验（四）用Yacc实现语法分析器

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2023年5月20日
学　　院	信息学院
课程名称	编译原理

实验（四）用Yacc实现语法分析器

目录

1 实验目的 **1**

2 实验内容 **1**

3 运行结果 **1**

4 支持前缀'-'和'!'运算符 **1**

A 附录：完整输出 **2**

1 实验目的

掌握语法分析器的构造原理，掌握Yacc的编程方法。

2 实验内容

用Yacc编写一个语法分析程序，使之与词法分析器结合，能够根据语言的上下文无关文法，识别输入的单词序列是否文法的句子。

3 运行结果

运行结果如图 1，完整输出见附录：完整输出。

```
-> % ./parser
factor-->num
unary-->factor
term-->unary
expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
stmt-->id=bool;
stmts-->stmts stmt
```

图 1: 运行结果

4 支持前缀'-'和'!'运算符

在这次实验中，我额外实现了对前缀'-'和'!'运算符的支持，即取相反数运算符和逻辑非运算符。

首先，需要对文法进行改写。该文法的优先级应该高于乘除法，但低于加减法。据此编写处理一元运算符的文法如下：

```
term : term '*' unary { printf("term-->term*unary\n"); }
     | term '/' unary { printf("term-->term/unary\n"); }
     | unary { printf("term-->unary\n"); }
     ;
unary : '!' unary { printf("unary-->!unary\n"); }
      | '-' unary %prec UMINUS{ printf("unary-->-unary\n"); }
      | factor { printf("unary-->factor\n"); }
      ;
factor : '(' bool ')' { printf("factor-->(bool)\n"); }
       | ID { printf("factor-->id\n"); }
       | NUM { printf("factor-->num\n"); }
```

```
    | REAL { printf("factor-->real\n"); }
    | TRUE { printf("factor-->true\n"); }
    | FALSE { printf("factor-->false\n"); }
    ;
```

同时，对于前缀'-'运算符需要处理结合性。在上述文法中通过标记UMINUS指定了%nonassoc，意为无结合性。这样，yacc会将形如a op b op c式的表达式视为错误。这样做的目的在于区分二元'-'运算符，即减法运算符。通过指定无结合性，二元'-'运算符和一元'-'运算符得以区分，不会产生冲突的问题。否则，yacc就会输出产生移入规约冲突的信息。

A 附录：完整输出

```
factor-->num
unary-->factor
term-->unary
expr-->term
rel-->expr
equality-->rel
join-->equality
bool-->join
stmt-->id=bool;
stmts-->stmts stmt
factor-->id
unary-->factor
term-->unary
expr-->term
factor-->num
unary-->factor
term-->unary
expr-->term
rel-->expr<=expr
equality-->rel
join-->equality
bool-->join
block-->{decls stmts}
stmt-->block
stmt-->while(bool)stmt
stmts-->stmts stmt
block-->{decls stmts}
program-->block
```