实验一 熟悉Oberon-0

1. **编写一个正确的Oberon-0 源程序**

   这里实现的是求最大公因子的程序，具体的程序内容为

```
MODULE test;
    CONST PI = 3;
    TYPE int = INTEGER;
         bool = BOOLEAN;
         struct = RECORD
            a : int;
            b : int;
            c : int
        END;
        arr = ARRAY 10 OF INTEGER;
    VAR a,b,c : int;
    PROCEDURE gcd(a:int;b:int;VAR c:int);
        BEGIN
            IF b # 0 THEN
                gcd (b,a MOD b,c)
            ELSE
                c := a
            END
        END gcd;
    BEGIN
        READ(a);
        READ(b);
        gcd(a,b,c);
        WRITE(c);
        WHILE c+5*2 > 10 DO
            c := 0;
            struct.c := 10;
            arr[5] := 4
        END
    END test.
```

2. **编写上述 Oberon-0 源程序的变异程序**

   这里编写了出现不同类型的错误的变异程序，具体程序内容不在报告中展示，放在testcases文件夹中，具体含义为

   001：IllegalSymbolException

   002：IllegalIntegerException

   003：IllegalIntegerRangeException

   004：IllegalOctalException

   005：IllegalIdentifierLengthException

   006：MismatchedCommentException

   007：MissingLeftParenthesisException

   008：MissingRightParenthesisException

   009：MissingOperatorException

010：MissingOperandException

011：TypeMismatchedException

012：ParameterMismatchedException

013：SyntacticException(其他语法错误，RECORD缺失end)

014：SemanticException(其他语意错误，MODULE前后标识符名称不一致)

015：SyntacticException(其他语法错误，缺失类型声明)

016：SemanticException(其他语意错误，未定义的标识符)

3. **讨论 Oberon-0 语言的特点**

关键字与保留字的区别：

- (这条是根据网络上对其他语言关键字与保留字的描述进行的总结)通常而言，保留字等同于关键字，或是保留字包含了关键字。保留字是指语言已经定义过的，或是为了之后的功能可扩展性而保留的字，用户在定义标识符时，不能再将保留字作为变量名和过程名使用，即定义的标识符不能与保留字同名。关键字是指在语言语法定义中具有特殊含义的字，它在语法中被赋予了含义，是语法的一部分。保留字与关键字主要区别在于保留字可能还包含有语法中没有定义的字，这些字是为了后续的扩展而进行保留的，这些字在之后的升级版本可能会被视为关键字。从实际上看，在词法规则定义中，我们需要为每个关键字都单独地建立一个转换图，而保留字的未定义部分只需要填入符号表中，而不需要建立转换图
- 对于Pascal、Oberon-0语言而言，保留字与关键字的主要区别在于：保留字是语言的一部分，其含义是语言定义好的，在后续标识符定义中，不能使标识符的名称与保留字同名；而关键字是语言预定义的程序流控制或是数据类型等，在后续程序中可以进行重定义，使其具有其他功能

Oberon-0表达式与其他高级语言的不同：

- Oberon-0的表达式中，运算的优先级与其他高级语言有所不同。在EBNF定义中，逻辑运算的优先级大于关系运算的优先级，这就导致了形如1<2 & 2<3这样的表达式在语言中会出现语义错误。另外，逻辑运算和算数运算的优先级是一致的，因此在设计程序时要注意根据语义添加上括号，否则容易出现类型不匹配的错误。
- Oberon-0的表达式中，部分运算符的表示与高级语言有所不同。如通常高级语言的求余为%号，而Oberon-0的求余为MOD，另外，除法、逻辑运算的运算符也有所不同
- 在Oberon-0表达式中，正、负号只能出现在一个simple_expression的开头，且优先级与加减号相同。即在算术运算中，正负号只能出现在开头，避免歧义，不能出现形如1+3*-2这类表达式。

更详细地讲，除了表达式外，Oberon-0语言的语法规则与其他高级语言有诸多不同

- IF语句和WHILE语句中都需要用END结尾标识结束。更通用地讲，每一个包含有语句序列的模块都需要用END标识结尾
- 支持过程的嵌套定义
- 不支持FOR循环和DO-WHILE循环
- 对于每一段statement_sequence，最后一个语句的最后不需要用分号;进行分隔，而C等高级语言每个语句都需要用分号标志结束
- Oberon-0支持ARRAY和RECORD数据类型，允许自定义类型(用某个标识符代替类型，与#define宏定义类似)
- Oberon-0程序中，定义与语句是完全分离的，每个模块或过程声明中都是先进行定义再执行语句，而不能将变量定义穿插在语句中
- Oberon-0不支持任何的类型转换，不支持函数声明(只有过程声明)

4. **讨论 Oberon-0 文法定义的二义性**

   我认为Oberon-0文法中已没有了二义性，它解决了其他高级程序设计语言中常见的二义性问题，主要为：

   - 悬挂else(Dangling else)问题：通常的高级程序设计语言中，往往会由于IF-ELSE语句中嵌套了IF语句，导致ELSE语句匹配具有二义性，如

     ```
     if(3>0)
         if(4>0) printf("1");
         else print("2");
     ```

     通常在这些高级程序设计语言中，采用最近匹配原则解决这个问题，即else与最近的if进行匹配。我们看在Oberon-0中相同含义的语句

     ```
     IF 3>0 THEN
         IF 4>0 THEN
             WRITE(1)
         ELSE
             WRITE(2)
         END
     END
     ```

     可以看到对于每一个IF语句，最后都需要跟一个END表示语句的结束，这就使得如果ELSE语句在END之前出现，那么它会属于内部的IF语句；如果ELSE语句在END之后出现，它属于外部的IF语句，这就解决了悬挂else的问题

   - 表达式的二义性问题。对于高级程序设计语言而言，解决表达式的二义性问题往往是根据优先级，在出现两种可能的action时选择其中一个合理的action来继续执行语法分析，而不是直接在文法中解决二义性问题，因为这会增加编译的复杂性。而在Oberon-0中，在文法层面就直接定义了二义性问题。通过增加非终结符号，并将不同优先级的运算符放在不同层次的非终结符号的产生式中，使得文法中直接体现了运算符的优先级，例如

     ```
     simple_expression = ["+" | "-"] term {("+" | "-" | "OR") term} ;
     term = factor {("*" | "DIV" | "MOD" | "&") factor} ;
     ```

     这里显式的定义了*、DIV、MODE和&优先级高于+、-、OR，因为非终结符term作为了一个整体出现在了simple_expression，因此在规约simple_expression之前，term已经被规约，作了运算。通过显式地在文法中定义运算符优先级，Oberon-0解决了表达式的二义性问题