# 实验一、熟悉 Oberon-0 语言定义



## 实验步骤 1.1、编写一个正确的 Oberon-0 源程序

*要求：覆盖了Oberon-0 语言提供的模块、声明（类型、常量、变量等）、过程声明与调用、语句、表达式等各种构造。*

```
Module SomeFuncs;
    PRICEDURE Factorial;
    VAR
        n, i, result: INTEGER;
    BEGIN
        i := 0;
        result := 1;
        Read(n);
        IF n = 0 THEN result := 0 END;
        WHILE i < n DO
            i := i + 1;
            result := i * result;
        END;
        Write(result);
        WriteLn;
    END Factorial;

    PRICEDURE StudentsInfo;
    TYPE
        ID = INTEGER;
        Info = RECORD
            userID : ID;
            age: INTEGER;
        END;
    VAR
        studentsList: ARRAY 10 OF Info;
        i : INTEGER;
    BEGIN
        i := 0;
        WHILE i < 10 DO
            studentsList[i].userID := 15331260;
            studentsList[i].age := 20;
            Write(studentsList[i].userID);
            WriteLn;
            Write(studentsList[i].age);
            WriteLn;
        END;
    END StudentsInfo;
```

# 实验步骤 1.2、编写上述 Oberon-0 源程序的变异程序

由于变异程序种类过多，所以在实验报告中不作书写，各种不同的错误根据下表查询子目录，例如第一个错误对应文件*w1.obr* 第二个错误对应文件*w2.obr*，以此类推。

## 错误：

1. **IllegalSymbolException**
   当识别一个单词时遇到不合法的输入符号(譬如@、$等符号)则抛出该异常。
2. **IllegalIntegerException**
   当整数常量(无论是十进制还是八进制)与其后的标识符之间无空白分隔时抛出该异常。
3. **IllegalIntegerRangeException**
   当识别出的整数常量(无论是十进制还是八进制)大于本文档约定的整数常量值最大限制(12)时抛出此异常。
4. **IllegalOctalException**
   当 0 开头的整数常量中含有 0~7 之外的符号(包括 8 和 9)时抛出该异常。
5. **IllegalIdentifierLengthException**
   当识别出的一个标识符长度超过最大限制(24)时抛出该异常。
6. **MismatchedCommentException**
   当“(”*开头的注释直至扫描到最后一个符号都找不到配对的*“)”时抛出该异常。
7. **MissingLeftParenthesisException**
   当分析到左右圆括号不匹配、且缺少左括号时则抛出该异常。
8. **MissingRightParenthesisException**
   当分析到左右圆括号不匹配、且缺少右括号时则抛出该异常。
9. **MissingOperatorException**
   异常 当分析到程序中缺少运算符时，或调用预定义函数缺少相应的参数时则抛出该异常。
10. **MissingOperandException**
    异常 当分析到程序中缺少操作数时则抛出该异常。
11. **TypeMismatchedException**
    当分析到表达式、赋值语句、或参数传递等构造中出现类型不兼容错误时则抛出该异常。

12. **ParameterMismatchedException**
    当分析到过程调用的实际参数数目与过程声明的形式参数数目不一致时则抛出该异常。注意，当实际参 数的数目与形式参数一致，只是存在某一参数的类型不兼容时，应抛出 TypeMismatchedException 异常而不是抛出本异常。

# 实验步骤 1.3、讨论 Oberon-0 语言的特点

## 1.大小写无关

Oberon-0 语言是大小写无关的，所以我们在编写程序时无须考虑大小写的影响，这给我们带来便利，也显示了语言的整洁性。但由于编程习惯的影响以及考虑到大小写影响代码的可读性的问题，在编写程序中还是会自觉地遵循一些过往的习惯对保留字采取了大写，变量名采取驼峰式去编程。

## 2.保留字与关键字的区分

在 Oberon-0语言中，与其前辈 Pascal 语言一样区别了保留字（Reserved Word）与关键字（Keyword） 两个概念。观察其保留字（IF、THEN、ELSIF 等）的作用可知，保留字的作用主要体现在程序语言的层次架构上，偏重于对程序块进行组织的功能。而关键字（INTEGER、WRITE、WRITELN）更多是用在实现程序的功能上，提供一些预定义的变量和函数。

另外，在实验二中构建词法表的时候发现，上述3个关键字均没有在EBNF中出现过，所以个人认为保留字是语言一开始在EBNF中就定义好的，而关键字要在词法分析的时候才能得到检测。

## 3.与 Java、C/C++等常见语言的表达式有何不同之处

首先，在限制整数常量允许表达范围的最大值的方法上有所不同，常见高级语言通常将此于2进制的存储模式相关联，而oberon-0则是限定了整数中数字的个数去限制表达范围，这有助于初学者理解使用。

其次，在声明变量类型时，oberon-0在标识符出现后才给出变量类型，而c语言等一般会在标识符出现前就给出类型。
再者，oberon-0在结构上采用
begin，end的结构，尽管清晰明了，但实际操作中造成编写麻烦，而c语言等则更多使用括号实现，给程序员带来更多便利。

# 实验步骤 1.4、讨论 Oberon-0 文法定义的二义性

显然，根据实验文档的介绍，该文法是无二义性的。

其实文法的设计上很好的避免了二义性的问题，使得文法无需使用算符优先表去处理一些优先级一级存在的二义行问题。同时利用扩展BNF的符号简化语言的设计。

对于优先级的的考虑，文法通过将不同优先级的运算符分开处理，使得 `*` 、 `DIV` 、 `MOD` 、 `|` 、 `&` 在 `+` 、 `-` 前就进行了规约，避免了二义性的问题，具体方法如下：

```
actual_parameters       =   "(" [expression {"," expression}] ")" ;

assignment              =   identifier selector ":=" expression ;

expression              =   simple_expression
                            [("=" | "#" | "<" | "<=" | ">" | ">=")
                            simple_expression] ;

simple_expression       =   ["+" | "-"] term {("+" | "-" | "OR") term} ;

term                    =   factor {("*" | "DIV" | "MOD" | "&") factor} ;
```

其中，由于使用EBNF，通过括号 `[]` 的使用，使得一元云算符与二元运算符的二义性问题得以简单解决。可见，由于EBNF的使用以及层次化的表达式定义，不存在优先级的二义性问题。

而对于语言结构上的二义性（if、else一类），文法通过引入END标识符，使得存在于c语言等的二义性不存在改文法，具体可见：

```
if_statement              =   "IF" expression "THEN"
                                 statement_sequence
                              {"ELSIF" expression "THEN"
                                 statement_sequence}
                              ["ELSE"
                                 statement_sequence]
                              "END" ;
```

通过引入 `begin` 和 `end` 使得文法设计更为简单。

# 实验心得体会

本次实验后，对关键字与保留字的区分上有了更多的了解，以往一向不作区分，就像编写c语言一样，怎么简单就怎么写，语言也没作区分，所以一直对该部分的定义模糊不清，但依然是觉得网上资料太少，大部分基于高级程序设计语言去解释，所以还是有一些概念上的模糊。

其次，由于实验初期无法对Oberon进行编译，无法知道自己的实验结果的正确性，很多步骤都需要后续实验弥补，给实验带来了不少的困难。

再者，通过学习本文法，更深入地了解到了无二义性文法与有二义性文法的区别，结合上一实验中opp应用的对比，可谓受益匪浅。