熟悉 Oberon-0 语言定义

编写一个正确的 Oberon-0源程序

程序主体内容:

1. 提示用户输入`n`和`r`的值。
2. 调用`Combination`过程计算组合数`C(n, r)`，其中使用到阶乘过程`Factorial`，后来发现数据类型受限且不使用RETURN返回计算结果，不能使用FOR循环，选择分别输出分子以及分母的乘子。
3. 计算从`0`到`maxNumber-1`的所有整数的阶乘，并存储在`factorials`数组中。
4. 调用`PrintArray`过程输出阶乘数组。

Oberon-0程序实现

```
MODULE FactorialExample;

CONST
  maxNumber = 10;

TYPE
  intArray = ARRAY maxNumber OF INTEGER;
  bool = BOOLEAN;

VAR
  n, r: INTEGER;
  factorials: intArray;

(* 计算整数 n 的阶乘 *)
PROCEDURE Factorial(n: INTEGER);
VAR
  result, i: INTEGER;
BEGIN
  result := 1;
  i := 1;
  WHILE i <= n DO
    result := result * i;
    i := i + 1
  END;
  WRITE(result);
END Factorial;

(* 计算组合数 C(n, r) = n! / (r! * (n - r)!) *)
PROCEDURE Combination(n, r: INTEGER);
BEGIN
  Factorial(n);
  Factorial(r);
  Factorial(n - r);
END Combination;

(* 打印数组内容 *)
PROCEDURE PrintArray(arr: intArray; size: INTEGER);
VAR
```

```
  i: INTEGER;
BEGIN
  i := 0;
  WHILE i < size DO
      Factorial(arr[i]);
  END;
END PrintArray;

BEGIN
  (* 输入 n 和 r *)
  Read(n);
  Read(r);

  (* 计算并输出组合数 C(n, r) *)
  Combination(n, r);

  (* 存储从 0 到 maxNumber-1 的阶乘的输入 *)
  n := 0;
  WHILE n < maxNumber DO
        factorials[n] := n + 1
  END;

  (* 输出阶乘数组 *)
  PrintArray(factorials, maxNumber)

END FactorialExample.
```

模块声明：定义了名为`FactorialExample`的模块。

常量声明：

- `maxNumber`：表示可以计算的最大数的阶乘，设置为10。

类型声明：

- `intArray`：定义一个整数数组类型，大小为`maxNumber`。
- `bool`：定义一个布尔类型别名。

变量声明：

- `n`, `r`, `factorialResult`, `combinationResult`：整数变量，用于存储用户输入和计算结果。
- `factorials`：整数数组，用于存储从0到`maxNumber-1`的阶乘结果。

过程声明：

- `Factorial`：计算整数`n`的阶乘。
- `Combination`：计算组合数`C(n, r)`。
- `PrintArray`：打印整数数组的内容。

表达式：

- `WHILE i <= n DO`：条件表达式

编写 Oberon-0 源程序的变异程序

词法错误

- **IllegalSymbolException**：非法符号异常

```
# factorial.001
# 正确写法
CONST
  maxNumber = 10;
# 错误写法
CONST
  maxNumber@ = 10;
```

- **IllegalIntegerException**：非法整数异常

```
# factorial.002
# 正确写法
CONST
maxNumber = 10;
# 错误写法
CONST
maxNumber = 1A;
```

- **IllegalIntegerRangeException**：非法整数范围异常，表示整数的值超出了允许的范围

```
# factorial.003
# 正确写法
result := 1;
# 错误写法
result := 1234567890123;
```

- **IllegalOctalException**：八进制数异常

```
# factorial.004
# 正确写法
result := 1;
# 错误写法
result := 01289;
```

- **IllegalIdentifierLengthException**：非法标识符长度异常

```
# factorial.005
# 正确写法
VAR
  result, i: INTEGER;
# 错误写法
VAR
  resultaaaaaaaaaaaaaaaaaaaaaaaa, i: INTEGER;
```

- **MismatchedCommentException**：注释不匹配异常

```
# factorial.006
# 正确写法
(* 计算整数 n 的阶乘 *)
# 错误写法
 计算整数 n 的阶乘 *)
# 注意：这里如果写的是"(* 计算整数 n 的阶乘"且之后还有注释的话,(*会选择与下一个注释的*)匹配,因而不会报错
```

语法错误

- **MissingRightParenthesisException**：缺少右括号异常

```
# factorial.007
# 正确写法
WRITE(result);
# 错误写法
WRITE(result ;
```

- **MissingLeftParenthesisException**：缺少左括号异常

```
# factorial.008
# 正确写法
PROCEDURE Factorial(n: INTEGER): INTEGER;
# 错误写法
PROCEDURE Factorial n: INTEGER): INTEGER;
```

- **MissingOperatorException**：缺少运算符异常

```
# factorial.009
# 正确写法
i := i + 1
# 错误写法
i := i  1
```

- **MissingOperandException**：缺少操作数异常

```
# factorial.010
# 正确写法
i := i + 1
# 错误写法
i := i +
```

语义错误

- **ParameterMismatchedException**：参数不匹配异常

```
# factorial.011
# 正确写法
Combination(n, r);
# 错误写法
Combination(n);
```

- **TypeMismatchedException**：类型不匹配异常

```
# factorial.012
# 正确写法
VAR
  result, i: INTEGER;
BEGIN
  result := 1;
# 错误写法
VAR
  result, i: INTEGER;
BEGIN
  result := (i < 10);
```

讨论 Oberon-0 语言的特点

保留字与关键字

1. 保留字：编程语言中已经预先定义并且具有特殊含义的单词，视为语言的组成部分，而不是普通的标识符或符号，保留字是语言的基础构建块，直接影响程序的结构和逻辑
2. 关键字：具有特殊含义的标识符，指定特定的操作、功能或语法结构，更加灵活

关键字在使用时需要注意避免与保留字冲突

语法规则

- 赋值语句
  - Oberon-0：使用`variable := expression;`
  - Java和C/C++：使用`variable = expression;`
- 数据类型声明
  - Oberon-0：`VAR variable : type;`
  - Java和C/C++：`type variable;`
- 条件表达式的语法
  - Oberon-0：通常使用`IF condition THEN statement ELSE statement END;`，即条件后跟THEN和ELSE分支，END结束
  - Java和C/C++：使用`if (condition) { statement } else { statement }`（Java）或`if (condition) { statement } else { statement }`（C/C++），即条件用圆括号括起来，语句放在大括号内
- 循环语句
  - Oberon-0：使用`WHILE condition DO statement END;`，即条件后跟DO和END
  - Java和C/C++：使用`while (condition) { statement }`（Java）或`while (condition) { statement }`（C/C++），也是条件用圆括号括起来，语句放在大括号内
- 布尔表达式
  - Oberon-0：使用关键字`TRUE`和`FALSE`表示
  - Java和C/C++：使用关键字`true`和`false`表示

讨论 Oberon-0 文法定义的二义性

Oberon-0语言的语法规则没有二义性，其中的赋值语句、条件语句、循环语句等都有明确的开始和结束标记，操作符的优先级和结合性也是明确的。

问题：为何在其他高级程序设计语言中常见的那些二义性问题在 Oberon-0 语言中并未出现？

Oberon-0语言的语法规则相对简洁，清晰，减少了不必要的歧义；在定义操作符优先级和结合性时比较严格，确保了表达式的解析顺序是可预测的；Oberon-0语言的语句和表达式都有明确的开始和结束符，有助于编译器或解释器准确地划分语法单元；对标识符的命名规则进行了规范，避免了在标识符中使用特殊字符或关键字，减少了表达式解析过程中的二义性