剑桥伪代码讲义

中文版

Contents

1 书写风格与排版 **3**
- 1.1 Keywords and case . . . 3
- 1.2 Identifiers . . . 3
- 1.3 Comments and indentation . . . 4

2 变量、常量与数据类型 **6**
- 2.1 The five data types . . . 6
- 2.2 DECLARE . . . 7
- 2.3 CONSTANT . . . 7

3 赋值、输入与输出 **9**
- 3.1 The assignment arrow . . . 9
- 3.2 INPUT and OUTPUT . . . 10

4 算术与逻辑 **11**
- 4.1 Arithmetic operators . . . 11
- 4.2 DIV and MOD . . . 11
- 4.3 Relational and logic operators . . . 12

5 字符串操作 **14**
- 5.1 LENGTH, UCASE and LCASE . . . 14
- 5.2 SUBSTRING . . . 15

6 选择结构 **16**
- 6.1 IF, THEN, ELSE, ENDIF . . . 16
- 6.2 CASE OF . . . 17

7 循环结构 **19**
- 7.1 FOR, the counted loop . . . 19
- 7.2 WHILE, the pre-condition loop . . . 20
- 7.3 REPEAT, the post-condition loop . . . 21

8 数组 **22**
- 8.1 One-dimensional arrays . . . 22
- 8.2 Two-dimensional arrays . . . 23

9 过程与函数 **25**
- 9.1 PROCEDURE and CALL . . . 25
- 9.2 FUNCTION and RETURN . . . 25
- 9.3 Parameters . . . 27

10 文件处理 **28**

10.1 Writing to a file 28
10.2 Reading a file to the end 29

11 A-Level 9618: 有何不同 **31**
11.1 Layout and operators 31
11.2 Parameters: BYVAL and BYREF 32
11.3 Records and other types 33

1 书写风格与排版

1.1 ■ Keywords and case

伪代码伪代码是剑桥考试局编写算法算法所使用的语言。它描述算法而无需使用通用编程语言。本网站解析一种定义的剑桥风格子集，以便您运行示例；考试作答应遵循该试卷指定的记法。

每个关键字都以**大写字母**书写：IF、REPEAT、PROCEDURE、OUTPUT。

```
DECLARE Count : INTEGER
Count ← 3
IF Count > 0
  THEN
    OUTPUT "Ready"
ENDIF
```

- 请写 IF，永远不要写 if 或 If。
- 关键字绝不能用作自定义名称：你不能将变量命名为 FOR。

常见错误

- 小写关键字。评分标准预期使用大写形式。
- 自创关键字。只有本讲义中列出的词汇才在考纲范围内。

1.2 ■ Identifiers

标识符是用于命名变量、常量、过程或函数的名称。函数。

```
DECLARE NumberOfPlayers : INTEGER
NumberOfPlayers ← 4
OUTPUT NumberOfPlayers
```

四条规则均出自考纲：

- 必须以**大写字母开头**：例如 Count，而非 count。
- 仅可包含字母和数字 — 不允许空格或标点符号。
- 不区分大小写，因此 Count 与 COUNT 被视为同一名称。请选定一种拼写方式并保持一致。选择一种拼写并始终如一地使用。
- 应使其具有表意功能。NumberOfPlayers 能让读者迅速理解含义；而 N 则不能。

常见错误

- count ← 0 — 以首字母小写开头。0478 要求首字母大写。
- Number of players — 名称内部不允许出现空格。

1.3 ■ Comments and indentation

注释以 // 开头，延续至行末。它是给人阅读的，算法会忽略它。读者；算法会忽略它。

```
// Swap the values of X and Y
DECLARE X, Y, Temp : INTEGER
X ← 1
Y ← 2
Temp ← X   // temporarily store X
X ← Y
Y ← Temp
OUTPUT X, " ", Y
```

缩进表示语句位于某个结构**内部**。考纲要求每级缩进四个空格，但有一个例外你必须掌握：每个层级四格，但你必须知道的一个例外情况：

```
DECLARE X, Y : INTEGER
X ← 7
Y ← 3
IF X > Y
  THEN
    OUTPUT "X is bigger"
  ELSE
    OUTPUT "Y is bigger or equal"
ENDIF
```

- THEN 和 ELSE 相对于其对应的 IF 缩进**两格**。
- 它们下方的语句缩进**四格**。
- ENDIF 与其自身的 IF 对齐。

```
IF X > Y
  THEN                                  two spaces in from IF
    OUTPUT "X is bigger"                four spaces: inside the branch
  ELSE
    OUTPUT "Y is bigger or equal"
ENDIF                                   lines up with its own IF
```

数空格数量：*0*，*2*，*4*

常见错误

- 将 `THEN` 放在 `IF` 行。这是 A-Level 9618 的排版格式 — 参见主题 11。
- 完全不缩进。评分员必须能看到结构才能给出结构分。

2 变量、常量与数据类型

2.1 ■ The five data types

0478 伪代码恰好有**五种**数据类型。请熟记列表 — 未列出名称的类型不得分不在其上的内容将被扣分。

类型	容纳	例子
INTEGER	整数	42、-7
REAL	可能带有小数部分的数字小数	3.14, -0.5
CHAR	单个字符	'A'
STRING	字符串	"Hello"
BOOLEAN	TRUE 或 FALSE（布尔值）	TRUE

```
DECLARE Age : INTEGER
DECLARE Price : REAL
DECLARE Grade : CHAR
DECLARE Name : STRING
DECLARE Passed : BOOLEAN
Age ← 17
Price ← 2.50
Grade ← 'A'
Name ← "Mei"
Passed ← TRUE
OUTPUT Name, " is ", Age, " and passed: ", Passed
```

- STRING 用**双引号**书写，CHAR 用**单引号**书写。
- TRUE 和 FALSE 是关键字，因此需大写且不加引号。

常见错误

- 不要写 FLOAT、DOUBLE 或 BOOL。这些是真实编程语言，而非本考纲内容。
- 用 'Hello' 表示字符串 — 单引号仅能包含一个字符。



五种数据类型：*INTEGER*、*REAL*、*CHAR*、*STRING* 和 *BOOLEAN*

2.2 ■ DECLARE

DECLARE 用于声明变量并在使用前固定其类型。

```
DECLARE <identifier> : <data type>
```

```
DECLARE Counter : INTEGER
DECLARE TotalToPay : REAL
Counter ← 1
TotalToPay ← 19.99
OUTPUT "Item ", Counter, " costs ", TotalToPay
```

同一类型的多个变量名可写在同一行，用逗号分隔：

```
DECLARE RoomLength, Width, Area : INTEGER
RoomLength ← 6
Width ← 4
Area ← RoomLength * Width
OUTPUT "Area = ", Area
```

- 在算法顶部附近，于变量首次使用**之前**进行声明。
- 变量在整个算法中保持其类型 — 一个 INTEGER 永远不会持有"Mei"。

常见错误

- 省略冒号：DECLARE Counter INTEGER。
- 使用未声明的变量。考试要求写出声明语句。



DECLARE 用于命名变量并确定其可存储值的类型

2.3 ■ CONSTANT

常量常量是一个在整个算法中固定不变的值 — 税率、π、班级人数等。只需命名一次，修改时只需更改一处即可。

```
CONSTANT <identifier> ← <value>
```

```
CONSTANT TaxRate ← 0.20
DECLARE Price, Tax : REAL
Price ← 50.00
Tax ← Price * TaxRate
OUTPUT "Tax to pay: ", Tax
```

- 在 **0478 中使用箭头符号**，与赋值操作完全相同。
- 该值必须是**字面量**字面量 — 数字、字符串、TRUE— 绝不能是计算表达式，也绝不能是另一个变量。
- 常量之后绝不允许被赋予新值。这正是常量的意义所在。

常见错误

- `CONSTANT TaxRate = 0.20`。= 形式是 A-Level 9618 拼写方式 — 参见主题 11。
- 在算法后续部分对常量进行赋值。

3 赋值、输入与输出

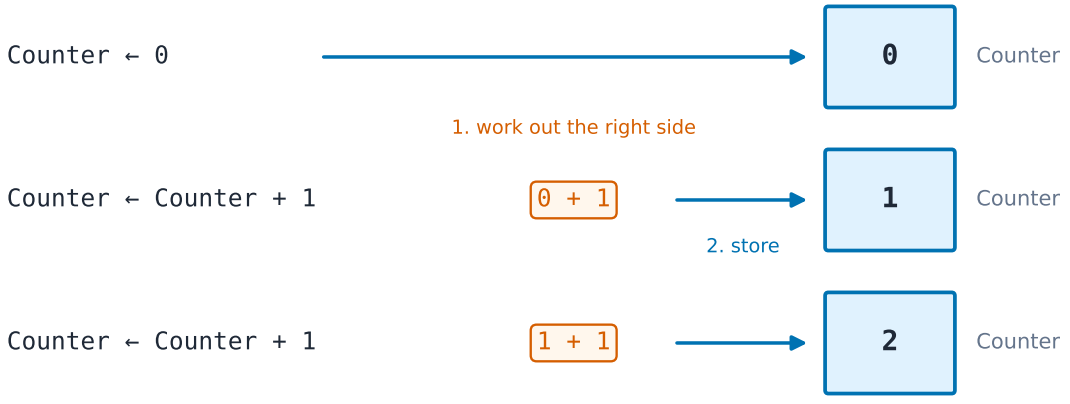
3.1 ■ The assignment arrow

赋值赋值运算符是 ←，即左向箭头。它表示取右侧的值放入左侧的框中。右侧先计算，然后存入框中。

```
<identifier> ← <value>
```

```
DECLARE Counter : INTEGER
Counter ← 0
Counter ← Counter + 1
Counter ← Counter + 1
OUTPUT Counter
```

`Counter ← Counter + 1` 作为数学式子看似不可能，但作为指令却非常自然：用当前框中的值计算右侧结果，再将结果存回框中。



右侧先计算，然后存入框中

- 如果你无法输入 ←，请写 `<-`。两者均可接受，此处及考试中均有效。
- 左侧必须是一个**单一标识符**或一个数组元素。绝不能是计算表达式。

常见错误

- `Counter = 0`。在伪代码中，`=` 表示**比较**而非赋值 — 容易混淆。将二者混淆是本论文中最常见的语法错误。
- `Counter + 1 ← Counter`。箭头指向被修改的对象。

3.2 ■ INPUT and OUTPUT

INPUT 从用户处读取值存入变量。OUTPUT 显示一个或多个值。

```
DECLARE Name : STRING
DECLARE Age : INTEGER
INPUT Name
INPUT Age
OUTPUT "Hello ", Name
OUTPUT "Next year you will be ", Age + 1
```

- INPUT 恰好需要**一个**变量，且该变量必须已被声明。
- OUTPUT 可接受任意数量的项目，由**逗号**分隔；它们依次打印彼此之间无任何间隔。
- 字面量字符串 literal string 在 OUTPUT 中使用双引号；变量则不需要。

3.2.1 提示用户

单独的 INPUT 在屏幕上不会显示任何内容，因此良好的算法应先输出提示信息：

```
DECLARE Radius : REAL
OUTPUT "Enter the radius: "
INPUT Radius
OUTPUT "The diameter is ", Radius * 2
```



提示先出现，然后是输入的文本，最后是回复

常见错误

- INPUT "Enter a number", X— 提示本身是一个独立的 OUTPUT 语句。
- OUTPUT "Total: " + Total。0478 用逗号连接输出项；+ 属于算术运算，而 & 归属于 A-Level 9618（主题 11）。
- 忘记加引号时：OUTPUT Hello 会将其视为名为 Hello 的变量来查找。

4 算术与逻辑

4.1 ■ Arithmetic operators

五个算术算术运算符；它们的优先级列于下表下方。

运算符	含义	A 等于 7，B 等于 2
+	add 加	A + B is 9
-	减法	A - B 等于 5
*	multiply 乘	A * B is 14
/	divide 除	A / B is 3.5
^	raise to a power 幂	A ^ B is 49

```
DECLARE A, B : INTEGER
A ← 7
B ← 2
OUTPUT "Sum: ", A + B
OUTPUT "Product: ", A * B
OUTPUT "Quotient: ", A / B
OUTPUT "Power: ", A ^ B
```

括号最先，然后是 ^，接着是 * 和 /，最后是 + 和 - – 即通常的优先级顺序优先级。每当读者需要停下来思考时，请使用括号。

```
DECLARE Total : REAL
Total ← (12 + 18) / 2
OUTPUT "Mean: ", Total
```

常见错误

- 期望 / 给出一个整数。7 / 2 是 3.5，而不是 3；若要得到整数，请使用 DIV.
- 写入 A x B 或 A ÷ B。请使用 * 和 /。

4.2 ■ DIV and MOD

整数除法整数除法回答两个不同的问题，且 0478 将每个问题分别表示为一个 **function** 函数，并在方括号中列出这两个值。

- DIV(a, b) – b 在 a 中能完整包含多少次，即 **quotient** 商。
- MOD(a, b) – 剩余部分是多少，即 **remainder** 余数。

```
OUTPUT DIV(10, 3)
OUTPUT MOD(10, 3)
OUTPUT DIV(17, 5)
OUTPUT MOD(17, 5)
```

两者均接受整数作为输入，并返回整数结果。



*17 分成每组 5 个：整除得到 3 个完整组，取余得到 2 个剩余项

4.2.1 它们的用途

MOD 是检验 整除的标准方法，因此可以用来判断一个数是否为偶数偶数：

```
DECLARE Number : INTEGER
Number ← 12
IF MOD(Number, 2) = 0
  THEN
    OUTPUT Number, " is even"
  ELSE
    OUTPUT Number, " is odd"
ENDIF
```

DIV 可将总数转换为整数单位 – 例如秒转分钟、便士转英镑：

```
DECLARE Seconds : INTEGER
Seconds ← 200
OUTPUT DIV(Seconds, 60), " minutes and ", MOD(Seconds, 60), " seconds"
```

常见错误

- 10 MOD 3. 这是 A-Level 9618 格式；**0478 使用函数形式** MOD(10, 3)。
- 混淆两者。DIV 给出整数部分；MOD 给出不符合的部分。

4.3 ■ Relational and logic operators

一个 **关系运算符**比较两个值并产生 TRUE 或 FALSE。

运算符	含义
=	等于
<>	不等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于

```
DECLARE Mark : INTEGER
Mark ← 55
OUTPUT "Pass: ", Mark >= 50
OUTPUT "Perfect: ", Mark = 100
OUTPUT "Not zero: ", Mark <> 0
```

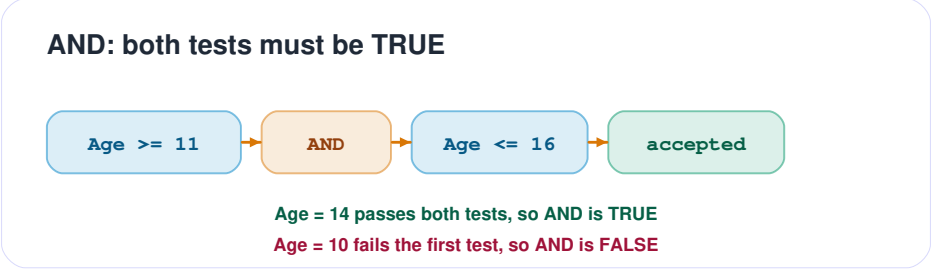
三个 **逻辑运算符**连接条件：

- AND – 两侧都必须是 TRUE。
- OR – 至少一侧必须是 TRUE。
- NOT – 将 TRUE 转换为 FALSE 再转换回来。

```
DECLARE Age : INTEGER
DECLARE HasTicket : BOOLEAN
Age ← 16
HasTicket ← TRUE
IF Age >= 15 AND HasTicket = TRUE
  THEN
    OUTPUT "Admitted"
ENDIF
IF NOT (Age > 18)
  THEN
    OUTPUT "Still a student rate"
ENDIF
```

常见错误

- <> 在此表示 “ 不等于 ” 符号 – 不是 !=，后者属于其他语言。
- IF Age >= 15 AND <= 18。每个 AND 的两侧都需要各自完整的比较：Age >= 15 AND Age <= 18.
- 在答案中称这些为 比较运算符。教学大纲中的术语是 **关系**。



AND 需要两个测试都为 *TRUE*：年龄 *14* 通过两项，年龄 *10* 未通过第一项

5 字符串操作

5.1 ■ LENGTH, UCASE and LCASE

0478 发布四个字符串例程，除此之外没有其他。其中三个在这里。

LENGTH(<identifier>) 返回字符串包含的字符数量 — 包括空格。

```
OUTPUT LENGTH("Happy Days")
OUTPUT LENGTH("")
```

UCASE(<identifier>) 将字符串转换为大写，LCASE(<identifier>) 转换为小写。两者都不会修改原始值 — 每个函数都会返回一个新值。

```
DECLARE Name : STRING
Name ← "Happy"
OUTPUT UCASE(Name)
OUTPUT LCASE(Name)
OUTPUT Name
```

最后一行仍然输出 Happy：若要保留更改，必须将其赋值回原变量。

```
DECLARE Name : STRING
Name ← "Happy"
Name ← UCASE(Name)
OUTPUT Name
```

5.1.1 忽略大小写进行比较

在两侧使用 UCASE 是接受任何输入方式答案的标准方法：

```
DECLARE Answer : STRING
Answer ← "yes"
IF UCASE(Answer) = "YES"
  THEN
    OUTPUT "Accepted"
ENDIF
```

常见错误

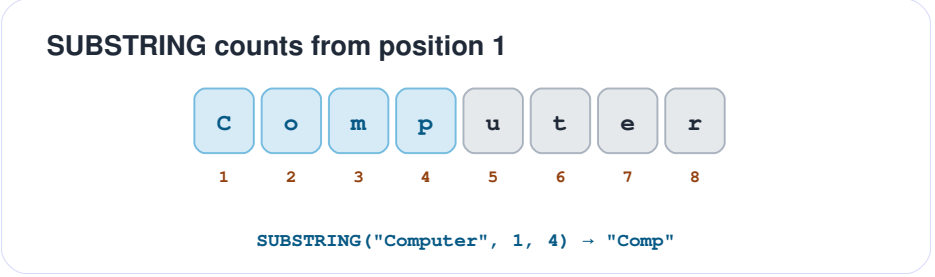
- 期望 UCASE(Name) 改变 Name。需将结果重新赋值。
- LENGTH 计算的是字符数而非单词数。LENGTH("Happy Days") 是 10。

5.2 ■ SUBSTRING

SUBSTRING(<identifier>, <start>, <length>) 从字符串中提取一段 — 起始于位置 start，取 length 个字符。

```
OUTPUT SUBSTRING("Happy Days", 1, 5)
OUTPUT SUBSTRING("Happy Days", 7, 4)
```

△ **计数从 1 开始**，而非 0。"Happy Days" 的第一个字符位于位置 1。



字符从 *1* 开始计数：*SUBSTRING(“Computer”, 1, 4)* 结果为 *“Comp”*

5.2.1 提取单个字符及剩余部分

```
DECLARE Word : STRING
Word ← "Computer"
OUTPUT "First letter: ", SUBSTRING(Word, 1, 1)
OUTPUT "Last letter: ", SUBSTRING(Word, LENGTH(Word), 1)
OUTPUT "Without the first: ", SUBSTRING(Word, 2, LENGTH(Word) - 1)
```

LENGTH 嵌套在 SUBSTRING 中，使得最后两行适用于任意长度的单词 — 硬编码 8 会在单词改变时立即失效。

5.2.2 逐字符读取字符串

```
DECLARE Word : STRING
DECLARE Index : INTEGER
Word ← "Cat"
FOR Index ← 1 TO LENGTH(Word)
    OUTPUT SUBSTRING(Word, Index, 1)
NEXT Index
```

常见错误

- 从 0 开始。位置 1 是第一个字符。
- 将第三个参数读作 结束位置。它是一个 **计数** — SUBSTRING(W, 2, 3) 从第二个字符起取三个字符。
- 请求的字符数超过字符串实际长度。

6 选择结构

6.1 ■ IF, THEN, ELSE, ENDIF

选择结构根据一个条件执行一组语句或另一组语句。

```
IF <condition>
  THEN
    <statements>
ENDIF
```

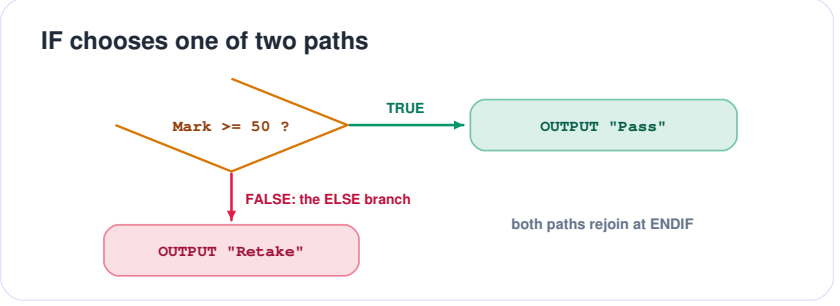
```
DECLARE Mark : INTEGER
Mark ← 72
IF Mark >= 50
  THEN
    OUTPUT "Pass"
ENDIF
```

对另一种情况添加 ELSE。两个分支中恰好只有一个被执行 — 不会同时执行，也不会都不执行。都不会。

```
DECLARE Mark : INTEGER
Mark ← 41
IF Mark >= 50
  THEN
    OUTPUT "Pass"
  ELSE
    OUTPUT "Fail"
ENDIF
```

排版是学生容易扣分的地方，因此请将其视为一种固定格式来记忆：

- IF 及其条件写在同一行，**条件后不留任何内容**。
- THEN 和 ELSE 各占一行，缩进 **两个**空格。
- 语句缩进 **四个**空格。
- ENDIF 回到其对应 IF 所在的列位置。



IF 在两条路径中选择一条；两者在 *ENDIF* 处汇合

6.1.1 嵌套

嵌套在 IF 中的 IF 可处理两个以上结果。每个结果都需要自己的 ENDIF。

```
DECLARE Mark : INTEGER
Mark ← 85
IF Mark >= 80
  THEN
    OUTPUT "Distinction"
  ELSE
    IF Mark >= 50
      THEN
        OUTPUT "Pass"
      ELSE
        OUTPUT "Fail"
    ENDIF
ENDIF
```

常见错误

- 遗漏 ENDIF。每一个 IF 都必须闭合，缺失任何一个都是语法错误，标记会一目了然地显示出来。标记会一目了然地显示出来。
- IF Mark >= 50 THEN OUTPUT "Pass" 写在一行上。**这种形式不存在于任何**课程大纲中；请写成三行。
- 将 IF Mark = 50 THEN 写成 IF Mark ← 50。箭头表示赋值；= 表示比较。

6.2 ■ CASE OF

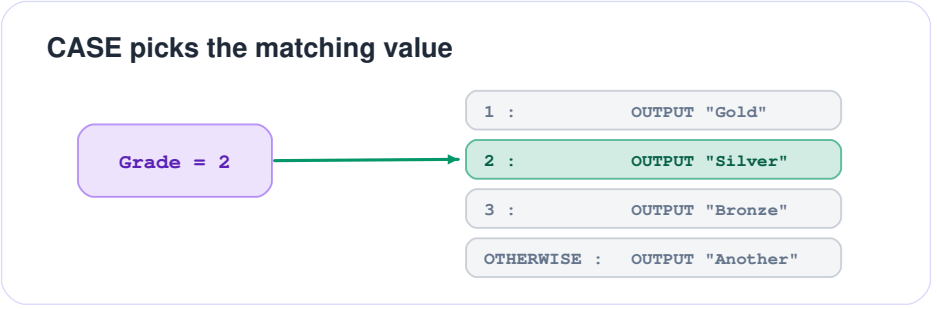
当一个变量与多个**单个值**进行比较时，使用 CASE OF 表述要清晰得多比一叠嵌套的 IFs 更清楚。

```
CASE OF <identifier>
    <value 1> : <statement>
    <value 2> : <statement>
    OTHERWISE <statement>
ENDCASE
```

```
DECLARE Choice : INTEGER
Choice ← 2
CASE OF Choice
    1 : OUTPUT "You chose north"
    2 : OUTPUT "You chose south"
    3 : OUTPUT "You chose east"
    4 : OUTPUT "You chose west"
    OTHERWISE OUTPUT "That is not a direction"
ENDCASE
```

- 每个分支都是一个**值**，后跟冒号，再跟要执行的操作。

- OTHERWISE 捕获所有被列出的值遗漏的内容。它是可选的，写一个是良好的编程习惯 — 这表示当用户输入意外内容时会发生什么。
- 第一个匹配的分支会运行，其余分支将被跳过。



CASE 将值与每个选项逐一比较；*Grade = 2* 匹配第二个

6.2.1 范围和一个字符

```
DECLARE Grade : CHAR
Grade ← 'B'
CASE OF Grade
    'A' : OUTPUT "Excellent"
    'B' : OUTPUT "Good"
    'C' : OUTPUT "Satisfactory"
    OTHERWISE OUTPUT "Unclassified"
ENDCASE
```

常见错误

- 使用 CASE OF 作为**条件**。它对一个变量进行值测试；而 Mark >= 50 需要一个 IF。
- 忘记 ENDCASE。
- 省略 OTHERWISE 然后疑惑为什么意外的值完全没有任何反应。

7 循环结构

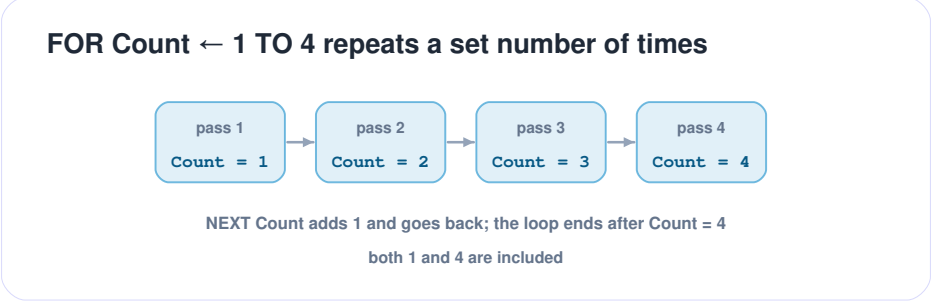
7.1 ■ FOR, the counted loop

迭代重复执行语句。当你**在开始之前就知道**需要循环多少次时，应使用 FOR 循环即计数控制型计数控制循环。

```
FOR <identifier> ← <value1> TO <value2>
    <statements>
NEXT <identifier>
```

```
DECLARE Index : INTEGER
FOR Index ← 1 TO 5
    OUTPUT "Line ", Index
NEXT Index
```

计数器从起始值开始，对于直到并**包含**结束值的每一个数值都会运行一次。`1 TO 5` 运行五次。



FOR Count ← 1 TO 4 运行四次循环，且同时包含 *1* 和 *4*

7.1.1 STEP（步长）

STEP 改变每次跳跃的大小。负步长用于递减计数。

```
DECLARE Index : INTEGER
FOR Index ← 2 TO 10 STEP 2
    OUTPUT Index
NEXT Index
FOR Index ← 3 TO 1 STEP -1
    OUTPUT "Countdown ", Index
NEXT Index
```

7.1.2 累加求和

最常见的用法：定义一个位于循环外部的变量，由循环对其不断累加。

```
DECLARE Index, Total : INTEGER
Total ← 0
FOR Index ← 1 TO 10
    Total ← Total + Index
NEXT Index
OUTPUT "Sum of 1 to 10 is ", Total
```

`Total ← 0` 必须放在循环**之前**。如果放在内部，每次迭代都会重置。

常见错误

- `NEXT` 命名了错误的计数器。嵌套循环的关闭顺序必须与打开顺序相反：先关闭内层的 `NEXT`。
- 假设 `1 TO 5` 运行四次。包含首尾两端。
- 在循环内部修改计数器。让 `FOR` 负责它。

7.2 ■ WHILE, the pre-condition loop

`WHILE` 循环在每次迭代**之前**测试条件，因此可能一次都不执行。适用于重复次数取决于循环运行过程中发生的事件的情况。

```
WHILE <condition> DO
    <statements>
ENDWHILE
```

```
DECLARE Total : INTEGER
Total ← 1
WHILE Total < 100 DO
    Total ← Total * 2
ENDWHILE
OUTPUT "First power of two past 100: ", Total
```

循环内部的某项操作最终必须使条件变为假，否则循环将永远无法结束 — 即**无限循环** (infinite loop)。

```
DECLARE Count : INTEGER
Count ← 5
WHILE Count > 0 DO
    OUTPUT Count
    Count ← Count - 1
ENDWHILE
OUTPUT "Lift off"
```

常见错误

- 忘记修改变量导致条件测试失败，从而产生无限循环。
- 遗漏了 DO 或 ENDWHILE。
- 在计数已知的情况下使用 WHILE。用 FOR 可以用一行代码清晰表达你的意图。

7.3 ■ REPEAT, the post-condition loop

REPEAT 循环在每次迭代**之后**测试条件，因此至少会执行**一次**。

```
REPEAT
    <statements>
UNTIL <condition>
```

这正是选择它的理由：用于验证输入（validating input），此时你必须先询问才能判断答案是否有效。

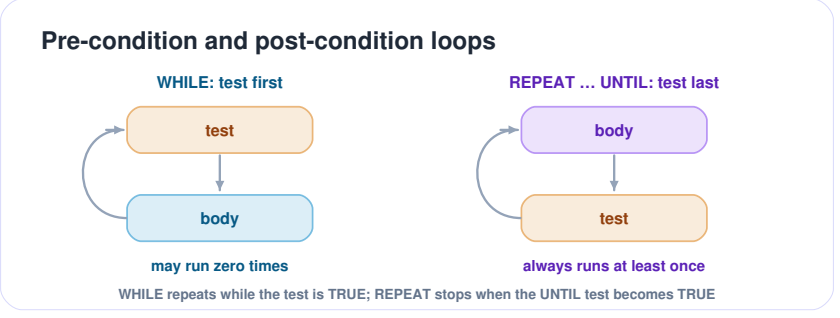
```
DECLARE Number : INTEGER
Number ← 0
REPEAT
    Number ← Number + 25
    OUTPUT "Trying ", Number
UNTIL Number >= 100
OUTPUT "Reached ", Number
```

△ UNTIL 表示**终止**循环的条件；WHILE 表示继续循环的条件 **使其持续运行**。两者互为反义，交换它们是常见的典型错误。

	测试时机	至少执行一次?	条件含义
FOR	计数器	否（空范围执行零次）	–
WHILE	之前	否	当此条件为真时继续执行
REPEAT	after	**是**	当此条件为真时停止

常见错误

- 将 UNTIL Number < 100 误写为 UNTIL Number >= 100。
- 在必须允许执行零次的循环体中使用 REPEAT。
- 忘记 UNTIL 会终止该结构 — 不存在 ENDREPEAT。



WHILE 先测试且可能执行零次；*REPEAT* 后测试且至少执行一次

8 数组

8.1 ■ One-dimensional arrays

数组 array 在一个名称下存储多个同类型的值。每个值位于一个 **下标** index。

```
DECLARE <identifier> : ARRAY[<lower>:<upper>] OF <data type>
```

```
DECLARE Names : ARRAY[1:3] OF STRING
Names[1] ← "Mei"
Names[2] ← "Sam"
Names[3] ← "Ana"
OUTPUT Names[2]
```

- 边界 bounds 写作 `lower:upper`，**两者均包含在内** —`[1:3]` 包含三个元素 element。
- Cambridge 数组通常从 **1** 开始，而非 0。
- 每个元素类型相同，由 `OF` 确定。



一个 *1-D* 数组：一个名称，多个值，每个值由其索引定位

8.1.1 使用 FOR 循环填充和读取数据

数组与计数循环相辅相成：计数器即为下标。

```
DECLARE Scores : ARRAY[1:5] OF INTEGER
DECLARE Index, Total : INTEGER
FOR Index ← 1 TO 5
    Scores[Index] ← Index * 10
NEXT Index
Total ← 0
FOR Index ← 1 TO 5
    Total ← Total + Scores[Index]
NEXT Index
OUTPUT "Total: ", Total
OUTPUT "Mean: ", Total / 5
```

8.1.2 查找最大值

```
DECLARE Scores : ARRAY[1:5] OF INTEGER
DECLARE Index, Largest : INTEGER
Scores[1] ← 42
Scores[2] ← 17
Scores[3] ← 93
Scores[4] ← 8
Scores[5] ← 55
Largest ← Scores[1]
FOR Index ← 2 TO 5
    IF Scores[Index] > Largest
      THEN
        Largest ← Scores[Index]
    ENDIF
NEXT Index
OUTPUT "Largest: ", Largest
```

将 Largest 初始化为**第一个元素**，切勿设为 0— 若数据含负数，0 将被错误判定为最大值，导致结果错误。

常见错误

- 从声明为 [1:5] 的数组中读取 Scores[6]。这属于越界访问。
- 编写 Scores(3)。数组使用方括号；圆括号调用函数。
- 声明 [1:5] 然后循环 0 TO 4。

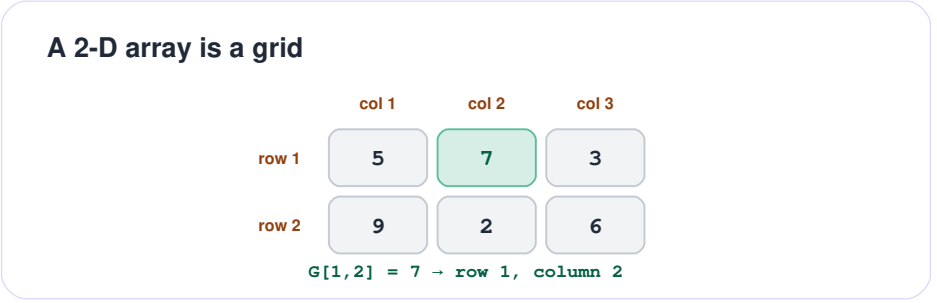
8.2 ■ Two-dimensional arrays

2 维二维数组是一个表格：包含行和列，需要两个索引。

```
DECLARE <identifier> : ARRAY[<lower1>:<upper1>, <lower2>:<upper2>] OF
↪  <data type>
```

```
DECLARE Grid : ARRAY[1:2, 1:3] OF INTEGER
Grid[1,1] ← 1
Grid[1,2] ← 2
Grid[1,3] ← 3
Grid[2,1] ← 4
Grid[2,2] ← 5
Grid[2,3] ← 6
OUTPUT Grid[2,3]
```

第一个索引是行行，第二个是列列。



一个 *2* 维数组是一个网格：*G[1, 2]* 表示第 *1* 行、第 *2* 列

8.2.1 嵌套循环遍历表格

每个维度维度对应一个循环。内层循环完成整行后，外层循环才继续移动。

```
DECLARE Grid : ARRAY[1:2, 1:3] OF INTEGER
DECLARE Row, Col : INTEGER
FOR Row ← 1 TO 2
    FOR Col ← 1 TO 3
        Grid[Row, Col] ← Row * Col
    NEXT Col
NEXT Row
FOR Row ← 1 TO 2
    FOR Col ← 1 TO 3
        OUTPUT "Grid[", Row, ",", Col, "] = ", Grid[Row, Col]
    NEXT Col
NEXT Row
```

△ 内层 NEXT 关闭的是 **内层**计数器。NEXT Row 在 NEXT Col 交叉之前尚未完成，循环就会继续并产生无意义结果 — 标记会立即指出这一点。

常见错误

- 混淆行和列。Grid[2,3] 是第 2 行、第 3 列。
- Grid[Row][Col]。剑桥考试写作中使用一对带逗号的方括号。
- 像上面那样交叉 NEXT 行。

9 过程与函数

9.1 ■ PROCEDURE and CALL

一个 **过程**过程是一组命名语句块。编写一次并从多个地方调用它，这就是 **分解**分解 — 整个纸笔 2 考试所依赖的理念。

```
PROCEDURE <identifier>
    <statements>
ENDPROCEDURE
```

通过名称调用过程，使用关键字 CALL:

```
PROCEDURE DefaultLine
    OUTPUT "------------------------------"
ENDPROCEDURE

CALL DefaultLine
OUTPUT "Report"
CALL DefaultLine
```

- 定义定义本身不会执行任何操作。只有 CALL 才能运行它。
- 当过程结束时，控制流返回到 CALL 之后的那一行。
- 过程 **不返回值**。如果需要返回值，你需要使用函数。

常见错误

- 单独编写 DefaultLine 来运行它。0478 需要 CALL。
- 遗漏 ENDPROCEDURE。
- 期望过程自身的变量在其外部存在。

9.2 ■ FUNCTION and RETURN

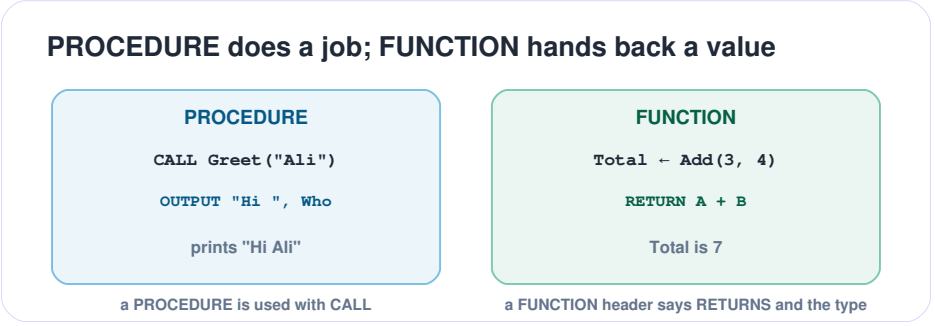
函数函数是一个返回值的命名例程。它在需要该值的地方书写即位于 OUTPUT 内部、赋值号右侧或条件语句中。

```
FUNCTION <identifier>(<parameters>) RETURNS <data type>
    <statements>
    RETURN <value>
ENDFUNCTION
```

```
FUNCTION SumSquare(Number1 : INTEGER, Number2 : INTEGER) RETURNS INTEGER
    RETURN Number1 * Number1 + Number2 * Number2
ENDFUNCTION

OUTPUT "Sum of squares = ", SumSquare(10, 20)
```

- RETURNS <data type> 在头部声明了返回值类型，且不可省略。
- RETURN 会**立即**结束函数执行 — 其后的代码均不会运行。
- 调用函数时从不使用 CALL；只需在需要使用其值的位置直接使用即可。



程序通过 *CALL* 调用执行任务；函数则返回一个值供后续使用

9.2.1 条件语句中的函数

```
FUNCTION IsEven(Number : INTEGER) RETURNS BOOLEAN
    RETURN MOD(Number, 2) = 0
ENDFUNCTION

DECLARE Index : INTEGER
FOR Index ← 1 TO 6
    IF IsEven(Index)
      THEN
        OUTPUT Index, " is even"
    ENDIF
NEXT Index
```

常见错误

- CALL SumSquare(10, 20) – CALL 调用的是过程而非函数。请在表达式中使用函数，例如 Answer ← SumSquare(10, 20)。
- 函数在某些路径上缺少 RETURN。所有出口路径都必须返回一个值。
- 头部遗漏了 RETURNS INTEGER。

9.3 ■ Parameters

一个**参数**参数是调用过程或函数时传入的值。每个参数都有其独立的数据类型。

```
PROCEDURE Line(Size : INTEGER)
    DECLARE Index : INTEGER
    FOR Index ← 1 TO Size
        OUTPUT "-"
    NEXT Index
ENDPROCEDURE

CALL Line(5)
CALL Line(20)
```

多个参数之间用逗号分隔，调用时的顺序必须与定义中的顺序一致：在定义中：

```
PROCEDURE Greet(Name : STRING, Times : INTEGER)
    DECLARE Index : INTEGER
    FOR Index ← 1 TO Times
        OUTPUT "Hello ", Name
    NEXT Index
ENDPROCEDURE

CALL Greet("Mei", 2)
CALL Greet("Sam", 1)
```

△ 在 **0478 传值传递**模式下，每个参数都是按值传递的：过程操作的是副本，因此在内部修改它不会改变外部值。若要获取返回值，请使用 FUNCTION 并 RETURN 该变量。（A-Level 9618 增加了 BYREF 以支持另一种行为 — 参见主题 11。）

```
FUNCTION Doubled(Number : INTEGER) RETURNS INTEGER
    RETURN Number * 2
ENDFUNCTION

DECLARE Value : INTEGER
Value ← 7
Value ← Doubled(Value)
OUTPUT Value
```

常见错误

- PROCEDURE Line(Size) 没有类型。
- 参数顺序错误 — 这属于 Greet(2, "Mei") 类型错误。
- 期望过程修改传入的变量。在 0478 中无法做到这一点。

10 文件处理

10.1 ■ Writing to a file

一个文件在项目结束后保留数据。0478 使用四个命令，且每个命令都命名了其操作的文件。

```
OPENFILE <file identifier> FOR WRITE
WRITEFILE <file identifier>, <data>
CLOSEFILE <file identifier>
```

```
OPENFILE "names.txt" FOR WRITE
WRITEFILE "names.txt", "Mei"
WRITEFILE "names.txt", "Sam"
CLOSEFILE "names.txt"
OUTPUT "Saved"
```

三种**文件模式**文件模式：

模式	功能说明
WRITE	创建一个新文件 – **原有内容全部丢失**
APPEND 追加	在现有内容末尾添加
READ	从头开始读取

```
OPENFILE "log.txt" FOR WRITE
WRITEFILE "log.txt", "first"
CLOSEFILE "log.txt"
OPENFILE "log.txt" FOR APPEND
WRITEFILE "log.txt", "second"
CLOSEFILE "log.txt"
OUTPUT "Both lines saved"
```

- 一个 WRITEFILE 写入**一行**。
- CLOSEFILE 是必须的。未关闭的文件可能导致尚未写入的数据丢失，这是常见的考试考点。写入，这属于标准考题答案。

常见错误

- 将 FOR WRITE 误作 FOR APPEND 使用，后者会静默覆盖原有内容。
- 忘记 CLOSEFILE。
- 打开一个已经打开的文件。

10.2 ■ Reading a file to the end

READFILE 将一行读入变量中。该变量必须先声明。

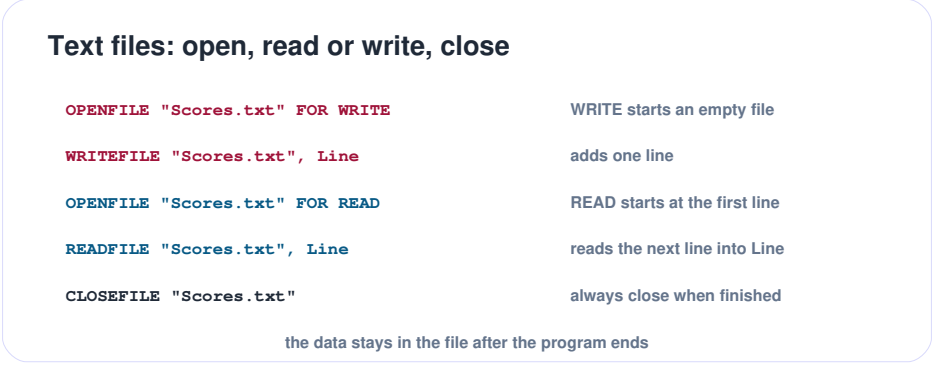
```
OPENFILE <file identifier> FOR READ
READFILE <file identifier>, <variable>
CLOSEFILE <file identifier>
```

通常你无法预知文件有多少行，因此需逐行读取**直到文件末尾**文件**结束** — 这正是 EOF 函数所报告的状态。

```
OPENFILE "names.txt" FOR WRITE
WRITEFILE "names.txt", "Mei"
WRITEFILE "names.txt", "Sam"
WRITEFILE "names.txt", "Ana"
CLOSEFILE "names.txt"

DECLARE Line : STRING
OPENFILE "names.txt" FOR READ
WHILE NOT EOF("names.txt") DO
    READFILE "names.txt", Line
    OUTPUT Line
ENDWHILE
CLOSEFILE "names.txt"
```

EOF("names.txt") 在每行读取完毕后变为 TRUE，因此 WHILE NOT EOF(...) 意味着 当仍有内容可读时。



打开文件用于写入或读取，逐行使用，然后关闭

10.2.1 边读边计数

```
OPENFILE "marks.txt" FOR WRITE
WRITEFILE "marks.txt", 40
WRITEFILE "marks.txt", 65
WRITEFILE "marks.txt", 88
CLOSEFILE "marks.txt"

DECLARE Mark, Count, Total : INTEGER
Count ← 0
Total ← 0
OPENFILE "marks.txt" FOR READ
WHILE NOT EOF("marks.txt") DO
    READFILE "marks.txt", Mark
    Count ← Count + 1
    Total ← Total + Mark
ENDWHILE
CLOSEFILE "marks.txt"
OUTPUT Count, " marks, mean ", Total / Count
```

常见错误

- 使用 `WHILE EOF(...)` 循环 — 仅在剩余内容已耗尽时执行一次。应改用 `NOT`。
- 因缺少 `EOF` 保护而越界读取。
- 打开一个从未被写入过的 `FOR READ` 文件。

11 A-Level 9618: 有何不同

11.1 ■ Layout and operators

剑桥出版**两份**伪代码规范，二者存在分歧。以下所有主题 1-10 对应 **IGCSE 0478**。A-Level **9618** 更改了以下细节 — 以及一名学生同时修读两门考试的学生若在一份试卷中混用另一套规范的写法，将被扣分。

	IGCSE 0478	**A-Level 9618**
`THEN`	单独占一行，缩进 2	在 `IF` 行上
常量	`CONSTANT Pi ← 3.142`	`CONSTANT Pi = 3.142`
`DIV` / `MOD`	函数：`DIV(10, 3)`	运算符：`10 DIV 3`
连接字符串	`OUTPUT` 中的逗号	`&` 运算符
标识符	必须以大写字母开头	允许混合大小写
`NEXT`	必须命名其计数器	良好习惯
`ROUND`, `RANDOM()`	两者均出版	**两者均不存在**

这是 0478 形式，在此处运行如下：

```
DECLARE Total : REAL
CONSTANT Rate ← 0.5
Total ← 10
IF MOD(7, 2) = 1
  THEN
    OUTPUT "Odd, half rate ", Total * Rate
ENDIF
```

9618 形式的同一算法书写如下 — 注意 `THEN` 位于 `IF` 行上，`=` 出现在 `CONSTANT` 中，且其两个值之间为 `MOD`；

```
DECLARE Total : REAL
CONSTANT Rate = 0.5
Total ← 10
IF 7 MOD 2 = 1 THEN
    OUTPUT "Odd, " & "half rate ", Total * Rate
ENDIF
```

△ **正确选项仅取决于你所参加的考试**。两种写法本身均无错误；每种写法在另一场考试中均属错误。本页演示区运行的是 **0478** 规则，因此若使用 9618 语法结构将被拒绝，并提示所属考纲 — 即一种快速自查自身习惯的方法。

常见错误

- 在 0478 答题纸的 `IF` 行上书写 `THEN`，因为教科书或一本网站曾如此展示。这是最常见的跨考纲混淆错误。
- `10 MOD 3` 出现在 0478 中。此处即为 `MOD(10, 3)`。
- `ROUND(3.14159, 2)` 出现在 9618 答卷中。9618 完全不支持 `ROUND`。

11.2 ■ Parameters: BYVAL and BYREF

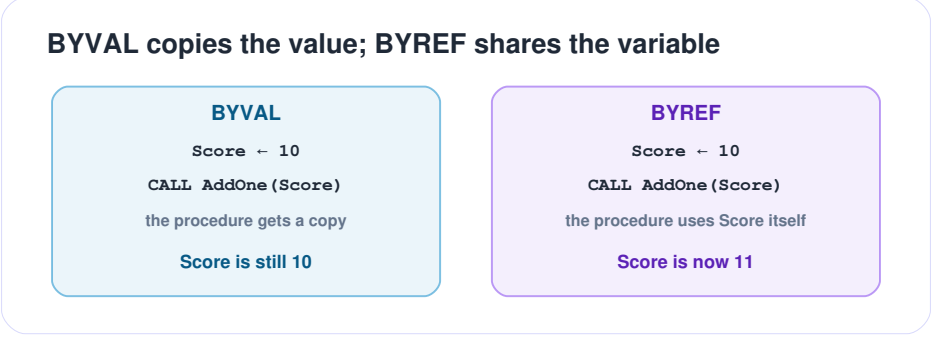
0478 对所有参数均采用 **传值 (topic 9)** 方式传递。9618 允许你自行选择，并在函数头中明确声明。在页眉中。

- BYVAL — 过程获得参数的 **副本**。修改副本不会影响外部原始变量。此为未显式指定时的默认行为。默认情况下，当两个单词均未写出时。
- BYREF – 该过程通过引用操作调用者的**原变量**，因此修改可见。调用之后。

此功能仅在 9618 中可用。它不能在此处运行，也不被 0478 答卷接受：

```
PROCEDURE Increase(BYREF Value : INTEGER)
    Value ← Value + 1
ENDPROCEDURE

DECLARE Count : INTEGER
Count ← 5
CALL Increase(Count)
OUTPUT Count      // prints 6
```



BYVAL 传递副本；*BYREF* 允许过程修改调用者的变量

11.2.1 返回值的 0478 方式

使用一个 FUNCTION 并赋值其返回值。这就是整个 0478 答案的内容，它运行如下：

```
FUNCTION Increased(Value : INTEGER) RETURNS INTEGER
    RETURN Value + 1
ENDFUNCTION

DECLARE Count : INTEGER
Count ← 5
Count ← Increased(Count)
OUTPUT Count
```

常见错误

- 在 0478 答案中书写 BYREF。该关键字不在该教学大纲内。
- 期望普通的 9618 参数像 BYREF 一样工作。没有该词则视为 BYVAL。

11.3 ■ Records and other types

9618 拥有数据结构 0478 所不具备的数据结构。你仅在 A-Level 阶段会遇到它们，且它们均不适用于本页面所述的 0478 规则。

记录将不同类型的字段归于一组（9618 章节 4）：

```
TYPE Student
    DECLARE Name : STRING
    DECLARE Mark : INTEGER
ENDTYPE

DECLARE Candidate : Student
Candidate.Name ← "Mei"
Candidate.Mark ← 91
OUTPUT Candidate.Name, " scored ", Candidate.Mark
```

枚举类型列出变量可取的唯一值：

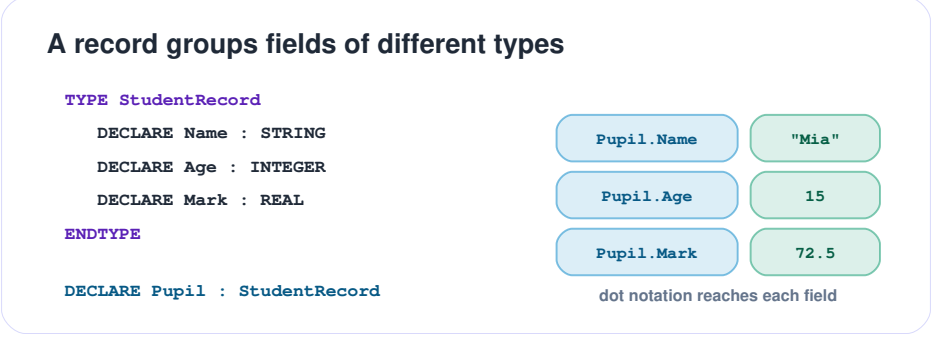
```
TYPE Vehicle = (Car, Bus, Taxi)
DECLARE MyRide : Vehicle
MyRide ← Taxi
```

9618 还增加了一种 DATE 数据类型、随机访问文件（OPENFILE … FOR RANDOM）、指针，以及面向对象的纸张所需的带有 CLASS … ENDCLASS 的类。

△ 0478 有 **五种**数据类型和数组，除此之外无其他内容（主题 2）。若一个 0478 答案声明了记录，则其回答的是不同的教学大纲。

常见错误

- 在 0478 答案中使用记录。应改用并行数组 — 每个字段对应一个数组，共享同一个索引。
- 误以为 TYPE 与 DECLARE 含义相同。TYPE 定义了一个 **新类型**；而 DECLARE 创建了一个现有类型的 **变量**。



记录将不同类型的字段归于一组；点记法可访问其中每一个