剑桥伪代码讲义

中文版

Contents

1 书写风格与排版

1.1 关键字与大小写

伪代码 (pseudocode) 是剑桥书写算法 (algorithm) 所用的语言。它不是真正的编程语言 —— 没有计算机会运行它 —— 但考试按真代码来评分, 所以书写形式很重要。

每个关键字 (keyword) 都用**大写**书写:`IF`、`REPEAT`、`PROCEDURE`、`OUTPUT`。

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

- 写 `IF`, 绝不写 `if` 或 `If`。
- 关键字永远不能用作你自己的名字: 不能把变量叫做 `FOR`。

常见错误

- 关键字用小写。评分方案要求大写。
- 自创关键字。只有本讲义中列出的词在考纲之内。

1.2 标识符

标识符 (identifier) 是你给变量 (variable)、常量 (constant)、过程 (procedure) 或函数 (function) 起的名字。

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

考纲给出四条规则:

- 必须**以大写字母开头**: 写 `Count`, 不写 `count`。
- 只能包含字母和数字 —— 不能有空格, 不能有标点。
- **不区分大小写**, 所以 `Count` 和 `COUNT` 是同一个名字。选定一种写法并保持一致。
- 名字要有意义。`NumberOfPlayers` 对得起读者花的时间,`N` 则不然。

常见错误

- `count ← 0`—— 以小写开头。0478 要求首字母大写。
- `Number of players`—— 名字内部不允许空格。

1.3 注释与缩进

注释 (comment) 以 `//` 开头, 一直到行尾。它写给人看, 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 (indentation) 表示哪些语句在某个结构**内部**。考纲要求每一层四个空格, 但有一个你必须记住的例外:

```
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` 相对于它们的 `IF` 缩进**两**个空格。
- 它们下面的语句缩进**四**个空格。
- `ENDIF` 与它自己的 `IF` 对齐。

常见错误

- 把 `THEN` 写在 `IF` 那一行。那是 A-Level 9618 的排版 —— 见主题 11。
- 完全不缩进。评分者必须看得见结构, 才能给出结构分。

2 变量、常量与数据类型

2.1 五种数据类型

0478 伪代码恰好有**五**种数据类型 (data type)。把这份清单记熟 —— 说出不在清单上的类型会丢分。

类型	存放	例子
`INTEGER`	整数	`42`、`-7`
`REAL`	带小数部分的数	`3.14`、`-0.5`
`CHAR`	单个字符	`'A'`
`STRING`	字符串	`"Hello"`
`BOOLEAN`	`TRUE` 或 `FALSE`	`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,`CHAR` 用**单**引号。
- `TRUE` 和 `FALSE` 是关键字, 所以大写且不加引号。

常见错误

- 写 `FLOAT`、`DOUBLE` 或 `BOOL`。那些属于真实的编程语言, 不属于本考纲。
- 用 `'Hello'` 表示字符串 —— 单引号只装一个字符。

2.2 DECLARE

`DECLARE` 为变量 (variable) 命名, 并在使用之前确定它的类型。

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

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

同一类型的多个名字写在一行, 用逗号隔开:

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

- 在**首次使用之前**声明变量, 写在算法开头附近。
- 变量在整个算法中保持它的类型 ——`INTEGER` 永远不会装下 `"Mei"`。

常见错误

- 漏掉冒号:`DECLARE Counter INTEGER`。
- 使用从未声明过的变量。考试要求写出声明。

2.3 CONSTANT

常量 (constant) 是在整个算法中固定不变的值 —— 税率、π、班级人数。命名一次意味着改动只需改一处。

```
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**, 与赋值完全一样。
- 值必须是**字面量** (literal)—— 一个数、一个字符串、`TRUE`—— 绝不能是计算式, 也不能是另一个变量。
- 常量之后永远不能被赋新值。这正是它存在的意义。

常见错误

- `CONSTANT TaxRate = 0.20`。`=` 的写法属于 A-Level 9618—— 见主题 11。
- 在算法后面给常量赋值。

3 赋值、输入与输出

3.1 赋值箭头

赋值 (assignment) 运算符是 ←, 一个向左的箭头。它的意思是把右边的值放进左边的盒子里。

```
<identifier> ← <value>
```

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

`Counter ← Counter + 1` 作为数学式看似不可能, 作为指令却再平常不过: 用盒子当前装的值算出右边, 再把结果存回去。

- 如果打不出 ←, 就写 `<-`。两者都被接受, 本站和考试都一样。
- 左边必须是**单个标识符**或一个数组元素, 绝不能是计算式。

常见错误

- `Counter = 0`。在伪代码中 `=` 是**比较**, 不是赋值 —— 混淆两者是本卷最常见的语法错误。
- `Counter + 1 ← Counter`。箭头指向被改变的那个东西。

3.2 INPUT 与 OUTPUT

`INPUT` 从用户读入一个值到变量中。`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` 恰好接受**一个**变量, 且该变量必须已经声明。
- `OUTPUT` 接受任意多项, 用**逗号**分隔; 它们依次输出, 中间不加任何东西。
- `OUTPUT` 中的字面量 (literal) 字符串加双引号, 变量不加。

3.2.1 提示用户

单独的 INPUT 在屏幕上什么也不显示, 所以写得好的算法会先输出一句提示 (prompt):

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

常见错误

- INPUT "Enter a number", X—— 提示是一条单独的 OUTPUT 语句。
- OUTPUT "Total: " + Total。0478 用逗号连接输出项;+ 是算术运算, 而 & 属于 A-Level 9618(主题 11)。
- 漏掉引号:OUTPUT Hello 会去找一个叫 Hello 的变量。

4 算术与逻辑

4.1 算术运算符

五个算术 (arithmetic) 运算符, 按计算器使用它们的顺序排列。

运算符	含义	A 为 7,B 为 2
+	加	A + B 得 9
-	减	A - B 得 5
*	乘	A * B 得 14
/	除	A / B 得 3.5
^	幂	A ^ B 得 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
```

先算括号, 再算 ^, 然后是 * 和 /, 最后是 + 和 -—— 通常的优先级 (precedence)。只要读者需要停下来想一想, 就加括号。

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

常见错误

- 以为 / 会给出整数。7 / 2 是 3.5, 不是 3; 要整数请用 DIV。
- 写成 A x B 或 A ÷ B。请用 * 和 /。

4.2 DIV 与 MOD

整数除法 (integer division) 回答两个不同的问题, 而 0478 把它们各写成一个**函数** (function), 两个值放在括号里。

- DIV(a, b)——b 能整着进入 a 多少次, 即**商** (quotient)。
- MOD(a, b)—— 剩下多少, 即**余数** (remainder)。

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

两者都接受整数, 也都返回整数。

4.2.1 它们的用途

MOD 是" 能否整除" 的标准检验, 因而也是判断一个数是否为偶数的方法:

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

DIV 把总量换算成整单位 —— 秒换分钟, 便士换英镑:

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

常见错误

- `10 MOD 3`。那是 A-Level 9618 的写法;**0478 使用函数形式** `MOD(10, 3)`。
- 把两者弄混。DIV 给出整数部分;MOD 给出装不下的部分。

4.3 关系运算符与逻辑运算符

关系运算符 (relational operator) 比较两个值, 产生 TRUE 或 FALSE。

运算符	含义
=	等于
<>	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于


```
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** (logic operator) 把条件连接起来:

- AND—— 两边都必须为 TRUE。
- OR—— 至少一边为 TRUE。
- NOT—— 把 TRUE 变成 FALSE, 反之亦然。

```
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

- 这里的不等于写作 <>, 而不是属于其他语言的 !=。
- `IF Age >= 15 AND <= 18`。AND 的每一边都需要完整的比较:`Age >= 15 AND Age <= 18`。
- 在答卷中称它们为比较运算符。考纲用语是**关系运算符**。

5 字符串操作

5.1 LENGTH、UCASE 与 LCASE

0478 只公布四个字符串 (string) 例程, 没有别的。其中三个在这里。

`LENGTH(<identifier>)` 返回字符串含有多少个字符 (character)—— 空格也算。

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

`UCASE(<identifier>)` 返回大写形式,`LCASE(<identifier>)` 返回小写形式。两者都不改变原值 (original)—— 各自交回一个新值。

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

最后那一行仍然输出 `Happy`: 要保留改动, 必须把结果赋回去。

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

5.1.1 比较时不必顾虑大小写

两边都套 `UCASE`, 是接受任意大小写输入的标准做法:

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

常见错误

- 以为 `UCASE(Name)` 会改变 `Name`。请把结果赋回去。
- `LENGTH` 数的是字符, 不是单词。`LENGTH("Happy Days")` 是 10。

5.2 SUBSTRING

`SUBSTRING(<identifier>, <start>, <length>)` 从字符串中取出一段 —— 从位置 `start` 开始, 取 `length` 个字符。

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

△ **计数从 1 开始**, 不是 0。`"Happy Days"` 的第一个字符在位置 1。

5.2.1 取一个字符, 以及取剩下的部分

```
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)
```

把 `LENGTH` 套进 `SUBSTRING`, 正是后两行对任意长度的词都成立的原因 —— 写死 `8` 的话, 词一变就错。

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 才是第一个字符。
- 把第三个参数当成结束位置。它是一个**个数** ——`SUBSTRING(W, 2, 3)` 从第二个字符起取三个。
- 要求的字符数超过字符串本身的长度。

6 选择结构

6.1 IF、THEN、ELSE、ENDIF

选择 (selection) 根据一个条件 (condition) 运行这一组语句或那一组。

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

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

加上 ELSE 处理另一种情况。两个分支 (branch) 中恰好运行一个 —— 绝不会两个都运行, 也绝不会一个都不运行。

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

排版是学生最常丢分的地方, 所以把它当作一个图形记下来:

- IF 和它的条件写在一行, **条件后面什么都不写**。
- THEN 和 ELSE 各占一行, 缩进**两**个空格。
- 语句缩进**四**个空格。
- ENDIF 回到它自己 IF 所在的列。

6.1.1 嵌套

IF 里面再放 IF, 可以处理超过两种结果。每一个都需要自己的 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

- 忘记 ENDIF。每个 IF 都要闭合, 少一个就是语法错误 (syntax error), 评分者一眼就能看出。
- 把 IF Mark >= 50 THEN OUTPUT "Pass" 写成一行。**两份考纲都没有这种写法**; 请写成三行。
- 把 IF Mark = 50 THEN 写成 IF Mark ← 50。箭头是赋值,= 才是比较。

6.2 CASE OF

当一个变量要与若干**单个值**逐一比较时,CASE OF 比一叠嵌套的 IF 清楚得多。

```
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
```

- 每个分支是一个**值**、一个冒号, 然后是要做的事。
- OTHERWISE 兜住所列值没有覆盖到的一切。它是可选的, 写上是好习惯 —— 用户输入意料之外的东西时, 走的就是它。
- 第一个匹配的分支运行, 其余的跳过。

6.2.1 一个范围, 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

- 用 CASE OF 处理**条件**。它把一个变量与若干值比较;`Mark >= 50` 需要 IF。
- 忘记 ENDCASE。
- 省略 OTHERWISE, 然后困惑于意料之外的值为什么毫无反应。

7 循环结构

7.1 FOR 计数循环

迭代 (iteration) 重复执行语句。当你在**开始之前**就知道要重复多少次时, 使用 FOR 循环 —— 计数控制 (count-controlled) 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 (counter) 从第一个值开始, 循环对直到第二个值 (**包含**它) 的每个值各运行一次。`1 TO 5` 运行五次。

7.1.1 STEP

STEP 改变每次跳跃的大小。负的步长会倒着数。

```
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 累加总和

最常见的用法: 循环之外有一个变量, 循环不断往里加。

```
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` 必须写在循环**之前**。写在里面, 它每一轮都会被清零。

常见错误

- NEXT 写错了计数器。嵌套循环必须按打开的相反顺序闭合: 先写内层的 NEXT。
- 以为 1 TO 5 运行四次。两端都包含。
- 在循环内部改动计数器。让 FOR 自己掌管它。

7.2 WHILE 前条件循环

WHILE 循环在每一轮**之前**检验, 因此它可能运行**零**次。当重复次数取决于循环运行中发生的事情时, 用它。

```
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
```

循环内部必须有东西最终使条件变为假, 否则循环永不结束 —— 这就是**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

- 忘记改动条件所检验的变量, 造成无限循环。
- 漏掉 DO 或 ENDWHILE。
- 在次数已知时用 WHILE。FOR 一行就把意思说清楚了。

7.3 REPEAT 后条件循环

REPEAT 循环在每一轮**之后**检验, 因此它总是**至少运行一次**。

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

“ 至少一次 ” 正是选择它的理由: 验证 (validating) 输入时, 你必须先问, 才能判断答案。

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

△ UNTIL 陈述的是**停止**循环的条件;WHILE 陈述的是**继续**循环的条件。两者相反, 弄反了是经典错误。

	何时检验	至少运行一次?	条件的含义
FOR	看计数器	否 (空区间运行零次)	—
WHILE	之前	否	为真时继续
REPEAT	之后	**是**	变为真时停止

常见错误

- 想写 UNTIL Number >= 100 却写成 UNTIL Number < 100。
- 在循环体必须能运行零次的场合用 REPEAT。
- 忘记 UNTIL 就是结构的结尾 —— 没有 ENDREPEAT 这种写法。

8 数组

8.1 一维数组

数组 (array) 在一个名字下存放同一类型的多个值。每个值位于一个**下标** (index) 上。

```
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 (bound) 写作 `lower:upper`, 而且**两端都包含** ——`[1:3]` 有三个元素 (element)。
- 剑桥的数组通常从 **1** 开始, 不是 0。
- 每个元素的类型相同, 由 `OF` 确定。

8.1.1 用 FOR 循环填充与读取

数组和计数循环天生一对: 计数器就是下标。

```
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 找出最大值

```
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
```

把 `Largest` 初始化为**第一个元素**, 绝不要用 0—— 遇到负数数据时,0 会胜出, 答案就错了。

常见错误

- 从声明为 `[1:5]` 的数组中读 `Scores[6]`, 这是越界 (out of bounds)。
- 写成 `Scores(3)`。数组用方括号; 圆括号是调用函数。
- 声明 `[1:5]` 却循环 `0 TO 4`。

8.2 二维数组

二维数组 (2-D array) 是一张表: 有行有列, 两个下标。

```
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]
```

第一个下标是行 (row), **第二个**是列 (column)。

8.2.1 嵌套循环遍历表格

每个维度 (dimension) 一层循环。内层循环走完一整行, 外层才前进一步。

```
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
```

△ 内层的 NEXT 闭合**内层**计数器。把 NEXT Row 写在 NEXT Col 之前就把两层循环交叉了, 毫无意义 —— 评分者一眼就看得出来。

常见错误

- 把行和列弄反。Grid[2,3] 是第 2 行、第 3 列。
- 写成 Grid[Row][Col]。剑桥写**一对**方括号, 里面用逗号。
- 如上所述把 NEXT 两行交叉。

9 过程与函数

9.1 PROCEDURE 与 CALL

过程 (procedure) 是一段有名字的语句块。写一次、在多处调用, 就是**分解** (decomposition)—— 整张试卷二都建立在这个思想上。

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

过程通过名字被**调用**, 要用关键字 CALL:

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

CALL DefaultLine
OUTPUT "Report"
CALL DefaultLine
```

- 定义 (definition) 本身什么也不做。只有 CALL 才会运行它。
- 过程结束后, 控制权回到 CALL 的下一行。
- 过程**不返回值**。需要返回值时, 你要的是函数。

常见错误

- 只写 DefaultLine 就想运行它。0478 要求写 CALL。
- 漏掉 ENDPROCEDURE。
- 以为过程内部的变量在外面还存在。

9.2 FUNCTION 与 RETURN

函数 (function) 是会交回一个值的过程。它写在需要那个值的地方 ——OUTPUT 里面、赋值号右边、条件里面。

```
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)
```

- 头部的 RETURNS <data type> 说明返回什么类型。它不是可选的。
- RETURN **立即**结束函数 —— 它后面的语句都不会运行。
- 函数**绝不**用 CALL 调用; 它出现在它的值该出现的地方。

9.2.1 把函数写进条件里

```
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)—— 那把答案扔掉了。
- 某条路径上没有 RETURN 的函数。每一条出口都必须返回一个值。
- 头部漏写 RETURNS INTEGER。

9.3 参数

参数 (parameter) 是调用过程或函数时交给它的值。每个参数都带自己的类型。

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

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


多个参数用逗号分隔, 调用时的顺序必须与定义时一致:

```
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 中每个参数都按值传递** (passed by value): 过程处理的是一个副本, 所以在里面改动它, 外面毫无变化。要取回一个值, 请用 FUNCTION 并 RETURN 它。(A-Level 9618 增加了 BYREF 来实现另一种行为 —— 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), 没有写类型。
- 调用时参数顺序错了 ——Greet(2, "Mei") 是类型错误。
- 指望过程去改变你传进去的变量。在 0478 里它做不到。

10 文件处理

10.1 写入文件

文件 (file) 在程序结束后仍保存数据。0478 用四条命令, 每一条都点名它所操作的文件。

```
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** (file mode):

模式	作用
WRITE	新建文件 —— **原有内容全部丢失**
APPEND	追加到已有内容的末尾
READ	从头读取

```
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"
```

- 一条 WRITEFILE 写**一行**。
- CLOSEFILE 不是可选的。未关闭的文件可能丢失仍在等待写出的数据, 这是一个常考的答案。

常见错误

- 本想 FOR APPEND 却写了 FOR WRITE, 悄无声息地毁掉旧内容。
- 忘记 CLOSEFILE。
- 打开一个已经打开的文件。

10.2 读取文件到末尾

READFILE 把一行读入一个变量。该变量必须先声明。

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

通常你并不知道有多少行, 所以要一直读到**文件末尾** —— 这正是 EOF 函数所报告的。

```
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") 在每一行都读完之后变为 TRUE, 所以 WHILE NOT EOF(...) 的意思是还有东西可读时。

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

- 写成 `WHILE EOF(...)`—— 那只在没有东西可读时才运行。你要的是 `NOT`。
- 循环没有 `EOF` 守卫, 读过了文件末尾。
- 对一个从未写入过的文件 `FOR READ`。

11 A-Level 9618: 有何不同

11.1 排版与运算符

剑桥公布了**两份**伪代码规范, 而且它们互相不一致。主题 1-10 讲的全部是 **IGCSE 0478**。A-Level **9618** 改动了下面这些细节 —— 同时学两者的学生, 常因把一种写进另一种的卷子而丢分。

	IGCSE 0478	A-Level 9618
`THEN`	**独占一行**, 缩进 2	写在 `IF` **那一行**
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` 中用逗号	`&` 运算符
标识符	必须以**大写**开头	允许混合大小写
`NEXT`	必须写出计数器	属于良好习惯
`ROUND`、`RANDOM()`	两者都有	**都没有**

这是 0478 的写法, 在本页可以运行:

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

同一算法的 9618 写法是这样 —— 注意 `THEN` 在 `IF` 那一行、`CONSTANT` 用 `=`、`MOD` 写在两个值之间:

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

△ **哪一种正确, 只取决于你考哪一张卷。**两者都不是错误, 但各自在对方的考试中都是错误。本页的运行环境按 **0478** 的规则执行, 所以 9618 的写法会被拒绝, 并给出一条指明其所属考纲的提示 —— 这是检查自己书写习惯的快捷办法。

常见错误

- 在 0478 答卷中把 `THEN` 写在 `IF` 那一行, 因为某本教材或某个网站是那样写的。这是最常见的跨考纲错误。

- 在 0478 中写 `10 MOD 3`。那里应写 `MOD(10, 3)`。
- 在 9618 答卷中写 `ROUND(3.14159, 2)`。9618 根本没有 `ROUND`。

11.2 参数:BYVAL 与 BYREF

0478 的**每个**参数都按值传递 (主题 9)。9618 让你选择, 并在头部写明。

- `BYVAL`—— 过程拿到一个**副本**。改动它, 外面毫无变化。不写这两个词时, 默认就是它。
- `BYREF`—— 过程操作的是调用者**自己的变量** (引用), 所以调用之后能看到改动。

这只属于 9618。它在本页不能运行, 在 0478 答卷中也不被接受:

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

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

11.2.1 0478 取回值的办法

用 `FUNCTION`, 并把它返回的东西赋回去。这就是 0478 答案的全部, 而且可以运行:

```
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 答卷中写 `BYREF`。该关键字不在那份考纲里。
- 以为 9618 中不加修饰的参数会像 `BYREF` 一样。不写那个词就是 `BYVAL`。

11.3 记录与其他类型

9618 拥有 0478 所没有的数据结构 (data structure)。你只会在 A-Level 遇到它们, 而且它们都不能在本页的 0478 规则下运行。

记录 (record) 把不同类型的字段归到一个名字下 (9618 第 4 节):

```
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 (enumerated type) 列出一个变量只能取的那些值:

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

9618 还增加了 `DATE` 数据类型、随机存取文件 (`OPENFILE … FOR RANDOM`)、指针, 以及面向对象卷所用的 `CLASS … ENDCLASS`。

△ 0478 只有**五**种数据类型和数组 (主题 2), 别无其他。在 0478 答卷中声明一个记录, 等于在回答另一份考纲。

常见错误

- 在 0478 答卷中使用记录。请改用平行数组 —— 每个字段一个数组, 共用一个下标。
- 以为 `TYPE` 和 `DECLARE` 是一回事。`TYPE` 定义一个**新类型**;`DECLARE` 创建一个已有类型的**变量**。