过程与函数

Ref As Pseudocode

PROCEDURE 与 CALL

子程序（subroutine）是一段有名字、完成一项任务的代码。写一次，就可以按需要多次使用。伪代码有两种子程序：过程和函数。

过程（procedure）用来完成一项任务，它不返回值。

```
PROCEDURE <identifier>()
   <statement(s)>
ENDPROCEDURE

PROCEDURE <identifier>(<param1> : <data type>, <param2> : <data type>
↪  ...)
   <statement(s)>
ENDPROCEDURE
```

用 CALL 来运行过程：

```
CALL <identifier>()
CALL <identifier>(<value1>, <value2> ...)
```

```
PROCEDURE DrawLine(LineLength : INTEGER, Symbol : CHAR)
   DECLARE Line : STRING
   DECLARE i : INTEGER
   Line ← ""
   FOR i ← 1 TO LineLength
      Line ← Line & Symbol
   NEXT i
   OUTPUT Line
ENDPROCEDURE

PROCEDURE ShowTitle()
   CALL DrawLine(12, '=')
   OUTPUT "  MAIN MENU"
   CALL DrawLine(12, '=')
ENDPROCEDURE

CALL ShowTitle()
CALL DrawLine(5, '*')
```

- **参数**（parameter）是写在过程头部、用来接收值的变量。LineLength 和 Symbol 就是 DrawLine 的参数。

- 在 CALL 中传进去的值叫作**实参**（argument）。实参的**顺序**和**类型**必须与参数一致。
- 没有参数的过程也要写空括号：ShowTitle()。
- 一条 CALL 本身就是一条完整的语句，单独占一行。

■ 为什么要用子程序?

- 同样的代码不必一遍又一遍地写。
- 每个子程序都可以单独编写和测试。
- 主程序变得简短、易读。

常见错误

- 调用过程时不写 CALL：应写 CALL DrawLine(5, '*')，而不是只写 DrawLine(5, '*')。
- 在过程头部漏写参数的数据类型。
- 忘记写 ENDPROCEDURE。

FUNCTION 与 RETURN

函数（function）和过程很像，但它会把**一个值返回**到调用它的地方。函数头部要写明返回值的数据类型。

```
FUNCTION <identifier>(<param1> : <data type> ...) RETURNS <data type>
   <statement(s)>
ENDFUNCTION
```

```
FUNCTION Max(Number1 : INTEGER, Number2 : INTEGER) RETURNS INTEGER
   IF Number1 > Number2 THEN
      RETURN Number1
   ELSE
      RETURN Number2
   ENDIF
ENDFUNCTION

DECLARE Distance : INTEGER
Distance ← 8
OUTPUT "Penalty fine = ", Max(10, Distance * 2)
```

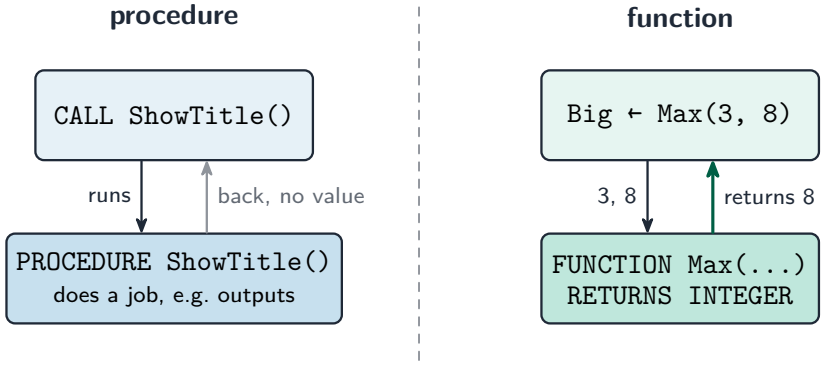
- RETURN 把值送回去。RETURN 一执行，函数就**立刻结束**，后面的语句都会被跳过。
- 函数调用本身**不是**一条语句，要把它用在**表达式里**：赋值、OUTPUT、条件，或者作为另一个调用的实参。
- 调用函数时**绝不能**用 CALL。

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

DECLARE Value : INTEGER
FOR Value ← 1 TO 4
   IF IsEven(Value) THEN
      OUTPUT Value, " is even"
   ENDIF
NEXT Value
```

■ 用过程还是函数?

- 子程序要算出调用者需要的**一个值**：用**函数**。
- 子程序是去**做**一件事（输出、修改数组、写文件），没有唯一的答案：用**过程**。



过程被调用去完成一项任务；函数返回一个值供使用

常见错误

- `CALL Max(3, 4)`。函数要用在表达式里：`Biggest ← Max(3, 4)`。
- 函数头部漏写 `RETURNS <data type>`，或者函数体里漏写 `RETURN` 语句。
- 题目要求函数**返回**一个值，你却在函数里把答案输出了。

BYVAL 与 BYREF

把值传给参数有两种方式。

- **按值传递**（by value，`BYVAL`）：子程序得到值的一个**副本**。子程序修改参数时，调用者的变量**不会**改变。
- **按引用传递**（by reference，`BYREF`）：子程序得到的是调用者的**那个变量本身**。子程序里的修改，调用结束后调用者**能**看到。

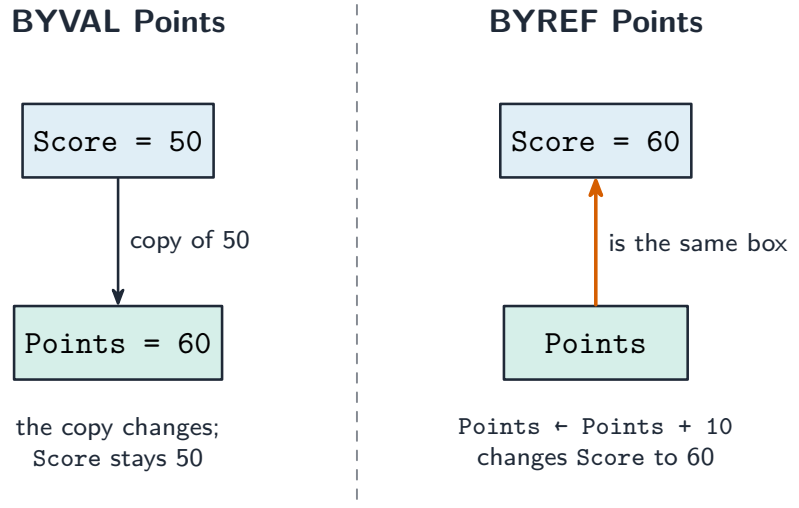
把关键字写在头部的参数前面。如果两个都不写，参数就**按值**传递。

```
PROCEDURE AddBonus(BYVAL Points : INTEGER)
   Points ← Points + 10
ENDPROCEDURE

PROCEDURE AddBonusRef(BYREF Points : INTEGER)
   Points ← Points + 10
ENDPROCEDURE

DECLARE Score : INTEGER
Score ← 50
CALL AddBonus(Score)
OUTPUT "After BYVAL: ", Score
CALL AddBonusRef(Score)
OUTPUT "After BYREF: ", Score
```

第一个 `OUTPUT` 显示 50（只有副本变了），第二个显示 60。



BYVAL 给过程一个副本；*BYREF* 给过程调用者自己的变量

■ 一个关键字管多个参数

如果连续几个参数用同一种方式传递，关键字只写一次。下面 X 和 Y **都**是 BYREF：

```
PROCEDURE Swap(BYREF X : INTEGER, Y : INTEGER)
   DECLARE Temp : INTEGER
   Temp ← X
   X ← Y
   Y ← Temp
ENDPROCEDURE

DECLARE A : INTEGER
DECLARE B : INTEGER
A ← 3
B ← 8
CALL Swap(A, B)
OUTPUT "A = ", A, ", B = ", B
```

■ 什么时候用 BYREF

- 过程需要送回**不止一个**结果。
- 过程需要修改调用者的变量，就像 Swap 那样。

不要在**函数**中按引用传递参数。函数用 RETURN 送回它的那一个值。

常见错误

- 以为修改普通（BYVAL）参数后，外面也能看到变化。它只是一个副本。
- 在 FUNCTION 头部使用 BYREF。
- 在 IGCSE 0478 的答案里写 BYREF。那个考纲里没有它。

局部变量与全局变量

变量的**作用域**（scope）是程序中可以使用这个变量的范围。

- **局部变量**（local variable）声明在子程序**内部**。它只在这个子程序运行期间存在，也只有这个子程序能用它。参数也是局部的。
- **全局变量**（global variable）声明在**主程序**中，在所有子程序之外。程序的任何部分都能使用它。

```
DECLARE Total : INTEGER

PROCEDURE AddToTotal(Amount : INTEGER)
   DECLARE Doubled : INTEGER
   Doubled ← Amount * 2
   Total ← Total + Doubled
ENDPROCEDURE

Total ← 0
CALL AddToTotal(5)
CALL AddToTotal(3)
OUTPUT "Total: ", Total
```

- `Total` 是**全局**的：过程可以修改它，主程序能看到结果 16。
- `Doubled` 和 `Amount` 是 `AddToTotal` 的**局部**变量，主程序不能使用它们。

■ 尽量使用局部变量

- 只使用参数和局部变量的子程序，可以单独测试、重复使用。
- 两个子程序都可以有一个叫 `i` 的局部变量，互不冲突。
- 如果许多子程序都去修改同一个全局变量，出了错就很难找。

所以：用**参数**把数据传进去，用 `RETURN`（或 `BYREF`）把结果送回来，全局变量只留给真正属于整个程序的少数几个值。

■ 练习

1. 写一个函数 `Area(Width : REAL, Height : REAL) RETURNS REAL`，然后用它输出一个 3.5×2.0 的长方形的面积。

2. 写一个过程 `MinMax`，接收三个整数，通过两个 `BYREF` 参数送回最小值和最大值。

<details> <summary> 参考答案 </summary>

```
FUNCTION Area(Width : REAL, Height : REAL) RETURNS REAL
   RETURN Width * Height
ENDFUNCTION

OUTPUT "Area: ", Area(3.5, 2.0)
```

```
PROCEDURE MinMax(A : INTEGER, B : INTEGER, C : INTEGER, BYREF Smallest :
↪  INTEGER, Largest : INTEGER)
   Smallest ← A
   Largest ← A
   IF B < Smallest THEN
      Smallest ← B
   ENDIF
   IF C < Smallest THEN
      Smallest ← C
   ENDIF
   IF B > Largest THEN
      Largest ← B
   ENDIF
   IF C > Largest THEN
      Largest ← C
   ENDIF
ENDPROCEDURE

DECLARE Low : INTEGER
DECLARE High : INTEGER
CALL MinMax(7, 2, 9, Low, High)
OUTPUT "Smallest ", Low, ", largest ", High
```

A、B 和 C 按值传递（默认方式）。Smallest 前面的 BYREF 同样适用于 Largest，因为关键字没有重复书写。

</details>