

11.3.1 Defining and calling procedures and functions, and choosing between them

Name: _____ Class: _____ Date: _____

Total: 30 marks

KEY WORDS parameter 参数 • procedure 过程 • function 函数 • return value 返回值
• argument 实参

Objective

Build the skills to answer exam questions on **structured programming** 结构化编程 that sheet 11.3 only starts. That sheet defines a **procedure** 过程 and a **function** 函数 and contrasts the two ways of passing a parameter; this one is about actually **writing** them: choosing which of the two a task needs, writing a header that earns its marks, using the right words for each part of a subroutine, and getting a value back to the caller.

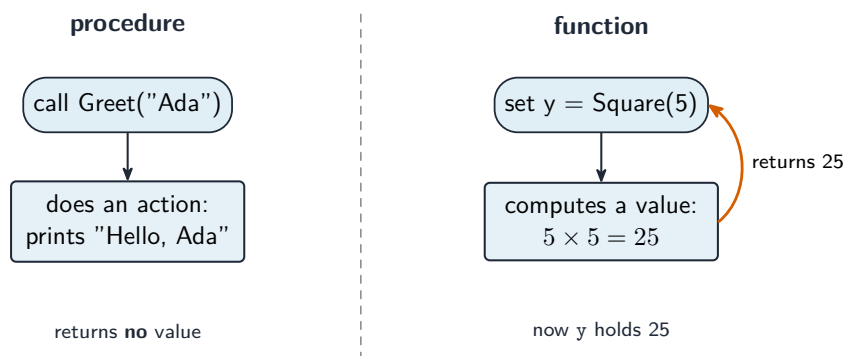
You must be able to:

- define and use a procedure, and explain where in an algorithm it is appropriate to use one
- define and use a function, and explain where it is appropriate to use one
- use the terminology: procedure and function **header**, **interface**, **parameter** 参数, **argument** 实参 and **return value** 返回值
- write efficient pseudocode for a module given only a description of what it must do

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

- Which one does the task need?



Ask one question: **does the caller need a value back?**

- **No, it just needs something done** – display a menu, print a report, update a file. Use a **procedure**, and call it on a line of its own with **CALL**.
- **Yes, it computes one value the caller then uses** – a total, a grade, a TRUE/FALSE answer. Use a **function**, because the **return value replaces the call** inside an expression: IF IsValid(Code) THEN.

The syllabus asks *where in the construction of an algorithm* each is appropriate, so answer in those terms: a procedure where the same group of steps is needed at several points, a function where a single value must be calculated and then used.

■ The words the mark scheme uses

```

FUNCTION Area(Length : REAL, Width : REAL) RETURNS REAL    <- header
    RETURN Length * Width                                   <- return
    ↪ value
ENDFUNCTION

Total <- Area(3.5, 2.0)                                     <- call,
    ↪ with arguments

```

Term	What it means here
header	the first line: the keyword, the name, the parameters and (for a function) RETURNS and its type
interface	what a caller must know to use it: its name, the parameters with their types and order, and what it returns
parameter	the variable inside the subroutine that receives a value – Length, Width
argument	the value the caller supplies – 3.5, 2.0
return value	the value handed back, which replaces the call in the expression – here 7.0

- The parameter is in the definition, the argument is at the call. Saying it the other way round loses the mark.

■ Writing a module that scores

```
FUNCTION CountAbove(BYVAL Limit : INTEGER) RETURNS INTEGER
    DECLARE Index, Count : INTEGER
    Count ← 0
    FOR Index ← 1 TO 50
        IF Score[Index] > Limit THEN
            Count ← Count + 1
        ENDIF
    NEXT Index
    RETURN Count
ENDFUNCTION
```

..... header: name, parameters, return type

..... local variables declared with a type

..... total initialised before the loop

..... loop over every element

..... the condition, with the right boundary

..... the update, inside the IF

..... every construct closed

..... the return value, after the loop

A “write a module” question is marked point by point, and the points are almost always these. Write them in this order and the marks follow.

1. **Header** – keyword, name, every parameter with its type, and RETURNS <type> for a function.
2. **Declare the locals** – including the loop counter.
3. **Initialise** any running total, count or string **before** the loop.
4. **Loop** – over the right range.
5. The **test** inside the loop, using the parameter rather than a fixed value.
6. The **update** inside the loop.
7. **Close every construct**: ENDIF, NEXT, ENDWHILE, ENDFUNCTION.
8. **Return** after the loop, not inside it.

Worked example. Write a function *IsLeap* that takes a year and returns *TRUE* if it is a leap year. A year is a leap year when it divides by 4, except that a century year must divide by 400.

```
FUNCTION IsLeap(Year : INTEGER) RETURNS BOOLEAN
    IF Year MOD 400 = 0 THEN
        RETURN TRUE
    ENDIF
    IF Year MOD 100 = 0 THEN
        RETURN FALSE
    ENDIF
    IF Year MOD 4 = 0 THEN
        RETURN TRUE
    ENDIF
    RETURN FALSE
ENDFUNCTION
```

- Testing in order of priority and returning as soon as a test passes removes the need for nested IFs – that is what “efficient pseudocode” means here.

■ Getting a value back from a procedure

A procedure returns nothing, so when the caller must see a change, pass the variable **by reference**:

```
PROCEDURE AddTo(BYREF Total : REAL, BYVAL Amount : REAL)
    Total <- Total + Amount
ENDPROCEDURE
```

- If the subroutine's whole job is to produce **one** value, prefer a function: `Total <- Total + Amount` at the call site is clearer than a `BYREF` parameter.

2 Practice

Now use the methods above.

2.1 For each task, state whether a **procedure** or a **function** is the better choice, and give a reason.

(a) display a menu of options on the screen [1]

(b) work out the discount on a price [1]

(c) exchange the values of two variables [1]

2.2 Write the **header** for a function `Area` that takes two real numbers and returns a real number. [2]

2.3 A program contains the statement `Total <- Area(3.5, 2.0)`. State the **arguments** in this call, and state what the **return value** replaces. [2]

2.4 Explain what is meant by the **interface** of a function.

[2]

2.5 State **one** reason for preferring a local variable to a global variable inside a subroutine.

[1]

3 Exam-style questions

3.1 Write a **function** `Initials()` that takes a person's full name as a string of two or more words separated by single spaces, and returns a string of their initials in capitals with no spaces. For example, `"ada byron lovelace"` returns `"ABL"`.

You may use `LENGTH`, `MID` and `UCASE`.

[6]

3.2 Write a **procedure** `Countdown()` that takes a positive integer `N` as a parameter, outputs the numbers from `N` down to 1 on separate lines, and then outputs "Go". [5]

3.3 A program works out a discounted price in three different places. The same three lines of code are used each time, but on different prices and rates. The programmer decides to replace them with a function `Discount()`.

(a) Write the **header** for `Discount()`. [2]

(b) Write the statement that calls `Discount()` for a price of 80.00 at a rate of 15, storing the result in `Final`. [1]

(c) State why a **function** suits this better than a procedure. [1]

3.4 A function `Grade()` takes an integer mark from 0 to 100 and returns a grade as a string: "A" for 70 or more, "B" for 60 or more, "C" for 50 or more, and "F" below 50.

Write the function, and write it so that no mark is tested more than it needs to be. [5]

Solutions

Each answer shows the steps, then the **result**.

2.1

- (a) a **procedure** – it performs an action and the caller needs no value back
- (b) a **function** – it computes one value that the caller then uses in an expression
- (c) a **procedure**, with both parameters passed **BYREF** – two values change, so there is no single value to return

2.2

- keyword, name and both parameters with their types
- **RETURNS REAL**

```
FUNCTION Area(Length : REAL, Width : REAL) RETURNS REAL
```

2.3

- the arguments are 3.5 and 2.0 – the values the caller supplies
- the return value **replaces the call** in the expression, so **Total** is assigned 7.0

2.4

- the interface is what a caller must know in order to use the function
- its name, its parameters with their types and order, and the type of the value it returns

2.5

- any one: the same identifier can then be used in another subroutine without a clash; its value cannot be changed accidentally by other parts of the program; the subroutine stays self-contained so it can be tested on its own and reused

3.1

- header with the parameter and **RETURNS STRING**
- declare the locals, including the loop counter, and start the result as ""
- loop over the whole name
- take the first character, and every character that follows a space
- concatenate it, in capitals, onto the result
- **RETURN** after the loop

```

FUNCTION Initials(FullName : STRING) RETURNS STRING
  DECLARE Result : STRING
  DECLARE Index : INTEGER
  Result <- ""
  FOR Index <- 1 TO LENGTH(FullName)
    IF Index = 1 OR MID(FullName, Index - 1, 1) = " " THEN
      Result <- Result & UCASE(MID(FullName, Index, 1))
    ENDIF
  NEXT Index
  RETURN Result
ENDFUNCTION

```

3.2

- PROCEDURE header with the parameter, and ENDPROCEDURE
- a loop that counts **down** from N to 1
- output each number inside the loop
- output "Go" once, **after** the loop

```

PROCEDURE Countdown(N : INTEGER)
  DECLARE Count : INTEGER
  FOR Count <- N TO 1 STEP -1
    OUTPUT Count
  NEXT Count
  OUTPUT "Go"
ENDPROCEDURE

```

- a WHILE loop that decreases a copy of N earns the same marks

3.3

(a) FUNCTION Discount(Price : REAL, Rate : REAL) RETURNS REAL

- one mark for the keyword, name and both parameters; one for RETURNS REAL

(b) Final <- Discount(80.00, 15)

(c) the caller needs the discounted price **back** to use it, and a function's return value replaces the call inside the expression

3.4

- header with the parameter and RETURNS STRING
- test the **highest** boundary first
- return as soon as a test passes, so no later test runs
- the remaining boundaries in descending order
- a final RETURN "F" for everything below 50

```
FUNCTION Grade(Mark : INTEGER) RETURNS STRING
  IF Mark >= 70 THEN
    RETURN "A"
  ENDIF
  IF Mark >= 60 THEN
    RETURN "B"
  ENDIF
  IF Mark >= 50 THEN
    RETURN "C"
  ENDIF
  RETURN "F"
ENDFUNCTION
```

- testing from the top down means each IF only has to check one boundary: a mark of 65 fails the first test and passes the second, so `Mark < 70 AND Mark >= 60` is never needed