

Exercise walk-through

11.3.3

Efficient pseudocode and breaking a task into modules

A-Level Computer Science

The six parts of a module answer

Half of Paper 2 is “write pseudocode for module X”. The scheme awards a mark **per feature**, so a module that is not finished still scores for every correct part.

1 The header

as the question gives it, with BYREF where an argument must change

2 Local declarations

DECLARE every local with its type; set counters and totals to 0

3 The loop

FOR over a given size; WHILE NOT EOF(...) for a file

4 The condition

the right comparison and the right boundary, on the right item

5 The update

inside the branch: the count increased, the value stored, or the message output

6 The end

RETURN once, after the loop; every IF, FOR and WHILE closed

Worked example: IsValid, the plan

Question

Write a function `IsValid(Code : STRING)` that returns `TRUE` when `Code` is two capital letters followed by four digits (the **format** `AB1234`), and `FALSE` otherwise.

- check the **length first**, so `MID` never reads past the end
- visit positions 1 to 6: capitals at 1 and 2, digits at 3 to 6
- `RETURN FALSE` at the first bad character, `RETURN TRUE` after the loop

```
FUNCTION IsValid(BYVAL Code : STRING)
    RETURNS BOOLEAN
    DECLARE Index : INTEGER
    DECLARE Ch : STRING
    IF LENGTH(Code) <> 6 THEN
        RETURN FALSE
    ENDIF
    // ... the character loop, next
```

Worked example: IsValid, each character

```
FOR Index ← 1 TO 6
  Ch ← MID(Code, Index, 1)
  IF Index ≤ 2 THEN
    IF Ch < "A" OR Ch > "Z" THEN
      RETURN FALSE
    ENDIF
  ELSE
    IF Ch < "0" OR Ch > "9" THEN
      RETURN FALSE
    ENDIF
  ENDIF
NEXT Index
RETURN TRUE
ENDFUNCTION
```

Validation like this
returns a BOOLEAN, so the
caller decides what to do:

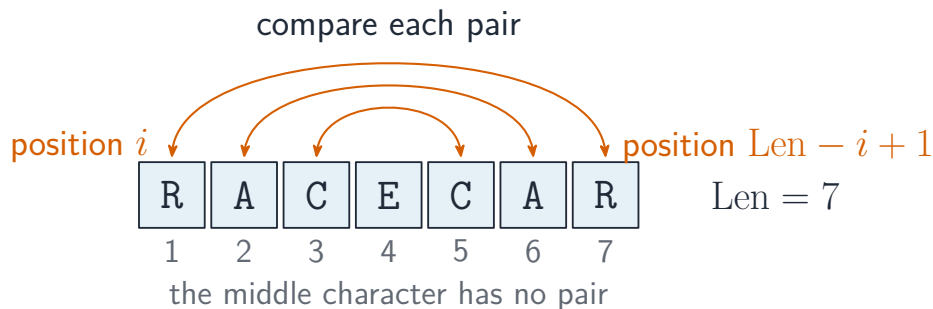
```
IF IsValid(Entry) THEN
  CALL Save(Entry)
ELSE
  OUTPUT "Invalid"
ENDIF
```

The **caller** outputs the
message: a function
calculates, a procedure acts.

Worked example: IsPalindrome, the method

Question

Write a function `IsPalindrome(Word : STRING)` that returns `TRUE` when `Word` reads the same backwards, such as "RACECAR".



Compare the characters from the two ends, moving inwards: position `Index` pairs with position `Len - Index + 1`, and only the first half needs testing.

Worked example: IsPalindrome, the function

```
FUNCTION IsPalindrome(BYVAL Word : STRING) RETURNS BOOLEAN
  DECLARE Len, Index : INTEGER
  Len ← LENGTH(Word)
  FOR Index ← 1 TO Len DIV 2
    IF MID(Word, Index, 1) <> MID(Word, Len - Index + 1, 1) THEN
      RETURN FALSE
    ENDIF
  NEXT Index
  RETURN TRUE
ENDFUNCTION
```

Len DIV 2 stops at the middle: an odd-length word's middle character has no pair, and needs no test.

Exercise 2.1 (a)

2.1 Each fragment does some work that is not needed. State what is not needed. [2]

(a)

```
IF Mark >= 70 THEN
    Grade ← "A"
ELSE
    IF Mark < 70 AND Mark >= 60 THEN
        Grade ← "B"
    ENDIF
ENDIF
```



Exercise 2.1 (a)

2.1 Each fragment does some work that is not needed. State what is not needed. [2]

(a)

```
IF Mark >= 70 THEN
    Grade ← "A"
ELSE
    IF Mark < 70 AND Mark >= 60 THEN
        Grade ← "B"
    ENDIF
ENDIF
```



(a) the test `Mark < 70`: the `ELSE` part is only reached when `Mark >= 70` is `FALSE`, so the mark is already known to be below 70

Exercise 2.1 (b)

2.1 Each fragment does some work that is not needed. State what is not needed. [2]

(b)

```
FOR Item ← 1 TO 100
    Rate ← 0.2
    INPUT Price
    OUTPUT Price * Rate
NEXT Item
```

Exercise 2.1 (b)

2.1 Each fragment does some work that is not needed. State what is not needed. [2]

(b)

```
FOR Item ← 1 TO 100
    Rate ← 0.2
    INPUT Price
    OUTPUT Price * Rate
NEXT Item
```

(b) `Rate ← 0.2` is inside the loop, so the same value is stored 100 times; it should be set once, before the loop (or be a constant)

Exercise 2.2

2.2 Rewrite this loop so that no work is repeated unnecessarily, and state the change you made. [3]

```
FOR Index ← 1 TO LENGTH(Names)
    OUTPUT UCASE(MID(Names, Index, 1))
NEXT Index
```



Exercise 2.2

2.2 Rewrite this loop so that no work is repeated unnecessarily, and state the change you made. [3]

```
FOR Index ← 1 TO LENGTH(Names)
    OUTPUT UCASE(MID(Names, Index, 1))
NEXT Index
```



2.2

- LENGTH(Names) is recomputed on every pass, so compute it once into a local variable

```
DECLARE Len : INTEGER
Len ← LENGTH(Names)
FOR Index ← 1 TO Len
    OUTPUT UCASE(MID(Names, Index, 1))
NEXT Index
```

- the rewritten loop, with the declaration

Exercise 2.3

2.3 State **two** things that make pseudocode inefficient.

[2]



Exercise 2.3

2.3 State **two** things that make pseudocode inefficient.

[2]

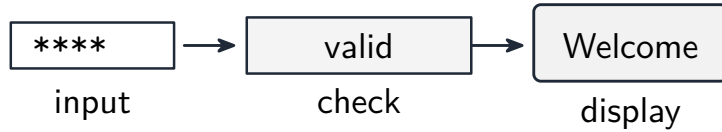


2.3

- any two: repeating a calculation inside a loop whose answer does not change; carrying on looping after the answer is known; testing a condition that an earlier test has already settled; using a loop or a long CASE where arithmetic would do

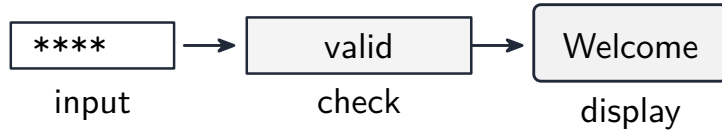
Exercise 2.4

2.4 A program must: input a password; check whether it is valid; and display a welcome screen. State, for each of the three jobs, whether it is better written as a **procedure** or a **function**, with a reason. [3]



Exercise 2.4

2.4 A program must: input a password; check whether it is valid; and display a welcome screen. State, for each of the three jobs, whether it is better written as a **procedure** or a **function**, with a reason. [3]



2.4

- input the password: a **procedure** – it performs an action and returns nothing the caller must have
- check whether it is valid: a **function** – it produces one value (TRUE/FALSE) that the rest of the algorithm uses in an IF
- display the welcome screen: a **procedure** – it does something, and there is no value to return

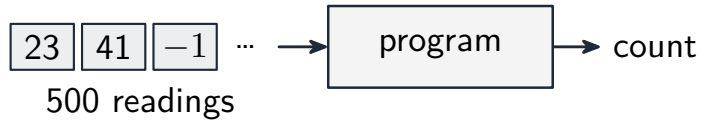
Setup for exercise 3.1

3.1 A module inputs 500 sensor readings, one at a time, and counts how many of them are above a given limit. A reading of -1 means “no reading”, and it must not be counted.

```
FUNCTION HowMany(Limit : INTEGER) RETURNS INTEGER
  DECLARE Index : INTEGER
  DECLARE Reading : INTEGER
  Count ← 0
  FOR Index ← 1 TO 500
    INPUT Reading
    IF Reading > Limit AND Reading <> -1 THEN
      IF Reading > Limit THEN
        Count ← Count + 1
      ENDIF
    ENDIF
  NEXT Index
  RETURN Count
ENDFUNCTION
```

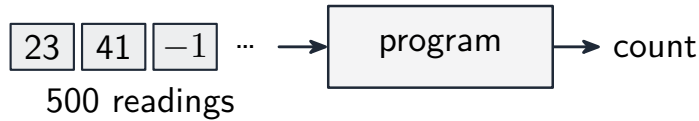
Exercise 3.1 (a)

(a) The limit is never negative. State **two** ways in which this pseudocode is inefficient or incorrect. [2]



Exercise 3.1 (a)

(a) The limit is never negative. State **two** ways in which this pseudocode is inefficient or incorrect. [2]



(a) the inner IF repeats a test that the outer IF has already made

- the limit is never negative, so a reading above the limit can never be -1: the test `Reading <> -1` is not needed; also `Count` is used without being declared

Exercise 3.1 (b)

(b) Rewrite the module efficiently.

[3]

Exercise 3.1 (b)

(b) Rewrite the module efficiently.

[3]

(b)

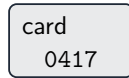
```
FUNCTION HowMany(Limit : INTEGER) RETURNS INTEGER
  DECLARE Index, Count, Reading : INTEGER
  Count ← 0
  FOR Index ← 1 TO 500
    INPUT Reading
    IF Reading > Limit THEN
      Count ← Count + 1
    ENDIF
  NEXT Index
  RETURN Count
ENDFUNCTION
```

- one test instead of three; Count declared as a local variable; the return kept outside the loop

Exercise 3.2

3.2 A program for a library must: find a member by their card number; work out the fine owed on a late book; and print a receipt.

Break the task into **three** modules. For each one, state whether it is a procedure or a function and write its header. [6]



member



fine



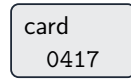
receipt

A large, empty rectangular box with a thin black border, intended for the student to write the module headers.

Exercise 3.2

3.2 A program for a library must: find a member by their card number; work out the fine owed on a late book; and print a receipt.

Break the task into **three** modules. For each one, state whether it is a procedure or a function and write its header. [6]



member



fine



receipt

A large, empty rectangular box with a thin border, intended for the student to write the headers for the three modules.

3.2

- find a member: a **function** – it produces one value the rest of the program uses

Full solution 3.2

3.2

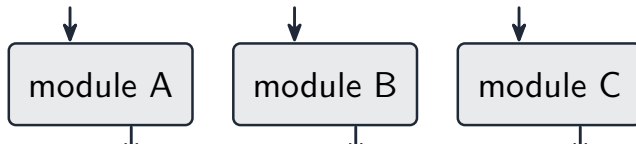
- find a member: a **function** – it produces one value the rest of the program uses
`FUNCTION FindMember(CardNumber : STRING) RETURNS INTEGER`
- work out the fine: a **function** – one value, used in an expression
`FUNCTION Fine(Days Late : INTEGER) RETURNS REAL`
- print a receipt: a **procedure** – it performs an action and returns nothing
`PROCEDURE PrintReceipt(MemberID : INTEGER, Amount : REAL)`
- sensible names and types are accepted; a module that both finds **and** prints would lose a mark, because it does two jobs

Exercise 3.3 (a)

3.3 A programmer writes each module so that it uses only its parameters and its own local variables.

(a) Explain why this makes a module easier to **test**.

[2]

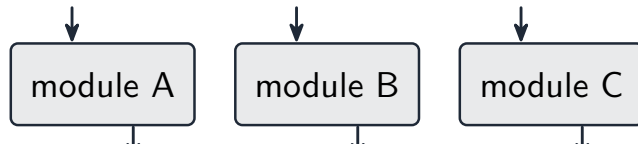


Exercise 3.3 (a)

3.3 A programmer writes each module so that it uses only its parameters and its own local variables.

(a) Explain why this makes a module easier to **test**.

[2]



(a) everything the module uses arrives through its **interface**, so it can be called on its own with chosen test values

- nothing outside it can change its variables, so the same inputs always give the same result and a fault is traceable to that module

Exercise 3.3 (b)

3.3 A programmer writes each module so that it uses only its parameters and its own local variables.

(b) State **two** other benefits of building a program from modules. [2]

Exercise 3.3 (b)

3.3 A programmer writes each module so that it uses only its parameters and its own local variables.

(b) State **two** other benefits of building a program from modules. [2]

(b) any two: the same module can be reused elsewhere in the program or in another program; several programmers can work on different modules at once; a change is made in one place only; the program is easier to read because each module has one job

Exercise 3.4

3.4 A function is written to return the position of the first space in a string.

```
FUNCTION FirstSpace(InString : STRING) RETURNS INTEGER
  DECLARE Index, Found : INTEGER
  Found ← 0
  FOR Index ← 1 TO LENGTH(InString)
    IF MID(InString, Index, 1) = " " THEN
      IF Found = 0 THEN
        Found ← Index
      ENDIF
    ENDIF
  NEXT Index
  RETURN Found
ENDFUNCTION
```

State **two** inefficiencies, and rewrite the loop so the function stops as soon as it has the answer. [4]

"ada lovelace" → FirstSpace → 4



Exercise 3.4

3.4 A function is written to return the position of the first space in a string.

```
FUNCTION FirstSpace(InString : STRING) RETURNS INTEGER
  DECLARE Index, Found : INTEGER
  Found ← 0
  FOR Index ← 1 TO LENGTH(InString)
    IF MID(InString, Index, 1) = " " THEN
      IF Found = 0 THEN
        Found ← Index
      ENDIF
    ENDIF
  NEXT Index
  RETURN Found
ENDFUNCTION
```

State **two** inefficiencies, and rewrite the loop so the function stops as soon as it has the answer. [4]

"ada lovelace" → FirstSpace → 4



3.4

- the inner IF Found = 0 is only needed because the loop keeps running after the answer is known
- LENGTH(InString) is recomputed on every pass

Full solution on next slide

Full solution 3.4

3.4

- the inner IF Found = 0 is only needed because the loop keeps running after the answer is known
- LENGTH(InString) is recomputed on every pass

```
FUNCTION FirstSpace(InString : STRING) RETURNS INTEGER
  DECLARE Index, Len : INTEGER
  Len ← LENGTH(InString)
  FOR Index ← 1 TO Len
    IF MID(InString, Index, 1) = " " THEN
      RETURN Index
    ENDIF
  NEXT Index
  RETURN 0
ENDFUNCTION
```

- returning immediately removes the flag and the inner test
- a final RETURN 0 for a string with no space, so a value comes back in both cases