Week 30

A-Level Computer Science

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

A-Level Computer Science Name: _____ Score: _____

1. A constant holds a fixed value that never changes while the program runs, whereas a variable can be reassigned.

   ☐ True ☐ False

2. ”A mark of 50 or more passes” is written as ‘IF Mark _____ 50 THEN‘.

   _____

3. The key difference between a procedure and a function is that a function:

   ☐ returns a value that can be used in an expression

   ☐ cannot take parameters

   ☐ never does any work

   ☐ must be global

4. Which are benefits of using a constant for a value such as a tax rate? Select **all** that apply.

   ☐ it is set once and cannot be changed accidentally by the program

   ☐ a change is made in one place and reaches every use

   ☐ the identifier gives the value a meaningful name

   ☐ the program runs twice as fast

   Tick every correct option.

5. A CASE statement is cleaner than nested IFs when you are:

   ☐ testing one value against several options

   ☐ repeating a block of code

   ☐ reading a file

   ☐ declaring a constant

6. A function ‘Square(x)‘ returns ‘x * x‘. What does the call ‘Square(5)‘ return?

   _____

7. What is the value of ‘7 DIV 2‘?

   _____

8. Which of these are valid guards in a Cambridge CASE statement? Select **all** that apply.

   ☐ ‘1:‘

   ☐ ‘1, 2, 3:‘

☐ ‘1 TO 5:‘

☐ ‘> 200:‘

☐ ‘OTHERWISE:‘

Tick every correct option.

9. A good reason to write a subroutine is that:

   ☐ the same logic is needed in more than one place

   ☐ you want to use more global variables

   ☐ the program is only one line long

   ☐ you want to avoid naming things

10. What is the value of ‘7 MOD 2‘?

    _____

11. Programming

Answers · 答案

A-Level Computer Science

1. **True**

   Constants (like Pi or a maximum score) make code clearer and let you change a recurring value in one place.

2. **>=**

   ”Or more” includes 50 itself, so the comparison is greater than or equal. ‘>‘ would fail a student on exactly 50.

3. **returns a value that can be used in an expression**

   A function returns a value (used in an expression); a procedure performs an action and returns nothing.

4. **it is set once and cannot be changed accidentally by the program; a change is made in one place and reaches every use; the identifier gives the value a meaningful name**

   The three mark-scheme benefits are safety, a single point of change, and readability. Speed is not one of them.

5. **testing one value against several options**

   CASE matches one value against many possibilities; deep nested IFs become hard to read.

6. **25**

   $5 \times 5 = 25$ — the value the function hands back to its caller (the frame popped off the call stack).

7. **3**

   DIV is integer division: $7 \div 2 = 3$ remainder 1, so DIV gives 3.

8. **‘1:‘; ‘1, 2, 3:‘; ‘1 TO 5:‘; ‘OTHERWISE:‘**

   A single value, a value list, a range and OTHERWISE. A comparison such as ‘> 200‘ is not a guard; anything not covered goes to OTHERWISE.

9. **the same logic is needed in more than one place**

   Subroutines remove duplication, give a named purpose, and can be tested in isolation.

10. **1**

    MOD gives the remainder: $7 \div 2$ leaves remainder 1.

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

11.1 Programming Basics

Name: _____ Class: _____ Date: _____ **Total: 35 marks**

KEY WORDS pseudocode 伪代码 • variables 变量 • constant 常量 • function 函数 • operators 运算符

Objective

Build the skills to answer exam questions on **programming basics** 编程基础 — constants, variables, operators and built-in functions.

You must be able to:

- declare **constants** and **variables**, and use **assignment**
- evaluate expressions with arithmetic, comparison and logic **operators** (including `DIV` and `MOD`)
- use **built-in functions** such as `LENGTH`, `LEFT`, `MID`, `UCASE`

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

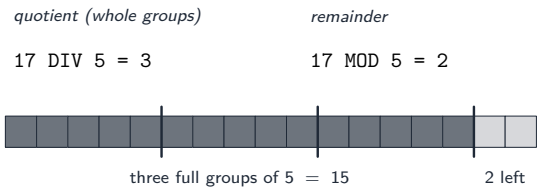
Constants and variables

A **constant** holds a value that never changes; a **variable** holds one that can. Declare with a type, assign with ←:

```
CONSTANT Pi ← 3.14159
DECLARE Radius : REAL
Radius ← 5
Area ← Pi * Radius * Radius
```

Operators: DIV and MOD

`DIV` is the **whole-number quotient**; `MOD` is the **remainder**:



Precedence (do first → last): `NOT` → `* / DIV MOD` → `+ -` → comparisons → `AND` → `OR`. So `3 + 4 * 2` = `3 + 8` = `11`. Use brackets when unsure.

Built-in functions

Function	Example	Result
`LENGTH(s)`	`LENGTH("Computer")`	`8`
`LEFT(s, n)`	`LEFT("Computer", 3)`	`"Com"`
`MID(s, start, len)`	`MID("Computer", 4, 3)`	`"put"`
`UCASE(s)`	`UCASE("hi")`	`"HI"`
`INT(x)`	`INT(3.9)`	`3`

`MID` counts characters from **1**. Use the exact names from the paper's reference list.

2 Practice

Now apply the methods above.

2.1 Write pseudocode to declare a **constant** `Pi` of `3.14159` and a **real** variable `Radius`. [2]

2.2 Evaluate `23 DIV 4` and `23 MOD 4`. [2]

2.3 Evaluate `3 + 4 * 2` (use precedence). [1]

2.4 State the result of `LENGTH("Computer")` and `LEFT("Computer", 3)`. [2]

3 Exam-style questions

3.1 Write pseudocode that inputs a `Price`, adds **20% tax**, and outputs the total. Use a **constant** for the tax rate. [4]

3.2 Evaluate each expression. [5]

Expression	Value
`7 DIV 2`	
`7 MOD 2`	
`(5 > 3) AND (2 > 4)`	
`NOT (1 = 1)`	
`MID("PROGRAM", 2, 3)`	

3.3 Write pseudocode that reads a word and outputs it in **capitals** followed by its **length**, using built-in functions. [3]

3.4 A program stores the marks of 30 students in a 1D array `Mark`, and reports how many passed (a mark of 50 or more).

(a) **Complete** the table by giving an appropriate data type and an example value for each variable. [4]

variable	purpose	data type	example value
`Mark[1]`	one student's mark		
`PassRate`	percentage that passed		
`TopName`	name of the best student		
`AllPassed`	did everyone pass?		

(b) **Write** pseudocode that counts the passes and calculates `PassRate`. [5]

(c) A programmer initialises the counter with `Count ← 0` before the loop. **Explain** what would happen if this line were left out. [2]

(d) **Describe** how an **IDE** helps a programmer locate a fault during **alpha testing**. [3]

(e) **State** the difference between **alpha** and **beta** testing. [2]

Solutions

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

2.1

- `CONSTANT` with a value, then `DECLARE … : REAL`

```
CONSTANT Pi ← 3.14159
DECLARE Radius : REAL
```

2.2

- `23 DIV 4 = 5` (integer quotient)
- `23 MOD 4 = 3` (remainder)

2.3

- multiply first (`4 * 2 = 8`), then add `3` → **11**

2.4

- `LENGTH("Computer") = 8`
- `LEFT("Computer", 3) = "Com"`

3.1

- constant for tax, `INPUT Price`, then compute and output

```
CONSTANT TaxRate ← 0.20
DECLARE Price : REAL
DECLARE Total : REAL
INPUT Price
Total ← Price * (1 + TaxRate)
OUTPUT "Total: ", Total
```

3.2

- `7 DIV 2 = 3`
- `7 MOD 2 = 1`
- `(5 > 3) AND (2 > 4) = FALSE`
- `NOT (1 = 1) = FALSE`
- `MID("PROGRAM", 2, 3) = "ROG"`

3.3

- `INPUT`, then `UCASE` and `LENGTH`

```
OUTPUT "Enter a word:"
INPUT Word
OUTPUT UCASE(Word)
OUTPUT "Length: ", LENGTH(Word)
```

3.4

(a) `Mark[1]` — `INTEGER`, e.g. `67`

- `PassRate` — `REAL`, e.g. `73.3`
- `TopName` — `STRING`, e.g. `"Wei"`
- `AllPassed` — `BOOLEAN`, e.g. `FALSE`

(b)

```
Count ← 0
FOR Index ← 1 TO 30
    IF Mark[Index] >= 50
      THEN
        Count ← Count + 1
    ENDIF
NEXT Index
PassRate ← Count / 30 * 100
OUTPUT PassRate
```

- initialise the counter; loop over all 30; correct condition; increment inside the `IF`; percentage calculated after the loop

(c) the counter would hold whatever value happened to be in that memory location, or the program would fail with an "unassigned variable" error

- either way the pass count and the rate would be wrong, and the error might not show every time it runs

(d) an IDE provides a **debugger**: breakpoints stop the program at a chosen line

- single-stepping runs one statement at a time
- a watch window shows the changing value of each variable, so the programmer sees exactly where the value first goes wrong

(e) **alpha** testing is done **inside** the development organisation, on incomplete software, by staff

- **beta** testing releases the near-finished software to selected **external users**, who report faults found in real use

1 A program is being developed to meet a particular customer requirement.

(a) The program contains these variables:

Variable	Data type
`MyChar`	`CHAR`
`MyString`	`STRING`
`MyInt`	`INTEGER`
`MyDOB`	`DATE`

Complete the table by filling in the gaps using functions and/or operators from the **insert**.

Each expression must be valid.

Expression
`MyString ← 'X' ..... MyString`
`MyChar ← ..... ("ABCD", ..... ,.....)`
`MyString ← ..... (..... (MyDOB))`
`MyInt ← ..... (..... (MyString) / 2)`

[4]

(b) Different test methods will be used at different stages of the program development.

Complete the table by identifying the test method that matches the test description.

The first row has been completed for you.

Test description	Test method
carried out as soon as a program module has been coded	alpha
carried out as program modules are being combined	
completed by the developers without referring to the code	
completed by the customer	

[3]

(c) During the alpha testing stage, an Integrated Development Environment (IDE) is used to help locate an error that has been identified. The IDE report window feature is used to examine the values assigned to variables.

Explain how **two** other IDE features are used together with the report window feature to help locate the error.

.....

.....

.....

.....

.....

..... [3]

1 **(a)** The table contains pseudocode examples.

Each example may contain statements that relate to one or more of:

- selection
- iteration (repetition)
- subroutines (procedures or functions).

Complete the table by placing **one or more** ticks ('✓') in each row.

Pseudocode example	Selection	Iteration	Subroutine

`IF Status = FALSE THEN`
`  FOR Count ← 1 TO 20`
`  CALL Reset(Count)`
`  NEXT Count`
`ENDIF`			

`OTHERWISE : Status ← TRUE`			
`WHILE AllDone() = TRUE`			
`30 : NextChar ← 'X'`			

[4]

(b) Complete the table by giving the appropriate data type.

Variable	Example data value	Data type
`Result`	`5.42`	
`MonthLetter`	`"JFMAMJJASOND"`	
`Birthday`	`15/11/2009`	

[3]

(c) Evaluate each expression in the table by using the data values shown in **(b)**.

Write 'ERROR' if the expression contains an error.

Expression	Evaluates to
`INT(Result) + 1 > 6`	
`NUM_TO_STR(LENGTH(MonthLetter))`	
`NUM_TO_STR(Result + "3.2")`	
`MID(MonthLetter, MONTH(Birthday) – 2, 1)`	

[4]

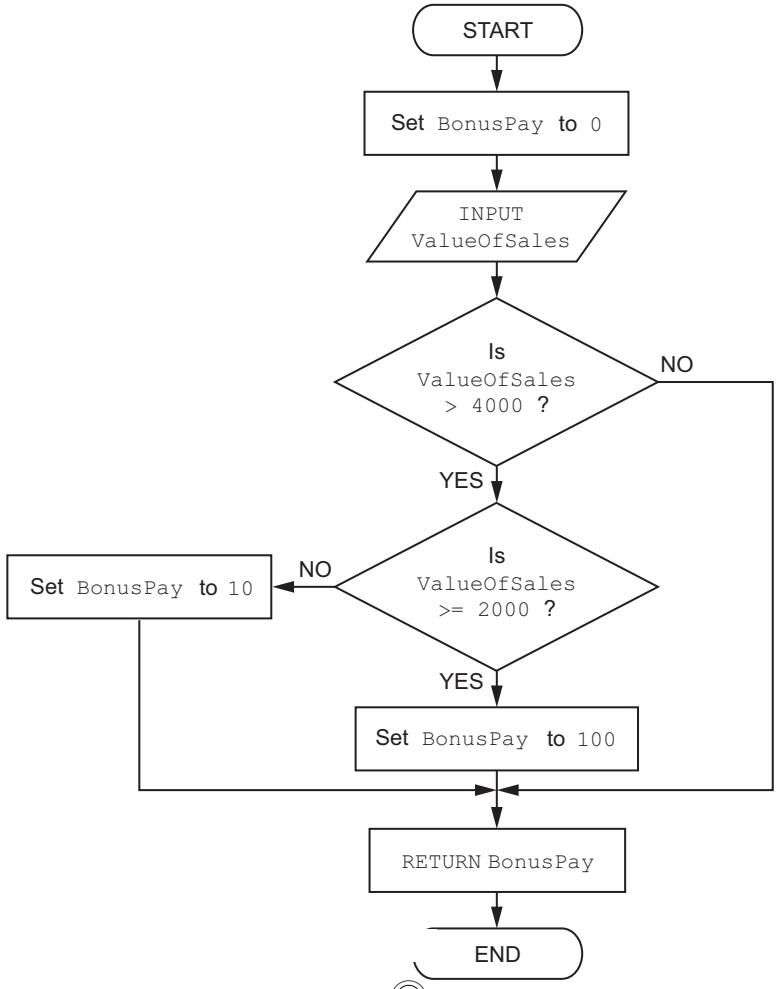
2 A program is being developed to calculate the pay of employees working for a company.

A function `CalculateBonus()` calculates bonus pay based on the value of sales.

Bonus pay is calculated as shown in the table.

Value of sales (in dollars)	Bonus pay (in dollars)
below 2000	0
between 2000 and 4000 inclusive	10
above 4000	100

A flowchart for the function `CalculateBonus()` has been designed.



(a) The flowchart contains logic errors.

One logic error is that `BonusPay` will **not** be set to `10` although the `ValueOfSales` input is between `2000` and `4000` inclusive.

(i) Explain why this error occurs.

.....

..... [1]

(ii) Explain how this error could be corrected.

.....

..... [1]

(b) There are different ways to reduce the risk of errors when developing the new program, such as the use of constants.

(i) State a value that could be replaced by a constant in the function `CalculateBonus()`

..... [1]

(ii) Explain how the use of constants helps to reduce the risk of programming errors.

.....

.....

.....

..... [2]

(iii) One other way that can reduce the risk of errors when writing the program is the use of library routines.

Explain how library routines can reduce the risk of programming errors.

.....

..... [1]

(c) All the logic errors have been corrected, and the program has been coded.

Identify and describe **two** other types of error that the program could contain.

Type of error .....

Description .....

.....

.....

Type of error .....

Description .....

.....

.....

[4]

1 A program will calculate the tax payable based on the cost of an item.

Calculations will occur at many places in the program and these involve the use of one of three tax rates.

Tax rate values represent a percentage. For example, a tax rate value of 5.23 represents 5.23%. In this case, the tax payable on an item costing $100 would be $5.23.

Tax rate values are used at several places within the program. One example is given in pseudocode as follows:

```
HighRate ← FALSE
CASE OF ItemCost
   <= 50  : TaxRate ← 3.75      // tax rate of 3.75%
   <= 200 : TaxRate ← 5.23      // tax rate of 5.23%
   > 200  : TaxRate ← 6.25      // tax rate of 6.25%
            HighRate ← TRUE
ENDCASE
TaxPayable ← ItemCost * TaxRate // tax payable
```

(a) The pseudocode contains a logical error.

Identify the error **and** suggest a correction.

Error .....

.....

Correction .....

.....

[2]

(b) During the design of the program, tax rate values have been used wherever they are needed as shown in the pseudocode example above. Tax rates do not change while the program runs.

(i) Identify a more appropriate way of representing the tax rate values in the final program.

..... [1]

(ii) Describe the benefits of your answer to part **(b)(i)** with reference to this program.

.....

.....

.....

.....

.....

.... [3]

(c) Give the **appropriate** data type for the variables in the following table, as used in the pseudocode:

Variable name	Data type
`HighRate`	
`TaxPayable`	

[2]

(d) The final `CASE` condition (`> 200`) in the pseudocode example could be replaced with a keyword.

Give the keyword.

..... [1]

1 **(a)** The following table contains pseudocode examples.

Each example may contain statements that relate to one or more of the following:

- selection
- iteration (repetition)
- subroutine (procedure or function).

Complete the table by placing **one or more** ticks ('✓') in each row.

Pseudocode example	Selection	Iteration	Subroutine

`FOR Index ⟵ 1 TO 3`
`  IF Safe[Index] = TRUE THEN`
`  Flap[Index] ⟵ 0`
`  ENDIF`
`NEXT Index`			

`CASE OF Compound(3)`			
`REPEAT UNTIL AllDone() = TRUE`			
`WHILE Result[3] <> FALSE`			

[4]

(b) Complete the table by giving the appropriate data type in each case.

Variable	Example data value	Data type
`Available`	`TRUE`	
`Received`	`"18/04/2021"`	
`Index`	`100`	

[3]

(c) Evaluate each expression in the table by using the data values shown in part **(b)**.

Write 'ERROR' if the expression contains an error.

Expression	Evaluates to
`Available AND NOT(Index > 100)`	
`Index MOD 30`	
`NUM_TO_STR(Index + "33")`	

[3]

6 In some countries, on the third Sunday in March, daylight saving time begins when clocks move forward by one hour.

A module `AdjustClock()` will take an integer parameter representing a year. The module will return an integer value representing the number of the day in March on which the clocks move forward.

For example, the following line of pseudocode would assign `DayNumber` the value 20:

```
DayNumber ← AdjustClock(2022)
```

Write pseudocode for the function `AdjustClock()`.

Date functions from the **insert** should be used in your solution.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

1 An algorithm is developed in pseudocode before being coded in a programming language.

(a) The following table shows four valid pseudocode assignment statements.

Complete the table by giving an appropriate data type to declare each of the variables `A`, `B`, `C` and `D`.

Assignment statement	Data type
`A ← LEFT(MyName, 1)`	
`B ← Total * 2`	
`C ← INT(ItemCost) / 3`	
`D ← "Odd OR Even"`	

[4]

(b) Other variables in the program have example values as shown:

Variable	Value
`Sorted`	`False`
`Tries`	`9`
`ID`	`"ZGAC001"`

Complete the table by evaluating each expression, using the example values.

Expression	Evaluates to
`Tries < 10 AND NOT Sorted`	
`Tries MOD 4`	
`TO_LOWER(MID(ID, 3, 1))`	
`LENGTH(ID & "xx") >= Tries`	

[4]

(c) The variable names `A`, `B`, `C` and `D` in part **(a)** are **not** good programming practice.

(i) State why these variable names are **not** suitable.

.....

..... [1]

(ii) Identify **one** problem that these variable names might cause.

.....

..... [1]

(iii) The choice of suitable variable names is one example of good programming practice.

Give **one other** example.

.....

..... [1]

1 **(a)** The following table contains pseudocode examples.

Each example may contain statements that relate to one or more of the following:
- selection
- iteration (repetition)
- input/output.

Complete the table by placing **one or more** ticks (✓) in each row.

Pseudocode example	Selection	Iteration	Input/Output

`FOR Index ← 1 TO 10`
`  Data[Index] ← 0`
`NEXT Index`			

`WRITEFILE ThisFile, "****"`			
`UNTIL Level > 25`			

`IF Mark > 74 THEN`
`  READFILE OldFile, Data`
`ENDIF`			

[4]

(b) Program variables have data types as follows:

Variable	Data type
`MyChar`	`CHAR`
`MyString`	`STRING`
`MyInt`	`INTEGER`

Complete the table by filling in each gap with a function (from the **insert**) so that each expression is valid.

Expression
`MyInt ← ..... (3.1415926)`
`MyChar ← ..... ("Elwood", 3, 1)`
`MyString ← ..... ( ..... (27.509))`
`MyInt ← ..... ( ..... ("ABC123", 3))`

[4]

(c) The variables given in part **(b)** are chosen during the design stage of the program development life cycle.

The choices are to be documented to simplify program maintenance.

State a suitable way of documenting the variables **and** give **one** piece of information that should be recorded, in addition to the data type.

.....

.....

..... [2]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

11.2 Constructs

Name: _____ Class: _____ Date: _____ **Total: 40 marks**

KEY WORDS selection 选择 • trace table 追踪表 • constructs 结构 • nested 嵌套

Objective

Build the skills to answer exam questions on **constructs** 结构 — selection (`IF`, `CASE`) and the three kinds of loop.

You must be able to:

- write `IF…ELSE`, **nested** `IF`, and `CASE` structures
- write a **count-controlled** (`FOR`), **pre-condition** (`WHILE`) and **post-condition** (`REPEAT`) loop
- **justify** which loop best suits a problem

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Selection: IF and CASE

```
IF Age >= 18 THEN
    OUTPUT "Adult"
ELSE
    OUTPUT "Minor"
ENDIF
```

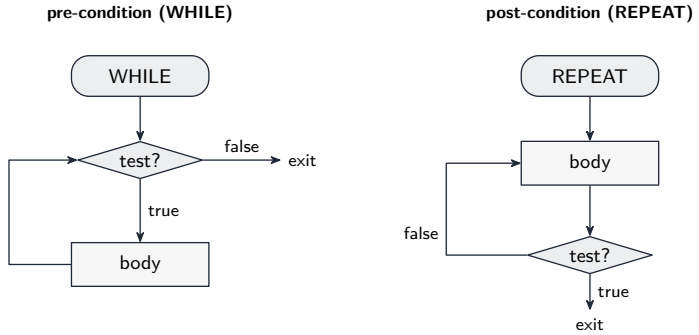
For one value against **several** options, a `CASE` is cleaner than deep nested `IF`s:

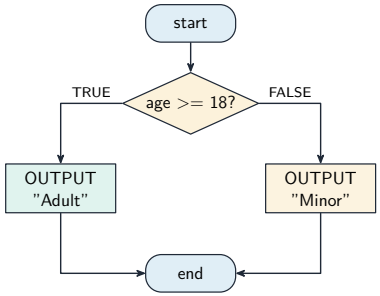
```
CASE OF Grade
    "A": OUTPUT "Excellent"
    "B": OUTPUT "Good"
    OTHERWISE: OUTPUT "Try again"
ENDCASE
```

The three loops

```
FOR i ← 1 TO 10          // count-controlled - known repeats
    OUTPUT i
NEXT i
```

`WHILE` tests **before** the body (may run **0** times); `REPEAT` tests **after** (runs **at least once**):





Selection (if-else) is the other core construct

```
WHILE Total < 100 DO        REPEAT
    INPUT n                     INPUT Password
    Total ← Total + n       UNTIL Password = "OPEN"
ENDWHILE
```


Choosing a loop

- repeats **known** in advance → `FOR`
- may need **zero** passes → `WHILE`
- needs **at least one** pass → `REPEAT`

Tracing a loop (trace table)

A **trace table** 追踪表 dry-runs code by hand: one column per variable (plus OUTPUT), one row each time a value changes. Trace this for `N = 3`:

```
Total ← 0
FOR i ← 1 TO N
    Total ← Total + i
NEXT i
OUTPUT Total
```

i	Total	OUTPUT
	0	
1	1	
2	3	
3	6	6

2 Practice

Now apply the methods above.

2.1 Write an `IF…ELSE` that outputs `"Pass"` when `Mark >= 50`, otherwise `"Fail"`. [2]

2.2 Rewrite this nested `IF` as a `CASE` structure. [3]

```
IF Grade = "A" THEN
    OUTPUT "Excellent"
ELSE
    IF Grade = "B" THEN
        OUTPUT "Good"
    ELSE
        OUTPUT "Try again"
    ENDIF
ENDIF
```

2.3 State which loop may run **zero** times and which runs **at least once**. [2]

2.4 Write a `FOR` loop that outputs the **even** numbers from 2 to 20. [2]

3 Exam-style questions

3.1 A menu uses a number `Choice`. Write a `CASE` structure: `1`→`"New"`, `2`→`"Open"`, `3`→`"Save"`, otherwise→`"Invalid"`. [4]

3.2 A program must keep asking for a password until the user types `"OPEN"`. Write the loop using the **most suitable** construct, and **justify** your choice. [3]

3.3 Write a **pre-condition** loop that reads numbers and adds them to a `Total`, stopping once `Total` reaches **100**. [3]

3.4 Complete the **trace table** for this algorithm when the input `X` is `5`. [4]

```
A ← 1
B ← 1
WHILE A < X DO
    B ← B * 2
    A ← A + 1
ENDWHILE
OUTPUT B
```

A	B	OUTPUT
1	1	

3.5 A program keeps the time of day in three global integer variables `HH`, `MM` and `SS`.

(a) A procedure `Tick()` is called once every second and must advance the time correctly, rolling over at 60 seconds, 60 minutes and 24 hours. **Write** pseudocode for `Tick()`. [6]

(b) **Complete** the trace table by dry running your procedure from the starting time shown, for **three** calls. [3]

call	HH	MM	SS
start	10	59	58
1			
2			
3			

(c) **Explain** why the three `IF` statements must be **nested** or tested in the right order, giving what would go wrong otherwise. [3]

(d) The variables are **global**. **Explain** one drawback of that, and **state** how you would restructure `Tick()` to avoid it. [3]

(e) **State** which loop construct you would use to run the clock until the user stops it, and **justify** your choice. [2]

Solutions

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

2.1

- `IF … ELSE … ENDIF` with the pass/fail condition

```
IF Mark >= 50 THEN
    OUTPUT "Pass"
ELSE
    OUTPUT "Fail"
ENDIF
```

2.2

- `CASE OF … ENDCASE` with two value branches and `OTHERWISE`

```
CASE OF Grade
    "A": OUTPUT "Excellent"
    "B": OUTPUT "Good"
    OTHERWISE: OUTPUT "Try again"
ENDCASE
```

2.3

- `WHILE` (pre-condition) may run **zero** times
- `REPEAT … UNTIL` (post-condition) runs **at least once**

2.4

- `FOR … STEP 2` from 2 to 20

```
FOR i ← 2 TO 20 STEP 2
    OUTPUT i
NEXT i
```

- *(outputting `i` only when `i MOD 2 = 0` also scores)*

3.1

- `CASE OF … ENDCASE` with three branches, the right outputs, and `OTHERWISE`

```
CASE OF Choice
    1: OUTPUT "New"
    2: OUTPUT "Open"
    3: OUTPUT "Save"
    OTHERWISE: OUTPUT "Invalid"
ENDCASE
```

3.2

- a **post-condition (`REPEAT … UNTIL`)** loop, because the password must be entered **at least once** before it can be tested


```
REPEAT
    INPUT Password
UNTIL Password = "OPEN"
```

3.3

- start `Total` at 0; loop while it is under 100; add each input

```
Total ← 0
WHILE Total < 100 DO
    INPUT n
    Total ← Total + n
ENDWHILE
```

3.4

- `B` doubles on each pass until `A` reaches 5

A	B	OUTPUT
1	1	
2	2	
3	4	
4	8	
5	16	16

- stops when $A = 5$; OUTPUT **16**

3.5

(a)

```
PROCEDURE Tick()
    SS ← SS + 1
    IF SS = 60
      THEN
        SS ← 0
        MM ← MM + 1
        IF MM = 60
          THEN
            MM ← 0
            HH ← HH + 1
            IF HH = 24 THEN HH ← 0 ENDIF
        ENDIF
    ENDIF
ENDPROCEDURE
```

- increment seconds; reset seconds and carry into minutes; reset minutes and carry into hours; hours wrap at 24; the carries nested, not independent; correct pseudocode syntax

(b) call 1: `10 59 59`

- call 2: `11 00 00`
- call 3: `11 00 01`

(c) the minutes can only roll over **because** the seconds just did, and the hours only because the minutes did

- if the three were tested independently, `MM` would be checked against 60 before it had been incremented on this tick, so the rollover would be a second late
- testing hours first would let the clock show `11:59:60` for one tick

(d) any global variable can be changed by **any** part of the program, so a fault anywhere can corrupt the time and be hard to trace

- pass the three values in as parameters and return the updated time — or hold them in one record passed by reference — so only `Tick()` can change them

(e) a **post-condition** loop, `REPEAT … UNTIL Stopped`, because the clock must tick at least once before the user has any chance to stop it

4 Study the algorithm:

```
DECLARE Chars : ARRAY[1:4] OF CHAR
DECLARE I, J : INTEGER
DECLARE Key : CHAR
I ← 2
Chars[1] ← 'D'
Chars[2] ← 'T'
Chars[3] ← 'H'
Chars[4] ← 'R'
WHILE I <= 4                              //Outer loop
   Key ← Chars[I]
   J ← I - 1
   WHILE J >= 0 AND Chars[J] > Key        //Inner loop
      Chars[J + 1] ← Chars[J]
      J ← J - 1
   ENDWHILE
   Chars[J + 1] ← Key
   I ← I + 1
ENDWHILE
```

(a) The outer loop structure used in the algorithm is **not** the most appropriate one to use.

State the type of loop structure that would be the most appropriate to use and justify why it is the most appropriate.

Loop structure .....

Justification .....

.....

[2]

(b) Complete the trace table by dry running the algorithm.

The first row has been completed.

I	Key	J	Chars[J]	Chars			
				[1]	[2]	[3]	[4]
2				'D'	'T'	'H'	'R'

[6]

2 A program uses three global integer variables `HH`, `MM` and `SS` to represent the current time in hours, minutes and seconds using the 24-hour clock notation.

Midnight would be represented as 00:00:00 (`HH:MM:SS`). If the variables `HH`, `MM` and `SS` contained the values 16, 30 and 10 respectively, then the time would be 16:30:10 or just after 4.30 in the afternoon.

A procedure `Tick()` will be called every second.
The procedure `Tick()` will:

- update the value in `SS` each time it is called
- update the values in `HH` and `MM` as appropriate
- call a procedure `CheckAlarm()` at the start of each minute
- call a procedure `NewDay()` whenever the time reaches midnight.

Complete the pseudocode for procedure `Tick()`.

```
PROCEDURE Tick()
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[6]

5 A program contains a global 1D array `Data` with 100 elements of type `INTEGER`.

The program contains a function `Process()` expressed in pseudocode as follows:

```
FUNCTION Process(Number : INTEGER, Label : STRING) RETURNS STRING
   DECLARE Index, Count : INTEGER
   DECLARE ReturnValue : STRING

   Count ← INT(100 / Number)
   Index ← Data[Number]

   CASE OF (Index MOD 2)
      0 : ReturnValue ← TO_UPPER(RIGHT(Label, Count))
      1 : ReturnValue ← "****"
   ENDCASE

   RETURN RetVal
ENDFUNCTION
```

(a) Run-time errors can be generated in different ways. For example, a run-time error will be generated if a function is called with invalid parameters.

The pseudocode contains three statements that could generate a run-time error.

Write the **three** statements **and** explain how each could generate a run-time error.

Statement 1 .....

Explanation .....

.....

Statement 2 .....

Explanation .....

.....

Statement 3 .....

Explanation .....

.....

[3]

(b) One type of run-time error can cause a program to stop responding ('freezing').

Identify a particular type of programming construct that can generate this type of error **and** explain why it occurs.

Construct .....

Explanation .....

.....

.....

[2]

(c) The function `Process()` contains a selection construct using a `CASE` structure.

Write pseudocode using a single selection construct with the same functionality **without** using a `CASE` structure.

.....

.....

.....

.....

..... [2]

2 An algorithm will:

1. prompt and input a sequence of 100 integer values, one at a time
2. sum the positive integers
3. output the result of the sum.

(a) Write pseudocode for the algorithm.

Assume the value zero is neither positive nor negative.

You must declare all variables used in the algorithm.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(b) The algorithm requires the use of basic constructs. One of these is sequence.

Identify **one other** basic construct required by the algorithm **and** describe how it is used.

Construct .....

Use .....

.....

[2]

4 An examination paper has a maximum of 75 marks. One of five pass grades (A to E) is assigned, depending on the mark obtained. The lowest mark for a given grade is known as the grade boundary.

A program is being written to process examination marks.

The five grade boundaries are stored in a global 1D array `GB` of type `INTEGER`, for example:

Index	Value	Comment
1	65	The minimum mark for an A grade.
2	57	The minimum mark for a B grade.
3	43	The minimum mark for a C grade.
4	35	The minimum mark for a D grade.
5	27	The minimum mark for an E grade.

Any paper that achieves a mark within 2 marks of a grade boundary must be checked. Using the given table, a paper with 45 marks would need to be checked.

(a) The pseudocode algorithm to determine whether a paper should be checked is as shown. The mark for the paper is stored in variable `Mark`. Global variables `Mark`, `Index`, `Upper` and `Lower` are declared as integers.

Complete the pseudocode.

```
FOR Index ← 1 TO .....
   Lower ← GB[Index] - 2
   Upper ← .....
   IF Mark ..... AND Mark ..... THEN
      OUTPUT "Check this paper"
   ENDIF
NEXT Index
```

[4]

(b) An alternative algorithm to determine if a paper needs to be checked uses a global 1D array `Check`, containing 76 elements of type `BOOLEAN`. The indices of the array are from 0 to 75 (inclusive), corresponding to the range of possible marks.

An element value in `Check` is `TRUE` if the index is within 2 marks of a grade boundary. For example, in the case where the C grade boundary is 43 the corresponding part of the `Check` array would be as follows:

Index	Value	
40	FALSE	
41	TRUE	
42	TRUE	
43	TRUE	← The grade boundary for a C grade
44	TRUE	
45	TRUE	
46	FALSE	

(i) The mark for a given paper is stored in variable `Mark`.

Describe how an algorithm would use the `Check` array to determine whether this paper should be checked.

.....

.....

.....

..... [2]

(ii) A procedure `GBInitialise()` will initialise the `Check` array using values from the `GB` array.

Note it can be assumed that the maximum grade boundary value for A is 70 and the minimum value for E is 15.

Write pseudocode for the procedure.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[6]

5 A global 1D array of strings contains three elements which are assigned values as shown:

```
Data[1] ← "aaaaaa"
Data[2] ← "bbbbbb"
Data[3] ← "cccccc"
```

Procedure `Process()` manipulates the values in the array.

The procedure is written in pseudocode as follows:

```
PROCEDURE Process(Format : STRING)
    DECLARE Count, Index, L : INTEGER
    DECLARE Result : STRING
    DECLARE C : CHAR

    Result ← "****"

    FOR Count ← 1 TO LENGTH(Format) STEP 2
        C ← MID(Format, Count, 1)
        L ← STR_TO_NUM(MID(Format, Count + 1, 1))

        Index ← (Count + 1) DIV 2

        CASE OF C
            'X' : Result ← TO_UPPER(Data[Index])
            'Y' : Result ← TO_LOWER(Data[Index])
            'Z' : Result ← "**" & Data[Index]
        ENDCASE

        Data[Index] ← LEFT(Result, L)
    NEXT Count

ENDPROCEDURE
```

(a) Complete the trace table by dry running the procedure when it is called as follows:

```
CALL Process("X3Y2W4")
```

Count	C	L	Index	Result	Data[1]	Data[2]	Data[3]

[6]

(b) The procedure is to be modified. If variable `C` is assigned a value other than `'X'`, `'Y'` or `'Z'`, then procedure `Error()` is called and passed the value of variable `C` as a parameter.

This modification can be implemented by adding a **single line** of pseudocode.

(i) Write the single line of pseudocode.

..... [1]

(ii) State where this new line should be placed.

..... [1]

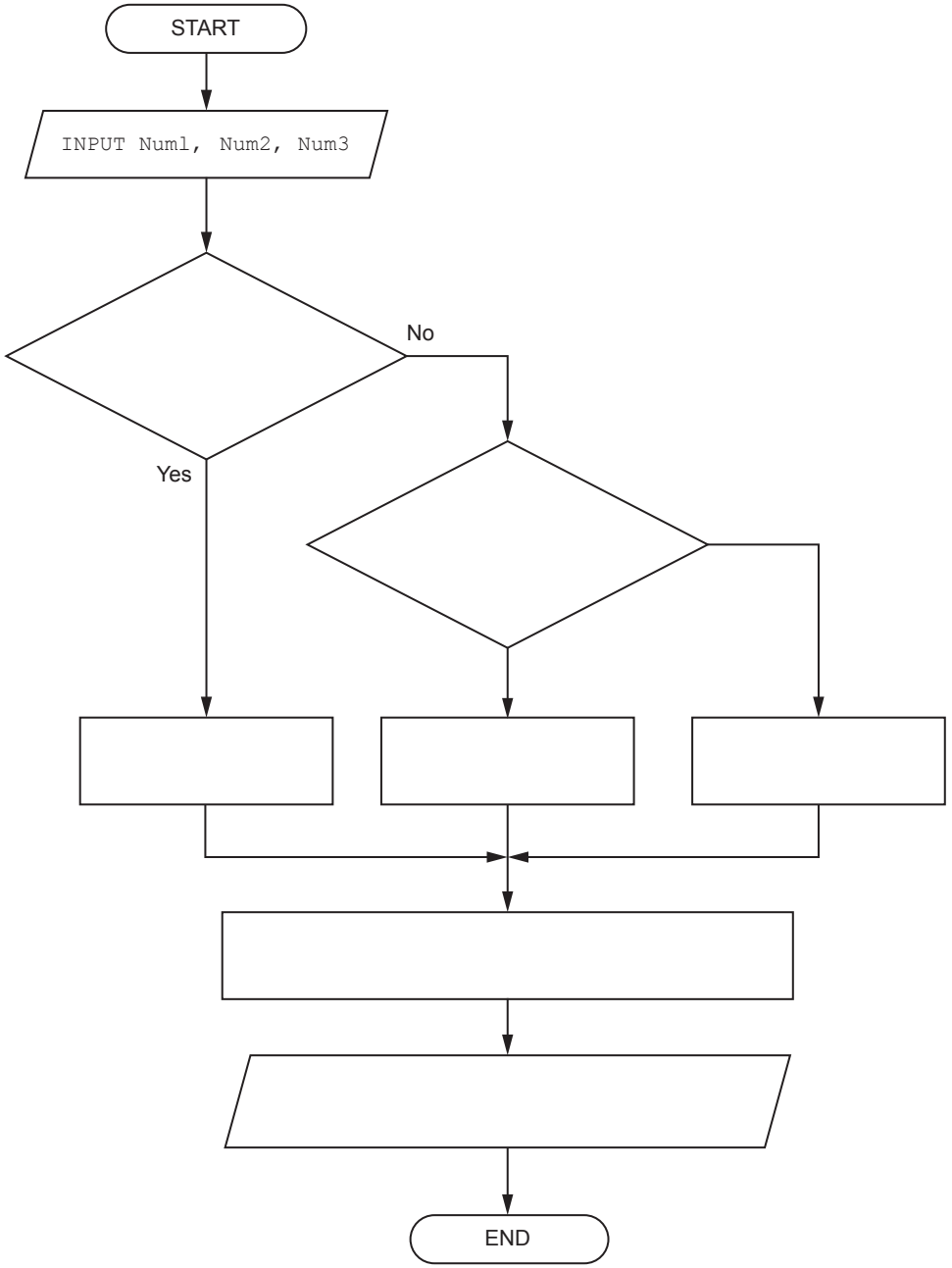
2 A program is being developed.

(a) An algorithm for part of the program will:

- input three numeric values and assign them to identifiers `Num1`, `Num2` and `Num3`
- assign the largest value to variable `Ans`
- output a message giving the largest value and the average of the three numeric values.

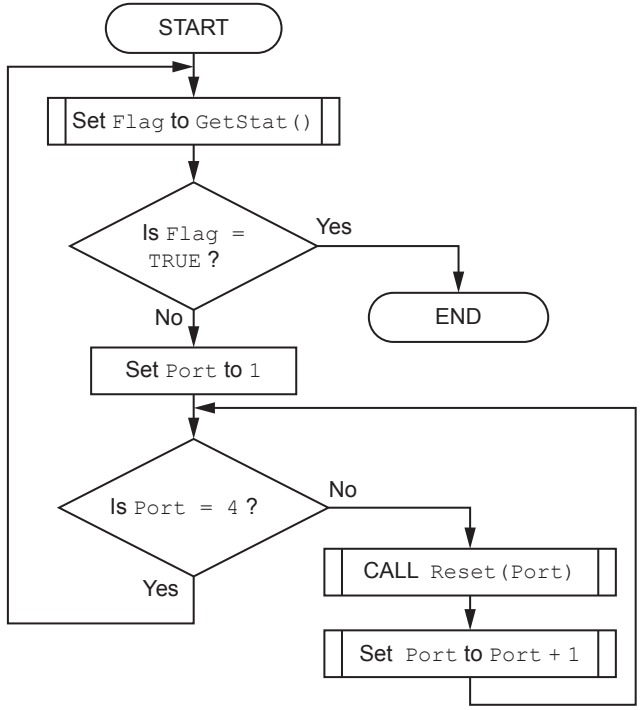
Assume the values are all different and are input in no particular order.

Complete the program flowchart on page 5 to represent the algorithm.



[5]

(b) A different part of the program contains an algorithm represented by the following program flowchart:



Write pseudocode for the algorithm.

.....

.....

.....

.....

.....

.....

.....

.....

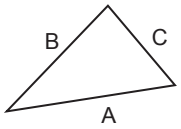
.....

.....

.....

[5]

4 A triangle has sides of length A, B and C.



In this example, A is the length of the longest side.

This triangle is said to be right-angled if the following equation is true:

$A \times A = (B \times B) + (C \times C)$

A procedure will be written to check whether three lengths represent a right-angled triangle. The lengths will be input in any sequence.

The procedure `IsRA()` will:

- prompt and input three integer values representing the three lengths
- test whether the three lengths correspond to the sides of a right-angled triangle
- output a suitable message.

The length of the longest side may **not** be the first value input.

Write pseudocode for the procedure `IsRA()`.

[5]

5 A program is being designed in pseudocode.

The program contains a global 1D array `Data` of type string containing 200 elements.

The first element has the index value 1.

A procedure `Process()` is written to initialise the values in the array:

```
PROCEDURE Process(Label : STRING)
   DECLARE Index : INTEGER
   Index ← 0
   INPUT Data[Index]
   WHILE Index < 200
      Index ← Index + 1
      CASE OF (Index MOD 2)
         0 : Data[Index] ← TO_UPPER(Label)
         1 : Data[Index] ← TO_LOWER(Label)
         OTHERWISE : OUTPUT "Alarm 1201"
      ENDCASE
   NEXT Index
   OUTPUT "Completed " & Index & " times"
ENDPROCEDURE
```

(a) (i) The pseudocode contains **two** syntax errors and **one** other error.

Identify the errors.

Syntax error 1 .....

.....

Syntax error 2 .....

.....

Other error .....

.....

[3]

(ii) The procedure contains a statement that is **not** needed.

Identify the pseudocode statement **and** explain why it is **not** needed.

Statement .....

Explanation .....

.....

[2]

(b) After correcting all syntax errors, the pseudocode is translated into program code which compiles without generating any errors.

When the program is executed it unexpectedly stops responding.

Identify the type of error that has occurred.

..... [1]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

11.3 Structured Programming

Name: _____ Class: _____ Date: _____ **Total: 17 marks**

KEY WORDS procedure 过程 • parameters 参数 • pass by reference 传引用 • pass by value 传值 • function 函数

Objective

Build the skills to answer exam questions on **structured programming** 结构化编程 — procedures, functions and parameters.

You must be able to:

- define and **call** a **procedure** and a **function**
- use **parameters**, including **by value** and **by reference**
- use the terms argument, parameter, return value, **local/global**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Procedure vs function

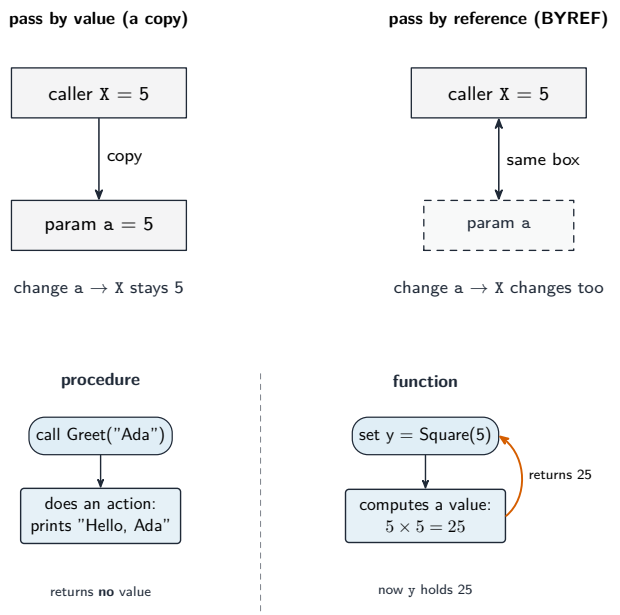
A **procedure** does an action and returns **nothing**; a **function** returns a **value** used in an expression.

```
PROCEDURE Greet(Name : STRING)
    OUTPUT "Hello, ", Name
ENDPROCEDURE
CALL Greet("Ada")

FUNCTION Square(x : INTEGER) RETURNS INTEGER
    RETURN x * x
ENDFUNCTION
Result ← Square(5) + 1      // Result = 26
```

■ Parameters: by value or by reference

A **parameter** is the variable that **receives** input; the value the caller supplies is the **argument**.



Procedures and functions structure a program

`Swap` must use **`BYREF`** so the new values stay in the caller:

```
PROCEDURE Swap(BYREF a : INTEGER, BYREF b : INTEGER)
    DECLARE temp : INTEGER
    temp ← a
    a ← b
    b ← temp
ENDPROCEDURE
```

■ Local, global, and when to use a subroutine

A **local** variable lives only inside its subroutine; a **global** is visible everywhere (prefer locals). Use a subroutine when logic **repeats**, has a **clear purpose**, or to **test** a part on its own.

2 Practice

Now apply the methods above.

2.1 State **one** difference between a **procedure** and a **function**. [1]

2.2 Define a procedure `Greet(Name)` that outputs `"Hello, "` followed by the name. [2]

2.3 Define a function `Cube(x)` that returns `x` cubed. [2]

2.4 State the difference between an **argument** and a **parameter**. [2]

3 Exam-style questions

3.1 Write a **function** `MaxOf(a, b)` that returns the larger of two integers. [3]

3.2 Explain **pass by value** and **pass by reference**, and state which one a `Swap` procedure must use, and why. [4]

3.3 State **two** benefits of using subroutines in a large program. [2]

3.4 State what is meant by a **local variable**. [1]

Solutions

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

2.1

- a **function returns a value** (used in an expression); a **procedure does not** return a value (it performs an action)

2.2

- `PROCEDURE … ENDPROCEDURE` with a parameter

```
PROCEDURE Greet(Name : STRING)
    OUTPUT "Hello, ", Name
ENDPROCEDURE
```

2.3

- `FUNCTION … RETURNS … ENDFUNCTION` with `RETURN`

```
FUNCTION Cube(x : INTEGER) RETURNS INTEGER
    RETURN x * x * x
ENDFUNCTION
```

2.4

- an **argument** is the value the caller passes in
- a **parameter** is the variable inside the subroutine that **receives** it

3.1

- `FUNCTION … RETURNS INTEGER`; compare; `RETURN` the larger

```
FUNCTION MaxOf(a : INTEGER, b : INTEGER) RETURNS INTEGER
    IF a > b THEN
        RETURN a
    ELSE
        RETURN b
    ENDIF
ENDFUNCTION
```

3.2

- **by value** — the routine gets a **copy**, so changes inside it do **not** affect the caller
- **by reference** — the routine shares the caller's actual variable, so changes **do** affect it
- `Swap` must use **pass by reference (`BYREF`)**
- so the swapped values remain in the caller's variables after the call

3.3

- any two: avoids repeating code; easier to read/maintain; can be tested on its own; supports decomposition/teamwork

3.4

- a variable declared **inside** a subroutine that exists **only while that subroutine runs** (and is not visible outside it)

4 A program contains a global 1D array `Number` consisting of 20 elements of type `REAL`

A procedure `Store()` will input a sequence of up to 20 real values, one value at a time. These values will be assigned to elements of the array using four steps:

Step 1: store the first value in the sequence in the first element of the array
Step 2: check each subsequent value input. If this value is larger than the previous value input, then assign the value to the next array element, otherwise go to **step 4**
Step 3: repeat from **step 2** unless the array is full
Step 4: output the count of the number of values stored in the array together with a suitable message.

(a) Complete the pseudocode for `Store()`

All variables used in the algorithm must be declared.

```
PROCEDURE Store()
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
ENDPROCEDURE
```

[6]

(b) The requirements of the program change:

- the number of values in the sequence is unknown, but may be higher than 20
- the values will need to be accessed by another program as data.

The data will be stored in a text file instead of an array.

(i) Give **two** benefits of using a text file instead of an array.

1 .....

.....

2 .....

.....

[2]

(ii) State any change that will need to be made before each value is written to the file.

.....

..... [1]

6 A string function `Compare()` will compare two strings.

The function will:
- take **four** parameters:
  - two strings, `String1` and `String2`
  - a character, `Position`, to indicate whether `String1` will be compared with the start (`'s'`), or the end (`'e'`) of `String2`
  - a Boolean, `CaseMatters`, to indicate whether an upper case character and lower case character (for example, `'A'` and `'a'`) are regarded as different (`TRUE`), or as the same (`FALSE`)
- return `FALSE` if `String2` has fewer characters than `String1`
- return `TRUE` if the comparison is true, otherwise return `FALSE`

For example:

Parameter				Return value
String1	**String2**	**CaseMatters**	**Position**	
"Cat"	"Catalogue"	TRUE	's'	TRUE
"CAT"	"Catalogue"	TRUE	's'	FALSE
"CAT"	"Catalogue"	FALSE	's'	TRUE
"Cat"	"Catalogue"	TRUE	'e'	FALSE
"GUE"	"Catalogue"	FALSE	'e'	TRUE
"Catalogue"	"Cat"	TRUE	's'	FALSE

Write pseudocode for the function `Compare()`

[7]

8 A program is being developed to manage student book loans from a college library.

The programmer has defined a record type to define each loan.

The data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

The programmer has defined a global array `Loan` to store 7000 loan records.

There are more elements in the array than books in the library. Unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

The programmer has defined a program module:

Module	Description
`CountLoans()`	• called with a parameter of type `STRING` representing a `StudentID` • counts the number of books currently on loan to the specified student • counts the number of books that the student has already returned • output both counts together with a suitable message

(a) Write pseudocode for module `CountLoans()`

[7]

(b) As a reminder, a global array `Loan` stores 7000 loan records with data items for each loan record:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

A new module is defined:

Module	Description
`NewLoan()`	• called with two parameters of type `STRING` representing a `StudentID` and a `BookID` • searches the array for an unused loan record • if found, updates the loan record and returns `TRUE`, otherwise returns `FALSE`

Write efficient pseudocode for `NewLoan()`

[7]

(c) When a book is returned, the loan record will have its `OnLoan` data set to `FALSE`

A new procedure `Archive()` will mark these records as unused after first saving them for future reference.

Explain how these records can be saved for future reference.

[2]

(d) It is decided to extend the program so that a new module `Reminder()` will send an email to the student three days before the book is due to be returned.

Outline the changes that will need to be made to the data stored and how this data will be used to generate the reminder email.

Data

Use

[2]

7 A program is being developed to implement a customer loyalty scheme for a coffee shop.

The programmer has decided that the following data items need to be stored for each customer:

Data item	Description
customer ID	a unique six-digit string
points	an integer value that is increased by one for every cup of coffee the customer orders

When a customer visits the shop and orders coffee, the scheme operates as follows:

- The total number of points is increased by the number of coffees ordered.
- If just one cup of coffee is ordered and the number of points goes above 10, then:
  - the cup of coffee they have just ordered is given to them free of charge
  - the number of points is reduced by 11.
- If the order is for multiple coffees and the number of points goes above 10, then:
  - they get one coffee free of charge for every 11 points
  - the number of points is reduced by 11 for each free coffee.

For example, the:

- customer currently has 9 points
- customer orders 16 cups of coffee
- total number of points now becomes 25
- customer gets 2 free coffees and now has 3 points left.

The programmer has defined a program module that is called every time a customer places an order:

Module	Description

`CustomerOrder()`	• called with two integer parameters:
	  ◦ the number of coffees ordered
	  ◦ the current number of points
	• output a suitable message giving the number of free coffees
	• return a value for the new points total.

(a) Write pseudocode for module `CustomerOrder()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

(b) A text file `Loyalty.txt` will be used to store the data items for the loyalty scheme. The data items for each customer will be stored on a separate line of the text file where each data item is separated by a comma:

`<CustomerID>,<Points>`

The contents of the text file `Loyalty.txt` will always be stored in ascending order by customer ID.

When the data items are read from or written to the text file `Loyalty.txt`, they may need to be converted to the appropriate data type.

Each customer has a unique customer ID starting at `"100001"` with this value increasing by one each time a new customer joins the loyalty scheme.

When a customer joins the loyalty scheme, they are assigned the next customer ID and value of points is set to 0.

For example, if the loyalty scheme has 204 customers and a new customer joins the loyalty scheme, the following line is added to the text file `Loyalty.txt`:

`"100205,0"`

You can assume that the number of customers in the loyalty scheme will never be more than 9000.

The programmer has defined a program module as follows:

Module	Description
`AddNewCustomers()`	• called with an integer parameter representing the number of new customers to be added to the loyalty scheme • adds a new line, containing the required information, to the text file `Loyalty.txt` for each customer added to the loyalty scheme • outputs each new customer ID added to the loyalty scheme

(i) Write pseudocode for module `AddNewCustomers()`

Assume that there is at least **one** customer already in the loyalty scheme.

[8]

(ii) The requirements for `AddNewCustomers()` are changed. There may be no existing customers in the loyalty scheme.

Explain the changes that will need to be made to the module `AddNewCustomers()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

3 A programmer has been asked to create a module `RollDice()` to simulate multiple rolls of a dice. This module will be used as part of a program for a game.

The module will:

step 1 – take a positive integer parameter representing the number of times the dice will be rolled

step 2 – simulate **one** roll of the dice by generating a random integer between 1 and 6 inclusive

step 3 – output the integer generated

step 4 – repeat, as required from **step 2**

step 5 – calculate the average value of the random integers generated

step 6 – return the average value.

Write pseudocode for the module `RollDice()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

5 A module `Parity()` takes a string as a parameter. The parameter has the identifier `BitString` and it represents a binary value.

The module will concatenate a single character to the end of `BitString` by applying one of the two rules:

- `'0'` if `Bitstring` contains an even number of 1s
- `'1'` if `Bitstring` contains an odd number of 1s.

The modified value of `BitString` is then returned.

For example:

Parameter	String returned	Explanation
`"0010010110"`	`"00100101100"`	there are an even number of 1s in the string passed to `Parity()` so a `'0'` is concatenated to the end of `BitString`
`"101010"`	`"1010101"`	there are an odd number of 1s in the string passed to `Parity()` so a `'1'` is concatenated to the end of `BitString`

Write pseudocode for the module `Parity()`

Assume the parameter `BitString` can only contain the characters `'0'` and `'1'`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

3 An algorithm will output the **last** three lines from a text file `Result.txt`

The lines need to be output in the same order as they appear in the file.

Assume:

- Three variables `LineX`, `LineY` and `LineZ` will store the three lines. These are of type string and all three variables have been initialised to an empty string.
- The file exists and contains **at least** three lines.

(a) The algorithm to output the lines is expressed in eight steps.

Complete the steps.

1. Open the file .....
2. Loop until .....
3. ..... and store in `ThisLine`
4. Assign `LineY` to `LineX`
5. Assign `LineZ` to `LineY`
6. Assign `ThisLine` to `LineZ`
7. After the loop, ...
8. Output `LineX`, `LineY`, `LineZ`

[4]

(b) Explain the purpose of steps 4, 5 and 6 in the algorithm from part **(a)**.

.....

.....

..... [1]

(c) The requirement changes, and the algorithm will now output three lines from the file, starting from a **given** line number.

The modified algorithm will be implemented as a function which will:

- be called with an integer parameter representing the given line number
- output three lines, starting at the given line
- return `TRUE` if the 3 lines are output, or `FALSE` if it was not possible to output the 3 lines.

Describe the changes that need to be made to **steps 2 to 8** of the algorithm given in part **(a)**.

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

4 A global integer variable `Tick` is always incremented every millisecond (1000 times per second) regardless of the other programs running.

The value of `Tick` can be read by any program but the value should not be changed.

Assume that the value of `Tick` does not overflow.

As an example, the following pseudocode algorithm would output `"Goodbye"` 40 seconds after outputting `"Hello"`.

```
DECLARE Start : INTEGER

OUTPUT "Hello"
Start ← Tick

REPEAT
   //do nothing
UNTIL Tick = Start + 40000

OUTPUT "Goodbye"
```

A program is needed to help a user to time an event such as boiling an egg.

The time taken for the event is known as the elapsed time.

The program contains a procedure `Timer()` which will:

- take two integer values representing an elapsed time in minutes and seconds
- use the value of variable `Tick` to calculate the elapsed time
- output a warning message 30 seconds before the elapsed time is up
- output a final message when the total time has elapsed.

For example, to set an alarm for 5 minutes and 45 seconds the program makes the following call:

```
CALL Timer(5, 45)
```

When 5 minutes and 15 seconds have elapsed, the program will output:

```
"30 seconds to go"
```

When 5 minutes and 45 seconds have elapsed, the program will output:

```
"The time is up!"
```


Write pseudocode for the procedure `Timer()`.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [6]

6 A shop sells sandwiches and snacks. The owner chooses a 'daily special' sandwich which is displayed on a board outside the shop. Each 'daily special' has two different fillings and is made with one type of bread.

The owner wants a program to randomly choose the 'daily special' sandwich.

The program designer decides to store the possible sandwich fillings in a 1D array of type string.

The array is declared in pseudocode as follows:

```
DECLARE Filling : ARRAY [1:35] OF STRING
```

Each element contains the name of one filling.

An example of the first five elements is as follows:

Index	Element value
1	"Cheese"
2	"Onion"
3	"Salmon"
4	"Anchovies"
5	"Peanut Butter"

A second 1D array stores the possible bread used:

```
DECLARE Bread : ARRAY [1:10] OF STRING
```

Each element contains the name of one type of bread.

An example of the first three elements is as follows:

Index	Element value
1	"White"
2	"Brown"
3	"Pitta"

Both arrays may contain unused elements. The value of these will be an empty string and they may occur anywhere in each array.

A procedure `Special()` will output a message giving the 'daily special' sandwich made from **two** randomly selected different fillings and **one** randomly selected bread.

Unused array elements must **not** be used when creating the 'daily special' sandwich.

Using the above examples, the output could be:

```
"The daily special is Cheese a    nion on Brown bread."
```


(a) Complete the pseudocode for the procedure `Special()`.

Assume that both arrays are global.

```
PROCEDURE Special()
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[7]

(b) The owner decides that some combinations of fillings do not go well together. For example, anchovies and peanut butter.

Describe how the design could be changed to prevent certain combinations being selected.

.....

.....

.....

.....

.....

.....

.....

..... [2]

8 A program is being developed to implement a game for up to six players.

During the game, each player assembles a team of characters. At the start of the game there are 45 characters available.

Each character has four attributes, as follows:

Attribute	Examples	Comment
Player	0, 1, 3	The player the character is assigned to.
Role	Builder, Teacher, Doctor	The job that the character will perform in the game.
Name	Bill, Lee, Farah, Mo	The name of the character. Several characters may perform the same role, but they will each have a unique name.
Skill level	14, 23, 76	An integer in the range 0 to 100, inclusive.

The programmer has defined a record type to define each character. The record type definition is shown in pseudocode as follows:

```
TYPE CharacterType
   DECLARE Player : INTEGER
   DECLARE Role : STRING
   DECLARE Name : STRING
   DECLARE SkillLevel : INTEGER
ENDTYPE
```

The `Player` field indicates the player to which the character is assigned (1 to 6). This field value is 0 if the character is **not** assigned to any player.

The programmer has defined a global array to store the character data, as follows:

```
DECLARE Character : ARRAY[1:45] OF CharacterType
```

At the start of the game all record fields are initialised, and all `Player` fields are set to 0

The programmer has defined a program module as follows:

Module	Description

`Count()`	• called with two parameters:
	  ○ an integer representing a player
	  ○ a string representing a character role
	• searches the `Character` array for characters with the given role that are assigned to the given player
	• counts the number of assigned characters and sums their total skill level
	• outputs the result of the search if characters with the given role are found, for example:
	`"Player 3 has 4 characters with the role of Teacher and the total skill level is 65"`
	• if no characters with the gi le are found, outputs:
	`"No characters with that role are assigned to this player"`

(a) Complete the pseudocode for module `Count()`.

```
PROCEDURE Count(ThisPlayer : INTEGER, ThisRole : STRING)
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```


[7]

(b) The `Character` array data has been saved in the text file `SaveFile.txt`
Each line of the file contains one element of the array (one record).

New modules are defined:

Module	Description

`Extract()` (already written)	• called with two parameters:
	  ○ a string representing a complete line from the text file
	  ○ an integer representing a field number (see structure below)
	• returns a string representing the required field

`Restore()`	• opens the text file `SaveFile.txt` • reads lines from the file and assigns values to each record in the `Character` array using data from each line of the file

As a reminder, the record structure is repeated here:

```
TYPE CharacterType
   DECLARE Player : INTEGER       //Field number 1
   DECLARE Role : STRING          //Field number 2
   DECLARE Name : STRING          //Field number 3
   DECLARE SkillLevel : INTEGER   //Field number 4
ENDTYPE
```

Write pseudocode for module `Restore()`.

You must use the module `Extract()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

(c) The game can last for several days and users often find that they have to close and rerun the game program many times in order to complete it.

Describe the benefit of using the file `SaveFile.txt` as described in part **(b)**.

.....

.....

.....

..... [2]

7 A fitness club has a computerised membership system. The fitness club offers a number of different exercise classes.

The following information is stored for each club member: name, home address, email address, mobile phone number, date of birth and the exercise(s) they are interested in.

(a) When an exercise class is planned, a new module will send personalised text messages to each member who has expressed an interest in that exercise. Members wishing to join the class send a text message back. Members may decide **not** to receive future text messages by replying with the message 'STOP'.

The process of abstraction is used to filter out unnecessary information.

(i) State **one** advantage of applying abstraction to this problem.

.....

..... [1]

(ii) Identify **three** items of information that will be required by the new module.
Justify your choices with reference to the given scenario.

Item 1 required .....

Justification .....

.....

Item 2 required .....

Justification .....

.....

Item 3 required .....

Justification .....

.....
[3]

(iii) Identify **two** operations that would be required to process data when the new module receives a text message back from a member.

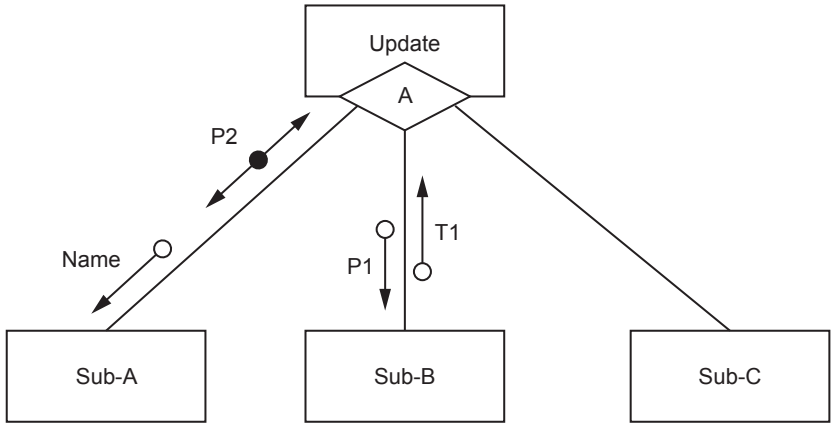
Operation 1 .....

.....

Operation 2 .....

.....
[2]

(b) The structure chart illustrates part of the membership program:



Data item notes:

- Name contains the name of a club member
- P1 and T1 are of type real.

(i) Explain the meaning of the diamond symbol (labelled with the letter A) in the chart.

.....

.....

.....

..... [2]

(ii) Write the pseudocode module headers for Sub-A and Sub-B.

Sub-A

.....

.....

Sub-B

.....

.....

[4]

7 A fitness club has a computerised membership system.

The system stores information for each club member: name, home address, email address, mobile phone number, date of birth and exercise preferences.

Many classes are full, and the club creates a waiting list for each class. The club adds details of members who want to join a class that is full to the waiting list for that class.

When the system identifies that a space is available in one of the classes, a new module will send a text message to each member who is on the waiting list.

(a) Decomposition will be used to break the new module into sub-modules (sub-problems).

Identify **three** sub-modules that could be used in the design **and** describe their use.

Sub-module 1 .....

Use .....

.....

.....

.....

Sub-module 2 .....

Use .....

.....

.....

.....

Sub-module 3 .....

Use .....

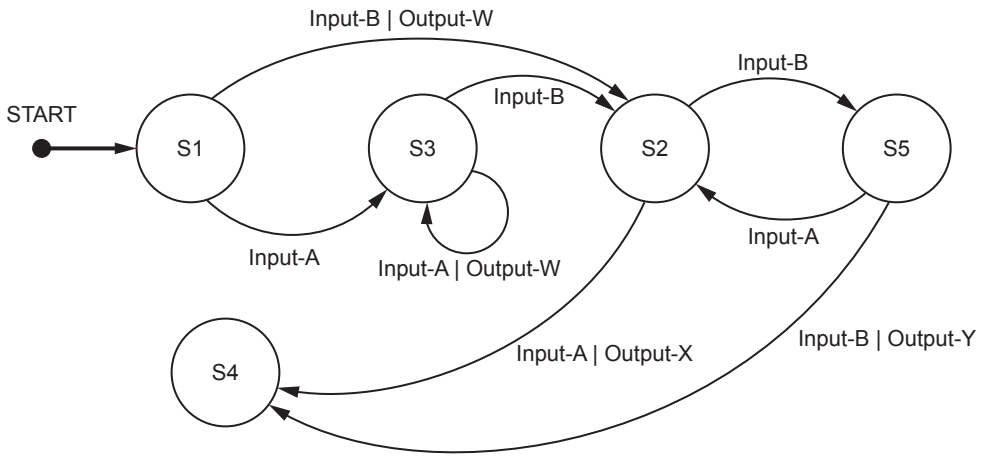
.....

.....

.....

[3]

(b) A different part of the program is represented by the following state-transition diagram.



(i) Complete the table to show the inputs, outputs and next states.

Assume that the current state for each row is given by the 'Next state' on the previous row. For example, the first Input-A is made when in state S1.

If there is no output for a given transition, then the output cell should contain 'none'.

The first two rows have been completed.

Input	Output	Next state
		S1
Input-A	none	S3
	Output-W	
	none	
Input-B		
Input-A		
		S4

[5]

(ii) Identify the input sequence that will cause the minimum number of state changes in the transition from S1 to S4.

..... [1]