Week 26

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. Which data type best stores a yes/no flag like ”in stock”?

   ☐ BOOLEAN

   ☐ STRING

   ☐ REAL

   ☐ DATE

2. An array stores:

   ☐ many items of the same type, reached by an index

   ☐ several fields of different types

   ☐ a single value

   ☐ only text

3. Why does a program write data to a file rather than keeping it in a variable?

   ☐ variables in RAM are lost when the program ends; a file persists

   ☐ files are always faster than RAM

   ☐ variables cannot hold numbers

   ☐ files use no storage

4. An Abstract Data Type (ADT) is defined by:

   ☐ what operations it offers, hiding how the data is stored

   ☐ the exact memory addresses it uses

   ☐ the programming language only

   ☐ the size of the screen

5. Pushing an item onto a stack that is already full causes a stack _____.

   _____

6. A real (floating-point) type should be used to store a measurement such as a temperature.

   ☐ True ☐ False

7. An array is a data structure holding many values of the _____ type under one name.

   _____

8. To add lines to the end of an existing file without losing its contents, open it FOR _____.

_____

9. A stack works in which order?

   ☐ LIFO — last in, first out

   ☐ FIFO — first in, first out

   ☐ random order

   ☐ sorted order

10. Which belong in the declaration and initialisation of an array-based stack? Select **all** that apply.

    ☐ an array with its size and element type

    ☐ a top-of-stack pointer initialised to 0

    ☐ a MaxSize the push operation compares against

    ☐ a front pointer for the next item to leave

    Tick every correct option.

10. Data Types and Structures

Answers · 答案

A-Level Computer Science

1. **BOOLEAN**

   A flag has two states, so BOOLEAN (TRUE/FALSE) is the precise choice — not the strings ”yes”/”no”.

2. **many items of the same type, reached by an index**

   An array is an ordered collection of same-type items accessed by index. (A record groups different types.)

3. **variables in RAM are lost when the program ends; a file persists**

   RAM is volatile, so data is lost on exit. A file on secondary storage persists between runs.

4. **what operations it offers, hiding how the data is stored**

   An ADT specifies the operations (the interface); the implementation is hidden and can change freely.

5. **overflow**

   Overflow = push when Top = MaxSize; popping from an empty stack (Top = 0) is underflow.

6. **True**

   Whole counts use integers; measurements use reals.

7. **same**

   Each value is reached by its index.

8. **APPEND**

   WRITE would start the file empty. APPEND keeps what is there and writes after it.

9. **LIFO — last in, first out**

   A stack is LIFO: the most recently pushed item is the first to be popped.

10. **an array with its size and element type; a top-of-stack pointer initialised to 0; a MaxSize the push operation compares against**

    A stack needs one pointer, Top. A front pointer belongs to a queue.

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

10.1 Data Types and Records

Name: _____ Class: _____ Date: _____ **Total: 37 marks**

KEY WORDS data type 数据类型 • record 记录 • record structure 记录结构 • array 数组 • index 索引

Objective

Build the skills to answer exam questions on **data types** 数据类型 and **records** 记录 — choosing the right type, and defining and using a record structure.

You must be able to:

- choose the most appropriate **data type** for a value
- explain the purpose of a **record** structure
- write pseudocode to **define** a record and to **read/write** its fields

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.

■ Choosing a data type

Pick the **smallest precise type** that fits the value:

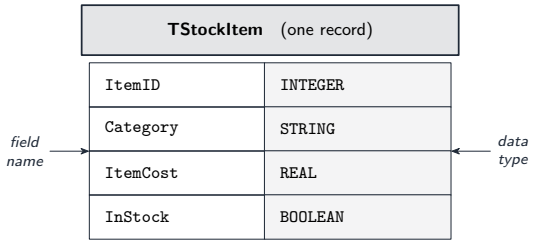
To store	Best type	Why
a count, an index, an ID	`INTEGER`	whole number
a price, a measurement	`REAL`	has a decimal part
a name, a sentence	`STRING`	text
a single grade letter	`CHAR`	exactly one character
”is it paid?”	`BOOLEAN`	a `TRUE`/`FALSE` flag

Exam method — “give the data type of this expression”. Look at the **operation**, not the inputs:

- one character — `LEFT(Name,1)` or `'A'` — gives a **CHAR**
- text in `"..."`, or joined with `&` — gives a **STRING**
- `+ - *` on whole numbers gives an **INTEGER**; any `/` or a decimal value gives a **REAL**
- a comparison (`>=`, `=`) or `AND`/`OR`/`NOT` gives a **BOOLEAN**

■ Defining a record

A **record** groups several fields of **different types** under one type name:




```
TYPE TStockItem
    DECLARE ItemID   : INTEGER
    DECLARE Category : STRING
    DECLARE ItemCost : REAL
    DECLARE InStock  : BOOLEAN
ENDTYPE
```

■ Using a record

Declare a variable of that type (or a whole **array of records**), then use **dot notation** to reach each field:

```
DECLARE Item1 : TStockItem
DECLARE Items : ARRAY[1:100] OF TStockItem

Item1.Category ← "Fruit"
Item1.ItemCost ← 2.50
OUTPUT Item1.Category, " costs ", Item1.ItemCost
```

2 Practice

Now apply the methods above.

2.1 Give the most appropriate data type for each value: (a) a student’s age; (b) a price in dollars; (c) whether a light is on. [3]

2.2 State the purpose of a **record** structure. [1]

2.3 A record type `TBook` must store a `Title` (text), `Pages` (whole number) and `InLibrary` (a flag). Write pseudocode to **define** it. [3]

2.4 `Book1` is a variable of type `TBook`. Write one pseudocode line to set its `Title` to `"Maths"`. [1]

3 Exam-style questions

3.1 The table shows four pseudocode assignment statements. Give an appropriate **data type** to declare each variable. [4]

Statement	Data type
`A ← LEFT(Name, 1)`	
`B ← Price * Quantity`	
`C ← Count + 1`	
`D ← (Score >= 50)`	

3.2 A factory stores data about each component in a record type `Component`:

Field	Example	Meaning
`Item_Num`	`123478`	a numeric ID
`Reject`	`FALSE`	rejected?
`Stage`	`'B'`	a stage letter
`Limit_1`	`13.5`	a value 0–100

(i) Write pseudocode to **declare the record structure** `Component`. [4]

(ii) A 1D array `Item` of **2000** elements will store all components. Write pseudocode to **declare** the `Item` array. [2]

3.3 State **two** benefits of using an **array of records** instead of several separate arrays. [2]

3.4 A bicycle-hire company stores one record for each hire. Each record holds a hire reference (up to 8 characters), the bicycle number, the date of hire, the number of hours hired, the cost per hour and whether the bicycle has been returned.

(a) **Complete** the table by giving an appropriate data type for each field. [3]

field	example value	data type
`HireRef`	`"HR004821"`	
`HoursHired`	`3`	
`CostPerHour`	`4.50`	
`Returned`	`TRUE`	

(b) **Write** pseudocode to declare the record structure `HireRecord`, including a field for the date. [5]

(c) A 1D array `Hire` of 500 elements will hold all the records. **Write** the pseudocode declaration for this array. [2]

(d) **Write** pseudocode to assign the value `4.50` to the cost per hour of the **first** record in the array. [2]

(e) **Outline** three benefits of storing the data as an array of records rather than as six separate parallel arrays. [3]

(f) **Evaluate** the expression `Hire[1].HoursHired * Hire[1].CostPerHour`, and **state** the data type of the result. [2]

Solutions

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

2.1

(a) `INTEGER`

(b) `REAL`

(c) `BOOLEAN`

2.2

- it groups **several fields of different data types** under **one identifier**, so values that belong together are stored as one unit

2.3

- `TYPE … ENDTYPE` with three fields of the right types

```
TYPE TBook
    DECLARE Title     : STRING
    DECLARE Pages     : INTEGER
    DECLARE InLibrary : BOOLEAN
ENDTYPE
```

2.4

- `Book1.Title ← "Maths"`

3.1

- `A` → **CHAR** (or STRING — one character from `LEFT`)
- `B` → **REAL** (a price/decimal)
- `C` → **INTEGER** (whole-number add)
- `D` → **BOOLEAN** (a comparison)

3.2

(i) `TYPE … ENDTYPE` with the four field types

```
TYPE Component
    DECLARE Item_Num : INTEGER
    DECLARE Reject   : BOOLEAN
    DECLARE Stage    : CHAR
    DECLARE Limit_1  : REAL
ENDTYPE
```

(ii) `DECLARE Item : ARRAY[1:2000] OF Component`

3.3

- any two: one identifier for all the data; easy to process with a **loop** and an index; all fields for one item stay together; easy to pass to a procedure

3.4

(a) `HireRef` — `STRING`

- `HoursHired` — `INTEGER`; `CostPerHour` — `REAL`
- `Returned` — `BOOLEAN`

(b)

```
TYPE HireRecord
    DECLARE HireRef : STRING
    DECLARE BicycleNumber : INTEGER
    DECLARE HireDate : DATE
    DECLARE HoursHired : INTEGER
    DECLARE CostPerHour : REAL
    DECLARE Returned : BOOLEAN
ENDTYPE
```

- `TYPE`/`ENDTYPE`; all six fields; correct types; `DATE` for the date; correct pseudocode syntax

(c) `DECLARE Hire : ARRAY[1:500] OF HireRecord`

(d) `Hire[1].CostPerHour ← 4.50`

(e) any three: the fields belonging to one hire are kept **together**, so a record can be passed to a procedure as a single item

- there is no risk of the arrays drifting **out of step** with one another
- adding a new field means changing one type definition instead of creating another array

(f) $3 \times 4.50 = \mathbf{13.50}$

- the result is `REAL`, because multiplying an `INTEGER` by a `REAL` gives a `REAL`

3 A program is needed to manage individual rentals in a car-hire business.

The data items for each rental will be held in a record structure of type `RentalRecord`
The programmer has started to define the items that will be needed:

Item	Example value	Comment
`RentalID`	`"AB1234"`	a unique alpha-numeric value
`CarID`	`241`	a numeric value used as an array index
`DisCode`	`'S'`	a letter indicating the type of discount offered
`Start`	`13/06/2025`	when the rental starts
`Duration`	`7`	the number of days of the rental
`Completed`	`FALSE`	`TRUE` when the car is returned and the rental charge paid

(a) (i) Write pseudocode to declare the record structure for type `RentalRecord`

.....

.....

.....

.....

.....

.....

.....

..... [4]

(ii) A 1D array `Rental` containing 500 elements is used to store the data for all rental records.

Write pseudocode to declare the `Rental` array.

.....

..... [2]

(b) State **three** benefits of using an array of records to store the data for all rentals.

1 .....

.....

2 .....

.....

3 .....

.....

[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]

1 The composite record data type, `ClubMember`, is defined in pseudocode as:

```
TYPE ClubMember
  DECLARE Code : INTEGER
  DECLARE LastName : STRING
  DECLARE FirstName : STRING
  DECLARE Telephone : STRING
  DECLARE JoinDate : DATE
  DECLARE Fees : REAL
  DECLARE FeesPaid : BOOLEAN
ENDTYPE
```

(a) (i) Write the **pseudocode** statement to set up a variable for one record of the composite data type, `ClubMember`.

.....

..... [1]

(ii) Write the **pseudocode** statements to assign the following values to the variable set up in part **(a)(i)**:

- `984632` to `Code`
- `TRUE` to `FeesPaid`

.....

.....

.....

..... [2]

(b) An enumerated data type, `Activity`, is required, so that a new field, `Choice`, can be added to the composite data type, `ClubMember`, to allow members to choose an activity.

(i) Write the **pseudocode** statement for the type declaration of `Activity` to hold the names of the available activities:

Badminton, Football, Golf, Snooker, Swimming, Tennis.

.....

.....

.....

..... [2]

(ii) Write the **new pseudocode** statement required to update the declaration of `Choice` in the definition of `ClubMember`.

.....

..... [1]

1 A program is being developed to control the production line in a factory.

(a) A number of different program life cycles are available for the development of a program.

(i) Explain the need for different program development life cycles.

.....

..... [1]

(ii) Coding is a stage in a program development life cycle.

State **one** consideration that would influence the choice of programming language.

.....

..... [1]

(b) The program has been in use for a number of months and adaptive maintenance is required.

(i) Give **three** reasons why adaptive maintenance may be required.

1 .....

2 .....

3 .....

[3]

(ii) As well as adaptive maintenance, other types of program maintenance may be needed.

Identify **one** other type of program maintenance.

..... [1]

(c) Pseudocode has been used to design modules for the program to control the production line.

The table shows **four** valid pseudocode expressions.

Complete the table by giving the data type of the evaluated expression.

Expression	Data type
`RIGHT(MachineCode, 4)`	
`Speed * 2.5`	
`NOT Status`	
`IS_NUM(Check)`	

[4]

(d) A global array `Product` is used as part of the pseudocode design being developed to control the production line. `Product` is used to store the number of rejected items each day.

The following pseudocode statement is used to assign a value to an element of the array:

```
Product[x, y] ← 23
```

The lower and upper bound values are shown in the table:

Variable	Lower bound	Upper bound
x	0	99
y	0	9

(i) State the number of dimensions of `Product`

..... [1]

(ii) State the total number of elements in array `Product`

..... [1]

(iii) Give the pseudocode declaration for the array `Product`

..... [2]

1 A program is being developed for the management of a sports centre.

(a) The programmer developing the software decides to use modules (procedures or functions) and local variables. They also decide not to use global variables.

(i) State **two** reasons why the programmer decides to use modules.

1 .....

.....

2 .....

.....

[2]

(ii) State **one** difference between local and global variables.

.....

..... [1]

(iii) State **two** benefits of using local variables.

1 .....

.....

2 .....

.....

[2]

(b) The pseudocode design contains a number of expressions.

Complete each pseudocode expression so that it evaluates to the value shown.

Refer to the **insert** for the list of pseudocode functions and operators.

Expression	Evaluates to
.....`(68)`	`'D'`
.....`(04/02/2025)`	2
.....`TRUE`	`FALSE`
.....`(..... ("Court1.4Upper",.....,.....))`	`1.4`

[5]

(c) The table lists some of the variables used in the program.

Complete the table by writing the most appropriate data type for each variable.

Variable	Use of variable	Data type
`MemberCount`	stores how many members have used the sports centre each day	
`TotalTakings`	stores the total amount of money taken each day	
`BookingConfirmed`	stores the state of a booking for an exercise class; a booking is either confirmed or **not** confirmed	
`MemberDOB`	to calculate when to send an email with a birthday message to a member	

[4]

1 Data types can be defined using pseudocode.

The composite record data type, `Departure`, is used to represent flights from Cambridge Airport and is defined in pseudocode as:

```
TYPE Departure
   DECLARE FlightNumber : STRING
   DECLARE Destination : STRING
   DECLARE FlightDate : DATE
   DECLARE Gate : STRING
   DECLARE Airline : STRING
ENDTYPE
```

A variable, `Flight1`, is declared in pseudocode as:

```
DECLARE Flight1 : Departure
```

(a) Write **pseudocode** to store the following details to `Flight1`:

Field	Data
`FlightNumber`	`SB2789`
`Destination`	`Dublin`
`FlightDate`	`30/07/2025`
`Gate`	`N03`
`Airline`	`Cambridge Airways`

.....

.....

.....

.....

.....

..... [3]

(b) The data type for `Gate` is changed to an enumerated data type, `GateID`.

(i) Write a **pseudocode** statement to declare `GateID` to hold the identity codes for the airport gates:

N01, N02, N03, W01, W02, W03, W04

.....

..... [2]

(ii) Write the new **pseudocode** statem equired to replace the declaration of `Gate` in `Departure`.

[1]

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]

4 A program includes the following assignment statement:

```
Result ← STR_TO_NUM(x) / STR_TO_NUM(y)
```

When the program evaluates the expression in the statement, it performs a calculation.

Variable `Result` is of type real and variables `x` and `y` are of type string.

Two checks are required before the calculation is performed:

1. The two strings represent valid numeric values.
2. The numeric value of string `y` is not zero.

(a) Identify the type of error that could occur if these checks are **not** carried out **and** state a cause of this error.

Type ....

....

Cause ....

.... [2]

(b) The designer considers implementing the **checks and calculation** as a module (a procedure or a function). One reason for this is that the same checks and calculations are performed at several places in the program.

Give **another** reason why this is a suitable approach **and** state what is avoided by this approach.

Reason ....

....

Avoided ....

.... [2]

Name: A-Level Computer Science: **w24 21**

(c) The module to perform the checks and calculation will be implemented as a function. The function will need to return both a real **and** a Boolean value. To achieve this a record type is defined in pseudocode as follows:

```
TYPE Result
   DECLARE Done : BOOLEAN
   DECLARE Value : REAL
ENDTYPE
```

The function `Evaluate()` will:

- take two parameters of type string representing the two numeric values
- return a variable of type `Result` with the `Done` field set to `FALSE` if either of the following applies:
  - at least one of the strings does **not** represent a valid numeric value
  - the numeric value of the string representing value `y` is zero
- otherwise return a variable of type `Result` with the `Done` field set to `TRUE` and the `Value` field assigned the result of the formula (based on the numeric value of the two parameters).

Write pseudocode for the function `Evaluate()`.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [6]

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]

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]

3 A factory needs a program to help manage its production of items.

Data will be stored about each item.

The data for each item will be held in a record structure of type `Component`.

The programmer has started to define the fields that will be needed as shown in the table.

Field	Example value	Comment
`Item_Num`	`123478`	a numeric value used as an array index
`Reject`	`FALSE`	`TRUE` if this item has been rejected
`Stage`	`'B'`	a letter to indicate the stage of production
`Limit_1`	`13.5`	any value in the range 0 to 100 inclusive
`Limit_2`	`26.4`	any value in the range 0 to 100 inclusive

(a) **(i)** Write pseudocode to declare the record structure for type `Component`.

.....

.....

.....

.....

.....

.....

..... [4]

(ii) A 1D array `Item` of 2000 elements will store the data for all items.

Write pseudocode to declare the `Item` array.

.....

..... [2]

(b) State **three** benefits of using an array of records to store the data for all items.

1 .....

.....

2 .....

.....

3 ....

..... [3]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

10.2 Arrays

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

KEY WORDS bounds 边界 • linear search 线性查找 • index 索引 • element 元素 • arrays 数组

Objective

Build the skills to answer exam questions on **arrays** 数组 — declaring 1-D and 2-D arrays, and processing them with loops.

You must be able to:

- use the array terms **element**, **index**, **bounds**, **dimension**
- choose and declare a **1-D** or **2-D** array
- write pseudocode to **process** array data (search, total, maximum)

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.

■ Array terms and 1-D arrays

An **array** is an ordered set of items of the **same type**, under one name, reached by an **index**.

- **element** — one item; **bounds** — the lowest and highest valid index; **dimension** — 1-D (a list) or 2-D (a table).

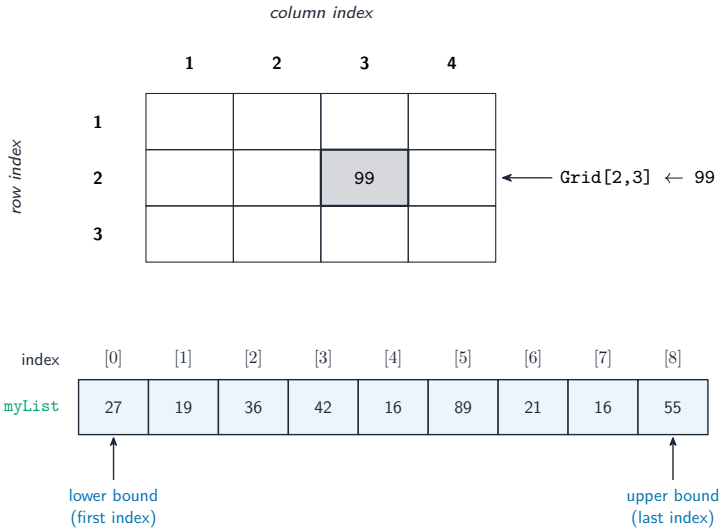
```
DECLARE Names : ARRAY[1:5] OF STRING
Names[3] ← "Cara"
OUTPUT Names[3]
```

Process every element with a `FOR` loop using the index:

```
FOR i ← 1 TO 5
    OUTPUT Names[i]
NEXT i
```


■ 2-D arrays

The first index is the **row**, the second the **column**:



A one-dimensional array indexes a list of elements

```
DECLARE Grid : ARRAY[1:3, 1:4] OF INTEGER
Grid[2, 3] ← 99
```

Visit every cell with **nested** loops (outer = rows, inner = columns).

■ Common operations

Linear search — check each element; use a flag so you can report "not found":

```
Found ← FALSE
FOR i ← 1 TO n
    IF A[i] = Target THEN
        OUTPUT "Found at ", i
        Found ← TRUE
    ENDIF
NEXT i
IF Found = FALSE THEN OUTPUT "Not found"
```

Maximum — set a running value to the first element, then sweep:

```
Max ← A[1]
FOR i ← 2 TO n
    IF A[i] > Max THEN Max ← A[i]
NEXT i
OUTPUT Max
```

2 Practice

Now apply the methods above.

2.1 Explain the array terms **element** and **bounds**. [2]

2.2 Declare a 1-D array `Scores` of **10** integers, then set the **4th** element to `75`. [2]

2.3 Write a `FOR` loop to output **all 10** elements of `Scores`. [2]

2.4 A cinema has **5** rows and **8** seats per row, each marked `'F'` (free) or `'X'`. Declare a suitable **2-D** array `Seats`. [2]

3 Exam-style questions

3.1 Array `Temps[1:n]` holds `n` real temperatures. Write pseudocode to find and output the **largest** value. [4]

3.2 Write pseudocode for a **linear search** of `IDs[1:n]` for a value `Wanted`. Output its position, or `"Not found"` if it is absent. [4]

3.3 A 2-D array `Sales[1:12, 1:4]` holds the monthly sales for 4 shops over 12 months. Write pseudocode to output the **total** of all sales. [3]

3.4 A weather station stores the last 24 hourly temperature readings in a 1D array `Reading` of type `REAL`, indexed from 1 to 24.

(a) **Complete** the pseudocode for the procedure `Store()`, which writes every reading to a text file, one value per line. [5]

```
PROCEDURE Store()
    DECLARE Index : INTEGER
    OPENFILE "temps.txt" FOR ...................
    FOR Index ← 1 TO ...................
        WRITEFILE "temps.txt", .....................
    .....................
    .....................
ENDPROCEDURE
```

(b) A file stores text, but `Reading` holds `REAL` values. **State** the change that must be made to each value before it is written. [1]

(c) A function `FindFirstAbove(Limit)` returns the **index** of the first reading above `Limit`, or -1 if there is none. **Comment** on why the programmer chose -1 as the initial value of the variable `FoundAt`, rather than 0. [2]

(d) **Write** pseudocode for `FindFirstAbove()`, stopping as soon as a reading is found. [5]

(e) The station is upgraded to store readings every **ten minutes** for a week. **State** the change to the array declaration, and **explain** one reason a **file** is used as well as the array. [3]

Solutions

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

2.1

- **element** — one item stored in the array
- **bounds** — the lowest and highest valid index values

2.2

- declare the array, then assign one element

```
DECLARE Scores : ARRAY[1:10] OF INTEGER
Scores[4] ← 75
```

2.3

- loop from 1 to 10 and output each element

```
FOR i ← 1 TO 10
    OUTPUT Scores[i]
NEXT i
```

2.4

- 2-D `ARRAY[…, …]` of `CHAR`
- `DECLARE Seats : ARRAY[1:5, 1:8] OF CHAR`

3.1

- initialise `Max` to the first element, then scan the rest

```
Max ← Temps[1]
FOR i ← 2 TO n
    IF Temps[i] > Max THEN
        Max ← Temps[i]
    ENDIF
NEXT i
OUTPUT Max
```

- compare + update; `OUTPUT` after the loop

3.2

- loop over the array, compare each element, report the position


```
Found ← FALSE
FOR i ← 1 TO n
    IF IDs[i] = Wanted THEN
        OUTPUT "Found at ", i
        Found ← TRUE
    ENDIF
NEXT i
IF Found = FALSE THEN
    OUTPUT "Not found"
ENDIF
```

- flag + `"Not found"` if nothing matched

3.3

- `Total ← 0`, then nested loops over the 2-D array

```
Total ← 0
FOR r ← 1 TO 12
    FOR c ← 1 TO 4
        Total ← Total + Sales[r, c]
    NEXT c
NEXT r
OUTPUT Total
```

- add `Sales[r, c]` and `OUTPUT`

3.4

(a) `OPENFILE "temps.txt" FOR WRITE`

- `FOR Index ← 1 TO 24`
- `WRITEFILE "temps.txt", NUM_TO_STRING(Reading[Index])`
- `NEXT Index`
- `CLOSEFILE "temps.txt"`

(b) each `REAL` must be **converted to a string** first, e.g. with `NUM_TO_STRING`

(c) a valid index runs from 1 to 24, so 0 is not a possible answer either — but -1 can never be mistaken for a **count** or a length

- if the array were ever indexed from 0, returning 0 would be indistinguishable from finding a match at the first element

(d)

```
FUNCTION FindFirstAbove(Limit : REAL) RETURNS INTEGER
    DECLARE FoundAt, Index : INTEGER
    FoundAt ← -1
    Index ← 1
    WHILE Index <= 24 AND FoundAt = -1
        IF Reading[Index] > Limit
          THEN FoundAt ← Index
          ELSE Index ← Index + 1
        ENDIF
    ENDWHILE
    RETURN FoundAt
ENDFUNCTION
```

- header with parameter and return type; `FoundAt` initialised to -1; loop with both conditions; correct comparison; `RETURN`

(e) $6 \times 24 \times 7 = \mathbf{1008}$ readings, so `DECLARE Reading : ARRAY[1:1008] OF REAL`

- the array is **volatile** — its contents are lost when the power goes or the program ends
- the file keeps the readings permanently and lets them be analysed later

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]

5 A program is being designed in pseudocode.

The program contains the following declaration for the global array `MyData`:

```
DECLARE MyData : ARRAY[1:10000] OF STRING
```

A function `FindFirst()` is written to search the array for a given string and to return the index of the first element where that string is found, or to return –1 if the string is **not** found.

The function is written in pseudocode as shown:

```
FUNCTION FindFirst(SearchString : STRING) RETURNS INTEGER
   DECLARE Index, FoundAt : INTEGER
   FoundAt ← -1
   FOR Index ← 1 TO 10000
      IF MyData[Index] = SearchString THEN // outer conditional clause
         IF FoundAt = -1 THEN              // inner conditional clause
            FoundAt ← Index
         ENDIF
      ENDIF
   NEXT Index
   RETURN FoundAt
ENDFUNCTION
```

(a) **(i)** Comment on why the programmer chose –1 as the initial value of `FoundAt`

.....

..... [1]

(ii) Explain the purpose of the **outer** conditional clause.

.....

.....

..... [1]

(iii) The **inner** conditional clause ensures that only the index of the **first** matching element, if any, is returned.

Explain how this clause works.

.....

.....

..... [1]

(b) The pseudocode does **not** use the most appropriate loop construct.

(i) Explain why this is **not** the most appropriate loop construct.

.....

.....

..... [1]

(ii) Suggest and justify a more appropriate loop construct that could be used.

Construct .....

Justification .....

.....

[2]

4 A quiz has nine questions. There are:

- five easy questions each worth 3 points
- four hard questions each worth 5 points.

At the end of the quiz the points for each correctly answered question are added to give a total score.

A check is made to test that the total score is valid using a 1D array `CheckTotal` of type `BOOLEAN`. Each index value of the array represents a possible total score. The corresponding element value is `TRUE` if the index value represents a valid total score and `FALSE` otherwise.

The first nine rows of the array are:

Index value	Element value	Comment
0	`TRUE`	valid total score (no correct answers)
1	`FALSE`	invalid total score
2	`FALSE`	invalid total score
3	`TRUE`	valid total score (one 3-point question correct)
4	`FALSE`	invalid total score
5	`TRUE`	valid total score (one 5-point question correct)
6	`TRUE`	valid total score (two 3-point questions correct)
7	`FALSE`	invalid total score
8	`TRUE`	valid total score (one 3-point question and one 5-point question correct)

For example, a total score of 6 points is valid; the value of the array element at index value 6 is `TRUE`

(a) Write pseudocode to declare `CheckTotal` and to set all elements of the array to `FALSE`

All variables used must be declared.

.....

.....

.....

.....

.....

.....

.....

.... [4]

(b) The pseudocode represents an algorithm to set the appropriate elements of the array to `TRUE`

Complete the pseudocode:

```
DECLARE EasyQ, HardQ : INTEGER

FOR EasyQ ← ..... TO 15 STEP .....

   FOR HardQ ← 0 TO ..... STEP .....

      CheckTotal[.....] ← TRUE

   NEXT HardQ

NEXT EasyQ
```

[5]

(c) The array elements have been assigned the required values.

A module `ValidateScore()` will take an integer value representing a total score as a parameter. It will return `TRUE` if the total score is valid, or `FALSE` if it is **not** valid.

Describe the algorithm for `ValidateScore()` using **four** steps.

Do **not** use pseudocode in your answer.

Step 1 .....

.....

Step 2 .....

.....

Step 3 .....

.....

Step 4 .....

.....

[4]

4 A program contains a global 1D array `Data` containing 20 elements of type `INTEGER`

A global string `NumString` represents a sequence of three-digit numbers, separated by commas. For example:

```
"101,456,219,754,328"
```

The string contains at least **four** three-digit numbers. The total number of three-digit numbers in the string is unknown.

A procedure `Store()` will:

- extract one three-digit number at a time from `NumString`
- convert each of the three-digit numbers extracted to an integer and assign this to the next array element, starting from index 1
- end when all three-digit numbers have been stored, or when the array is full.

Complete the pseudocode for `Store()`

All local variables used must be declared.

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

[6]

5 A global 1D array `Num` of integers contains four elements. A program assigns values to these elements as shown:

```
Num[1] ← 1
Num[2] ← 2
Num[3] ← 5
Num[4] ← 3
```

A procedure `Process()` manipulates the values in the array.

The procedure is written in pseudocode:

```
PROCEDURE Process(Start : INTEGER)
   DECLARE CaseVar, Index, Count : INTEGER

   Index ← Start
   Count ← 0

   WHILE Count <= 20
      CaseVar ← Num[Index]
      CASE OF CaseVar
         1 : Num[Index] ← Num[Index] + Index //clause one
             Index ← Index + 1
             Count ← Count + 1

         2 : Num[Index] ← Num[Index] + Index //clause two
             Index ← Index + 2
             Count ← Count + 2

         3 : Num[Index] ← Num[Index] * 2     //clause three
             Index ← Index + 3
             Count ← Count - 1

         4 : Count ← Count + 4               //clause four
         OTHERWISE : Count ← 20
      ENDCASE

      Index ← (Index MOD 4) + 1

   ENDWHILE

ENDPROCEDURE
```


(a) Complete the trace table by dry running the procedure when it is called in the statement:
`CALL Process(1)`

Index	CaseVar	Count	Num[1]	Num[2]	Num[3]	Num[4]

[5]

(b) As a reminder, the CASE structure in the pseudocode is:

```
CASE OF CaseVar
   1 : Num[Index] ← Num[Index] + Index //clause one
       Index ← Index + 1
       Count ← Count + 1

   2 : Num[Index] ← Num[Index] + Index //clause two
       Index ← Index + 2
       Count ← Count + 2

   3 : Num[Index] ← Num[Index] * 2     //clause three
       Index ← Index + 3
       Count ← Count - 1

   4 : Count ← Count + 4               //clause four
   OTHERWISE : Count ← 20
ENDCASE
```

The CASE structure could be optimised by combining two existing clauses.

(i) Identify the CaseVar values for these **two** existing clauses.

..... [1]

(ii) Write pseudocode for a single clause to replace the **two** clauses identified in **(b) (i)**.

.....

.....

..... [2]

6 A factory produces food items. The items must be used within a certain number of days after their production date. The number of days is known as the shelf life. It is different for each type of item but is always a whole number in the range 1 to 21 (inclusive).

The latest date that an item can be used is called the 'use-by' date.

A program is needed to produce labels which show the 'use-by' date.

Part of the program is a function GetDate() which will:

- take two parameters: a production date and a value representing the shelf life
- return the corresponding 'use-by' date.

The program contains a global 1D array DaysInMonth of type integer which stores the number of days in each month (index 1 is January):

Index	Value
1	31
2	28
3	31
4	30
11	30
12	31

Note: Leap years are **not** considered

(a) An algorithm uses the array DaysInMonth to calculate a 'use-by' date. An alternative design would involve the use of multiple selection statements.

An array-based technique is often used when there is a large number of different values to check and where no pattern exists.

One advantage of using an array-based technique is the speed of execution compared to the use of multiple selection statements.

Give **two other** advantages of using an array for this type of operation rather than a solution based on multiple selection statements.

1 .....

.....

2 .....

.....
[2]

(b) Complete the pseudocode for the function `GetDate()`.

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

```
FUNCTION GetDate(ProductionDate : DATE, ShelfLife : INTEGER) RETURNS DATE
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDFUNCTION
```


[7]

8 An exam 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. For example, if the grade boundary for an A grade is 65 marks, then any candidate who achieves a mark of 65 or above will be awarded an A. A grade of U is awarded for marks below the E grade boundary.

The five grade boundaries are stored in a global 1D array `GradeBoundary` of type integer.

For example:

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

A global 2D array `Result` of type integer contains candidate marks for the exam. Each row relates to one candidate. Column 1 contains the candidate mark and column 2 contains the unique candidate ID.

For example, for the fourth and fifth candidates:

Element	Mark
`Result[4, 1]`	56
`Result[5, 1]`	54

Element	ID
`Result[4, 2]`	1074832
`Result[5, 2]`	2573839

There are more rows in the array than candidates who sit the exam. Any unused rows will be at the end of the array.

Candidate papers that are given a mark within two marks of any grade boundary must be checked.

For example, given the values in the example grade boundaries above, any paper that was awarded between 41 and 45 marks (inclusive) would need to be checked.

A program is being written to identify papers that need to be checked.

The programmer has defined the first program module as follows:

Module	Description
`CheckMark()`	• called with a parameter of type integer representing a candidate mark • returns `TRUE` if the mark is within 2 of any of the five grade boundaries, otherwise returns `FALSE`

(a) Write pseudocode for module `CheckMark()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

(b) A second module is defined:

Module	Description
`CheckAll()`	• called with a parameter of type integer representing the number of candidate marks in the `Result` array • uses `CheckMark()` to check each candidate mark • for each paper that needs to be checked, write the corresponding candidate ID on a separate line in a new file named `GRList.txt` • outputs a message with a count of how many papers need to be checked

Write pseudocode for module `CheckAll()`.

`CheckMark()` must be used to check each individual mark.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

(c) The requirement changes. Instead of a new file, the module described in part **(b)** needs to add the corresponding candidate ID for each paper that needs to be checked to an **existing** file.

Explain the change that will need to be made to `CheckAll()`.

.....

..... [1]

2 A program is being developed to process bank card information.

When a card number is displayed, all the characters **except** the last four are replaced with the asterisk character `'*'`.

Card numbers are stored as strings. The strings are between 10 and 20 characters in length.

The function `Conceal()` will take a string representing a card number and return a modified string.

Example strings:

Original string	Modified string
`"1234567890"`	`"******7890"`
`"1234567897652"`	`"*********7652"`
`"1234567890123456"`	`"************3456"`

(a) The function `Conceal()` will:

- take a numeric string as a parameter representing the card number
- return a string in which the asterisk character replaces all except the last four characters of the card number parameter.

Write pseudocode for the function `Conceal()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[6]

(b) The requirements have been changed. `Conceal()` will now be written as a procedure which will process 100 card numbers each time it is called.

The card numbers will be stored in a 2D array `CardNumber`. The original string will be stored in column one and the modified string in column two.

(i) Write pseudocode to declare the array.

.....

..... [2]

(ii) The new procedure `Conceal()` will write the modified string to the corresponding element in column two.

The array `CardNumber` is passed as a parameter to the new procedure `Conceal()`.

Identify how this parameter should be specified in the new procedure header.

.....

..... [1]

1 A program sorts string data using different sorting methods.

(a) The text file `Data.txt` stores string data items. Each data item is on a new line in the text file.

The function `ReadData()`:

- has a local array of strings that can store 45 items
- reads each line of data and stores it in the array
- returns the array.

Write program code for the function `ReadData()`.

Save your program as **Question1_N24**.

Copy and paste the program code into part **1(a)** in the evidence document.

[6]

(b) The function `FormatArray()` takes an array of strings as a parameter. It concatenates the contents of the array into one string with a space between each array element. The function returns the concatenated string.

(i) Write program code for `FormatArray()`.

Save your program.

Copy and paste the program code into part **1(b)(i)** in the evidence document.

[2]

(ii) The main program:

- calls `ReadData()` and stores the returned array
- calls `FormatArray()` with the returned array and outputs the returned string.

Write program code for the main program.

Save your program.

Copy and paste the program code into part **1(b)(ii)** in the evidence document.

[3]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(b)(iii)** in the evidence document.

[1]

(c) The function `CompareStrings()`:

- takes two strings as parameters
- compares each string, one character at a time, to identify which string comes first alphabetically. If the first two characters are the same, the second character of each string is compared. This continues until the two characters are different.

The function:

- returns `1` if the first parameter comes before the second alphabetically
- returns `2` if the second parameter comes before the first alphabetically.

Write program code for `CompareStrings()`.

Assume that all strings are in lower case.

Assume that a difference between two strings will always be identified before the end of one string is reached.

Do **not** use an in-built string comparison function.

The strings must be compared one character at a time.

Save your program.

Copy and paste the program code into part **1(c)** in the evidence document.

[4]

(d) The function `Bubble()` takes an array of strings as a parameter and sorts the data into ascending alphabetical order, using a bubble sort. The bubble sort uses `CompareStrings()` to compare each string.

The function returns the sorted list.

(i) Write program code for `Bubble()`.

Save your program.

Copy and paste the program code into part **1(d)(i)** in the evidence document.

[3]

(ii) Write program code to amend the main program to:

- call `Bubble()` with the unsorted array as a parameter
- call `FormatArray()` with the sorted array and output the returned string.

Save your program.

Copy and paste the program code into part **1(d)(ii)** in the evidence document.

[2]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(d)(iii)** in the evidence document.

[1]

1 A program sorts string data using different sorting methods.

(a) The text file `Data.txt` stores string data items. Each data item is on a new line in the text file.

The function `ReadData()`:

- has a local array of strings that can store 45 items
- reads each line of data and stores it in the array
- returns the array.

Write program code for the function `ReadData()`.

Save your program as **Question1_N24**.

Copy and paste the program code into part **1(a)** in the evidence document.

[6]

(b) The function `FormatArray()` takes an array of strings as a parameter. It concatenates the contents of the array into one string with a space between each array element. The function returns the concatenated string.

(i) Write program code for `FormatArray()`.

Save your program.

Copy and paste the program code into part **1(b)(i)** in the evidence document.

[2]

(ii) The main program:

- calls `ReadData()` and stores the returned array
- calls `FormatArray()` with the returned array and outputs the returned string.

Write program code for the main program.

Save your program.

Copy and paste the program code into part **1(b)(ii)** in the evidence document.

[3]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(b)(iii)** in the evidence document.

[1]

(c) The function `CompareStrings()`:

- takes two strings as parameters
- compares each string, one character at a time, to identify which string comes first alphabetically. If the first two characters are the same, the second character of each string is compared. This continues until the two characters are different.

The function:

- returns `1` if the first parameter comes before the second alphabetically
- returns `2` if the second parameter comes before the first alphabetically.

Write program code for `CompareStrings()`.

Assume that all strings are in lower case.

Assume that a difference between two strings will always be identified before the end of one string is reached.

Do **not** use an in-built string comparison function.

The strings must be compared one character at a time.

Save your program.

Copy and paste the program code into part **1(c)** in the evidence document.

[4]

(d) The function `Bubble()` takes an array of strings as a parameter and sorts the data into ascending alphabetical order, using a bubble sort. The bubble sort uses `CompareStrings()` to compare each string.

The function returns the sorted list.

(i) Write program code for `Bubble()`.

Save your program. Copy and paste the program code into part **1(d)(i)** in the evidence document.

[3]

(ii) Write program code to amend the main program to:

- call `Bubble()` with the unsorted array as a parameter
- call `FormatArray()` with the sorted array and output the returned string.

Save your program. Copy and paste the program code into part **1(d)(ii)** in the evidence document.

[2]

(iii) Test your program.

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into part **1(d)(iii)** in the evidence document.

[1]

4 A global 1D array `Data` contains 100 elements of type integer.

A function `Check()` will:

- total the element values in odd index locations (1, 3, 5 ... 97, 99)
- total the element values in even index locations (2, 4, 6 ... 98, 100)
- return one of three strings ‘Odd’, ‘Even’ or ‘Same’ to indicate which total is the greater, or whether the totals are the same.

Write pseudocode for the function `Check()`.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [6]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

10.3 Files

Name: _____ Class: _____ Date: _____ **Total: 31 marks**

KEY WORDS file 文件 • text file 文本文件 • end of file 文件结束 • record 记录

Objective

Build the skills to answer exam questions on **text files** 文本文件 — why files are needed, and reading and writing them in pseudocode.

You must be able to:

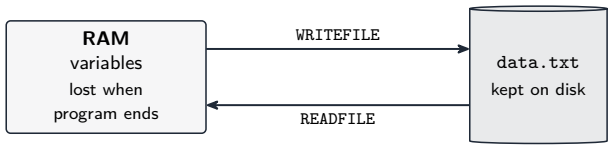
- explain **why** a program uses files
- write pseudocode to **read** a text file line by line using `EOF`
- write pseudocode to **write** or **append** lines to a text file

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.

■ Why files

Variables live in **RAM** and are lost when the program ends. A **file** on disk is **kept** between runs — so it stores data permanently (high scores, records) and lets programs share data.



■ Reading a text file

Open it, read each line until **end of file**, then **close** it:

```
OPENFILE "data.txt" FOR READ
WHILE NOT EOF("data.txt") DO
    READFILE "data.txt", LineString
    OUTPUT LineString
ENDWHILE
CLOSEFILE "data.txt"
```

`EOF` returns `TRUE` when there is nothing left to read — test it **before** each `READFILE`.

■ Writing and appending

`FOR WRITE` creates a new file (or **overwrites** an old one); `FOR APPEND` keeps the existing lines and adds after them:

```
OPENFILE "log.txt" FOR WRITE
FOR i ← 1 TO 100
    WRITEFILE "log.txt", "Event " & i
NEXT i
CLOSEFILE "log.txt"
```

Always **close** a file — otherwise buffered writes can be lost and the file may stay locked.

2 Practice

Now apply the methods above.

2.1 State **one** reason a program needs to use a file rather than only variables. [1]

2.2 State when the function `EOF` returns `TRUE`. [1]

2.3 Write pseudocode to **open** `"scores.txt"` for reading and then **close** it. [2]

2.4 State **one** reason you must always **close** a file. [1]

3 Exam-style questions

3.1 Write pseudocode to read **every line** of `"names.txt"` and output each line. [4]

3.2 Write pseudocode to write the numbers **1 to 50**, one per line, to a new file `"nums.txt"`. [3]

3.3 A file `"members.txt"` holds one name per line. Write pseudocode to **count** how many names are in the file and output the total. [4]

3.4 A sports club stores one line per member in a text file `members.txt`. Each line holds the membership number, the surname and the class of membership, separated by the character `|`.

```
40118|Chen|Full
40119|Osei|Junior
```

(a) **Explain** why a **separator** character is needed, and **state** one property the chosen character must have. [3]

(b) **Write** pseudocode for a procedure `CountClass(Wanted)` that reads the whole file and outputs how many members are in the class given by `Wanted`. [6]

(c) The club adds two new items for each member: a phone number and an emergency contact, **both encrypted**. **Explain** one problem this could cause for the file format, and **describe** how you would solve it. [3]

(d) **Compare** storing this data in a **serial** file with storing it in a **sequential** file ordered by membership number, for the task of finding one member. [3]

Solutions

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

2.1

- any one: data is **kept** after the program ends; it stores large amounts permanently; it lets programs share data / restart from a saved state

2.2

- when the **end of the file** has been reached — there is nothing left to read

2.3

- open for read, then close

```
OPENFILE "scores.txt" FOR READ
CLOSEFILE "scores.txt"
```

2.4

- any one: so **buffered writes are saved**; so the file is not left **locked** for other programs

3.1

- open, loop until EOF, read and output, then close

```
OPENFILE "names.txt" FOR READ
WHILE NOT EOF("names.txt") DO
    READFILE "names.txt", Line
    OUTPUT Line
ENDWHILE
CLOSEFILE "names.txt"
```

3.2

- open for write, loop 1 to 50, then close

```
OPENFILE "nums.txt" FOR WRITE
FOR i ← 1 TO 50
    WRITEFILE "nums.txt", i
NEXT i
CLOSEFILE "nums.txt"
```

3.3

- start `Count` at 0, then read every record

```
Count ← 0
OPENFILE "members.txt" FOR READ
WHILE NOT EOF("members.txt") DO
    READFILE "members.txt", Name
    Count ← Count + 1
ENDWHILE
CLOSEFILE "members.txt"
OUTPUT Count
```

- increment inside the loop; close and output

3.4

(a) the three items are of different lengths, so without a marker the program cannot tell where one field ends and the next begins

- the separator must be a character that can **never appear inside the data itself** — a surname could contain a space or a hyphen, so | is safer than either

(b)

```
PROCEDURE CountClass(Wanted : STRING)
    DECLARE Line, ThisClass : STRING
    DECLARE Count : INTEGER
    Count ← 0
    OPENFILE "members.txt" FOR READ
    WHILE NOT EOF("members.txt")
        READFILE "members.txt", Line
        ThisClass ← the text after the second "|" in Line
        IF ThisClass = Wanted THEN Count ← Count + 1 ENDIF
    ENDWHILE
    CLOSEFILE "members.txt"
    OUTPUT Count
ENDPROCEDURE
```

- header with parameter; counter initialised; `OPENFILE … FOR READ`; `WHILE NOT EOF` with `READFILE`; the field extracted and compared; `CLOSEFILE` and output

(c) encrypted data is arbitrary bytes, so it may **contain the separator character itself**, which would split one field into two and corrupt every field after it on that line

- encode the encrypted values in a form that cannot contain | (hex or Base64), or use fixed-length fields instead of a separator

(d) in a **serial** file the records are in the order they were added, so a search must read from the start until it finds the member: on average half the file

- a **sequential** file is ordered by membership number, so the search can stop as soon as it passes the wanted key, and the file can be searched far faster
- the cost is that inserting a new member into a sequential file means rewriting the file to keep the order, whereas a serial file just appends

3 A text file `OldFile.txt` contains IDs, names and email addresses for members of a club. Three information items are stored for each member and each item is stored on a separate line.

The example shows the information for the first two members in the first six lines of the file:

Line in file	Information item	Example data
1	member 1 ID	`"AB1234"`
2	member 1 name	`"Freddie Jones"`
3	member 1 email address	`"FreddieJ909@Cambridge.org"`
4	member 2 ID	`"BC2345"`
5	member 2 name	`"Sue Smith"`
6	member 2 email address	`"Sue1024@Cambridge.org"`

The member ID string is always two letters followed by four digits.

The file design is to be changed so that information for each user is stored as a single line of the file, with the character `'\'` used as a separator between data items.

For example, the single line for member 1 will be:

`"AB1234\Freddie Jones\FreddieJ909@Cambridge.org"`

An algorithm will produce a new file `NewFile.txt` from the contents of `OldFile.txt`

Assume:

- `LineX`, `LineY` and `LineZ` are of type `STRING` and are used to store the three items of information for each member
- `NewString` is a temporary variable of type `STRING`
- `OldFile.txt` exists and contains valid data.

(a) The algorithm to create the new file is expressed in steps.

Complete the following numbered steps:

Step 1 : open the file ..... in .....

Step 2 : open the file ..... in .....

Step 3 : read a line from ..... and store in .....

Step 4 : read a line from ..... and store in .....

Step 5 : read a line from ..... and store in .....

Step 6 : set `NewString` to .....

Step 7 : write `NewString` to .....

Step 8 : repeat from step ..... until .....

Step 9 : close both files. (1)

[6]

(b) The character `'\'` has been chosen as a separator.

Explain why this is a suitable character.

.....

..... [1]

(c) Two new information items are to be stored for each member. Both new items are encrypted; they can each contain any character (including `'\'`) and can be of any length up to a maximum of 99 characters.

For example:

Information item	Example data
member 1 ID	`"AB1234"`
member 1 name	`"Freddie Jones"`
member 1 email address	`"FreddieJ909@Cambridge.org"`
member 1 new information 1	`"En/98&*(|?\/D7iP"`
member 1 new information 2	`"23\CoboL"`

Explain the additional file design changes needed so that:

- all the information items for each member are stored on a single line of `NewFile.txt`
- it is possible to extract each information item after the line is read.

Assume the `'\'` character is not used in any email address.

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [3]

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]

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 The first three characters of a `StudentID` represent a tutor ID. Each student has one tutor.
`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 8000 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
`LoanStatus()`	• called with two parameters of type `STRING` representing a `StudentID` and a `BookID` • outputs a message saying whether a given loan has been returned or not • outputs a warning message if a record of the given loan is **not** found

(a) Write efficient pseudocode for the module `LoanStatus()`

Assume that each combination of `StudentID` and `BookID` can occur only once.

[8]

(b) A second module is defined:

Module	Description
`LoansPerTutor()`	• called with a parameter of type `STRING` representing a tutor ID (as a reminder, the first three characters of a `StudentID` represent a tutor ID) • returns an integer value representing the number of books currently on loan to students who have the given tutor

Reminder: unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

Write pseudocode for the module `LoansPerTutor()`

[7]

(c) It is decided to mark as unused all records for book loans that have been returned. The data for these records will first be written to a new text file for archive purposes.

(i) The archive program will automatically generate a meaningful filename each time it is run.

Outline a meaningful format to use for the filename.

[2]

(ii) There is a problem that will need to be overcome before the data items can be written to a text file.

As a reminder, the data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book The first three characters of a `StudentID` represent a tutor ID. Each student has one tutor.
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

Explain the problem.

[1]

(iii) One way of storing the data items in a text file is to store each data item on a separate line.

Identify **one** benefit and **one** drawback of this way of storing the data.

Benefit

Drawback

[2]

2 A program reads string data from a text file into a linear queue and then compresses the data.

The queue is created using the global 1D array `Queue`. The queue can store up to 100 elements. Each element is initialised to the empty string `""`

The queue has two pointers that are global to the program:

- `QueueHead` that points to the index of the first element in the queue, initialised to `-1`
- `QueueTail` that points to the index of the last element in the queue, initialised to `-1`

The global variable `NumberItems` stores the number of elements stored in the queue, initialised to `0`

(a) Write program code to declare and initialise `Queue`, `QueueHead`, `QueueTail` and `NumberItems`

Save your program as **Question2_N25**.

Copy and paste the program code into part **2(a)** in the evidence document.

[2]

(b) The function `Enqueue()` takes a string as a parameter. The function checks if the queue is full, and returns Boolean `FALSE` if the queue is full.

If the queue is not full, the parameter is inserted into the next position in `Queue`. The function updates the appropriate pointer(s), updates `NumberItems` and then returns Boolean `TRUE`

Write program code for `Enqueue()`

Save your program.

Copy and paste the program code into part **2(b)** in the evidence document.

[5]

(c) The function `Dequeue()` returns the string `"False"` if the queue is empty.

If the queue is not empty, the function returns the next element in the queue. The function updates the appropriate pointer(s) and updates `NumberItems`

Write program code for `Dequeue()`

Save your program.

Copy and paste the program code into part **2(c)** in the evidence document.

[3]

(d) The text file `BinaryData.txt` stores individual binary digits, `'1'` and `'0'`. Each digit is on a new line. For example, the first line in the text file stores `'1'`, the second line stores `'1'`

The procedure `ReadData()` reads in each line from the text file `BinaryData.txt` and inserts it into the queue using the appropriate method.

The procedure needs to work for a text file with any number of lines up to a maximum of 100.

Write program code for `ReadData()`

Save your program.

Copy and paste the program code into part **2(d)** in the evidence document.

[5]

(e) The string data in the text file is compressed.

The compression algorithm counts the number of times each binary digit appears consecutively, then stores the binary digit followed by the number of times it appears. The algorithm stores the compressed data in a single string.

For example, if the text file contains the data:

```
1
1
0
0
0
1
1
1
```

The compression algorithm will create the string `"120313"` because there are two `'1'` digits, followed by three `'0'` digits, followed by three `'1'` digits.

The procedure `Compress()` uses `Dequeue()` to remove each element from the queue in turn. The procedure then compresses the data following the compression algorithm described. The new compressed string is stored in the global variable `NewString`

You can assume that one binary digit will never appear more than nine times consecutively in the sequence.

You can assume that there will always be at least one item in the queue.

Write program code for `Compress()`

Save your program.

Copy and paste the program code into part **2(e)** in the evidence document.

[6]

(f) **(i)** Write program code for the main program to:

- call `ReadData()`
- call `Compress()`
- output the content of the compressed string.

Save your program. Copy and paste the program code into part **2(f)(i)** in the evidence document.

[2]

(ii) Test your program.

Take a screenshot of the output(s).

Save your program. Copy and paste the screenshot(s) into part **2(f)(ii)** in the evidence document.

[1]

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]

4 A programmer is developing a new stock control program for a shop owner to produce sales reports.

(a) A text file `Sales.txt` stores strings representing the number of items sold.

The file contains 52 lines, each representing the number of items sold in each week of the year.

For example:

File line number	File data
1	"350"
2	"502"
3	"434"
49	"492"
50	"872"
51	"772"
52	"201"

The example shows that 502 items were sold in week 2.

The owner requires a module to output all week numbers where the number of items sold is greater than 500.

Describe an algorithm that produces the required module.

Do **not** include pseudocode statements in your answer.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(b) Once the program has been completed, it is tested using the walkthrough method.

Describe the walkthrough method and explain how it can be used to identify errors.

.....

.....

.....

.....

.....

.....

.....

..... [3]

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
loyalty points	an integer value that depends on how often the customer visits the coffee shop
last visit date	the date the customer last visited the coffee shop

The programmer has defined a program module:

Module	Description
`UpdateVisit()`	• called with two parameters: 1. loyalty points of type `INTEGER` 2. last visit date of type `DATE` • change the loyalty points: ○ increase by **four** if the last visit date and the current date are in the same month ○ otherwise increase by **one** • change the last visit date to the current date • the changed values must be available to the code that follows the call to `UpdateVisit()`

(a) (i) Write pseudocode for module `UpdateVisit()`

Assume the customer has made at least **one** previous visit.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[5]

(ii) The programmer decides to amend the `UpdateVisit()` module so that loyalty points are further increased if the customer visits the coffee shop the same day of the week as their last visit.

Write the pseudocode for this condition.

.....

..... [1]

(b) A text file `Loyalty.txt` will be used to store the data items for the loyalty scheme.

The text file `Loyalty.txt` contains only one line of data for each customer.

Each data item is separated by a comma:

`<CustomerID>,<LoyaltyPointsString>,<LastVisitDateString>`

`LastVisitDateString` is always eight characters in length in the format: DDMMYYYY.

An example of how a line of data will be stored in the text file `Loyalty.txt` is:

`"100123,132,05032025"`

This example shows that the customer with an ID of 100123 had 132 loyalty points following their last visit to the coffee shop on 05/03/2025.

If a customer visits the coffee shop on a Monday, the value of their loyalty points is increased by 10.

The programmer has defined **two** more program modules:

Module	Description
`FindCustomer()` (is already written)	• called with a parameter of type `STRING` that represents a customer ID • returns the line of data read from the text file `Loyalty.txt` containing the data items for that customer
`MondayCheck()`	• called with a parameter of type `STRING` that represents the customer ID of a customer • calls `FindCustomer()` • extracts the loyalty points from the string returned by `FindCustomer()` • increases the loyalty points for the current customer by 10 if the day visited is a Monday • returns an integer value representing the loyalty points

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

The module `FindCustomer()` must be used and assume it returns a valid string for the current customer.

[8]

(ii) A date is stored in `LastVisitDateString` in the format:

DDMMYYYY

DD is a 2-digit string
MM is a 2-digit string
YYYY is a 4-digit string.

For example, the date 03/09/2024 is stored as `"03092024"`

An algorithm is needed to convert the string stored in `LastVisitDateString` to data type `DATE` which is then stored in the variable `LastVisitDate`.

Complete the pseudocode for this algorithm:

Assume that `LastVisitDateString` has been declared and contains valid data.

```
DECLARE DayInt, MonthInt, YearInt : INTEGER

DECLARE LastVisitDate : DATE
```

[4]

3 The text file `HighScoreTable.txt` stores the top seven player scores for a game.

The data in the file is stored in the order:

Player ID

Game level

Score

Each item of data is stored on a new line. For example, the first set of data in the file is:

Player ID: GHEH

Game level: 3

Score: 10

(a) The data about the players and their scores is stored in a 2D array of strings with the identifier `HighScores`.

The first dimension of the array has seven elements: one for each player. The second dimension of the array has three elements: one for the player ID, one for the game level and one for the score. All data is stored in the array as strings.

`HighScores` is declared local to the main program, and all elements are initialised to an empty string, for example `""`.

Write program code to declare and initialise `HighScores`.

Save your program as **Question3_N24**. Copy and paste the program code into part **3(a)** in the evidence document.

[2]

(b) The function `ReadData()` reads the data from `HighScoreTable.txt` and stores the data in a 2D array. The function returns the 2D array.

The function uses exception handling when opening and reading data from the file.

Write program code for `ReadData()`.

Save your program. Copy and paste the program code into part **3(b)** in the evidence document.

[5]

(c) The procedure `OutputHighScores()` takes a 2D array as a parameter. It outputs each player's ID, level and score in the order they are in the array. The outputs are in the format:

GHEH reached level 3 with a score of 10

Write program code for `OutputHighScores()`.

Save your program. Copy and paste the program code into part **3(c)** in the evidence document.

[2]

(d) The scores in `HighScoreTable.txt` are **not** in order.

The player scores are first sorted by the game level they reached. For example, all players that reached level 5 are higher in the high score table than players that reached level 4.

The players that reached the same level are then sorted in descending order of score.

An example sorted high score table is:

Position	Player ID	Game level	Score
1	GFED	5	25
2	HJKM	4	21
3	RERR	4	19
4	TTYU	4	15
5	WSVG	3	20
6	PPTR	3	15
7	SNQT	2	10

The function `SortScores()` uses a bubble sort to sort the data as described and returns the sorted array.

Write program code for `SortScores()`.

Save your program. Copy and paste the program code into part **3(d)** in the evidence document.

[4]

(e) (i) Amend the main program to implement these steps in the order given:

- call `ReadData()` and store the returned array in `HighScores`
- output `"Before"`
- call `OutputHighScores()` with `HighScores` as a parameter
- call `SortScores()` and store the returned array in `HighScores`
- output `"After"`
- call `OutputHighScores()` with `HighScores` as a parameter.

Save your program.

Copy and paste the program code into part **3(e)(i)** in the evidence document.

[2]

(ii) Test your program

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **3(e)(ii)** in the evidence document.

[2]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

10.4 Introduction to Abstract Data Types (ADT)

Name: _____ Class: _____ Date: _____ **Total: 19 marks**

KEY WORDS linked list 链表 • queue 队列 • stack 栈 • pointer 指针 • node 节点

Objective

Build the skills to answer exam questions on **Abstract Data Types** 抽象数据类型 (ADTs) — the stack, queue and linked list, and how they are implemented with arrays.

You must be able to:

- explain that an **ADT** is data **plus** the operations on it
- use a **stack** (LIFO), a **queue** (FIFO) and a **linked list**
- describe how each is implemented using an **array**

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.

What is an ADT

An **ADT** is a **collection of data** together with a **set of operations** on it. You use it through its operations only — **how** it is stored is hidden, so the implementation can change without breaking your code.

Stack — LIFO (Last In, First Out)

- **push** — add to the **top**; **pop** — remove from the **top**.
- Stored in `Stack[1:MaxSize]` with an integer `Top` (0 = empty).
- **Push**: full when `Top = MaxSize` (**overflow**); else `Top ← Top + 1`, `Stack[Top] ← x`.
- **Pop**: empty when `Top = 0` (**underflow**); else return `Stack[Top]`, `Top ← Top - 1`.

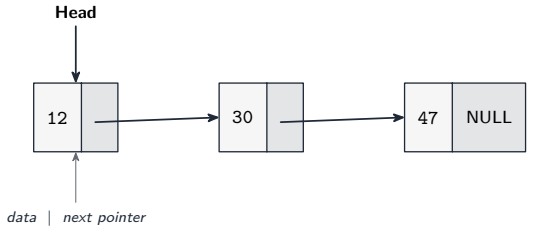
Queue — FIFO (First In, First Out)

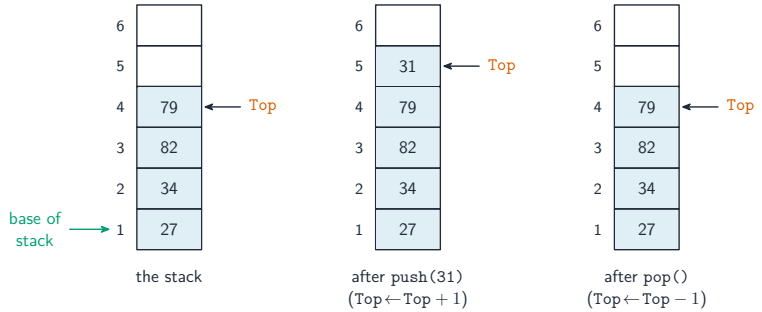
- **enqueue** — add to the **rear**; **dequeue** — remove from the **front**.
- Stored in `Queue[1:MaxSize]` with a `Front` pointer, a `Rear` pointer and a `Count`.
- **Enqueue**: full when `Count = MaxSize`; else `Rear ← (Rear MOD MaxSize) + 1`, `Queue[Rear] ← x`, `Count ← Count + 1`.

- **Dequeue**: empty when `Count = 0`; else return `Queue[Front]`, `Front ← (Front MOD MaxSize) + 1`, `Count ← Count - 1`.
- The `MOD` makes it a **circular array**, so the pointers wrap round instead of running off the end.

Linked list

Each **node** holds a value and a **pointer** to the next node. `Head` marks the first node; the last node's pointer is `NULL`.





A stack is another abstract data type (push/pop)

Traverse — follow the pointers from `Head` to `NULL`:

```
Current ← Head
WHILE Current <> NULL DO
    OUTPUT Current.Value
    Current ← Current.Next
ENDWHILE
```

Advantage over an array: cheap insert/delete (adjust pointers). Disadvantage: slow random access (you must follow pointers from `Head`).

2 Practice

Now apply the methods above.

2.1 State what an **Abstract Data Type** is. [1]

2.2 State the order rule of a **stack** and of a **queue** (use the terms LIFO / FIFO). [2]

2.3 Name the stack operation that **adds** an item and the one that **removes** an item. [2]

2.4 In a linked list, state what the **Head** pointer identifies and what the **last node's** pointer holds. [2]

3 Exam-style questions

3.1 A stack is implemented as `Stack[1:10]` with an integer `Top`.

(i) Write pseudocode for `Push(NewItem)` that checks for **overflow**. [4]

(ii) State what **underflow** means. [1]

3.2 The items `A`, `B`, `C`, `D` are pushed onto an empty stack in that order, then **two** items are popped. State the two items removed (**in order**) and which item is now on top. [3]

3.3 A linked list stores nodes that each have a `Value` and a `Next` pointer.

(i) Describe how to **traverse** the list and output every value. [3]

(ii) State **one** advantage of a linked list over an array. [1]

Solutions

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

2.1

- a **collection of data together with the operations** that can be performed on it (implementation hidden)

2.2

- stack = **LIFO** (Last In, First Out)
- queue = **FIFO** (First In, First Out)

2.3

- add = **push**
- remove = **pop**

2.4

- `Head` identifies the **first node** in the list
- the last node's pointer holds **NULL** (a null/sentinel pointer = end of list)

3.1

(i) test full, else increment `Top` and store

```
IF Top = 10 THEN
    OUTPUT "Stack full (overflow)"
ELSE
    Top ← Top + 1
    Stack[Top] ← NewItem
ENDIF
```

(ii) trying to **pop from an empty stack** (`Top = 0`) — there is nothing to remove

3.2

- popped: **D**, then **C** (LIFO order)
- now on top: **B**

3.3

(i) start at `Head`

- repeatedly output the current node's value and move to the node its `Next` pointer gives
- stop when the pointer is `NULL`

(ii) any one: items can be **inserted/deleted** without shifting others (just change pointers); the list can grow/shrink at run time

3 A program uses a stack to hold up to 30 integer values. The stack is implemented using a global integer variable and a global 1D array.

The array is declared in pseudocode:

```
DECLARE ThisStack : ARRAY[1:30] OF INTEGER
```

Stack design notes:

- The global variable `SP` acts as a stack pointer. `SP` contains the array index of the last value pushed onto the stack.
- If the stack is empty, then `SP` is assigned the value zero.
- The first item added to the stack will be stored in `ThisStack[1]`
- `SP` is incremented each time an item is added to the stack.

A function `Pop()` is written to remove an item from `ThisStack`. The function returns an item of type `PopData` which is defined in pseudocode:

```
TYPE PopData
   DECLARE Data   : INTEGER
   DECLARE Exists : BOOLEAN
ENDTYPE
```

The value removed from the stack is assigned to `Data` and `Exists` is set to `TRUE`. If it is **not** possible to remove a value from the stack, then `Exists` is set to `FALSE`

Complete the pseudocode for `Pop()`

```
FUNCTION Pop() RETURNS PopData

   DECLARE ThisPop : .....

   IF ..... THEN

      PopData.Exists ← ..... // Stack is empty

   ELSE

      PopData.Data ← .....

      PopData.Exists ← .....

      SP ← .....

   ENDIF

   RETURN ThisPop

ENDFUNCTION
```

[5]

9 **(a)** A stack has been implemented using pseudocode to store a maximum of 100 string items using the global variables in the following table:

Identifier	Data type	Description	Initialisation value
`Base`	`INTEGER`	pointer for the bottom of the stack	`0`
`Top`	`INTEGER`	pointer for the top of the stack	`-1`
`StackArray`	`STRING`	1D array to implement the stack	`[0:99]`
`Max`	`INTEGER`	maximum number of items in the stack	`100`

The value of `Top` is incremented each time a data item is added to the stack and decremented every time a data item is removed.

(i) Complete the **pseudocode** for the function to remove a data item from the stack.

```
FUNCTION Pop() .....
   DECLARE DataItem : STRING
   DataItem ← ""

   IF ..... THEN

      DataItem ← .....

      Top ← .....
   ELSE
      DataItem ← "You cannot remove data; the stack is empty"
   ENDIF

   .....
ENDFUNCTION
```

[5]

(ii) Write the **pseudocode** to output the data item removed from the stack with an appropriate message.

.....

..... [1]

(b) A stack is used to implement recursion.

State the **three** essential features of recursion.

1 .....

.....

2 .....

.....

3 .....

.....

[3]

10 Explain what is meant by **exception handling**.
Include an example of a possible cause of an exception in your answer.

Explanation .....

.....

.....

.....

Example .....

[3]

2 A program reads string data from a text file into a linear queue and then compresses the data.

The queue is created using the global 1D array `Queue`. The queue can store up to 100 elements. Each element is initialised to the empty string `""`

The queue has two pointers that are global to the program:

- `QueueHead` that points to the index of the first element in the queue, initialised to `-1`
- `QueueTail` that points to the index of the last element in the queue, initialised to `-1`

The global variable `NumberItems` stores the number of elements stored in the queue, initialised to `0`

(a) Write program code to declare and initialise `Queue`, `QueueHead`, `QueueTail` and `NumberItems`

Save your program as **Question2_N25**.

Copy and paste the program code into part **2(a)** in the evidence document.

[2]

(b) The function `Enqueue()` takes a string as a parameter. The function checks if the queue is full, and returns Boolean `FALSE` if the queue is full.

If the queue is not full, the parameter is inserted into the next position in `Queue`. The function updates the appropriate pointer(s), updates `NumberItems` and then returns Boolean `TRUE`

Write program code for `Enqueue()`

Save your program.

Copy and paste the program code into part **2(b)** in the evidence document.

[5]

(c) The function `Dequeue()` returns the string `"False"` if the queue is empty.

If the queue is not empty, the function returns the next element in the queue. The function updates the appropriate pointer(s) and updates `NumberItems`

Write program code for `Dequeue()`

Save your program.

Copy and paste the program code into part **2(c)** in the evidence document.

[3]

(d) The text file `BinaryData.txt` stores individual binary digits, `'1'` and `'0'`. Each digit is on a new line. For example, the first line in the text file stores `'1'`, the second line stores `'1'`

The procedure `ReadData()` reads in each line from the text file `BinaryData.txt` and inserts it into the queue using the appropriate method.

The procedure needs to work for a text file with any number of lines up to a maximum of 100.

Write program code for `ReadData()`

Save your program.

Copy and paste the program code into part **2(d)** in the evidence document.

[5]

(e) The string data in the text file is compressed.

The compression algorithm counts the number of times each binary digit appears consecutively, then stores the binary digit followed by the number of times it appears. The algorithm stores the compressed data in a single string.

For example, if the text file contains the data:

```
1
1
0
0
0
1
1
1
```

The compression algorithm will create the string `"120313"` because there are two `'1'` digits, followed by three `'0'` digits, followed by three `'1'` digits.

The procedure `Compress()` uses `Dequeue()` to remove each element from the queue in turn. The procedure then compresses the data following the compression algorithm described. The new compressed string is stored in the global variable `NewString`

You can assume that one binary digit will never appear more than nine times consecutively in the sequence.

You can assume that there will always be at least one item in the queue.

Write program code for `Compress()`

Save your program.

Copy and paste the program code into part **2(e)** in the evidence document.

[6]

(f) **(i)** Write program code for the main program to:

- call `ReadData()`
- call `Compress()`
- output the content of the compressed string.

Save your program.

Copy and paste the program code into part **2(f)(i)** in the evidence document.

[2]

(ii) Test your program.

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot(s) into part **2(f)(ii)** in the evidence document.

[1]

5 Stacks and queues are both abstract data types.

A stack uses a top-of-stack pointer to indicate the location of the last item added to the stack.

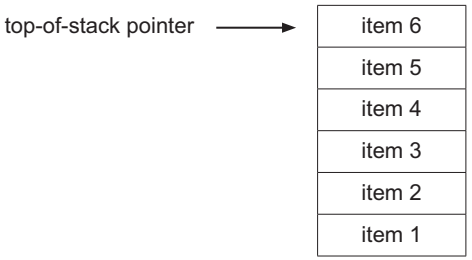
A queue uses two pointers:

- a front pointer to indicate the location of the next item to be removed from the queue
- a rear pointer to indicate the location of the next item to be added to the queue.

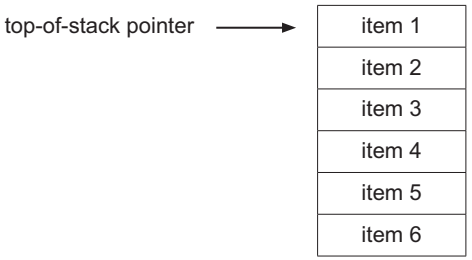
A queue can be used to reverse the items stored on a stack.

For example, if a stack contains six items:

Initial state of the stack:



Final state of the stack when the items have been reversed:



Describe how the queue could be used to reverse the items that are currently stored on the stack.

Your description must include how the pointers are used in both the stack and queue.

Assume:

- The stack initially contains an unknown number of items.
- The queue can store all the items currently stored on the stack.
- The queue is initially empty.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

2 A programmer uses a number of Abstract Data Types (ADT) in the programs that she is developing.

(a) A stack using memory locations 400 to 409 is used in one program.

The diagram represents the current state of the stack.

A variable `TopOfStack` pointer indicates the last value added to the stack.

Stack

Memory location	Value	
409		
408		
407		
406		
405		
404		
403	'P'	← `TopOfStack`
402	'X'	
401	'D'	
400	'B'	

(i) Complete the answer column in the following table:

	Answer
the memory location of the value that has been on the stack for the longest time	
the number of consecutive **push** operations that will result in the `TopOfStack` variable containing 409	

[2]

(ii) The diagram shows the current state of the stack.

Stack

Memory location	Value	
409		
408		
407	'A'	← `TopOfStack`
406	'C'	
405	'F'	
404	'K'	
403	'B'	
402	'S'	
401	'R'	
400	'D'	

The following sequence of operations are performed:

```
PUSH 'T'
POP
POP
PUSH 'Z'
PUSH 'X'
POP
PUSH 'Y'
```

Complete the following diagram to show the state of the stack after the operations have been performed.

Stack

Memory location	Value
409	
408	
407	
406	
405	
404	
403	
402	
401	
400	

[3]

(b) In a different program, the programmer decides to implement a linked list using an array of records.

(i) The array is represented in the diagram.

A pointer value of `-1` indicates the end of the linked list.

Complete the pointer field to produce a linked list that is in alphabetical order.

Index	Data	Pointer
0	"Neptune"	
1	"Jupiter"	
2	"Saturn"	
3	"Earth"	
4	"Mercury"	
5	"Uranus"	

[3]

(ii) The variable `StartPointer` contains the index of the first item in the linked list.

State the value of the variable `StartPointer`

..... [1]

(c) A third program is also being created to manage a queue.

Describe **two** features of a queue.

Feature one .....

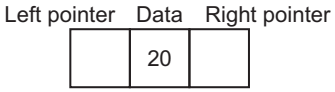
.....

Feature two .....

.....

[2]

4 **(a)** A linked list of nodes is used to store an ordered list of integers. Each node consists of the data, a left pointer and a right pointer, for example:

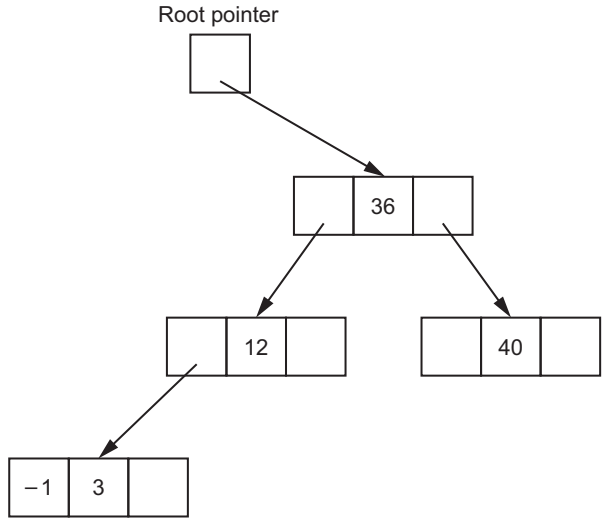


The linked list will be organised as a binary tree.

−1 is used to represent a null pointer.

Complete the binary tree, including null pointers, to show how the data will be organised after the following integers have been added:

6, 15, 41, 66



[4]

(b) Describe what is meant by recursion.

.....

.....

.....

..... [2]

(c) A binary tree is a suitable Abstract Data Type (ADT) that a designer can implement using recursive algorithms.

Identify **one other** ADT that a designer can implement using recursive algorithms.

..... [1]

3 A program uses a stack to hold up to 60 numeric values.
The stack is implemented using two integer variables and a 1D array.

The array is declared in pseudocode as shown:

```
DECLARE ThisStack : ARRAY[1:60] OF REAL
```

The stack operates as follows:

- Global variable `SP` acts as a stack pointer that points to the next available stack location. The value of `SP` represents an array index.
- Global variable `OnStack` represents the number of values currently on the stack.
- The stack grows upwards from array element index 1.

(a) **(i)** Give the initial values that should be assigned to the **two** variables.

`SP` .....

`OnStack` .....
[1]

(ii) Explain why it is **not** necessary to initialise the array elements before the stack is used.

.....

.....

.....

..... [2]

(b) A function to add a value to `ThisStack` is expressed in pseudocode as shown.
The function will return a value to indicate whether the operation was successful or not.

Complete the pseudocode by filling in the gaps.

```
FUNCTION Push(ThisValue : REAL) RETURNS BOOLEAN
   DECLARE ReturnValue : BOOLEAN
   IF ..... THEN
      RETURN ..... // stack is already full
   ENDIF
   ..... ← ThisValue
   SP ← .....
   OnStack ← OnStack + 1
   RETURN TRUE
ENDFUNCTION
```

[4]

3 The implementation of a linked list uses an integer variable and a 1D array `List` of type `Node`.

Record type `Node` is declared in pseudocode as follows:

```
TYPE Node
   DECLARE Data : STRING
   DECLARE Pointer : INTEGER
ENDTYPE
```

The array `List` is declared in pseudocode as follows:

```
DECLARE List : ARRAY[1:200] OF Node
```

The linked list will operate as follows:

- Integer variable `HeadPointer` will store the array index for the first node in the linked list.
- The `Pointer` field of a node contains the index value of the next array element (the next node) in the linked list.
- The value 0 is used as a null pointer.

(a) (i) State why the value 0 has been selected as the null pointer.

.....

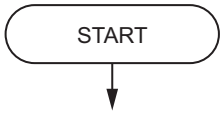
..... [1]

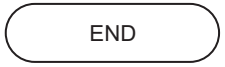
(ii) Give the range of valid values that could be assigned to variable `HeadPointer`.

..... [1]

(b) The array `List` will be initialised so that each node points to the following node. The last node will contain a null pointer.

Complete the program flowchart to represent the algorithm for this operation.





[4]

(c) An algorithm outputs the `Data` field from all nodes in the array `List`. The order the `Data` is output should be the same order it is stored in the linked list.

Describe the algorithm in **four** steps.

Do **not** use pseudocode statements in your answer.

Step 1 .....

.....

.....

Step 2 .....

.....

.....

Step 3 .....

.....

.....

Step 4 .....

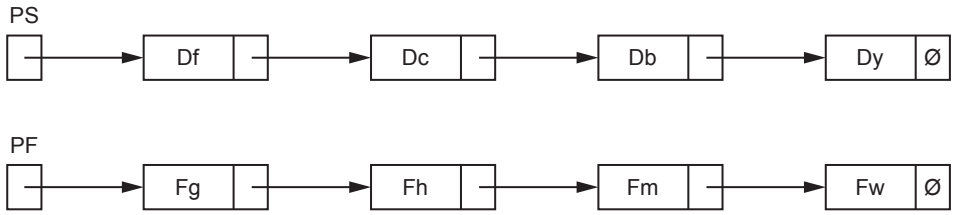
.....

.....

[4]

3 The diagram shows an Abstract Data Type (ADT) representation of a linked list after data items have been added.

- PS is the start pointer.
- PF is the free list pointer.
- Labels Df, Dc, Db and Dy represent the data items of nodes in the list.
- Labels Fg, Fh, Fm and Fw represent the data items of nodes in the free list.
- The symbol Ø represents a null pointer.



(a) Describe the linked list immediately after initialisation, before **any** data items are added.

.....

.....

.....

.....

.....

..... [3]

(b) A program will be written to include a linked list to store alphanumeric user IDs.

The design uses two variables and two 1D arrays to implement the linked list.
Each array element contains data of a single data type and **not** a record.

The statements below describe the design.

Complete the statements.

The two variables will be of type ..... .

The two variables will be used as ..... to the arrays.

The values stored in the two variables will indicate .....

..... .

The first 1D array will be of type ..... .

The first 1D array will be used to ..... .

The second 1D array will be of type ..... .

The second 1D array will be used to ..... .

[5]