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
```

(1)

[6]

(b) The requirements of the program change:

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

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

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

1 .....

.....

2 .....

.....
[2]

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

.....

..... [1]

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

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

For example:

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

Write pseudocode for the function `Compare()`

[7]

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

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

The data items are:

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

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

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

The programmer has defined a program module:

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

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

[7]

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

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

A new module is defined:

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

Write efficient pseudocode for `NewLoan()`

.... [7]

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

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

Explain how these records can be saved for future reference.

.....

.....

.....

..... [2]

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

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

Data .....

.....

.....

.....

Use .....

.....

.....

.....

[2]

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

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

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

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

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

For example, the:

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

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

Module	Description

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

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

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

`<CustomerID>,<Points>`

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

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

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

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

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

`"100205,0"`

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

The programmer has defined a program module as follows:

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

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

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

.....

.....

.....

.....

.....

.....

.....

[8]

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

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

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

The module will:

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

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

step 3 – output the integer generated

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

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

step 6 – return the average value.

Write pseudocode for the module `RollDice()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

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

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

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

The modified value of `BitString` is then returned.

For example:

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

Write pseudocode for the module `Parity()`

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

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

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

Assume:

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

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

Complete the steps.

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

[4]

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

.....

.....

..... [1]

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

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

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

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

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

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

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

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

```
DECLARE Start : INTEGER

OUTPUT "Hello"
Start ← Tick

REPEAT
   //do nothing
UNTIL Tick = Start + 40000

OUTPUT "Goodbye"
```

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

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

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

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

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

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

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

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

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

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

Write pseudocode for the procedure `Timer()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

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

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

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

The array is declared in pseudocode as follows:

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

Each element contains the name of one filling.

An example of the first five elements is as follows:

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

A second 1D array stores the possible bread used:

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

Each element contains the name of one type of bread.

An example of the first three elements is as follows:

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

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

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

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

Using the above examples, the output could be:

```
"The daily special is Cheese and Onion on Brown bread."
```

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

Assume that both arrays are global.

```
PROCEDURE Special()
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[7]

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

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

.....

.....

.....

.....

.....

.....

.....

..... [2]

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

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

Each character has four attributes, as follows:

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

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

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

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

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

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

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

The programmer has defined a program module as follows:

Module	Description

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

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

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```


[7]

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

New modules are defined:

Module	Description

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

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

As a reminder, the record structure is repeated here:

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

Write pseudocode for module `Restore()`.

You must use the module `Extract()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

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

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

.....

.....

.....

..... [2]

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

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

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

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

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

.....

..... [1]

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

Item 1 required .....

Justification .....

.....

Item 2 required .....

Justification .....

.....

Item 3 required .....

Justification .....

.....

[3]

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

Operation 1 .....

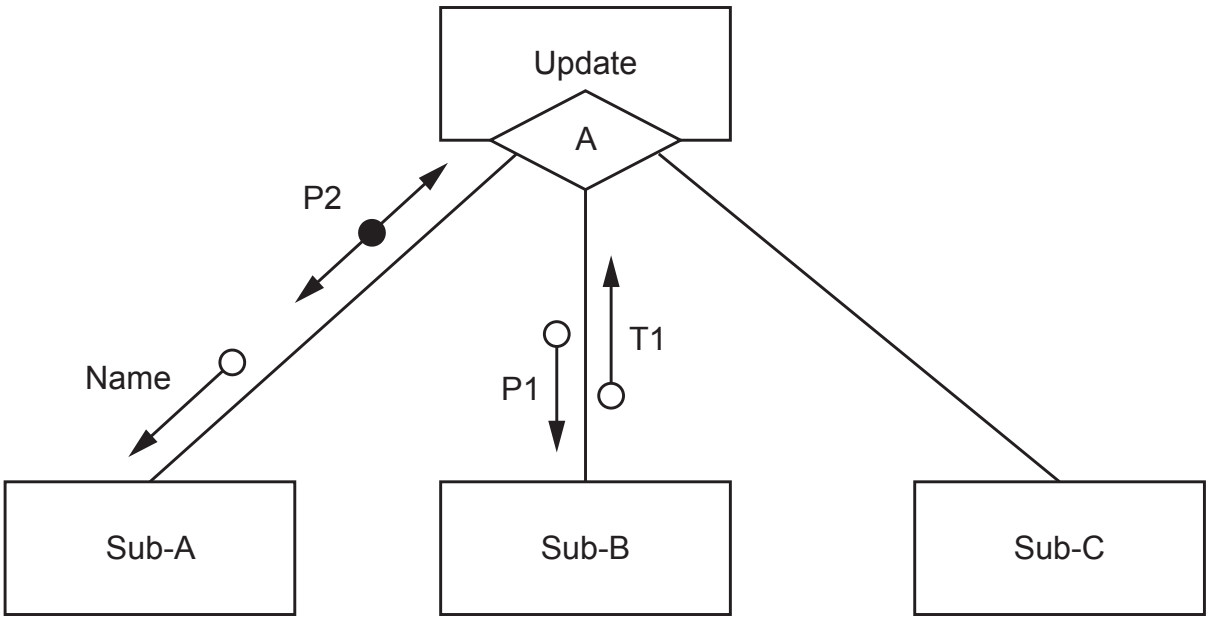
.....

Operation 2 .....

.....

[2]

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



Data item notes:

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

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

.....

.....

.....

..... [2]

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

Sub-A

.....

.....

Sub-B

.....

.....

[4]

7 A fitness club has a computerised membership system.

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

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

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

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

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

Sub-module 1 .....

Use .....

.....

.....

.....

Sub-module 2 .....

Use .....

.....

.....

.....

Sub-module 3 .....

Use .....

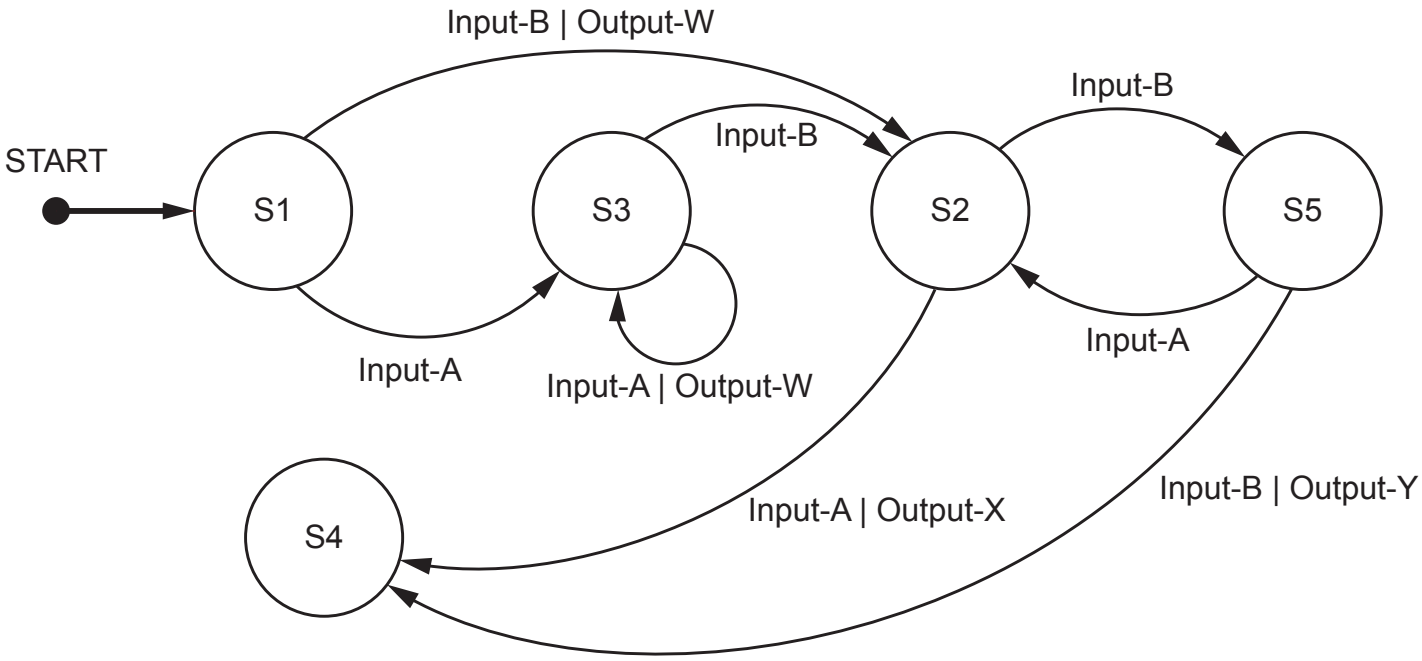
.....

.....

.....

[3]

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



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

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

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

The first two rows have been completed.

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

[5]

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

..... [1]