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]

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]