Cambridge Assessment International Education

Cambridge International AS & A Level

CANDIDATE NAME	

CENTRE NUMBER						CANDIDATE NUMBER					

5904136477

COMPUTER SCIENCE **9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 2022**

2 hours

You must answer on the question paper.

You will need: Insert (enclosed)

INSTRUCTIONS
- Answer **all** questions.
- Use a black or dark blue pen.
- Write your name, centre number and candidate number in the boxes at the top of the page.
- Write your answer to each question in the space provided.
- Do **not** use an erasable pen or correction fluid.
- Do **not** write on any bar codes.
- You may use an HB pencil for any diagrams, graphs or rough working.
- Calculators must **not** be used in this paper.

INFORMATION
- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [ ].
- No marks will be awarded for using brand names of software packages or hardware.
- The insert contains all the resources referred to in the questions.

This document has **20** pages. Any blank pages are indicated.

[Turn over

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

1 **(a)** A programmer is developing an algorithm to solve a problem. Part of the algorithm would be appropriate to implement as a subroutine (a procedure or a function).

(i) State **two** reasons why the programmer may decide to use a subroutine.

1 ....

2 ....

- When a task which is repeated / reused / performed in several places
- When a part of an algorithm performs a specific task
- Reduces complexity of program / program is simplified // subroutine already available
- Testing / debugging / maintenance is easier

[2]

(ii) A procedure header is shown in pseudocode:

```
PROCEDURE MyProc(Count : INTEGER, Message : STRING)
```

Give the correct term for the identifiers `Count` and `Message` **and** explain their use.

Term .... Parameter(s) ....

Use .... to pass values / arguments to the procedure ....

[2]

(b) The algorithm in **part (a)** is part of a program that will be sold to the public.
All the software errors that were identified during in-house testing have been corrected.

Identify **and** describe the additional test stage that may be carried out before the program is sold to the public.

Test stage .... Beta testing ....

Description ....

1 Testing carried out by a small group of (potential) users
2 Users will check that the software works as required / works in the real world / does not contain errors
3 Users will feedback problems / suggestions for improvement
4 Problems / suggestions identified are addressed (before the program is sold)

[4]

(c) Part of an identifier table is shown:

Variable	**Type**	**Example value**
`FlagDay`	`DATE`	`23/04/2004`
`CharList`	`STRING`	`"ABCDEF"`
`Count`	`INTEGER`	`29`

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

Expression	**Evaluation**
`MID(CharList, MONTH(FlagDay), 1)`	'D'
`INT(Count / LENGTH(CharList))`	4
`(Count >= 29) AND (DAY(FlagDay) > 23)`	FALSE

[3]

2 **(a)** An algorithm will process data from a test taken by a group of students. The algorithm will prompt and input the name and test mark for each of the 35 students.

The algorithm will add the names of all the students with a test mark of less than 20 to an existing text file `Support_List.txt`, which already contains data from other group tests.

(i) Describe the steps that the algorithm should perform.

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

1 Open file in APPEND mode (and subsequent Close)
2 Prompt and Input a student name and mark
3 If mark greater than or equal to 20 jump to step 5
4 Write only the name to the file
5 Repeat from Step 2 for 35 times / the number of students

[5]

(ii) Explain why it may be better to store the names of the students in a file rather than in an array.

Data in a file is saved after the computer is switched off / stored permanently // no need to re-enter the data when the program is re-run

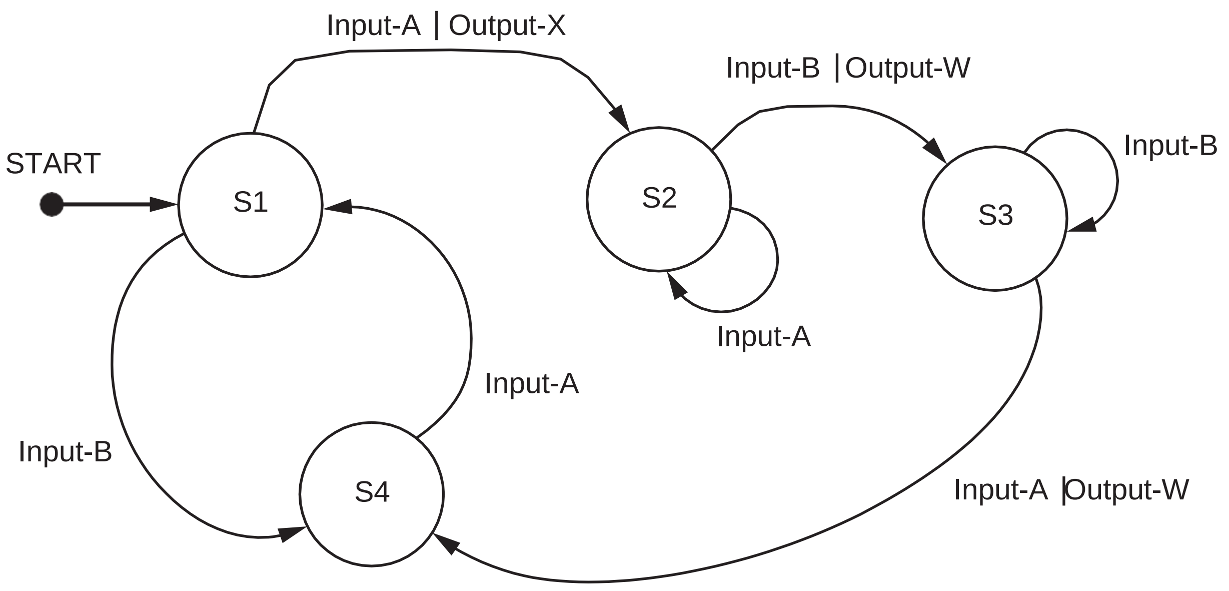
[1]

(iii) Explain why `WRITE` mode cannot be used in the answer to **part 2(a)(i)**.

So that existing file data is not overwritten.

[1]

(b) Examine the following state-transition diagram.



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

Input	Output	Next state
▨	▨	S1
Input-A	**Output-X**	**S2**
Input-A	**(none)**	S2
Input-B	Output-W	**S3**
Input-A	Output-W	**S4**

[4]

3 A stack is used in a program to store string data which needs to be accessed in several modules.

(a) A stack is an example of an Abstract Data Type (ADT).

Identify **one other** example of an ADT **and** describe its main features.

Name: Queue

Features:
1 Each queue element contains one data item
2 A Pointer to the front / start of the queue
3 A Pointer to the back / end of the queue
4 Data is added at back / end and removed from front / start // works on a FIFO basis
5 May be circular

Name: Linked List

Features:
1 Each node contains data and a pointer to the next node
2 A Pointer to the start of the list
3 Last node in the list has a null pointer
4 Data may be added / removed by manipulating pointers (not moving data)
5 Nodes are traversed in a specific sequence
6 Unused nodes are stored on a free list // a free-list pointer to the Free List

[3]

(b) Explain how the stack can be implemented using an array.

1 Declare a (1D) array of data type `STRING`
2 The number of elements in that array corresponds to the size of the required stack
3 Declare an integer / variable for `StackPointer`
4. Declare an integer / variable for the size of the stack // for the max value of `StackPointer`
5 Use the `StackPointer` as an index to the array
6 Pointers and variables initialised to indicate empty stack
7 Store each item on the stack as one array element / Each stack item maps to one array element
8 Attempt to describe Push **and** Pop operations
9 Push **and** Pop routines need to check for full or empty conditions

[5]

(c) A second stack is used in the program. The diagram below shows the initial state of this stack. Value X is at the top of the stack and was the last item added.

Upper-case letters are used to represent different data values.

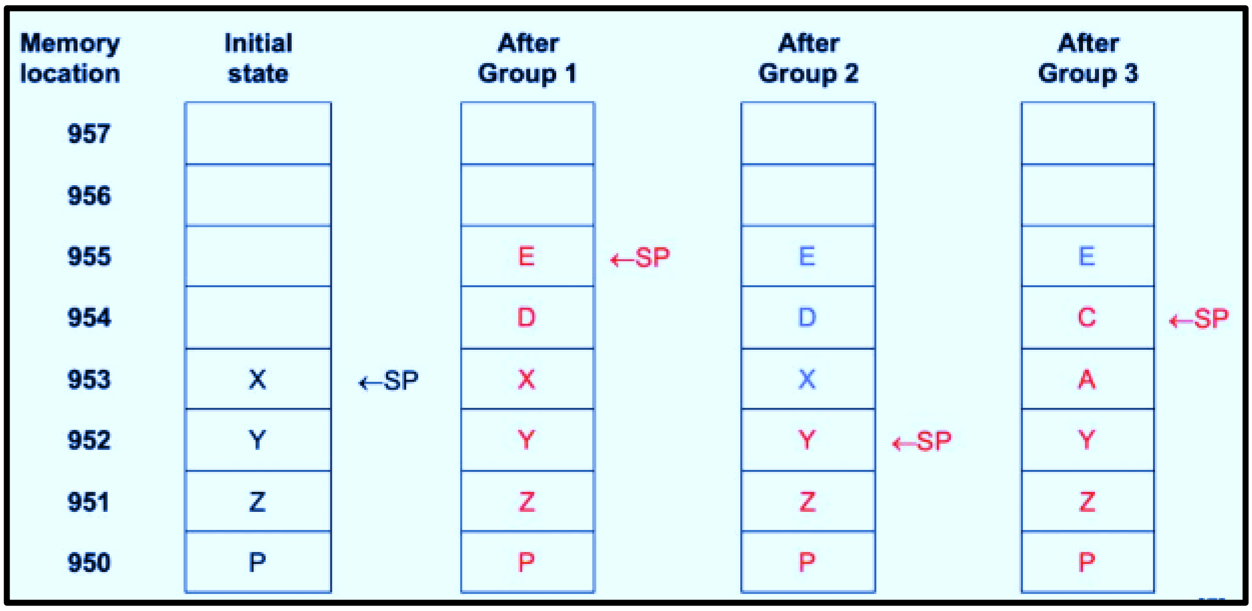
Stack operations are performed in three groups as follows:

```
Group 1:
                PUSH D
                PUSH E

Group 2:
                POP
                POP
                POP

Group 3:
                PUSH A
                PUSH B
                POP PUSH
                C
```

Complete the diagram to show the state of the stack **after** each group of operations has been performed.

Include the current stack pointer (SP) **after** each group. [5]

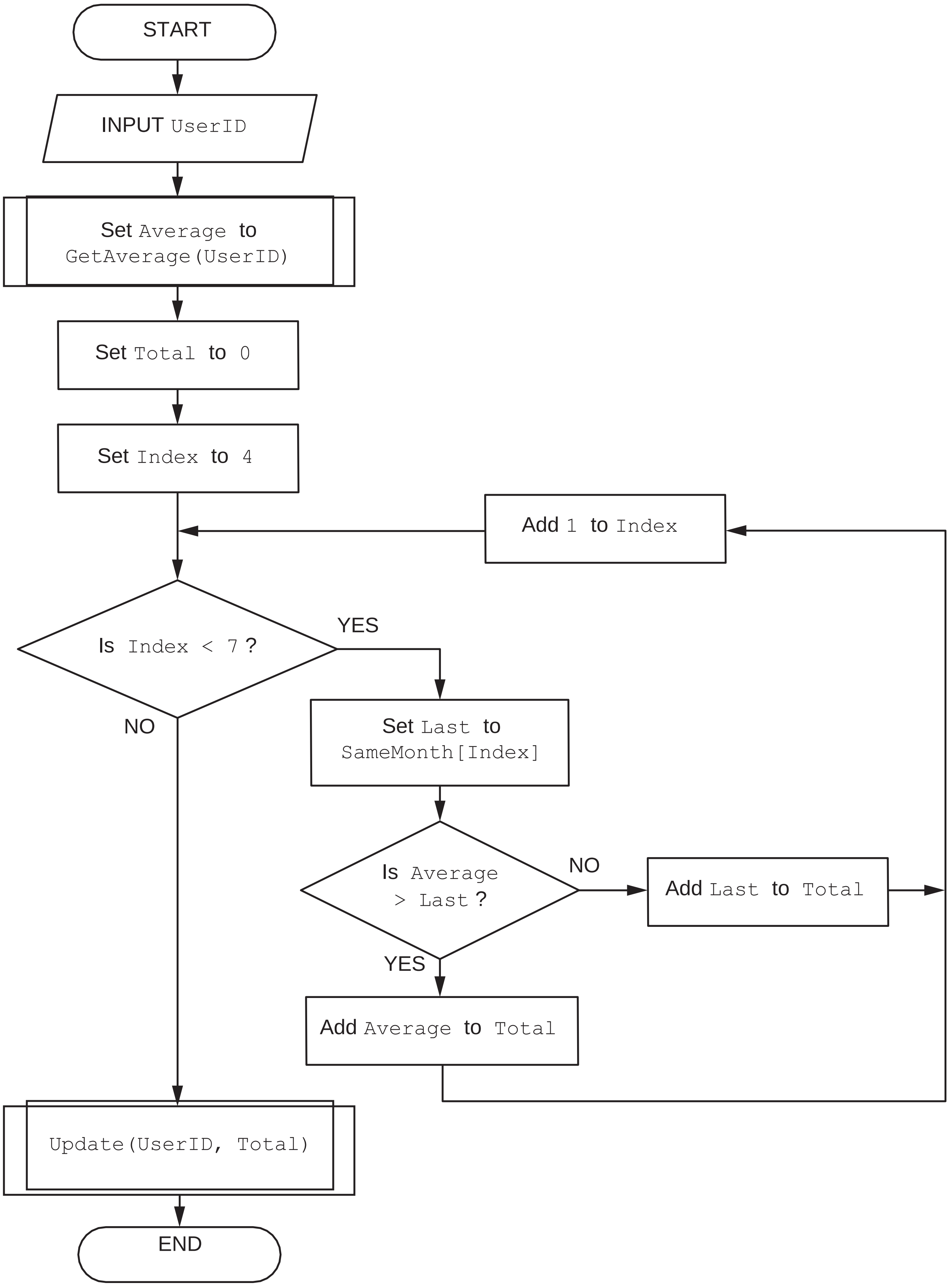
Memory location	Initial state		After Group 1	After Group 2	After Group 3
957					
956					
955					
954					
953	X	←SP			
952	Y				
951	Z				
950	P				



Memory location	Initial state		After Group 1		After Group 2		After Group 3	
957								
956								
955			E	←SP	E		E	
954			D		D		C	←SP
953	X	←SP	X		X		A	
952	Y		Y		Y	←SP	Y	
951	Z		Z		Z		Z	
950	P		P		P		P	

BLANK PAGE

4 The program flowchart represents a simple algorithm.



(a) Write the equivalent pseudocode for the algorithm represented by the flowchart.

```
INPUT UserID
Average ← GetAverage(UserID)
Total ← 0
Index ← 4

WHILE Index < 7 // REPEAT
   Last ← SameMonth[Index]
   IF Average > Last THEN
      Total ← Total + Average
   ELSE
      Total ← Total + Last
   ENDIF
   Index ← Index + 1
ENDWHILE // UNTIL Index = 7

CALL Update(UserID, Total)
```

```
INPUT UserID
Average ← GetAverage(UserID)
Total ← 0
FOR Index ← 4 TO 6
   Last ← SameMonth[Index]
   IF Average > Last THEN
      Total ← Total + Average
   ELSE
      Total ← Total + Last
   ENDIF
NEXT Index

CALL Update(UserID, Total)
```

[6]

(b) Give the name of the iterative construct in the flowchart.

Pre-condition (loop) / count-controlled (loop) [1]

5 Examine the following pseudocode.

```
IF A = TRUE THEN
   IF B = TRUE THEN
      IF C = TRUE THEN
         CALL Sub1()
      ELSE
         CALL Sub2()
      ENDIF
   ENDIF
ELSE
   IF B = TRUE THEN
      IF C = TRUE THEN
        CALL Sub4()
      ELSE
        CALL Sub3()
      ENDIF
   ELSE
      IF C = FALSE THEN
        CALL Sub3()
      ELSE
        CALL Sub4()
      ENDIF
   ENDIF
ENDIF
```

A programmer wants to re-write the pseudocode as **four** separate `IF...THEN...ENDIF` statements, each containing a single `CALL` statement. This involves writing a single, simplified logic expression as the condition in each statement.

Write the amended pseudocode.

```
1  IF A AND B AND C THEN
      CALL Sub1()
   ENDIF

2  IF (A AND B) AND NOT C THEN
      CALL Sub2()
   ENDIF

3  IF (NOT A) AND (NOT C) THEN
      CALL Sub3()
   ENDIF

4  IF (NOT A) AND C THEN
      CALL Sub4()
   ENDIF
```

1

2

3

4

[4]

BLANK PAGE

6 **(a)** The factorial of an integer number is the product of all the integers from that number down to 1.

In general, the factorial of n is n × (n−1) × ... × 2 × 1

For example, the factorial of 5 is 5 × 4 × 3 × 2 × 1 = 120

In this question, n will be referred to as the `BaseNumber`.

A function `FindBaseNumber()` will:

- be called with a positive, non-zero integer value as a parameter
- return `BaseNumber` if the parameter value is the factorial of the `BaseNumber`
- return −1 if the parameter value **is not** a factorial.

For example:

Parameter value	**Value returned**
120	5
12	−1
6	3
1	1

`FindBaseNumber(12)` will return −1 because 12 is not a factorial.You

may use the rest of this page for rough working.

Write pseudocode for the function `FindBaseNumber()`.

```
FUNCTION FindBaseNumber(ThisValue : INTEGER) RETURNS
INTEGER
   DECLARE Num, Try : INTEGER
   DECLARE Found : BOOLEAN

   Num ← 0
   Found ← FALSE
   Try ← 1

    WHILE Try <= ThisValue AND Found = FALSE
       Num ← Num + 1
       Try ← Try * Num
       IF Try = ThisValue THEN //BaseNumber found
          Found ← TRUE
       ENDIF
    ENDWHILE

    IF Found = TRUE THEN
       RETURN Num
    ELSE
       RETURN -1
    ENDIF

ENDFUNCTION
```

[7]

(b) A program is written to allow a user to input a sequence of values to be checked using the function `FindBaseNumber()`.

The user will input one value at a time. The variable used to store the user input has to be of type string because the user will input 'End' to end the program.

Valid input will be converted to an integer and passed to `FindBaseNumber()` and the return value will be output.

Complete the table by giving **four** invalid strings that may be used to test distinct aspects of the required validation. Give the reason for your choice in each case. [4]

Input	Reason for choice

Input	Reason for choice
"Aardvark"	Non-numeric (and not "End")
"27.3"	Numeric but not an integer
"-3" // "0"	A non-positive integer
""	An empty string

BLANK PAGE

7 A teacher is designing a program to perform simple syntax checks on programs written by students.

Two global 1D arrays are used to store the syntax error data. Both arrays contain 500 elements.

- Array `ErrCode` contains integer values that represent an error number in the range 1 to 800.
- Array `ErrText` contains string values that represent an error description.

The following diagram shows an example of the arrays.

Index	**`ErrCode`**	**`ErrText`**
1	`10`	`"Invalid identifier name"`
2	`20`	`"Bracket mismatch"`
3	`50`	`"Undeclared variable"`
4	`60`	`"Type mismatch in assignment"`
…		
500	`999`	`<Undefined>`

Note:

- There may be less than 500 error numbers so corresponding elements in both arrays may be unused. Unused elements in `ErrCode` have the value `999`. The value of unused elements in `ErrText` is undefined.
- Values in the `ErrCode` array are stored in ascending order but not all values may be present, for example, there may be no error code `31`.

The teacher has defined two program modules as follows:

Module	**Description**

`OutputError()`	• takes two parameters as integers:
	o a line number in the student's program
	o an error number
	• searches for the error number in the `ErrCode` array:
	o if found, outputs the corresponding error description and the line number, for example:
	`"Bracket mismatch on line 34"`
	o if not found, outputs the line number and a warning, for example:
	`"Unknown error on line 34"`

`SortArrays()`	sorts the arrays into ascending order of `ErrCode`

(a) Write **efficient** pseudocode for module `OutputError()`.

```
PROCEDURE OutputError(LineNum, ErrNum : INTEGER)
  DECLARE Index : INTEGER

  Index ← 0

  // Search until ErrNum found OR not present OR end of array

  REPEAT
     Index ← Index + 1
  UNTIL ErrCode[Index] >= ErrNum OR Index = 500

  IF ErrCode[Index] = ErrNum THEN
     OUTPUT ErrText[Index], " on line ", LineNum //Found
  ELSE
     OUTPUT "Unknown error on line ", LineNum     //Not found
  ENDIF

ENDPROCEURE
```

[6]

(b) Write an **efficient** bubble sort algorithm in pseudocode for module `SortArrays()`.

```
PROCEDURE SortArrays()
   DECLARE TempInt, J, Boundary : INTEGER
   DECLARE TempStr : STRING
   DECLARE NoSwaps : BOOLEAN

   Boundary ← 499

   REPEAT
      NoSwaps ← TRUE
      FOR J ← 1 TO Boundary
         IF ErrCode[J]> ErrCode[J+1] THEN
            //first swap ErrCode elements
            TempInt ← ErrCode[J]
            ErrCode[J] ← ErrCode[J+1]
            ErrCode[J+1] ← TempInt
            //now swap corresponding ErrText elements
            TempStr ← ErrText[J]
            ErrText[J] ← ErrText[J+1]
            ErrText[J+1] ← TempStr
            NoSwaps ← FALSE
         ENDIF
      NEXT J
      Boundary ← Boundary - 1
   UNTIL NoSwaps = TRUE

ENDPROCEDURE
```

[8]

(c) Two 1D arrays were described at the beginning of the question. Both arrays contain 500 elements.

- Array `ErrCode` contains integer values that represent an error number in the range 1 to 800.
- Array `ErrText` contains string values that represent an error description.

The two arrays will be replaced by a single array. A user-defined data type (record structure) has been declared as follows:

```
TYPE ErrorRec
   DECLARE ErrCode : STRING
   DECLARE ErrText : STRING
ENDTYPE
```

(i) State the error in the record declaration.

`ErrCode` should be an `INTEGER` // `ErrCode` should not be a `STRING`

[1]

(ii) State **two** benefits of using the single array of the user-defined data type.

1 Array of records can store mixed data types / multiple data types under a single identifer
2 Tighter / closer association between `ErrCode` and `ErrText` // simpler code as fields may be referenced together // values cannot get out of step as with two arrays
3 Program easier to design / write / debug / test / maintain / understand

[2]

(iii) Write the declaration for the single array in pseudocode.

```
DECLARE Error : ARRAY[1:500] OF ErrorRec
```

[1]

BLANK PAGE