Cambridge Assessment International Education

Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER | | | | | |

CANDIDATE NUMBER | | | | |

7344032364

COMPUTER SCIENCE **9618/21**

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.

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

1 **(a)** An algorithm includes a number of complex calculations. A programmer is writing a program to implement the algorithm and decides to use library routines to provide part of the solution.

State **three** possible benefits of using library routines in the development of the program.

1 .....

1 They are tried and tested so free from errors
2 They perform a function that you may not be able to program yourself (for example **encryption**)
3 They are readily available / speed up development time

2 .....

3 .....

.....

[3]

(b) The following pseudocode is part of a program that stores names and test marks for use in other parts of the program.

```
DECLARE Name1, Name2, Name3 : STRING
DECLARE Mark1, Mark2, Mark3 : INTEGER
INPUT Name1
INPUT Mark1
INPUT Name2
INPUT Mark2
INPUT Name3
INPUT Mark3
```

(i) The pseudocode needs to be changed to allow for data to be stored for up to 30 students.

Explain why it would be good practice to use arrays to store the data.

1 Algorithm to process / search / organise the data is easier to implement // Values may be accessed via a loop-controlled variable / iterated through using index
2 Makes the program easier to design / code / test / understand
3 Multiple instances referenced via a single identifier / so fewer identifiers needed // Easier to amend the program when the number of students increases

.....

.....

.....

.....

.....

.....

.....

[3]

(ii) The following pseudocode statement includes array references:

```
OUTPUT "Student ", Name[Count], " scored ", Mark[Count]
```

State the purpose of the variable `Count` and give its data type.

Purpose ..... **Purpose**: It identifies / references an individual array element // provides the index to the array

Data type Integer .....

[2]

(c) The pseudocode statements in the following table may contain errors.

State the error in each case or write 'NO ERROR' if the statement contains no error.

Assume that any variables used are of the correct type for the given function.

Statement	Error
`IF EMPTY ← "" THEN`	Should be "=" not ←
`Status ← IS_NUM(-23.4)`	Parameter should be a string (or char) // should not be a real
`X ← STR_TO_NUM("37") + 5`	NO ERROR
`Y ← STR_TO_NUM("37" + "5")`	Wrong operator – should be & or Parameter is not a string

[4]

2 A system is being developed to help manage a car hire business. A customer may hire a car for a number of days.

An abstract model needs to be produced.

(a) Explain the process of abstraction **and** state **four** items of data that should be stored each time a car is hired.

Explanation .....
- Abstraction is used to filter out information / data that is not necessary for the task

.....
- To keep only information / data that is necessary for the task

Item 1 .....

Item 2 .....
- Car details: ID, Car Registration, car type etc
- Customer details: ID, name, address, licence details etc
- Start date (of hire)
- Return date / Number of days (of hire)
- Cost of hire

Item 3 .....

Item 4 .....

[3]

(b) Identify **two** operations that would be required to process the car hire data.

Operation 1 .....
1 Input customer details
2 Input car details
3 Input payment details
4 Create hire / start hire
5 Return car / end hire
6 Change / check car status (hired / available / written-off)
7 Cancel hire
8 Process payment / calculate hire cost

.....

Operation 2 .....

.....

[2]

3 A 1D array `Data` of type integer contains 200 elements. Each element has a unique value.

An algorithm is required to search for the largest value and output it.

Describe the steps that the algorithm should perform.

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

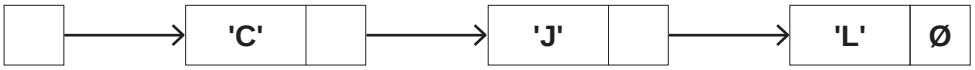
1 Declare a variable / an integer `Max`
2 Assign value of first / any element to `Max`
3 Set up a loop to repeat 200 times / from start to end of array
4 Use the loop counter as the array index
5 If value of current element is greater than `Max`...
6 ...then assign value to `Max`
7 After the loop, Output `Max`

[5]

4 **(a)** The following diagram shows an Abstract Data Type (ADT) representation of an ordered linked list. The data item stored in each node is a single character. The data will be accessed in alphabetical order.

The symbol Ø represents a null pointer.

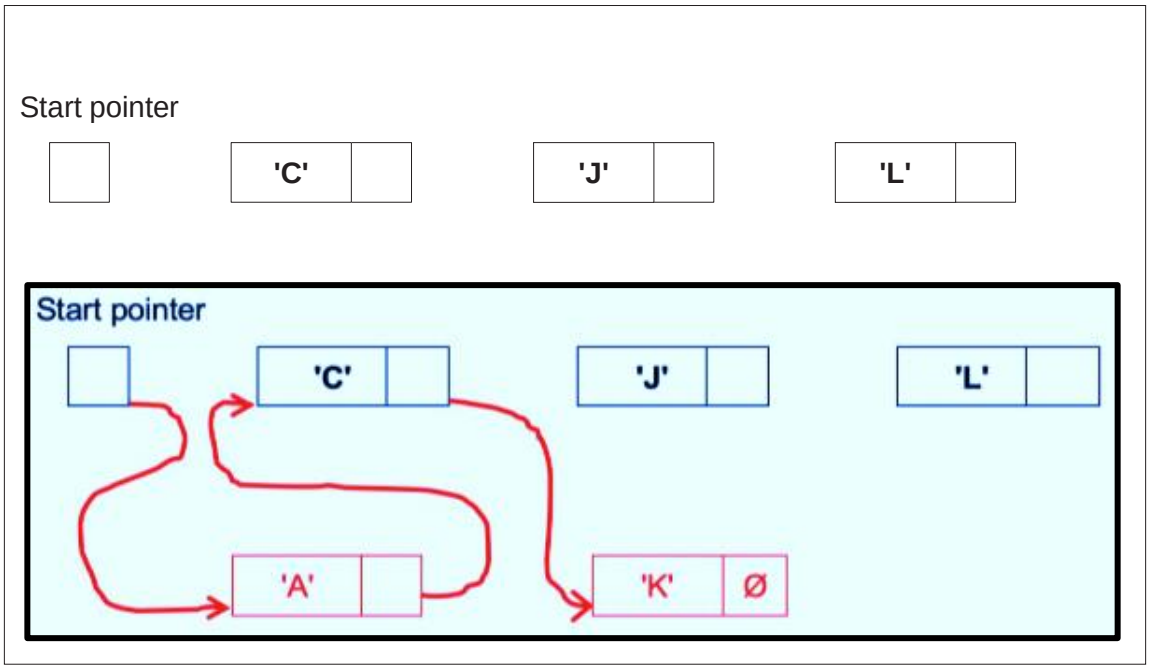
Start pointer



(i) Nodes with data 'A' and 'K' are added to the linked list. Nodes with data 'J' and 'L' are deleted.

After the changes, the data items still need to be accessed in alphabetical order.

Complete the diagram to show the new state of the linked list.



[4]

(ii) The original data could have been stored in a 1D array in which each element stores a character.

For example:

'C'	'J'	'L'		

Explain the advantages of making the changes described in **part (a)(i)** when the data is stored in the linked list instead of an array.

1 Pointers determine the ordering of data // only the pointers need to be changed when data changed

2 Easier to add / delete data (to maintain correct sequence) in a linked list // description of moving data to maintain correct sequence when array used

[2]

(iii) Explain the disadvantages of making the changes described in **part (a)(i)** when the data is stored in the linked list instead of an array.

1 Need to store pointers as well as data
2 More complex (to setup / implement)

[2]

(b) A program will store data using a linked list like the one shown in **part (a)**.

Explain how the linked list can be implemented.

1 Declare **two** (1D) arrays
2 One for data, one for pointers
3 Elements from same index represent one node
4 Declare an integer / variable for `StartPointer` // explain its use
5 Define appropriate value for null pointer // explain its use
6 Declare an integer / variable for `FreeList` pointer // explain its use
7 Routines are needed to add / delete / search

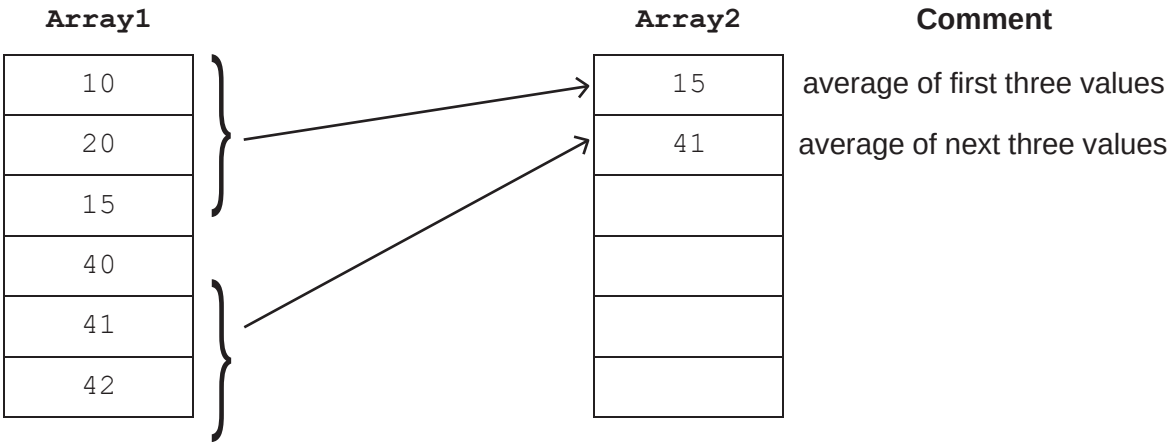
[4]

5 A program uses two 1D arrays of type integer. `Array1` contains 600 elements and `Array2` contains 200 elements.

`Array1` contains sample values read from a sensor. The sensor always takes three consecutive samples and all of these values are stored in `Array1`.

A procedure `Summarise()` will calculate the average of three consecutive values from `Array1` and write the result to `Array2`. This will be repeated for all values in `Array1`.

The diagram below illustrates the process for the first six entries in `Array1`.



Write pseudocode for the procedure `Summarise()`.

```
PROCEDURE Summarise()
   DECLARE Value : REAL
   DECLARE IxA, IxB : INTEGER // Index variables

   IxB ← 1

   FOR IxA ← 1 TO 598 STEP 3
      Value ← Array1[IxA] + Array1[IxA + 1] + Array1[IxA + 2]
      Value ← Value / 3
      Array2[IxB] ← INT(Value)
      IxB ← IxB + 1
   NEXT IxA

ENDPROCEDURE
```

..... [5]

BLANK PAGE

6 The following pseudocode algorithm attempts to check whether a string is a valid email address.

```
FUNCTION IsValid(InString : STRING) RETURNS BOOLEAN
   DECLARE Index, Dots, Ats, Others : INTEGER
   DECLARE NextChar : CHAR
   DECLARE Valid : BOOLEAN

   Index ← 1
   Dots ← 0
   Ats ← 0
   Others ←  0
   Valid ← TRUE

   REPEAT
      NextChar ← MID(InString, Index, 1)
      CASE OF NextChar
         '.' : Dots ← Dots + 1
         '@' : Ats ← Ats + 1
               IF Ats > 1 THEN
                  Valid ← FALSE
               ENDIF
         OTHERWISE : Others ← Others + 1
      ENDCASE

      IF Dots > 1 AND Ats = 0 THEN
         Valid ← FALSE
      ELSE
         Index ← Index + 1
      ENDIF

   UNTIL Index > LENGTH(InString) OR Valid = FALSE

   IF NOT (Dots >= 1 AND Ats = 1 AND Others > 8) THEN
      Valid ← FALSE
   ENDIF

   RETURN Valid

ENDFUNCTION
```

(a) Part of the validation is implemented by the line:

```
IF NOT (Dots >= 1 AND Ats = 1 AND Others > 8) THEN
```

State the values that would result in the condition evaluating to `TRUE`.

.....

.....

.....

- Condition evaluates to TRUE if bracket contents evaluate to FALSE:
- Bracket contents evaluate to FALSE if:
- Dots: zero / less than one
  or
- Ats: not equal to one
  or
- Others: less than nine

[1]

(b) (i) Complete the trace table by dry running the function when it is called as follows:

```
Result ← IsValid("Liz.123@big@net")
```

Index	NextChar	Dots	Ats	Others	Valid

[5]

Index	NextChar	Dots	Ats	Others	Valid
		0	0	0	TRUE
1	'L'			1	
2	'i'			2	
3	'z'			3	
4	'.'	1			
5	'1'			4	
6	'2'			5	
7	'3'			6	
8	'@'		1		
9	'b'			7	
10	'i'			8	
11	'g'			9	
12	'@'		2		FALSE

(ii) State the value returned when `IsValid()` is called using the expression shown in **part (b)(i)**.

FALSE ..... [1]

7 A simple arithmetic expression is stored as a string in the format:

<Value1><Operator><Value2>

An operator character is one of the following: '+' '-' '*' '/'

Example arithmetic expression strings:

```
"803+1904"
"34/7"
```

(a) A procedure `Calculate()` will:

- take an arithmetic expression string as a parameter
- evaluate the expression
- output the result.

Assume:

- the string contains only numeric digits and a single operator character
- Value1 and Value2 represent integer values
- Value1 and Value2 are unsigned (they will not be preceded by '+' or '−').

(i) Write pseudocode for the procedure `Calculate()`.

```
PROCEDURE Calculate(Expression : STRING)
   DECLARE Val1, Val2, Index : INTEGER
   DECLARE Result : REAL
   DECLARE Par1, Par2, Par3 : STRING

   CONSTANT PLUS = '+'
   CONSTANT MINUS = '-'
   CONSTANT MULTIPLY = '*'
   CONSTANT DIVIDE = '/'

   FOR Index ← 1 TO LENGTH(Expression) //search for
operator
      ThisChar ← MID(Expression, Index, 1)
      IF IS_NUM(ThisChar) = FALSE THEN
         Par1 ← LEFT(Expression, Index - 1)
         Par2 ← ThisChar
         Par3 ← RIGHT(Expression, LENGTH(Expression) -
Index)
      ENDIF
   NEXT Index

   Val1 ← STR_TO_NUM(Par1)
   Val2 ← STR_TO_NUM(Par3)

   CASE OF Par2
      PLUS     : Result ← Val1 + Val2
      MINUS    : Result ← Val1 - Val2
      MULTIPLY : Result ← Val1 * Val2
      DIVIDE   : Result ← Val1 / Val2
   ENDCASE

   OUTPUT Result

ENDPROCEDURE
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

(ii) `Calculate()` is changed to a function that returns the value of the evaluated expression.

Write the header for the function in pseudocode.

..... `FUNCTION Calculate(Expression : STRING) RETURNS REAL` .....

..... [1]

(b) A string representing an arithmetic expression could be in the correct format but be impossible to evaluate.

Give an example of a correctly formatted string and explain why evaluation would be impossible.

Example string ..... "23/0" (Any divide by zero example) .....

Explanation .....

..... **Reason**: The result is infinity / cannot be represented / is undefined // will cause the program to crash .....

.....

[2]

8 A teacher is designing a program to perform simple syntax checks on programs written by students. Student programs are submitted as text files, which are known as project files.

A project file may contain blank lines.

The teacher has defined the first program module as follows:

Module	Description
`CheckFile()`	• takes the name of an existing project file as a parameter of type string • returns `TRUE` if the file is valid (it contains at least 10 non-blank lines), otherwise returns `FALSE`

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

```
FUNCTION CheckFile(Thisfile : STRING) RETURNS BOOLEAN
   DECLARE Valid : BOOLEAN
   DECLARE ThisLine : STRING
   DECLARE Count : INTEGER

   Count ← 0
   Valid ← FALSE
   OPEN ThisFile FOR READ

   WHILE NOT EOF(ThisFile) AND Valid = FALSE
      READFILE ThisFile, ThisLine
      IF ThisLine <> "" THEN
         Count ← Count + 1
         IF Count > 9 THEN
            Valid ← TRUE
         ENDIF
      ENDIF
   ENDWHILE

   CLOSEFILE ThisFile
   RETURN Valid

ENDFUNCTION
```

.....

.....

.....

..... [7]

Further modules are defined as follows:

Module	Description
`CheckLine()`	• takes a line from a project file as a parameter of type string • returns zero if the line is blank or contains no syntax error, otherwise returns an error number as an integer

`CountErrors()`	• takes two parameters:
	  ○ the name of a project file as a string
	  ○ the maximum number of errors as an integer
	• uses `CheckFile()` to test the project file. Outputs an error message and ends if the project file is not valid
	• calls `CheckLine()` for each line in the project file
	• counts the number of errors
	• outputs the number of errors or a warning message if the maximum number of errors is exceeded

(b) `CountErrors()` is called to check the project file `Jim01Prog.txt` and to stop if more than 20 errors are found.

Write the pseudocode statement for this call.

```
CALL CountErrors("Jim01Prog.txt", 20)
```

[2]

(c) Write pseudocode for module `CountErrors()`. Assume `CheckFile()` and `CheckLine()` have been written and can be used in your solution.

```
PROCEDURE CountErrors(ThisFile : STRING, MaxErrors :
INTEGER)
   DECLARE ErrCount, ThisError : INTEGER
   DECLARE ThisLine : STRING

   ErrCount ← 0

   IF CheckFile(ThisFile) = FALSE THEN
      OUTPUT "That program file is not valid"
   ELSE
      OPEN ThisFile FOR READ

      REPEAT
         READFILE, ThisFile, ThisLine
         ThisError ← CheckLine(ThisLine)
         IF ThisError <> 0 THEN
            ErrCount ← ErrCount + 1
         ENDIF
      UNTIL ErrCount > MaxErrors OR EOF(ThisFile)

      IF EOF(ThisFile) = FALSE THEN
         OUTPUT "Check terminated - too many errors"
      ELSE
         OUTPUT "There were ", ErrCount, " errors."
      ENDIF

   CLOSEFILE ThisFile
   ENDIF

ENDPROCEDURE
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

(d) Module `CheckLine()` includes a check for syntax errors.

Two examples of syntax error that **cannot** be detected from examining a **single** line are those involving selection and iteration.

Give **two other** examples.

1 .....

.....

.....

2 .....

.....

.....

[2]

1 Incorrect block structure. Missing keyword denoting part of block (for example `ENDPROCEDURE, ENDFUNCTION, ENDTYPE`)
2 Data type errors, for example, assigning an integer value to a string
3 Identifier used before it is declared
4 Incorrect parameter use

BLANK PAGE

BLANK PAGE

BLANK PAGE