Cambridge Assessment International Education

Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

5664831112

COMPUTER SCIENCE **9608/42**

Paper 4 Further Problem-solving and Programming Skills **October/November 2021**

2 hours

You must answer on the question paper.

No additional materials are needed.

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.

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

1 An array, `NumberArray`, stores 100 integer values. The array needs to be sorted into ascending numerical order.

(a) Describe how an insertion sort will sort the data in `NumberArray`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

(b) Another type of sorting algorithm is a bubble sort.

The procedure `Bubble()` takes an array as a parameter. It performs a bubble sort on the array. The sorting algorithm stops as soon as all the elements are in ascending order.

Complete the procedure `Bubble()`.

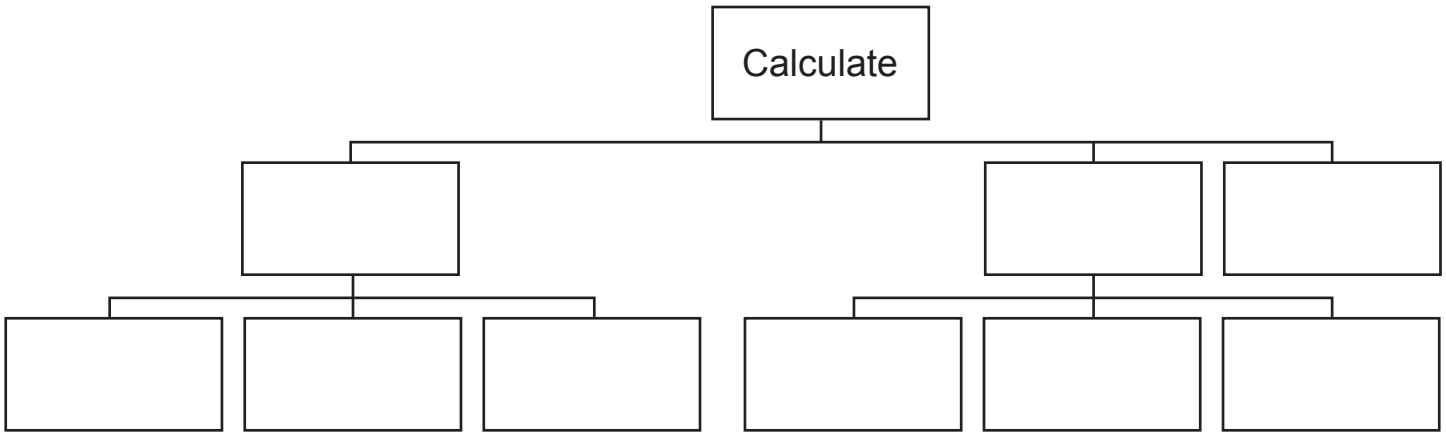
```
PROCEDURE Bubble(BYREF NumberArray : ARRAY[0 : 99] OF INTEGER)
    DECLARE Outer : INTEGER
    DECLARE Swap : BOOLEAN
    DECLARE Inner : INTEGER
    DECLARE Temp : INTEGER
    Outer ← LENGTH(NumberArray) - 1
    REPEAT
        Inner ← .....
        Swap ← FALSE
        REPEAT
            IF NumberArray[Inner] > NumberArray[Inner + 1]
                THEN
                    Temp ← NumberArray[Inner]
                    NumberArray[Inner] ← NumberArray[Inner + 1]
                    NumberArray[Inner + 1] ← Temp
                    Swap ← .....
            ENDIF
            Inner ← Inner + 1
        UNTIL Inner = .....
        Outer ← Outer - 1
    UNTIL Swap = ..... OR Outer = .....
ENDPROCEDURE
```

[5]

2 Complete the JSP structure diagram for the following pseudocode procedure.

```
PROCEDURE Calculate()
    INPUT Number1
    INPUT Number2
    INPUT Command
    IF Command = 1
        THEN
            Value ← Function1(Number1, Number2)
        ELSE
            IF Command = 2
                THEN
                    Value ← Function2(Number1, Number2)
                ELSE
                    Value ← Function3(Number1, Number2)
            ENDIF
    ENDIF
    OUTPUT Value
ENDPROCEDURE
```

JSP structure diagram



[4]

3 A user has to choose a new password to create an account. It is recommended that the password has at least two of the following elements:

- upper-case letter
- numeric character
- symbol.

The system outputs:

- `"Strong"` if there are at least two of the elements
- `"Medium"` if there is only one of the elements
- `"Weak"` if there are none of the elements.

Complete the following decision table for the password system described.

		Rules							
Conditions	One or more upper-case letters	N	Y	N	Y	N	Y	N	Y
	One or more numeric characters	N	N	Y	Y	N	N	Y	Y
	One or more symbols	N	N	N	N	Y	Y	Y	Y
Actions	Strong								
	Medium								
	Weak								

[3]

4 Each node of a binary tree is a record. Each record has a left pointer, an integer data value between 0 and 100 inclusive, and a right pointer.

For example:

Item	Example data
`LeftPointer`	2
`Data`	34
`RightPointer`	3

(a) Write **pseudocode** to declare the record with the identifier `Node`.

.....

.....

.....

.....

.....

.....

..... [2]

(b) Write **pseudocode** to declare a new node, `Node100`, **and** assign 100 to its data value, 1 to the left pointer and 4 to the right pointer.

.....

.....

.....

.....

.....

.....

..... [3]

(c) The ordered binary tree is stored as a 1D global array named `BinaryTree` of type `Node`.

`RootNode` and `FreePointer` are declared as global variables.

A null pointer is represented by `-1`.

The current state of the binary tree is shown in the following table:

RootNode	0	**Index**	**LeftPointer**	**Data**	**RightPointer**
		[0]	1	23	3
FreePointer	6	[1]	-1	5	2
		[2]	-1	8	4
		[3]	5	100	-1
		[4]	-1	9	-1
		[5]	-1	88	-1
		[6]	-1	null	-1
		[7]	-1	null	-1

(i) State the purpose of the free pointer.

.....

..... [1]

(ii) Identify an appropriate integer value to represent null data.

..... [1]

(iii) Draw the current state of the binary tree.

[2]

(iv) The procedure `AddData()`:

- takes the node to be added to the tree as a parameter
- finds the location for the node to be stored
- stores the node in the next free array index
- stores `-1` in the new node's `LeftPointer` and `RightPointer`
- updates the pointers in the other nodes
- updates `FreePointer`.

Complete the pseudocode for the procedure `AddData()`.

```
PROCEDURE AddData(NewNode)
    BinaryTree[FreePointer] ← .....
    BinaryTree[FreePointer].LeftPointer ← -1
    BinaryTree[FreePointer].RightPointer ← -1
    DECLARE PositionFound : BOOLEAN
    DECLARE PointerCounter : INTEGER
    PositionFound ← .....
    PointerCounter ← RootNode
    WHILE NOT .....
        IF ..... < BinaryTree[PointerCounter].Data
            THEN
                IF BinaryTree[PointerCounter].LeftPointer = -1
                    THEN
                        BinaryTree[PointerCounter].LeftPointer ← FreePointer
                        PositionFound ← TRUE
                    ELSE
                        PointerCounter ← BinaryTree[PointerCounter].LeftPointer
                ENDIF
            ELSE
                IF BinaryTree[PointerCounter].RightPointer = -1
                    THEN
                        BinaryTree[PointerCounter].RightPointer ← FreePointer
                        PositionFound ← TRUE
                    ELSE
                        PointerCounter ← BinaryTree[PointerCounter].RightPointer
                ENDIF
        ENDIF
    ENDWHILE
    FreePointer ← FreePointer .....
ENDPROCEDURE
```

[5]

5 Study the following recursive pseudocode algorithm.

```
FUNCTION Recursive(Num1, Num2 : INTEGER) RETURNS INTEGER
    IF Num1 > Num2
        THEN
            RETURN 10
        ELSE
            IF Num1 = Num2
                THEN
                    RETURN Num1
                ELSE
                    RETURN Num1 + Recursive(Num1 * 2, Num2)
            ENDIF
    ENDIF
ENDFUNCTION
```

(a) The function is called as follows:

```
Recursive(1, 15)
```

Dry run the function and complete the trace table. Give the final return value.

Trace table:

Function call	Num1	Num2	Return value

Final return value .....

Working .....

.....

.....

.....

.....

[4]

(b) Rewrite the function `Recursive()` in **pseudocode**, using an **iterative algorithm**.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

6 Details of errors generated in a program are stored in a stack.

Details of each error are stored in a record structure, `Error`.

(a) State which error will be the first retrieved from the stack.

.....

..... [1]

(b) The stack is implemented as a 1D array with the identifier `ErrorArray`.

The pointer `LastItem` stores the position of the last error in the array.

(i) The function, `AddItemToStack`, takes the next error, the array, and pointer as parameters.

If the stack is full, the function returns `FALSE`; otherwise it adds the error to the stack, changes the pointer's value and returns `TRUE`.

Complete the following pseudocode for the function `AddItemToStack`.

```
FUNCTION AddItemToStack(BYREF ErrorArray : ARRAY[0 : 99] OF Error,
                        BYREF LastItem : INTEGER,
                        BYVALUE Error1 : Error) RETURNS BOOLEAN

    IF LastItem = .....

        THEN

            RETURN .....

        ELSE

            ErrorArray[LastItem + 1] ← .....

            LastItem ← .....

            RETURN .....

    ENDIF

ENDFUNCTION
```

[4]

(ii) Explain the reasons why `ErrorArray` and `LastItem` are passed by reference, but `Error1` is passed by value.

.....

.....

.....

.....

.....

..... [3]

(iii) The function `RemoveItem` takes the next error from the stack and returns it.

If there are no errors in the stack, it returns the global record `NullError`.

Complete the pseudocode algorithm `RemoveItem`.

```
FUNCTION RemoveItem(BYREF ErrorArray : ARRAY[0 : 99] OF Error,
                    BYREF LastItem : INTEGER) RETURNS Error

    DECLARE ItemToRemove : Error

    IF ..........................................

        THEN

            RETURN ..........................................

        ELSE

            ItemToRemove ← ErrorArray[..........................................]

            LastItem ← LastItem - 1

            RETURN ..........................................

ENDFUNCTION
```

[3]

(iv) The errors that have been processed are stored in a global queue, `ErrorComplete`.

The function `Enqueue` adds a record to `ErrorComplete`:

```
Enqueue(ErrorToAdd)
```

`Enqueue()` returns `TRUE` if the record is successfully added to the queue, and returns `FALSE` if the queue is full.

The procedure `RunError()` should:

- remove a record from the stack using the function `RemoveItem()`
- output an appropriate message if there were no records in the stack
- if an error record is returned, add the record to the queue using the function `Enqueue()`
- if the record is added to the queue, output an appropriate message
- if the record is not added to the queue, output an appropriate message.

Complete the pseudocode procedure `RunError()`.

```
PROCEDURE RunError(BYREF ErrorComplete : ARRAY[0 : 99] OF Error,
                   BYREF ErrorArray : ARRAY[0 : 99] OF Error)
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[5]

7 A treasure box is hidden within a computer game.

The box has a code that needs to be entered to allow the user into the box. The box contains up to 10 objects that are defined as being of the class `FieldObject`. The definition for the class `Box` is:

Box	
`Size : STRING`	`// small, medium or large`
`Contents : ARRAY[0 : 9] OF FieldObject`	`// the 10 items the box holds`
`Lock : STRING`	`// the code to unlock the box`

`Strength : INTEGER`	`// the strength of the box`
	`// decreases by 1 each time an`
	`// incorrect code is entered`

`Constructor()`	`// instantiates an object of the Box`
	`// class and assigns initial values`
	`// to the attributes`

`Unlock()`	`// checks if the code is correct to` `// unlock the box`
`GetContents()`	`// returns the array`
`SetSize()`	`// sets the size of the box`
`SetContents()`	`// sets the contents of the box`
`SetLock()`	`// sets the lock code`
`SetStrength()`	`// sets the strength`

(a) The constructor creates a new instance of a box. It takes the size of the box, one item of content and the lock code as parameters. The strength is initialised to `100`.

Write **program code** to create the class and constructor for `Box`.

Do **not** write the code for `Unlock()` or any of the set or get methods. Use the constructor for your chosen language.

Programming language .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(b) The player inputs the code to unlock the box. Each time they enter an incorrect code, the strength of the box decreases by 1. If the strength of the box becomes 0, the box automatically unlocks.

The class `Box` has a method `Unlock()` that:

- takes the code entered as a parameter and checks if it matches the code to unlock the box
- returns `TRUE` if the parameter matches the unlock code
- subtracts 1 from `Strength` if the parameter does not match the unlock code
- checks if the new value of `Strength` is less than 1
- returns `TRUE` if the new value of `Strength` is less than 1, otherwise it returns `FALSE`.

Write **program code** for the method `Unlock()`.

Programming language .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(c) The text file, `Progress.txt`, stores data about the player's previous progress.

The procedure `LoadGame()`:

- opens the text file in read mode
- takes the data from the file and stores the data in the variable `GameData`
- raises an exception with an appropriate message output if it cannot find the file.

Write **program code** for the procedure `LoadGame()`.

Programming language .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

8 A game stores details about characters.

A declarative programming language is used to represent the following knowledge base:

```
01 hair(blonde).

02 hair(black).

03 hair(red).

04 face(glasses).

05 face(moustache).

06 face(beard).

07 person(ismail).

08 person(anisha).

09 person(kim).

10 person(kyle).

11 has(kyle, glasses).

12 has(kyle, beard).

13 has(anisha, red).

14 has(kyle, black).
```

These clauses have the following meaning:

Clause	Explanation
01	Hair can be blonde
04	Glasses can be on the face
08	Anisha is a person
12	Kyle has a beard
13	Anisha has red hair

A person, `X`, is a selected person if they have black hair and either a moustache or a beard.

Write a rule to represent this condition.

```
SelectedPerson(X)

IF
```

.....

.....

.....

..... [2]

BLANK PAGE