Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

* 8 9 7 1 0 0 7 5 4 4 *

COMPUTER SCIENCE **9608/43**

Paper 4 Further Problem-solving and Programming Skills **May/June 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.

DC (CJ/CGW) 198995/3

1 A vending machine allows users to insert coins to purchase an item.

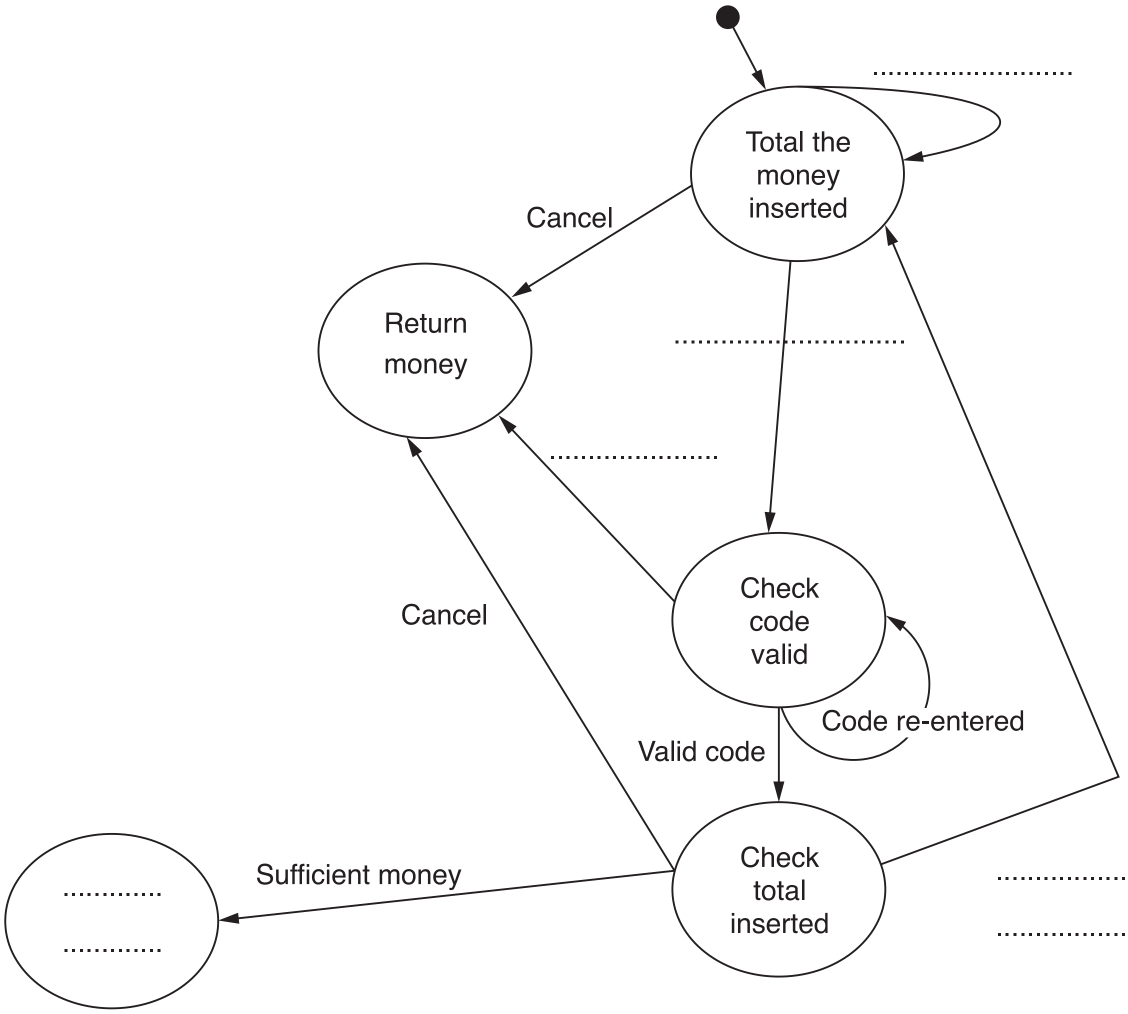
The user then enters the code for the item they would like the machine to dispense (give out). The user must re-enter the code until it is valid.

If the code is valid but the user has not inserted enough money for the item chosen, the machine waits for more coins to be inserted. The user then has to re-enter the code.

The user can press cancel at any time to return the money inserted into the machine.

(a) The state-transition diagram shows the different states of the vending machine.

Complete the state-transition diagram.



[5]

(b) The vending machine is part of a program that is written using object-oriented programming (OOP). The vending machine makes use of two classes that are described in the following tables.

All attributes are declared as private.

foodItem	
`name : STRING`	`// the name of the item of food`
`code : STRING`	`// the code to be entered for that item to be` `// selected`
`cost : REAL`	`// the cost of the item`
`constructor(nameP, codeP,` `costP)`	`// creates an instance of foodItem` `// takes the name, code and cost as parameters`

`getCode()`
`getCost()`
`getName()`
	`// returns the code for the item`
	`// returns the cost of the item`
	`// returns the name of the item`

vendingMachine	

`items : ARRAY[0:3] OF foodItem`
`moneyIn : REAL`
	`// stores four items of type foodItem`
	`// stores the total money inserted by the`
	`// user, initialised to 0 in the constructor`

`constructor(item1, item2,`
`item3, item4)`
	`// creates an instance of vendingMachine,`
	`// takes four objects of type foodItem as`
	`// parameters and stores them in array items`

`insertMoney()`	`// takes the value of the coin as a parameter` `// and adds it to moneyIn`
`checkValid ()`	`// takes a code as a parameter and checks it is` `// valid against the food item codes`
`getItemName()`	`// takes the array index as a parameter and` `// returns the name of the food items`

(i) Write **program code** to declare the class `vendingMachine`. You are only required to write program code for the attribute declarations and the constructor.

If you are writing in Python, include attribute declarations using comments.

Use your programming language's constructor method.

Programming language ....

Program code

....

....

....

....

....

....

....

....

....

....

.... [4]

(ii) The method `checkValid()` takes the food item code as a parameter. It checks the code against each element in `items` and returns:

- `–1` if the code is not valid
- `–2` if the code is valid, but the `moneyIn` is less than the cost of the item
- the index of the item, if the code is valid and the `moneyIn` is greater than or equal to the cost of the item.

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

Programming language ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

.... [5]

(iii) Four objects of type `foodItem` are declared with the identifiers:

`chocolate, sweets, sandwich, apple`

Write **program code** to declare an instance of `vendingMachine` with the identifier `machineOne` and the objects: `chocolate, sweets, sandwich, apple`.

Programming language ....

Program code

....

....

....

.... [2]

2 Peter uses a record structure, `customer`, to store data about customers. The data includes:

- a unique customer ID between 10 000 and 99 999
- the customer's first name
- the customer's last name
- the customer's telephone number (for example, +44 1234567891).

(a) Write **pseudocode** to define the record type `customer`.

....

....

....

....

....

....

....

.... [3]

(b) The customer records are stored in a random file. The location of each record is calculated as a hash value using:

```
(customer.customerID modulus 1000) + 2
```

(i) Calculate the hash value for each of the customer IDs in the following table.

Customer ID	Hash value
40 125	
10 131	

[1]

(ii) Two or more records could have the same hash value that results in a collision.

Explain how the hashing algorithm can be designed to handle collisions.

....

....

....

....

....

.... [3]

(iii) The function, `getCustomer()`:

- takes the customer ID as a parameter
- passes the customer ID to the function `getRecordLocation()`, which returns the calculated hash value
- reads and returns the record from the hashed location in the file `customerRecords.dat`

You can assume that both the file and the record being accessed exist.

Write **pseudocode** for the function `getCustomer()`.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

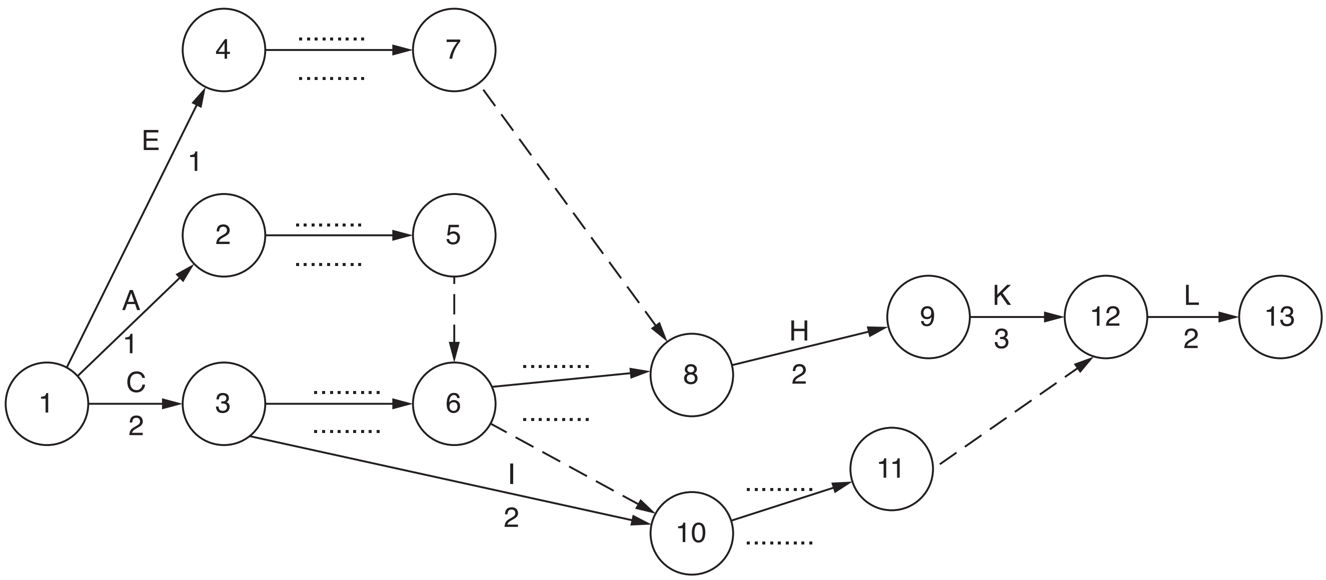
.... [5]

3 Alix manages a team of programmers who are creating a new computer game.

Alix has listed some of the tasks, along with their estimated time to complete and their immediate predecessors in the following table:

Task	Description	Predecessors	Time to complete (weeks)
A	Design character	–	1
B	Program character movement	A	1
C	Design level 1	–	2
D	Program level 1	C	2
E	Design robot	–	1
F	Program robot movement	E	1
G	Integrate character in level 1	B, D	2
H	Integrate robot in level 1	F, G	2
I	Design level 2	C	2
J	Program level 2	D, I	2
K	Test level 1	H	3
L	Integrate character and robot into level 2	J, K	2

(a) Complete the Program Evaluation Review Technique (PERT) chart for the tasks in the table.



[5]

(b) Explain how the tasks in the table can be divided between the team to allow concurrency of tasks.

....

....

....

....

....

.... [2]

(c) Explain the benefits of the team using program libraries in the development of the program.

....

....

....

....

....

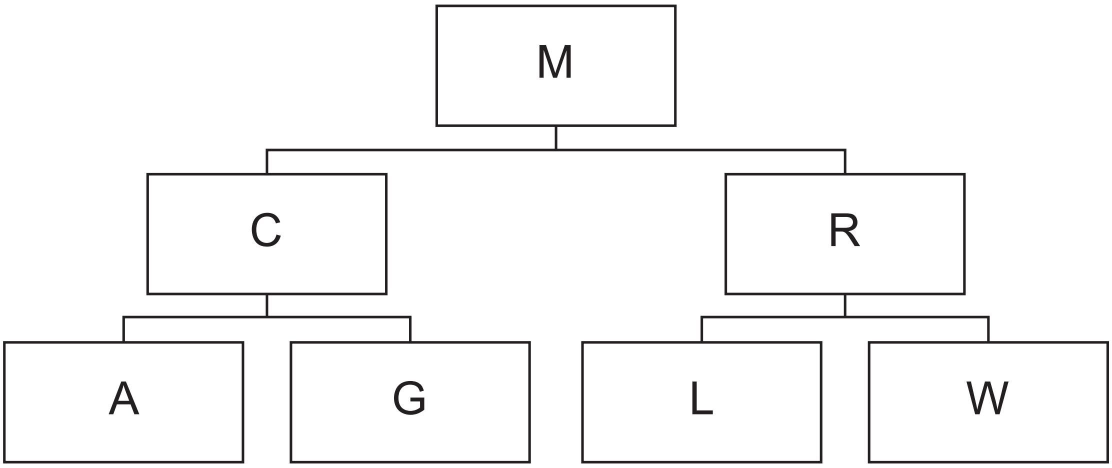
.... [3]

(d) Identify **two** features in an editor that the developers can use to help them create their programs.

Feature 1 ....

Feature 2 ....
[2]

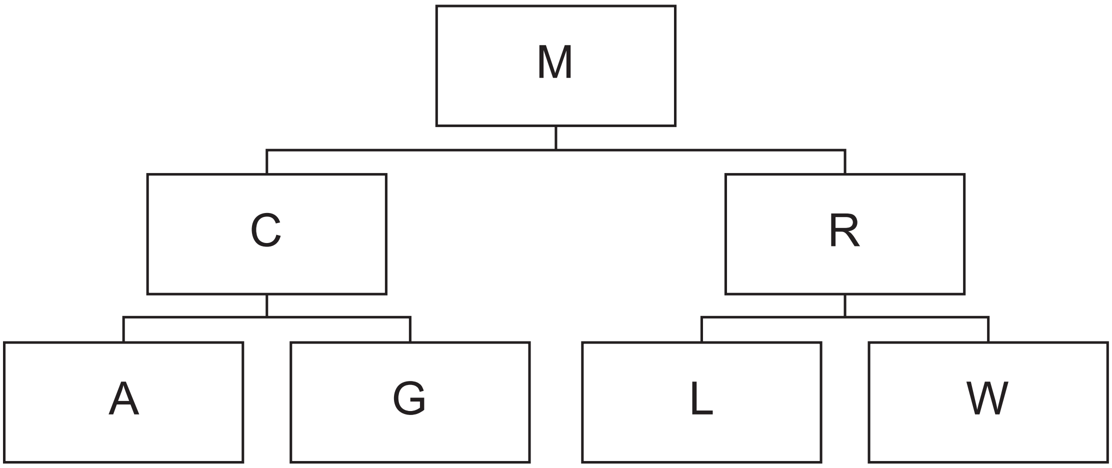
4 Chon creates a binary tree structure to store options that the user can select from a menu (M) in his program.



(a) There are four new options that need to be added.

If option G is selected, the user must choose either option D or option H. If option L is selected, the user must choose either option J or option P.

Complete the following binary tree by adding options D, H, J and P.



[2]

(b) Each node in the binary tree is stored using the following record structure:

```
TYPE node
   leftPointer : INTEGER
   data : STRING
   rightPointer : INTEGER
ENDTYPE
```

The tree is stored as a 1D array, `binaryTree`. Null pointers are represented by `−1`.

(i) The table shows the contents of the three fields in each record stored in the 1D array `binaryTree`.

Complete the table to show the contents of `binaryTree` from **part (a)**.

rootPointer	

freePointer	

Index	**leftPointer**	**data**	**rightPointer**
0		M	
1		C	
2		A	
3		L	
4		G	
5		R	
6		W	
7		J	
8		D	
9		P	
10		H	
11			

[4]

(ii) Write **pseudocode** to declare the array `binaryTree` to store up to 100 objects of type `node`.

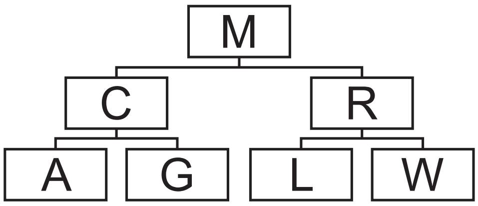
....

....

....

.... [2]

(iii) A pre-order traversal on the following tree would output M C A G R L W



The pre-order traversal can be written as a recursive procedure:

1. output the root node
2. follow the left pointer and repeat from step 1
3. follow the right pointer and repeat from step 1.

Complete the **pseudocode** recursive procedure `preOrder()`.

```
PROCEDURE preOrder(BYVALUE rootPointer : INTEGER)
```

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

```
ENDPROCEDURE
```

[6]

5 A binary search algorithm searches for data in a sorted array.

(a) The pseudocode function `binarySearch()` performs a binary search to find a given value in the global array, `dataArray`. If the value is found, the function returns its index. If the value is not found, the function returns `-1`.

Complete the **pseudocode** for the function `binarySearch()`.

```
FUNCTION binarySearch(BYVALUE upper, lower, searchValue : INTEGER)
                                                    RETURNS INTEGER

  DECLARE flag : INTEGER

  DECLARE mid : INTEGER

  flag ← -2

  mid ← 0

  WHILE flag <> -1

    mid ← lower + ((upper – lower) ...........................................)

    IF upper < lower

      THEN

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

      ELSE

        IF dataArray(mid) < searchValue

          THEN

            ........................................... ← ...........................................

          ELSE

            IF dataArray(mid) > searchValue

              THEN

                ........................................... ← ...........................................

              ELSE

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

            ENDIF

        ENDIF

    ENDIF

  ENDWHILE

ENDFUNCTION
```

[4]

(b) The binary search algorithm can be written recursively.

Write **program code** for a recursive function `recursiveBinarySearch()`.

Programming language ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [5]

6 The table shows assembly language instructions for a processor that has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

Label	Instruction		Explanation
	Op code	**Operand**	
	`LDM`	`#n`	Immediate addressing. Load the number n to ACC
	`LDD`	`<address>`	Direct addressing. Load the contents of the location at the given address to ACC
	`LDX`	`<address>`	Indexed addressing. Form the address from <address> + the contents of the Index Register. Copy the contents of this calculated address to ACC
	`LDR`	`#n`	Immediate addressing. Load the number n to IX
	`STO`	`<address>`	Store contents of ACC at the given address
	`ADD`	`<address>`	Add the contents of the given address to ACC
	`INC`	`<register>`	Add 1 to the contents of the register (ACC or IX)
	`AND`	`<address>`	Bitwise AND operation of the contents of ACC with the contents of <address>
	`XOR`	`<address>`	Bitwise XOR operation of the contents of ACC with the contents of <address>
	`OR`	`<address>`	Bitwise OR operation of the contents of ACC with the contents of <address>
	`OUT`		Output to screen the character whose ASCII value is stored in ACC
	`CMP`	`<address>`	Compare the contents of ACC with the contents of <address>
	`CMP`	`#n`	Compare the contents of ACC with number n
	`JPE`	`<address>`	Following a compare instruction, jump to <address> if the compare was True
	`JPN`	`<address>`	Following a compare instruction, jump to <address> if the compare was False
	`JMP`	`<address>`	Jump to the given address
	`END`		Return control to the operating system
`<label>:`	`<op code>`	`<operand>`	Labels an instruction
`<label>:`	`<data>`		Gives a symbolic address <label> to the memory location with contents <data>

An algorithm takes each letter of a stored 5-letter word and checks if the letter is upper case.

If the letter is upper case, it outputs the letter.

If the letter is not upper case, it converts the letter to upper case and then outputs it.

All ASCII upper case letters have `010` as the three most significant bits.

Assume each letter is alphabetic.

Complete the assembly language program for the algorithm described using the instruction set provided on the previous page.

Instruction			Comment
Label	**Op code**	**Operand**	
	LDR	#0	// load zero to IX
			// load count and check if it is 5
	JPE	endP	// jump to end
	LDX	word	// load letter from indexed address word
			// check if it is upper case
	CMP	#0	
	JPE	output	// jump to output if it is upper case
	LDX	word	// load letter from indexed address word
			// convert to upper case
output:	OUT		// output the character
			// increase count by 1
	INC	IX	// increase IX by 1
	JMP	start	// return to start
endP:	end		// end the program
word:	B01001000		
	B01101111		
	B01110101		
	B01110011		
	B01100101		
mask1:	B00100000		
mask2:	B11011111		
count:	0		

[6]

7 Giles is writing a program that uses a stack.

The stack stores up to 1000 integers in the 1D array, `stackArray`.

(a) The procedure `setUpStack()` takes two parameters:

- the array, `stackArray`
- a pointer to the last element pushed onto the stack, `topOfStack`

The procedure initialises all array elements to `−1` and the pointer to `−1`.

Write **pseudocode** for the procedure `setUpStack()`.

....

....

....

....

....

....

....

....

.... [3]

(b) The function `pop()` pops and returns the item from the top of the stack. If the stack is empty, it returns `−1`.

Write **pseudocode** for the function `pop()`.

....

....

....

....

....

....

....

....

....

.... [3]

BLANK PAGE

BLANK PAGE