A

Surname ____________________

Other Names ____________________

Centre Number ____________________

Candidate Number ____________________

Candidate Signature ____________________

GCSE
COMPUTER SCIENCE

Paper 1 Computational thinking and problem-solving

8520/1

Monday 13 May 2019 Morning

Time allowed: 1 hour 30 minutes

There are no additional materials required for this paper.

At the top of the page, write your surname and other names, your centre number, your candidate number and add your signature.

[Turn over]

JUN198520101

INSTRUCTIONS

- **Use black ink or black ball-point pen. Use pencil only for drawing.**
- **Answer ALL questions.**
- **You must answer the questions in the spaces provided.**
- **Do all rough work in this book. Cross through any work you do not want to be marked.**
- **You are free to answer questions that require a coded solution in whatever format you prefer as long as your meaning is clear and unambiguous.**
- **You must NOT use a calculator.**

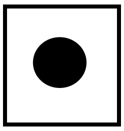
INFORMATION

- **The total number of marks available for this paper is 80.**

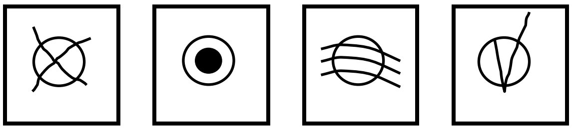
ADVICE

For the multiple-choice questions, completely fill in the lozenge alongside the appropriate answer.

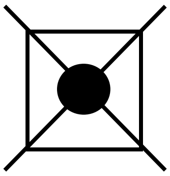
CORRECT METHOD



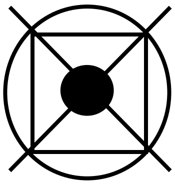
WRONG METHODS



If you want to change your answer you must cross out your original answer as shown.



If you wish to return to an answer previously crossed out, ring the answer you now wish to select as shown.



DO NOT TURN OVER UNTIL TOLD TO DO SO

Answer ALL questions.

0 1 **The pseudo-code in FIGURE 1 assigns two string values to two variables.**

FIGURE 1

```
title ← 'computer science'
level ← 'gcse'
```

0 1 . 1 **Shade ONE lozenge that shows the length of the contents of the variable** `level` **in FIGURE 1. [1 mark]**

- ○ **A** 1
- ○ **B** 2
- ○ **C** 3
- ○ **D** 4

0 1 . 2 **Shade ONE lozenge that shows the result of concatenating the variable `title` with the variable `level` in FIGURE 1. [1 mark]**

- ◯ **A** `'computer science gcse'`
- ◯ **B** `'Computer Science GCSE'`
- ◯ **C** `'computersciencegcse'`
- ◯ **D** `'computer sciencegcse'`

[Turn over]

0 1 . 3 **Shade ONE lozenge to show which of the following strings is a substring of the variable** `title` **in FIGURE 1. [1 mark]**

- ○ **A** `'compsci'`
- ○ **B** `'puters'`
- ○ **C** `'sci'`
- ○ **D** `'tersci'`

0 1 . 4 **The Unicode character code of `title[0]`, which is `'c'`, is 99.**

Shade ONE lozenge to show the Unicode character code of the character `level[3]` in FIGURE 1. [1 mark]

- ○ **A 95**
- ○ **B 99**
- ○ **C 101**
- ○ **D 103**

[Turn over]

4

0 2 **The three examples of code shown in FIGURE 2 are all equivalent to one another.**

FIGURE 2

Example 1	**Example 2**	**Example 3**

`a ← 4`
`b ← 3`
`IF a = b THEN`
`   c ← a + b`
`ENDIF`
	` MOV R0, #4`
	` MOV R1, #3`
	` CMP R0, R1`
	` BNE end`
	` ADD R2, R0, R1`
	`end:`
	` HLT`
		`1001 0000 0100 0000`
		`1001 0001 0011 0000`
		`0100 0000 0001 0000`
		`1010 0101 0000 0000`
		`1100 0010 0000 0001`
		`1111 0000 0000 0000`

BLANK PAGE

[Turn over]

0 2 . 1 **Shade ONE lozenge to show the statement that is true about FIGURE 2, on page 8. [1 mark]**

- ◯ **A** None of the examples of code is in a low-level language.
- ◯ **B** Only one of the examples of code is in a low-level language.
- ◯ **C** Only two of the examples of code are in low-level languages.
- ◯ **D** All three of the examples of code are in low-level languages.

0 2 . 2 **Explain why a developer, who is good at both low-level and high-level programming, would normally use high-level languages when writing programs. [4 marks]**

BLANK PAGE

0 2 . 3 **Statements A and B refer to two different types of program translator.**

STATEMENT A: This type of translator can convert a high-level language program into machine code. The source code is analysed fully during the translation process. The result of this translation can be saved, meaning the translation process does not need to be repeated.

STATEMENT B: This type of translator was used to convert the code in EXAMPLE 2 to the code in EXAMPLE 3 in FIGURE 2, on page 8.

State the type of program translators referred to in statements A and B. [2 marks]

Statement A: ______________________________

____

____

Statement B: ______________________________

____

____

0 3 **A cake recipe uses 100 grams of flour and 50 grams of sugar for every egg used in the recipe.**

FIGURE 3 shows the first line of an algorithm that will be used to calculate the amount of flour and sugar required based on the number of eggs being used. The number of eggs is entered by the user.

FIGURE 3

```
eggsUsed ← USERINPUT
```

0 3 . 1 **Shade ONE lozenge to show which of the following lines of code correctly calculates the amount of flour needed in grams. [1 mark]**

- ○ **A** `flourNeeded ← USERINPUT`
- ○ **B** `flourNeeded ← eggsUsed * USERINPUT`
- ○ **C** `flourNeeded ← eggsUsed * 100`
- ○ **D** `flourNeeded ← eggsUsed * 50`

03.2 Shade ONE lozenge to show which programming technique has been used in all of the lines of code in QUESTION 03.1. [1 mark]

- ◯ **A** Assignment
- ◯ **B** Indefinite iteration
- ◯ **C** Nested iteration
- ◯ **D** Selection

[Turn over]

03.3 **The developer wants to use validation to ensure that the user can only enter a positive number of eggs, ie one egg or more. The maximum number of eggs that can be used in the recipe is eight.**

Develop an algorithm, using either pseudo-code or a flowchart, so that the number of eggs is validated to ensure the user is made to re-enter the number of eggs used until a valid number is entered.

You should assume that the user will always enter an integer. [4 marks]

[Turn over]

6

BLANK PAGE

[Turn over]

0 4 . 1 **Complete the trace table for the algorithm shown in FIGURE 4 for when the user enters the value 750 when prompted. [4 marks]**

FIGURE 4

```
constant PAYLOAD_SIZE ← 250
constant HEADER_SIZE ← 50
OUTPUT 'Enter the number of bits of data to
be sent'
dataToBeSent ← USERINPUT
totalSize ← PAYLOAD_SIZE + HEADER_SIZE
numberOfPackets ← 0
REPEAT
   dataToBeSent ← dataToBeSent – totalSize
   numberOfPackets ← numberOfPackets + 1
UNTIL dataToBeSent ≤ 0
```

totalSize	dataToBeSent	numberOfPackets
	750	

0 4 . 2 **State why both** `PAYLOAD_SIZE` **and** `HEADER_SIZE` **from the algorithm in FIGURE 4 did not need to be included in the trace table. [1 mark]**

0 4 . 3 **Shade ONE lozenge to show which of the following best represents the input and output to/from the algorithm in FIGURE 4. [1 mark]**

- ○ **A** **Input:** `dataToBeSent`, **output:** `numberOfPackets`
- ○ **B** **Input:** `numberOfPackets`, **output:** `totalSize`
- ○ **C** **Input:** `totalSize`, **output:** `dataToBeSent`

[Turn over]

0 4 . 4 **A developer looks at the algorithm in FIGURE 4, on page 20, and realises that the use of iteration is unnecessary if they use a combination of the** `DIV` **and** `MOD` **operators.**

- `DIV` **calculates integer division, eg** `11 DIV 4 = 2`
- `MOD` **calculates the remainder after integer division, eg** `11 MOD 4 = 3`

The programmer realises that she can rewrite the algorithm by replacing the `REPEAT-UNTIL` **structure with code that uses selection,** `MOD` **and** `DIV` **instead.**

Complete this new algorithm, on the opposite page, by stating the code that should be written in the boxes labelled A, B and C. This new algorithm should calculate the same final result for the variable `numberOfPackets` **as the original algorithm in FIGURE 4. [3 marks]**

```
constant PAYLOAD_SIZE ← 250
constant HEADER_SIZE ← 50
OUTPUT 'Enter the number of bits of
data to be sent'
dataToBeSent ← USERINPUT
totalSize ← PAYLOAD_SIZE +
HEADER_SIZE
numberOfPackets ← dataToBeSent DIV
totalSize

 IF [ A ] MOD [ B ] > 0 THEN

   numberOfPackets ← [ C ]
 ENDIF
```

A ______________________________

____

B ______________________________

____

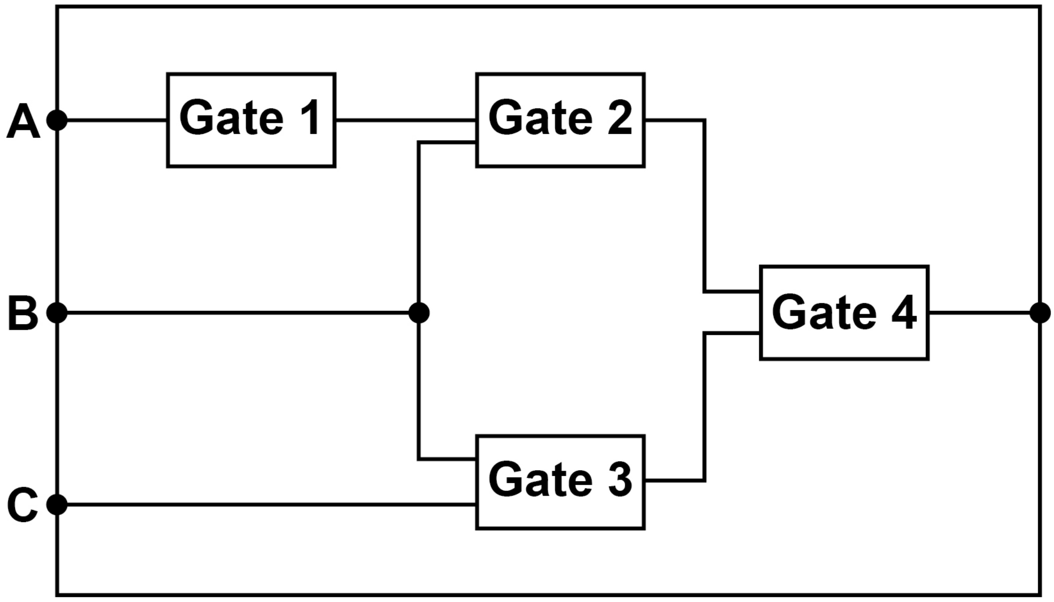
C ______________________________

____

9

0 5 **The expression** `(B AND (NOT A)) OR (B AND C)` **can be represented by the logic circuit shown in FIGURE 5. In the circuit the logic gates are marked with labels instead of their proper symbols.**

FIGURE 5



0 5 . 1 **State the name of the logic gate used at Gate 1 in FIGURE 5. [1 mark]**

____

____

0 5 . 2 **State the name of the logic gate used at Gate 2 in FIGURE 5. [1 mark]**

____

____

0 5 . 3 **Draw the logic circuit symbol in the space below for the logic gate used at Gate 3 in FIGURE 5. [1 mark]**

0 5 . 4 **Draw the logic circuit symbol in the space below for the logic gate used at Gate 4 in FIGURE 5. [1 mark]**

[Turn over]

0 5 . 5 **Complete the truth table for the Boolean expression:**

`(X AND Y) OR (NOT X)`

[3 marks]

X	Y	X AND Y	NOT X	(X AND Y) OR (NOT X)
0	0			
0	1			
1	0			
1	1			

0 5 . 6 **A truth table for the complex Boolean expression:**

`(A1 AND (NOT A2) AND A3) OR (A1 AND A2 AND A3)`

is shown in FIGURE 6, on the opposite page.

FIGURE 6

A1	A2	A3	OUTPUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Shade ONE lozenge which shows a simpler expression which is the equivalent of the original, more complex, expression. [1 mark]

○ **A** `NOT A1`

○ **B** `A2 OR A3`

○ **C** `A1 AND (NOT A2)`

○ **D** `A1 AND A3`

0 6 **Run length encoding (RLE) is a form of compression that creates frequency/data pairs to describe the original data.**

For example, an RLE of the bit pattern `0000001110111 1` **could be** `6 0 3 1 1 0 4 1` **because there are six** `0`**s followed by three** `1`**s followed by one** `0` **and finally four** `1`**s.**

The algorithm in FIGURE 7, on the opposite page, is designed to output an RLE for a bit pattern that has been entered by the user.

Five parts of the code labelled `L1`, `L2`, `L3`, `L4` **and** `L5` **are missing.**

- **Note that indexing starts at zero.**

FIGURE 7

```
pattern ← L1
i ← L2
count ← 1
WHILE i < LEN(pattern)-1
   IF pattern[i] L3 pattern[i+1]
THEN
      count ← count + 1
   ELSE
      L4
      OUTPUT pattern[i]
      count ← 1
   ENDIF
   L5
ENDWHILE
OUTPUT count
OUTPUT pattern[i]
```

0 6 . 1 **Shade ONE lozenge to show what code should be written at point `L1` of the algorithm. [1 mark]**

- ◯ **A** `OUTPUT`
- ◯ **B** `'RLE'`
- ◯ **C** `True`
- ◯ **D** `USERINPUT`

BLANK PAGE

0 6 . 2 **Shade ONE lozenge to show what value should be written at point `L2` of the algorithm. [1 mark]**

- **A** `-1`
- **B** `0`
- **C** `1`
- **D** `2`

0 6 . 3 **Shade ONE lozenge to show what operator should be written at point `L3` of the algorithm. [1 mark]**

- **A** =
- **B** ≤
- **C** <
- **D** ≠

[Turn over]

0 6 . 4 **Shade ONE lozenge to show what code should be written at point `L4` of the algorithm. [1 mark]**

- ○ **A** `count`
- ○ **B** `count ← count - 1`
- ○ **C** `count ← USERINPUT`
- ○ **D** `OUTPUT count`

0 6 . 5 **Shade ONE lozenge to show what code should be written at point `L5` of the algorithm. [1 mark]**

- ○ **A** `i ← i * 2`
- ○ **B** `i ← i + 1`
- ○ **C** `i ← i + 2`
- ○ **D** `i ← i DIV 2`

0 6 . 6 **State a run length encoding of the series of characters** `ttjjeeess` **[2 marks]**

[Turn over]

BLANK PAGE

0 6 . 7 **A developer implements the algorithm shown in FIGURE 7, on page 29, and tests their code to check that it is working correctly. The developer tests it only with the input bit pattern that consists of six zeros and it correctly outputs** `6 0`.

Using example test data, state THREE further tests that the developer could use to improve the testing of their code. [3 marks]

0 7 **A developer creates the algorithm shown in FIGURE 8 to provide support for users of a new brand of computer monitor (display).**

- **Line numbers are included but are not part of the algorithm.**

FIGURE 8

```
1   OUTPUT 'Can you turn it on?'
2   ans ← USERINPUT
3   IF ans = 'no' THEN
4      OUTPUT 'Is it plugged in?'
5      ans ← USERINPUT
6      IF ans = 'yes' THEN
7         OUTPUT 'Contact supplier'
8      ELSE
9         OUTPUT 'Plug it in and
          start again'
10     ENDIF
11  ELSE
12     OUTPUT 'Is it connected to the
       computer?'
13     ans ← USERINPUT
14     IF ans = 'yes' THEN
15        OUTPUT 'Contact supplier'
16     ELSE
17        OUTPUT 'Connect it to the
          computer'
18     ENDIF
19  ENDIF
```

0 7 . 1 **Shade ONE lozenge to show which programming technique is used on line 3 of the algorithm in FIGURE 8. [1 mark]**

- ○ **A Assignment**
- ○ **B Iteration**
- ○ **C Selection**

0 7 . 2 **Shade ONE lozenge to show the data type of the variable `ans` in the algorithm in FIGURE 8. [1 mark]**

- ○ **A Date**
- ○ **B Integer**
- ○ **C Real**
- ○ **D String**

[Turn over]

BLANK PAGE

0 7 . 3 **Regardless of what the user inputs, the same number of** `OUTPUT` **instructions will always execute in the algorithm shown in FIGURE 8, on page 36.**

State how many `OUTPUT` **instructions will execute whenever the algorithm is run. [1 mark]**

0 7 . 4 **The phrase** `'Contact supplier'` **appears twice in the algorithm in FIGURE 8.**

State the TWO possible sequences of user input that would result in `'Contact supplier'` **being output. [2 marks]**

Sequence 1:

Sequence 2:

0 7 . 5 **Another developer looks at the algorithm shown in FIGURE 8 and makes the following statement.**

“At the moment if the user enters ‘y’ or ‘n’ they will sometimes get unexpected results. This problem could have been avoided.”

Explain why this problem has occurred and describe what would happen if a user entered ‘y’ or ‘n’ instead of ‘yes’ or ‘no’.

You may include references to line numbers in the algorithm where appropriate. You do NOT need to include any additional code in your answer. [3 marks]

[Turn over]

8

0 8 . 1 **State the comparisons that would be made if the binary search algorithm was used to search for the value 30 in the following array (array indices have been included above the array).**

0	1	2	3	4	5	6
1	6	14	21	27	31	35

[3 marks]

0 8 . 2 **For a binary search algorithm to work correctly on an array of integers, what property must be true about the array? [1 mark]**

[Turn over]

4

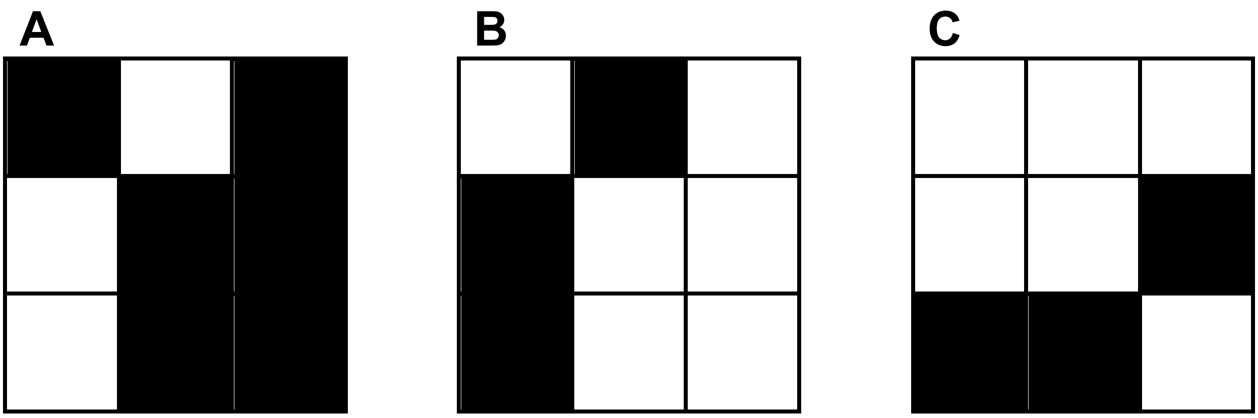
0 9 **A black and white image can be represented as a two-dimensional array where:**

- **0 represents a white pixel**
- **1 represents a black pixel.**

Two images are exact inverses of each other if:

- **every white pixel in the first image is black in the second image**
- **every black pixel in the first image is white in the second image.**

For example, B is the inverse of A but C is not the inverse of A:



A developer has started to create an algorithm that compares two 3x3 black and white images, `image1` **and** `image2`**, to see if they are exact inverses of each other.**

Complete the algorithm in pseudo-code, ensuring that, when the algorithm ends, the value of the variable `inverse` **is** `true` **if the**

two images are inverses of each other or `false` **if they are not inverses of each other.**

The algorithm should work for any 3x3 black and white images stored in `image1` **and** `image2`.

- **Note that indexing starts at zero.**

```
image1 ← [ [0, 0, 0], [0, 1, 1],
           [1, 1, 0] ]
image2 ← [ [1, 1, 1], [1, 1, 0],
           [0, 0, 1] ]
inverse ← true
i ← 0
WHILE i ≤ 2
   j ← 0
   WHILE j ≤ 2
```

[6 marks]

[Turn over]

BLANK PAGE

[Turn over]

6

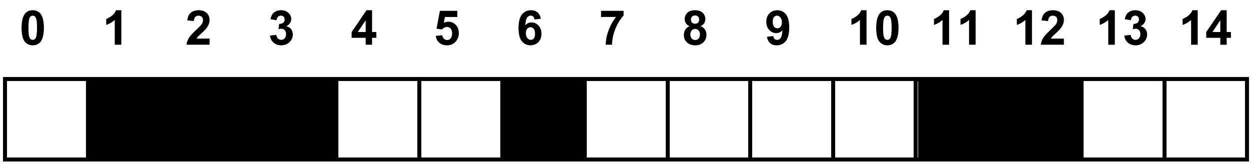
[Turn over]

1 0 A developer wants to simulate a simple version of the game of Battleships™. The ships are located on a one-dimensional array called `board`. There are always three ships placed on the board:

- one 'carrier' that has size three
- one 'cruiser' that has size two
- one 'destroyer' that has size one.

The size of the board is always 15 squares. A possible starting configuration is shown in FIGURE 9 where the indices are also written above the board.

FIGURE 9



The carrier, for example, is found at locations `board[1]`, `board[2]` **and** `board[3]`.

A player makes a guess to see if a ship (or part of a ship) is located at a particular location. If a ship is found at the location then the player has 'hit' the ship at this location.

Every value in the `board` array is 0, 1 or 2.

- **The value 0 is used to indicate an empty location.**
- **The value 1 is used to indicate if a ship is at this location and this location has NOT been hit.**
- **The value 2 is used to indicate if a ship is at this location and this location has been hit.**

The developer identifies one of the sub-problems and creates the subroutine shown in FIGURE 10.

FIGURE 10

```
SUBROUTINE F(board, location)
   h ← board[location]
   IF h = 1 THEN
      RETURN true
   ELSE
      RETURN false
   ENDIF
ENDSUBROUTINE
```

BLANK PAGE

10.1 **The subroutine in FIGURE 10, on page 51, uses the values** `true` **and** `false`**. Each element of the array** `board` **has the value** `0`, `1` **or** `2`.

State the most appropriate data type for these values. [2 marks]

VALUES	DATA TYPE
`true, false`	
`0, 1, 2`	

10.2 **The developer has taken the overall problem of the game Battleships and has broken it down into smaller sub-problems.**

State the technique that the developer has used. [1 mark]

[Turn over]

10.3 **The identifier for the subroutine in FIGURE 10, on page 51, is `F`. This is not a good choice. State a better identifier for this subroutine and explain why you chose it. [2 marks]**

New subroutine identifier: ____________________

____

Explanation: ______________________________

____

____

____

10 . 4 **The variable `h` in the subroutine in FIGURE 10, on page 51, is local to the subroutine. State TWO properties that only apply to local variables. [2 marks]**

[Turn over]

1 0 . 5 **Develop a subroutine that works out how far away the game is from ending.**

The subroutine should:

- **have a sensible identifier**
- **take the board as a parameter**
- **work out AND OUTPUT how many hits have been made**
- **work out how many locations containing a ship have yet to be hit and:**
  - **if 0 then output** `'Winner'`
  - **if 1, 2 or 3 then output** `'Almost there'`.

[11 marks]

END OF QUESTIONS

18

BLANK PAGE

BLANK PAGE

For Examiner's Use	
Question	Mark
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
TOTAL	

IB/M/IK/Jun19/8520/1/E4

1 9 6 G 8 5 2 0 / 1