AQA

SPECIMEN MATERIAL

GCSE
COMPUTER SCIENCE
8520/1

Paper 1

Specimen 2015 am/pm Time allowed: 1hr 30mins

Materials
There are no additional materials required for this paper.

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.
- Some questions will require you to shade a lozenge. If you make a mistake cross through the incorrect answer.
- Do all rough work in this book. Cross through any work that 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 marks for questions are shown in brackets.
- The maximum mark for this paper is 80.
- You are reminded of the need for good English and clear presentation in your answers.

Please write clearly, in block capitals, to allow character computer recognition.

Centre number ☐☐☐☐☐ Candidate number ☐☐☐☐

Surname

Forename(s)

Candidate signature ____________________

Answer **all** questions in the space provided.

0 1 Two computer programs that add together two numbers are shown in **Figure 1**. One is written in a high-level language and the other is written in a low-level language.

Figure 1

High-level program	**Low-level program**

`x = 12`
`y = 3`
`z = x + y`
	`0001 1100`
	`0010 0001`
	`0001 0011`
	`0010 0010`
	`0011 0001`
	`0010 0011`

0 1 . 1 Shade **two** lozenges to give the reasons why computer programs are most commonly written in high-level languages instead of low-level languages.

[2 marks]

A It saves the programmer time. ⬭

B The programs will always run faster. ⬭

C Program code written in high-level language is often easier for humans to understand. ⬭

D Computers understand only high-level languages. ⬭

E The programs are often easier for a computer to decode. ⬭

0 1 . 2 The low-level program shown in **Figure 1** is written in machine code.

Suggest **two** reasons why it would have been better for the programmer to use assembly language instead of machine code.

[2 marks]

Reason 1 ____

____

Reason 2 ____

____

4

0 2 **Figure 2** contains a subroutine that returns a value.

Figure 2

```
SUBROUTINE TotalOut(a, b)
  c ← a + b
  WHILE a < c
    a ← a + 1
    b ← b - a
  ENDWHILE
  RETURN b
ENDSUBROUTINE
```

0 2 . 1 Complete the trace table below when the subroutine call `TotalOut(3, 4)` is made (you may not need to use all of the rows in the table):

[3 marks]

a	b	c

0 2 . 2 A programmer mistakenly tries to shorten the subroutine in **Figure 2** by replacing the lines:

```
c ← a + b
WHILE a < c
```

With the line:

```
WHILE a < (a + b)
```

Explain why this change is a mistake.

[2 marks]

___

___

___

___

0 2 . 3 What value is returned by the subroutine call `TotalOut(x, 0)` where x is any positive integer?

[1 mark]

6

There are no questions printed on this page

Turn over for the next question

DO NOT WRITE ON THIS PAGE
ANSWER IN THE SPACES PROVIDED

0 3 Black and white bitmap images can be encoded using 1 to represent black and 0 to represent white and arranging the pixels in a two-dimensional array.

The algorithm in **Figure 3** calculates the number of black pixels in a bitmap image stored in a two-dimensional array called `image`, which has 4 rows and 3 columns.

Four parts of the code labelled **L1**, **L2**, **L3** and **L4** are missing.

Figure 3

- For this algorithm, array indexing starts at 1.

```
num_of_black ← L1
num_of_rows ← 4
num_of_cols ← 3
x ← 1
WHILE x L2 num_of_rows
  y ← 1
  WHILE y ≤ num_of_cols
    IF image[x][y] = L3 THEN
      num_of_black ← num_of_black + 1
    ENDIF
    y ← L4
  ENDWHILE
  x ← x + 1
ENDWHILE
```

0 3 . 1 Shade **one** lozenge to show which value should be written at point **L1** of the algorithm in **Figure 3**.

[1 mark]

A -1 ⬭

B 0 ⬭

C 1 ⬭

0 3 . 2 Shade **one** lozenge to show which operator should be written at point **L2** of the algorithm in **Figure 3**.

[1 mark]

A = ⬭

B ≥ ⬭

C ≤ ⬭

0 3 . 3 Shade **one** lozenge to show which value should be written at point **L3** of the algorithm in **Figure 3**.

[1 mark]

A -1 ⬭

B 0 ⬭

C 1 ⬭

0 3 . 4 Shade **one** lozenge to show what code should be written at point **L4** of the algorithm in **Figure 3**.

[1 mark]

A `x` ⬭

B `x + 1` ⬭

C `y` ⬭

D `y + 1` ⬭

4

0 4 The algorithm in **Figure 4** is the binary search algorithm designed to search for a value within an array.

Figure 4

- Line numbers are included but are not part of the algorithm.
- For this algorithm, array indexing starts at 1.

```
1    val ← 43
2    arr ← [3, 5, 13, 43, 655, 872]
3    left ← 1
4    right ← LENGTH(arr)
5    WHILE left ≠ right
6      mid ← (left + right) DIV 2
7      IF val ≤ arr[mid] THEN
8        right ← mid
9      ELSE
10       left ← mid + 1
11     ENDIF
12   ENDWHILE
```

0 4 . 1 Complete the trace table for the algorithm in **Figure 4** (you may not need to use all of the rows in the table). The final value of `left` is already given.

[5 marks]

val	**left**	**right**	**mid**	**arr[mid]**
	4			

0 4 . 2 Why would the binary search algorithm shown in **Figure 4** not work when the array `arr` contains `[5, 3, 13, 872, 655, 43]`?

[1 mark]

0 4 . 3 There are alternative statements that could have been used on line 5 of the algorithm shown in **Figure 4** that would not change the functionality of the algorithm.

Shade **one** lozenge to show which of the following lines could **not** replace line 5 in **Figure 4** as it would change the functionality of the algorithm.

[1 mark]

New Line 5

A `WHILE left < right`

B `WHILE NOT (left = right)`

C `WHILE left < right AND left > right`

D `WHILE left < right OR left > right`

Question 4 continues on the next page

0 4 . 4 The final value of `left` in the algorithm shown in **Figure** 4 is 4. A programmer realises that they can use this value to check whether `val` has been found or not in the algorithm shown in **Figure 4**.

The programmer wants to extend the algorithm and introduce a new variable called `found` that is `true` when the value has been found in the array or `false` otherwise.

Write the pseudo-code or draw the flowchart that is needed to **extend** the algorithm so that when the algorithm finishes, the new variable `found` is:

- `true` when `val` is found in `arr`
- `false` when `val` is not found in `arr`

This code should follow on from the end of the algorithm in **Figure 4**.

[4 marks]

11

0 5 . 1 Calculate the file size in bits for a two minute sound recording that has used a sample rate of 1000 Hertz (Hz) and a sample resolution of 5 bits.

You should show your working.

[3 marks]

0 5 . 2 Another sound file has a size of 24,000 bits. What is 24,000 bits in kilobytes?

You should show your working.

[2 marks]

Question 5 continues on the next page

0 5 . 3 Sound files are stored as bit patterns. Bit patterns are often compressed.

Compress the following bit pattern using run length encoding.

```
0000 0011 1111 1000 0000 0000 0111 1111
```

[4 marks]

0 5 . 4 Shade **one** lozenge which shows the **true** statement about run length encoding:

[1 mark]

A It will always make a file smaller. ⬭

B It is most effective on data that appears random. ⬭

C It will not lose any of the original data. ⬭

10

0 6 **Figure 5** shows the start of an algorithm.

Figure 5

```
OUTPUT 'enter the 24 hour number (0-23)'
hour ← USERINPUT
```

The algorithm in **Figure 5** asks the user to enter a number between 0 and 23 that represents an hour using the 24 hour clock. The input is stored in a variable called `hour`.

Extend the algorithm in **Figure 5**, using either pseudo-code or a flowchart, so that it outputs the equivalent time using the 12 hour clock, ie a number between 1 and 12, followed by either `am` or `pm`.

For example:

- If the user enters `0`, the program outputs `12` followed by `am`.
- If the user enters `4`, the program outputs `4` followed by `am`.
- If the user enters `12`, the program outputs `12` followed by `pm`.
- If the user enters `15`, the program outputs `3` followed by `pm`.

You can assume that the variable `hour` is an integer and that the value that the user inputs will be between 0 and 23.

[7 marks]

7

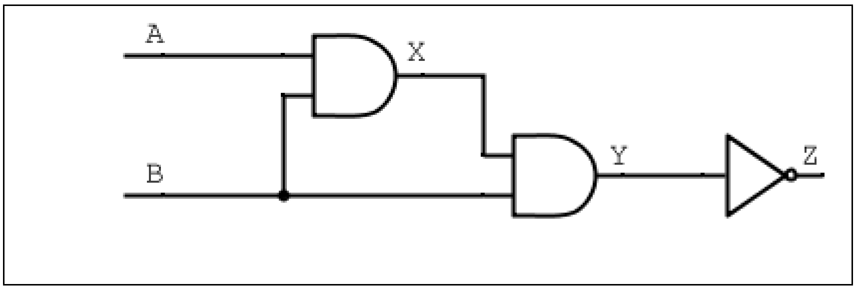
0 7 . 1 Complete the truth table for the OR logic gate:

[1 mark]

A	B	A OR B
0	0	
0	1	
1	0	
1	1	

0 7 . 2 Complete the truth table for the logic circuit shown in **Figure 6**.

Figure 6



[3 marks]

A	B	X	Y	Z
0	0			
0	1			
1	0			
1	1			

0 7 . 3 A logic circuit is being developed for an automatic door system:

- The automatic door has two sensors, one on either side of the door, sensor **F** and sensor **B.** The door opens when either of these sensors is activated.
- The door system can also be turned on/off using a manual switch, **S.** The door will not open unless **S** is on.
- The output from this logic circuit, for whether the door is open or not, is **D**.

Complete the logic circuit diagram for this system:

[3 marks]

7

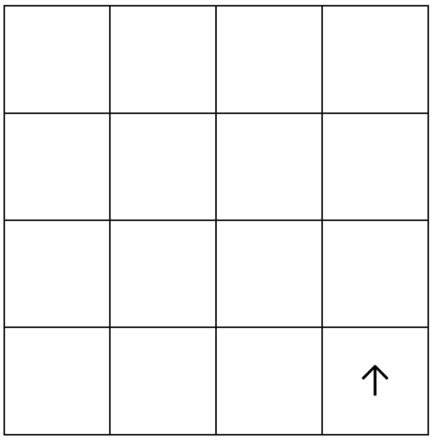
0 8 Four separate subroutines have been written to control a robot.

- `Forward(n)` moves the robot `n` squares forward.
- `TurnLeft()` turns the robot 90 degrees left
- `TurnRight()` turns the robot 90 degrees right
- `ObjectAhead()` returns `true` if the robot is facing an object in the next square or returns `false` if this square is empty.

0 8 . 1 Draw the path of the robot through the grid below if the following program is executed (the robot starts in the square marked by the ↑ facing in the direction of the arrow).

```
Forward(2)
TurnLeft()
Forward(1)
TurnRight()
Forward(1)
```

[3 marks]

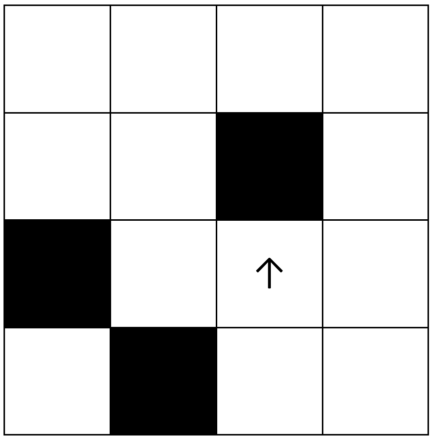


Question 8 continues on the next page

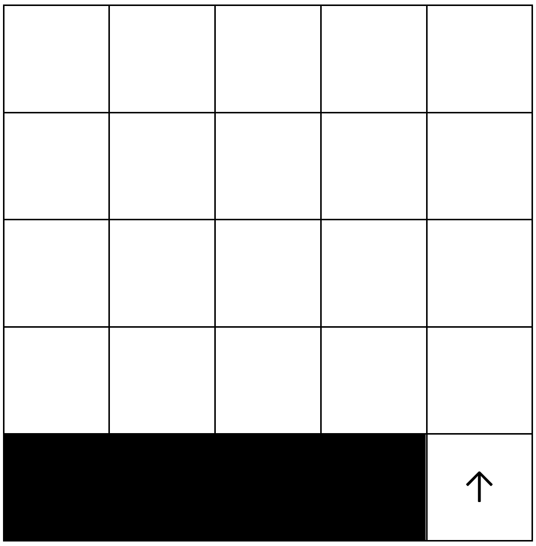
0 8 . 2 Draw the path of the robot through the grid below if the following program is executed (the robot starts in the square marked by the ↑ facing in the direction of the arrow). If a square is black then it contains an object.

```
WHILE ObjectAhead() = true
  TurnLeft()
    IF ObjectAhead() = true THEN
      TurnRight()
      TurnRight()
    ENDIF
  Forward(1)
ENDWHILE
Forward(1)
```

[3 marks]



0 8 . 3 A robot needs to visit every square in the following grid that does not contain an object:



The objects are shown as black squares.

Complete the algorithm by writing the following instructions in the correct places (you will need to use each instruction exactly once):

```
Forward(distance)
distance ← distance - 1
distance ← 4
TurnLeft()
TurnLeft()
```

```
_______________________________

WHILE distance > 0

    ___________________________

    ___________________________

    Forward(distance)

    ___________________________

    ___________________________

ENDWHILE
```

[4 marks]

10

0 9 **Figure 7** shows a famous algorithm that calculates the largest whole number that can be divided into two given numbers without leaving a remainder. For example, given the two numbers 12 and 16, 4 is the largest whole number that divides into them both.

Figure 7

Line numbers have been included but are not part of the algorithm.

```
1   num1 ← USERINPUT
2   num2 ← USERINPUT
3   WHILE num1 ≠ num2
4     IF num1 > num2 THEN
5       num1 ← num1 – num2
6     ELSE
7       num2 ← num2 – num1
8     ENDIF
9   ENDWHILE
```

0 9 . 1 Complete the trace table for the algorithm in **Figure 7** when the user enters 15 and then 39 (you may not need to use all of the rows in the table).

The first line has been completed for you.

[4 marks]

num1	num2
15	39

0 9 . 2 State the line number from the algorithm in **Figure 7** where selection is first used.

[1 mark]

0 9 . 3 State the line number from the algorithm in **Figure 7** where iteration is first used.

[1 mark]

0 9 . 4 How many lines in the algorithm in **Figure 7** use variable assignment?

[1 mark]

0 9 . 5 A programmer wants to implement the algorithm in **Figure 7** in their code. Explain why it could be a benefit to implement this as a subroutine.

[2 marks]

9

Turn over for the next question

1 0 The variables a, b and c in **Figure 8** are assigned string values.

Figure 8

```
a ← 'bitmap'
b ← 'pixel'
c ← 'bit'
```

1 0 . 1 Shade **one** lozenge which shows the concatenation of the variables a and b shown in **Figure 8.**

[1 mark]

concatenation of the variables a and b

A bitmap pixel ⬭

B bitmappixel ⬭

C ab ⬭

Strings can also be represented as arrays of characters. For instance, the three statements below are an alternative to the statements shown in **Figure 8** where those strings are now being represented as arrays of characters.

```
a ← ['b','i','t','m','a','p']
b ← ['p','i','x','e','l']
c ← ['b','i','t']
```

- For the following questions, array indexing starts at 1.

1 0 . 2 Shade **two** lozenges which correspond to the **two true** statements about these arrays.

[2 marks]

A a[1] = b[1] OR a[1] = c[1] ⬭

B LENGTH(b) ≥ LENGTH(a) ⬭

C NOT (a[6] = b[1] AND a[3] = c[3]) ⬭

D a = a[5] ⬭

E a[LENGTH(a) – LENGTH(c)] = c[3] ⬭

1 0 . 3 Develop a subroutine called `Prefix`, using either pseudo-code or a flowchart, which takes two character arrays called `word` and `pre` as parameters and determines whether the **start** of the character array `word` is the same as **all** of the character array `pre`.

For example using the character arrays:

```
a ← ['b','i','t','m','a','p']
b ← ['p','i','x','e','l']
c ← ['b','i','t']
```

- `a` starts with the same sequence of characters as `c` so `Prefix(a, c)` would return `true`
- `b` does not start with the same sequence of characters as `c` so `Prefix(b, c)` would return `false`.

Your subroutine should also:

- work for character arrays of all lengths greater than `0`
- **not** assume that the length of `pre` is less than the length of `word`.

The start of your subroutine has been completed for you.

```
SUBROUTINE Prefix(word, pre)
```

[9 marks]

12

END OF QUESTIONS

DO NOT WRITE ON THIS PAGE
ANSWER IN THE SPACES PROVIDED

Acknowledgement of copyright holders and publishers

Permission to reproduce all copyright material has been applied for. In some cases, efforts to contact copyright holders have been unsuccessful and AQA will be happy to rectify any omissions of acknowledgements in future papers if notified.