



---

# AS

# COMPUTER SCIENCE

## Paper 1

---

Time allowed: 1 hour 45 minutes

### Materials

For this paper you must have:

- a computer
- a printer
- appropriate software
- the Electronic Answer Document
- an electronic version and a hard copy of the Skeleton Program
- an electronic version and a hard copy of the Preliminary Material
- an electronic version of the Data Files **puzzle1.txt**, **puzzle1P.txt** and **puzzle1S.txt**

You must **not** use a calculator.

### Instructions

- Type the information required on the front of your Electronic Answer Document.
- Before the start of the examination make sure your **Centre Number**, **Candidate Name** and **Candidate Number** are shown clearly **in the footer** of every page (not the front cover) of your Electronic Answer Document.
- Enter your answers into the Electronic Answer Document.
- Answer **all** questions.
- Save your work at regular intervals.

### Information

- The marks for questions are shown in brackets.
- The maximum mark for this paper is 75.
- No extra time is allowed for printing and collating.
- The question paper is divided into **three** sections.

### Advice

You are advised to allocate time to each section as follows:

**Section A** – 20 minutes; **Section B** – 25 minutes; **Section C** – 60 minutes.

### At the end of the examination

Tie together all your printed Electronic Answer Document pages and hand them to the Invigilator.

### Warning

It may not be possible to issue a result for this paper if your details are not on every page of your Electronic Answer Document.

---

---

### Section A

You are advised to spend no more than **20 minutes** on this section.

Enter your answers to **Section A** in your Electronic Answer Document. You **must save** this document at regular intervals.

Question **03** in this section asks you to write program code **starting from a new program/project/file**.

You are advised to save your program at regular intervals.

---

0	1
---	---

The algorithm, represented using pseudo-code in **Figure 1**, describes a method to access numbers in the data structure, `List`, shown in **Table 1**.

**Figure 1**

```

SUBROUTINE A(S, X, Y)
  P ← -1
  WHILE P = -1 AND X ≤ Y
    Z ← (X + Y) DIV 2
    IF List[Z] = S THEN
      P ← Z
    ELSE
      IF List[Z] < S THEN
        X ← Z + 1
      ELSE
        Y ← Z - 1
      ENDIF
    ENDIF
  ENDWHILE
  RETURN P
ENDSUBROUTINE

```

The DIV operator calculates the whole number part resulting from an integer division, for example,  $10 \text{ DIV } 3 = 3$

Complete **Table 2** by hand-tracing the algorithm in **Figure 1** when the following statement is executed.

$$\text{Result} \leftarrow A(38, 0, 18)$$

You may not need to use all the rows in **Table 2**.

The first row of **Table 2** has already been completed for you.

**Table 1****List**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
2	8	12	18	25	29	36	42	49	51	57	61	68	71	79	83	84	91	97

**Table 2**

S	X	Y	P	Z	List[Z]
38	0	18	-1		
<b>Result:</b>					

Copy the contents of all the unshaded cells in **Table 2** into your Electronic Answer Document.

**[5 marks]****Turn over for the next question****Turn over ►**

0 2

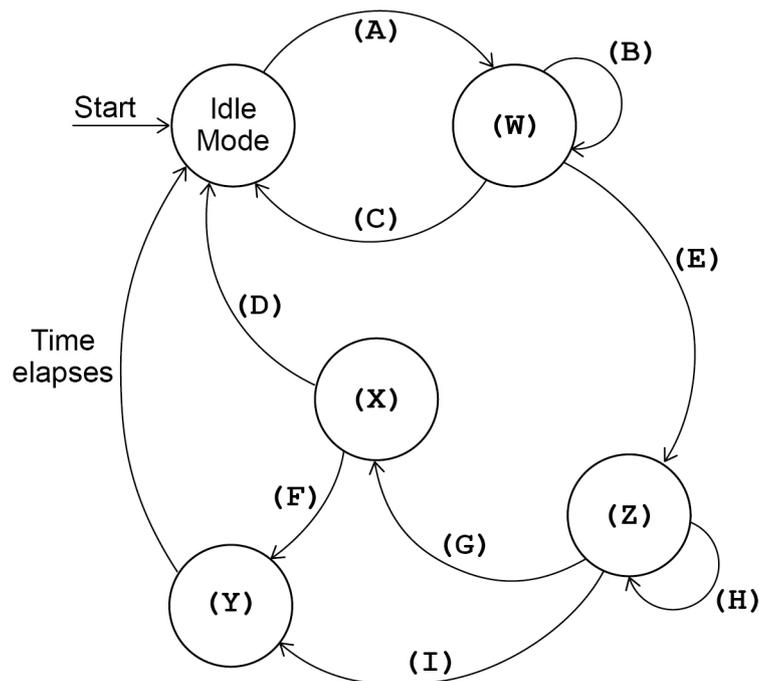
A parking meter has an Add hours button (+), an Accept button, a coin slot, a payment card reader, a Cancel button and a number keypad.

The system operates in a specific sequence:

- the system is initially in Idle Mode
- when the user presses the + button the system goes into Select Hours Mode with the parking time set to 1 hour and the payment owed set to £1.00
  - each time the user presses the + button again, the number of hours' parking time increases by 1 and the payment owed increases by £0.50
  - when the user presses the Accept button the system goes into Payment Due Mode and the user is able to make payments using cash or a payment card
  - the user can cancel the operation by pressing the Cancel button
- using cash:
  - each time the user inserts a coin (except the final coin), the value of it is deducted from the payment owed
  - when the final coin that completes the payment is inserted, the system goes into Paid Mode
- using a payment card:
  - when the user inserts a payment card into the card reader, the meter goes into a mode that allows the user to enter their PIN
  - the user then enters their PIN on the keypad
  - if the PIN is correct, the system goes into Paid Mode; otherwise the system goes into Idle Mode
- the system remains in Paid Mode until the time paid for has elapsed.

**Figure 2** shows a partially completed state transition diagram that represents the operation of the parking meter. Four of the states are labelled (W) to (Z) and events are labelled (A) to (I).

**Figure 2**



Complete **Table 3** by filling in the unshaded cells with the correct labels from **Figure 2**. You should write:

- which labels **(A)** to **(I)** represent which event(s)
- which labels **(w)** to **(z)** represent which state.

Some of the cells in the table may need to be assigned more than one label.

Each label **must** only be used once.

**Table 3**

Event / State	Label(s): (A) to (I), (w) to (z)
Card Payment Mode	
Enter correct PIN	
Enter incorrect PIN	
Insert a coin (except final coin)	
Insert final coin	
Insert payment card	
Paid Mode	
Payment Due Mode	
Press Accept	
Press Cancel	
Press + button	
Select Hours Mode	

Copy the contents of all the unshaded cells in **Table 3** into your Electronic Answer Document.

**[6 marks]**

**Turn over for the next question**

**Turn over ►**

**0 3****Figure 3** shows an algorithm represented using pseudo-code.**Figure 3**

```

C ← 0
D ← 0
S ← 0
T ← 0
WHILE C < 3 AND D < 3
    T ← T + 1
    N1 ← generate random integer between 1 and 6 inclusive
    N2 ← generate random integer between 1 and 6 inclusive
    OUTPUT N1, N2
    S ← S + N1 + N2
    IF N1 = 6 OR N2 = 6 THEN
        C ← C + 1
    ENDIF
    IF N1 = N2 THEN
        D ← D + 1
    ENDIF
ENDWHILE
A ← S DIV (T * 2)
OUTPUT C, D, A

```

The DIV operator calculates the whole number part resulting from an integer division, for example,  $10 \text{ DIV } 3 = 3$

**Table 4** lists the DIV operators for each of the available programming languages. You should refer to the row for your programming language.

**Table 4**

Programming language	DIV
C#	/
Java	/
Pascal	div
Python	//
VB.NET	\

**What you need to do:**

**Task 1**

Write a program to implement the algorithm in **Figure 3**.

**Task 2**

Test that your program works:

- run your program.

**Evidence that you need to provide**

Include the following evidence in your Electronic Answer Document.

**03.1** Your PROGRAM SOURCE CODE for **Task 1**. **[8 marks]**

**03.2** SCREEN CAPTURE(S) showing the test described in **Task 2**. **[1 mark]**

**Turn over for the next section**

**Turn over ►**

---

**Section B**

You are advised to spend no more than **25 minutes** on this section.

Enter your answers to **Section B** in your Electronic Answer Document. You **must save** this document at regular intervals.

These questions refer to the **Preliminary Material** and the **Skeleton Program**, but do **not** require any additional programming.

Refer **either** to the **Preliminary Material** issued with this question paper **or** your electronic copy.

---

**0 4 . 1** State the most appropriate data type to use for whole numbers. **[1 mark]**

**0 4 . 2** State the identifier of a variable in the **Skeleton Program** that can only represent whole numbers. **[1 mark]**

**0 5** State the identifier **and** data type of a variable in the **Skeleton Program** that can only represent two different values. **[2 marks]**

**0 6** The **Skeleton Program** uses the data structure `Answer`.

**0 6 . 1** State what the second element of this data structure, `Answer[1]`, is used for. **[1 mark]**

**0 6 . 2** Explain the purpose of the third element of this data structure, `Answer[2]`, **and** how it is used. **[3 marks]**

**0 7** The grid size of the puzzle is currently 9 and sub-grids are 3 x 3.

If the grid size were to be increased to 16, the sub-grids would be 4 x 4 and the hexadecimal digits 0 1 2 3 4 5 6 7 8 9 A B C D E F could be used to solve the puzzle.

Describe **two** changes that would need to be made to the subroutine `DisplayGrid` to enable this.

You should not make any changes to the **Skeleton Program** to answer this question. **[2 marks]**

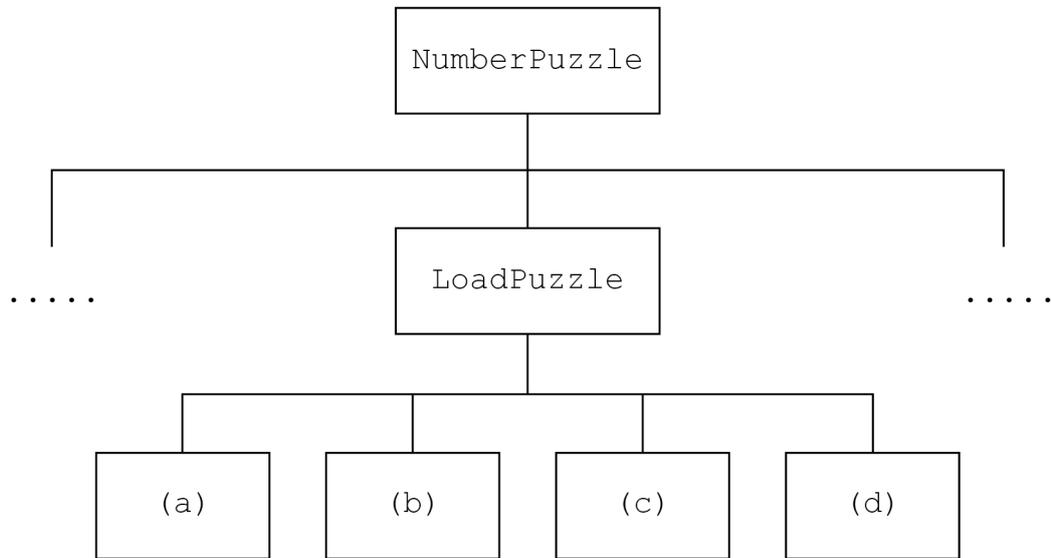
- 0 8** Explain what is meant by decomposition. **[2 marks]**
- 0 9**. **1** Explain the differences between definite and indefinite iteration. **[2 marks]**
- 0 9**. **2** State the identifier of a subroutine in the **Skeleton Program** that contains **definite** iteration. **[1 mark]**
- 0 9**. **3** State the identifier of a subroutine in the **Skeleton Program** that contains **indefinite** iteration. **[1 mark]**
- 1 0**. **1** Explain what exception handling is used for. **[2 marks]**
- 1 0**. **2** State the identifier of a subroutine in the **Skeleton Program** that performs exception handling **and** give an example of a circumstance that might cause an exception within that subroutine. **[2 marks]**
- 1 1**. **1** What is a subroutine? **[1 mark]**
- 1 1**. **2** The use of subroutines promotes code re-use.  
Describe, with references to subroutine(s) in the **Skeleton Program**, an example of how a subroutine has been re-used. **[1 mark]**
- 1 1**. **3** Describe an advantage of re-using subroutines. **[1 mark]**

**Question 11 continues on the next page**

**Turn over ►**

**1 1 . 4** Figure 4 shows an incomplete hierarchy chart for part of the **Skeleton Program**.

**Figure 4**



Complete **Table 5** by writing the labels that should appear in each of the boxes (a) to (d) in **Figure 4**.

**Table 5**

Box	Label
(a)	
(b)	
(c)	
(d)	

Copy the contents of all the unshaded cells in **Table 5** into your Electronic Answer Document.

**[2 marks]**

**Turn over for the next section**

**Turn over ►**

### Section C

You are advised to spend no more than **60 minutes** on this section.

Enter your answers to **Section C** in your Electronic Answer Document. You **must save** this document at regular intervals.

These questions require you to load the **Skeleton Program** and to make programming changes to it.

**1 2**

This question adds further validation to the **Skeleton Program**.

When a puzzle is loaded, some cells already contain numbers. These cells are referred to as protected cells.

**Figure 5** shows the numbers in the puzzle grid cells when **puzzle1** is first loaded. These are the protected cells for this puzzle.

**Figure 5**

	1	2	3	4	5	6	7	8	9
	===.===.===			===.===.===			===.===.===		
1	8	.	.	5	.	.	.	.	7
	.....			.....			.....		
2	9	.	.	5	.	.	4	.	.
	.....			.....			.....		
3	4	.	1	.	.	6	.	.	.
	===.===.===			===.===.===			===.===.===		
4	.	.	.	7	.	.	1	.	6
	.....			.....			.....		
5	1	.	.	4	.	.	6	.	3
	.....			.....			.....		
6	.	5	.	8	.	.	1	.	.
	===.===.===			===.===.===			===.===.===		
7	.	.	.	.	1	.	.	4	.
	.....			.....			.....		
8	.	.	.	2	.	.	7	.	1
	.....			.....			.....		
9	2	.	.	.	.	.	5	.	6
	===.===.===			===.===.===			===.===.===		

The **Skeleton Program** is to be changed so that the user cannot change the contents of any protected cells.

The subroutine `SolvePuzzle` needs to be modified so that the user cannot change a digit in a protected cell but can still enter a digit into an empty cell or change a digit that they have previously entered.

**What you need to do:****Task 1**

Amend the subroutine `SolvePuzzle` so that it checks every cell reference entered by the user against the protected cell references in `Puzzle` and only allows the contents of the cell to be changed if the cell referenced by `CellInfo` is not a protected cell.

If a protected cell is referenced, an appropriate error message should be displayed.

**Task 2**

Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter P
- load **puzzle1**
- enter S
- enter 117
- enter 323
- enter 993
- enter 853
- enter 854

**Evidence that you need to provide**

Include the following evidence in your Electronic Answer Document.

**1 2 . 1** Your PROGRAM SOURCE CODE for the entire subroutine `SolvePuzzle`.  
[7 marks]

**1 2 . 2** SCREEN CAPTURE(S) showing the requested test described in **Task 2**.  
The SCREEN CAPTURE(S) need(s) to show the puzzle grid before and after the described test.  
[1 mark]

**Turn over for the next question**

**Turn over ►**

1 3

This question adds further validation to the **Skeleton Program**. The subroutine `SolvePuzzle` asks the user to enter coordinates and a digit. The **Skeleton Program** is to be changed so that if the digit entered by the user already exists in the referenced row, column or sub-grid, the digit cannot be used.

**Figure 6** shows the numbers in the puzzle grid cells of `puzzle1` when first loaded.

**Figure 6**

	1	2	3	4	5	6	7	8	9
	===.===.===			===.===.===			===.===.===		
1	8	.	.	5	.	.	.	.	7
	.....			.....			.....		
2	9	.	.	5	.	.	4	.	.
	.....			.....			.....		
3	4	.	1	.	.	6	.	.	.
	===.===.===			===.===.===			===.===.===		
4	.	.	.	7	.	.	1	.	6
	.....			.....			.....		
5	1	.	.	4	.	.	6	.	3
	.....			.....			.....		
6	.	5	.	8	.	.	1	.	.
	===.===.===			===.===.===			===.===.===		
7	.	.	.	.	1	.	.	4	.
	.....			.....			.....		
8	.	.	.	2	.	.	7	.	1
	.....			.....			.....		
9	2	.	.	.	.	.	5	.	6
	===.===.===			===.===.===			===.===.===		

**What you need to do:****Task 1**

Write a new subroutine, `DuplicateDigit`, which takes `PuzzleGrid`, `Row`, `Column` and `Digit` as parameters. The subroutine should return `True` if the digit entered by the user is already present in the row, column or sub-grid of the cell the user has entered. Otherwise, the subroutine should return `False`.

**Task 2**

Amend the subroutine `SolvePuzzle` so that it uses `DuplicateDigit` to check the user input and only allows the digit to be placed into the puzzle grid if the digit is not already present in the row, column or sub-grid of the cell the user has entered. If the digit is already present, an appropriate message should be given to the user.

**Task 3**

Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter L
- load `puzzle1`
- enter S
- enter 178
- enter 819
- enter 124
- enter 989
- enter 555

**Evidence that you need to provide**

Include the following evidence in your Electronic Answer Document.

**1 3 . 1** Your PROGRAM SOURCE CODE for the entire subroutine `DuplicateDigit` and the entire subroutine `SolvePuzzle`. **[8 marks]**

**1 3 . 2** SCREEN CAPTURE(S) showing the requested test described in **Task 3**.  
The SCREEN CAPTURE(S) need(s) to show the puzzle grid before and after the described test. **[1 mark]**

**Turn over for the next question**

**Turn over ►**

**1 4**

This question extends the functionality of the **Skeleton Program**.

The user is to be allowed to clear the contents of the most recently changed cells. To do this, they will enter a negative integer when prompted to enter a row, column and digit. The integer will represent the number of cells to be cleared.

For example, if the user enters  $-3$ , the three most recently entered digits should each be replaced by a space in the puzzle grid and the `Answer` data structure should be updated.

If the number of cells to be cleared is greater than the number of entries in the `Answer` data structure, all digits entered by the user should be cleared.

### What you need to do:

#### Task 1

Write a new subroutine, `ClearEntries`, that clears the required number of previously entered digit(s) from the puzzle grid and updates the `Answer` data structure as described above.

#### Task 2

Amend the subroutine `SolvePuzzle` to test user input for a negative value. If the user input is a negative integer, the subroutine should call `ClearEntries` with the necessary parameters and then re-display the puzzle grid.

#### Task 3

Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter P
- load `puzzle1`
- enter S
- enter  $-x$
- enter  $-1$
- enter  $-5$

### Evidence that you need to provide

Include the following evidence in your Electronic Answer Document.

**1 4 . 1**

Your PROGRAM SOURCE CODE for the entire subroutine `ClearEntries` and the entire subroutine `SolvePuzzle`.

[12 marks]

**1 4 . 2**

SCREEN CAPTURE(S) showing the requested test described in **Task 3**.

The SCREEN CAPTURE(S) need(s) to show the puzzle grid before and after the described test.

[1 mark]

**END OF QUESTIONS**

**There are no questions printed on this page**

**There are no questions printed on this page**

**There are no questions printed on this page**

---

**There are no questions printed on this page**

**Copyright information**

For confidentiality purposes, all acknowledgements of third-party copyright material are published in a separate booklet. This booklet is published after each live examination series and is available for free download from [www.aqa.org.uk](http://www.aqa.org.uk).

Permission to reproduce all copyright material has been applied for. In some cases, efforts to contact copyright-holders may have been unsuccessful and AQA will be happy to rectify any omissions of acknowledgements. If you have any queries please contact the Copyright Team.

Copyright © 2022 AQA and its licensors. All rights reserved.

