AQA

A-level
COMPUTER SCIENCE
7517/1

Paper 1

Mark scheme

June 2024

Version: 1.0 Final

246A7517/1/MS

Mark schemes are prepared by the Lead Assessment Writer and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the students' responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation each associate analyses a number of students' scripts. Alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

No student should be disadvantaged on the basis of their gender identity and/or how they refer to the gender identity of others in their exam responses.

A consistent use of 'they/them' as a singular and pronouns beyond 'she/her' or 'he/him' will be credited in exam responses in line with existing mark scheme criteria.

Further copies of this mark scheme are available from aqa.org.uk

Level of response marking instructions

Level of response mark schemes are broken down into levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are marks in each level.

Before you apply the mark scheme to a student's answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student's answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level, ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The descriptors on how to allocate marks can help with this. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student's answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner's mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the Indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.

A-level Computer Science

Paper 1 (7517/1) – applicable to all programming languages A, B, C, D and E

June 2024

The following annotation is used in the mark scheme:

; – means a single mark
// – means an alternative response
/ – means an alternative word or sub-phrase
A. – means an acceptable creditworthy answer
R. – means reject answer as not creditworthy
NE. – means not enough
I. – means ignore
DPT. – means 'Don't penalise twice'. In some questions a specific error made by a candidate, if repeated, could result in the loss of more than one mark. The **DPT** label indicates that this mistake should only result in a candidate losing one mark, on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated.

Examiners are required to assign each of the candidate's responses to the most appropriate level according to **its overall quality**, and then allocate a single mark within the level. When deciding upon a mark in a level, examiners should bear in mind the relative weightings of the assessment objectives

eg

In question **07.1**, the marks available for the AO3 elements are as follows:

AO3 (design)	4 marks
AO3 (programming)	8 marks

Where a candidate's answer only reflects one element of the AO, the maximum mark they can receive will be restricted accordingly.

Question			Marks
01		**All marks AO1 (understanding)** **Example answers** Easier to test/debug as each subroutine can be tested separately; Easier to understand the code if sensible identifiers are used for subroutine names; Code can be easily reused as each subroutine is independent of rest of program; Makes it easier to work as a team of programmers as each subroutine can be worked on independently; Can be used as often as needed without having to write the code each time; Makes program easier to maintain/update (in future) as fewer changes will need to be made to make an update; Allows the use of recursive techniques because subroutines can call themselves; Easier to understand the code as each subroutine can be considered in isolation; Less likely to be errors in the code due to reuse of code; Reduces/eliminates side effects (eg unexpected change to value in global variable) through use of local variables; **Max 3** **Notes for examiners** Each advantage must be different. Mark should only be awarded if there is an explanation of how the advantage is achieved. Mark should only be awarded if the explanation is relevant for the stated advantaged.	**3**

Question			Marks
02	1	**All marks AO1 (understanding)** When an item is removed from a linear queue; **A.** by implication that problem arises when items have been deleted all other items need to be shuffled up one // this can result in unusable space in the array // this can result in only being able to add a limited number of items to the queue (eg only ever being able to add five items to a queue which uses an array of size 5); **Alternative answer** No need to shuffle items up one // no unusable spaces; After deleting an item from a circular queue;	2
02	2	**All marks AO1 (understanding)** 1. Check that the queue is not already empty // check to see if the current size is 0; **R.** reference to array instead of queue 2. if it is then deal with (underflow) error // (If it is not then) process/dequeue the front item in the queue; **R.** reference to array instead of queue (unless reference to array is made using value of front pointer) **R.** if dequeue would only happen under some of the correct circumstances 3. Reduce the value of the variable used to store the current size by 1; 4. Check if the front is in the last position of the array and if it is set it to the first position in the array (**A.** 0 or 1 instead of first position in the array); 5. (Else) add 1 to the value of the front pointer; **Alternative to mark points 1 and 3 if no current size variable is used:** calculate the current size; compare this to 0; **Alternative to mark points 1 and 3 if no current size variable is used:** check to see if the front/rear is –1;; **Alternative to mark points 4 and 5:** add 1 to the value of the front pointer; then set it to the remainder from dividing the value of front by the maximum size of the queue; **Alternative to mark points 4 and 5:** add 1 to the value of the front pointer; then if the pointer has passed the end of the array set it to point to the start of the array. **DPT.** Rear instead of front **Max 4 if any errors**	5

Question			Marks
03	1	**All marks AO1 (knowledge)** (Using a program/algorithm/method to) determine if a program will halt; **Max 1 for the following points, but only award mark if 1st mark was awarded**: without running the program; for a particular input;	**2**
03	2	**Mark is for AO1 (understanding)** It demonstrates that there are non-computable problems // it demonstrates that there are some problems for which there is no algorithm that can solve them // it demonstrates that there are undecidable problems;	**1**

Question			Marks
04	1	**Mark is for AO1 (knowledge)** The number of members of a set // the number of elements in a set; **A.** the size of a set	1
04	2	**Mark is for AO2 (apply)** The empty set is also a subset of **R** // {}/ Ø is also a subset of **R**; **NE.** they are not the only subsets of **R**	1
04	3	**Mark is for AO2 (apply)** 1 // one;	1
04	4	**Mark is for AO1 (understanding)** It means either the element (immediately) before or the element (immediately) after // or // alternation;	1

04	5	**All marks AO2 (apply)**
		
		`a(bb)*|(bb)+`
		
		Mark as follows:
		
		1 mark: expression contains `a(bb)*` **R.** `bb*`
		1 mark: expression contains `(bb)+` **R.** `bb+`
		Max 1 mark if any errors
		
		Alternative answer
		`a|a?(bb)+`
		
		Mark as follows:
		
		1 mark: expression contains `a|a?`
		1 mark: expression contains `(bb)+` **R.** `bb+`
		Max 1 mark if any errors
			2

		Alternative answer `(a|bb)(bb)*` **Mark as follows:** **1 mark:** expression contains `a|bb` **1 mark:** expression contains `(bb)*` **R.** `bb*` **Max 1 mark if any errors**	

04	**6**	**All marks AO2 (apply)**
		
		`(ab|b)(bb)*`
		
		Mark as follows:
		
		1 mark: expression contains `ab|b` // `(ab)|b`
		1 mark: expression contains `(bb)*` **R.** `bb*`
		Max 1 mark if any errors
		
		Alternative answer
		
		`a?b(bb)*`
		
		Mark as follows:
		
		1 mark: expression contains `a?b`
		1 mark: expression contains `(bb)*` **R.** `bb*`
		Max 1 mark if any errors
		
		Note for examiners
		Any regular expression that would match with an expression that starts with an optional `a` followed by a compulsory `b` should get (at least) one mark.
			2

Question			Marks
05	1	**Mark is for AO1 (understanding)** Reverse Polish (Notation) // RPN; **A.** Postfix	1
05	2	**All marks AO2 (apply)** `523*+4+;;` **Mark as follows:** **1 mark:** `23*` in expression; **1 mark:** correct order of operands with `+` symbols either side of the `4`; **Max 1 mark if any errors**	2

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>06</td><td>1</td><td>All marks AO1 (understanding)

Connected;
Undirected;

A. explanations of connected/undirected</td><td>2</td></tr>
<tr><td>06</td><td>2</td><td>All marks AO2 (apply)

<table>
<tr><th></th><th></th><th colspan="3">Temp</th><th></th><th></th></tr>
<tr><th>Done</th><th>Pos</th><th>[0]</th><th>[1]</th><th>[2]</th><th>Current</th><th>OUTPUT</th></tr>
<tr><td>False</td><td>-1</td><td></td><td></td><td></td><td>0</td><td></td></tr>
<tr><td></td><td>0</td><td>0</td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td>1</td><td></td><td>1</td><td></td><td>2</td><td></td></tr>
<tr><td></td><td>2</td><td></td><td></td><td>2</td><td>-1</td><td>Y</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>-1</td><td></td></tr>
<tr><td></td><td>1</td><td></td><td></td><td></td><td></td><td>K</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td></td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>1</td><td></td><td>3</td><td></td><td>-1</td><td>H</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>-1</td><td></td></tr>
<tr><td></td><td>0</td><td></td><td></td><td></td><td></td><td>U</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>4</td><td></td></tr>
<tr><td></td><td>-1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>0</td><td>4</td><td></td><td></td><td>-1</td><td>M</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>-1</td><td></td></tr>
<tr><td></td><td>-1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>True</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Mark as follows:

1. <code>Done</code> set to False, <code>Pos</code> set to –1 and <code>Current</code> set to 0
2. <code>Pos</code> set to 0, <code>Temp[0]</code> set to 0 and <code>Current</code> set to 1
3. <code>Pos</code> set to 1, <code>Temp[1]</code> set to 1 and <code>Current</code> set to 2
4. First output is Y
5. <code>Temp[2]</code> set to 2 with no other values after this, <code>Temp[1]</code> set to 3 with no other values after this and <code>Temp[0]</code> set to 4 with no other values after this;
6. <code>Done</code> column correct
7. Correct order for K, H, U and M in output column with no incorrect outputs

I. unnecessary repeated values in a column

Max 6 if any errors</td><td>7</td></tr>
</table>

06	**3**	**All marks AO2 (analyse)** There is no child node to the right of any node // each child node is to the left of its parent node;; **Note for examiners:** if answer is not correct then max one mark for any of the following points: • each node has, at most, one child node **R.** each node has one child node • there are no nodes with two child nodes • tree has a depth of five	**2**
06	**4**	**Mark is for AO2 (analyse)** Stack // LIFO (data structure);	**1**
06	**5**	**Mark is for AO2 (analyse)** Change the line `Current ← Dir1[Current]` to `Current ← Dir2[Current]` Change the line `Current ← Dir2[Temp[Pos]]` to `Current ← Dir1[Temp[Pos]];` // Change `Dir1` to `Dir2` and change `Dir2` to `Dir1`;	**1**

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>07</td><td>1</td><td>

4 marks for AO3 (design) and 8 marks for AO3 (programming)

Mark Scheme

<table>
<tr><th>Level</th><th>Description</th><th>Mark Range</th></tr>
<tr><td>4</td><td>A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met.</td><td>10–12</td></tr>
<tr><td>3</td><td>There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays relevant prompts, inputs the required data, has at least one iterative structure and at least one selection structure and uses appropriate variables to store most of the needed data. An attempt has been made to test for increasing and decreasing numbers, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made.</td><td>7–9</td></tr>
<tr><td>2</td><td>A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as, although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.</td><td>4–6</td></tr>
<tr><td>1</td><td>A program has been written and a few appropriate programming language statements have been written, but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised.</td><td>1–3</td></tr>
</table>

</td><td>12</td></tr>
</table>

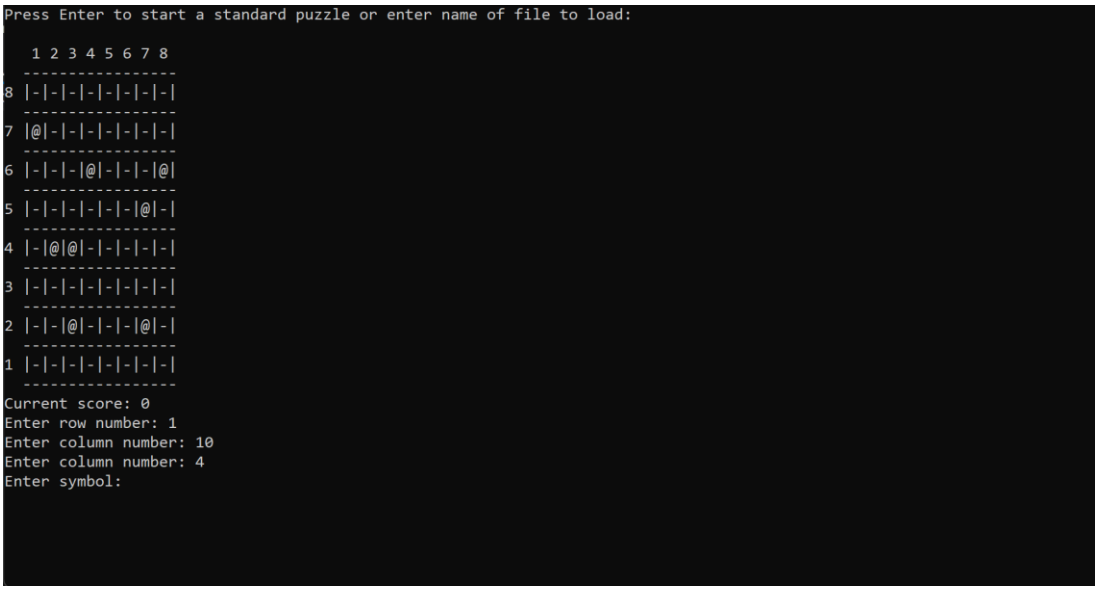
		Guidance **Evidence of AO3 design – 4 points:** Evidence of design to look for in responses: 1. Identifying that an iteration structure is needed that repeats a number of times based on the number of digits in the number entered by the user. 2. Identifying that selection structures (**A.** equivalent) for the three possible outcomes (bouncy, not bouncy, perfectly bouncy) is needed. 3. Identifying that variables with suitable data types are needed to store the number of digits followed by a larger digit and the number of digits followed by a smaller digit (**A.** any suitable equivalent). 4. Recognising the need to use input as an integer for the indefinite iteration and as a string to access individual digits // attempting to use remainder division with a power of ten. Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Evidence for AO3 programming – 8 points:** Evidence of programming to look for in response: 5. User input being assigned to appropriate variable. **A.** array of integers as long as at least 8 digits are allowed. 6. Indefinite iteration with correct condition containing attempt to get user input. 7. Iteration structure that repeats the correct number of times (one less than number of digits). 8. Compares two consecutive digits. 9. Selection structure with no incorrect contents for when next digit is larger than current digit. **R.** if any incorrect conditions. 10. Selection structure with no incorrect contents for when next digit is less than current digit. **R.** if any incorrect conditions. 11. Correctly detects that a number with all digits the same is an increasing number and correctly detects that a number with all digits the same is a decreasing number // correctly detects that a number with all digits the same is not a bouncy number 12. Selection structure(s) (**A.** equivalent) after iterative structure – for the three possible outcomes (bouncy, not bouncy, perfectly bouncy). **A.** would output messages that a perfectly bouncy number is perfectly bouncy and also bouncy. **R.** if bouncy number would result in output of perfectly bouncy. **R.** if no attempt made to detect bouncy number **R.** if no attempt made to detect perfectly bouncy number **Max 11** if any errors	

07	**2**	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		*Must match code from **07.1**, including messages on screen capture(s) matching those in code.*
		*Code for **07.1** must be sensible.*
		
		Screen captures showing the integer(s) entered and result(s) of each of the tests;
		I. order of tests
		
		`Enter a number: -3`
		`Enter a number: 14982`
		`perfectly bouncy`
		`>>>`
		`Enter a number: 1234`
		`not bouncy`
		`>>>`
		
		Note for examiners: example screen captures shown here match the order of the test data given in the question but there is no requirement for the tests to be done in any particular order.
			1

Question			Marks
08	1	**Mark is for AO1 (knowledge)** Joining (two) strings (to form one string // into one);	1
08	2	**Mark is for AO2 (analyse)** One // 1;	1
08	3	**Mark is for AO2 (analyse)** Zero // 0;	1

Question			Marks
09	1	**Mark is for AO2 (apply)** Because it is only called by methods inside the class // because it is never used/called from outside the class; **NE.** It is only called by/from the method `GetSymbol`	1
09	2	**Mark is for AO1 (understanding)** A local variable can only be used in the method/subroutine/program block in which it is declared whereas a private attribute can be used/accessed anywhere in the class;	1

Question			Marks
10	1	**Mark is for AO2 (analyse)** `LoadPuzzle` // `AttemptPuzzle`; **I.** case **A.** `__LoadPuzzle` (Python only) **A.** `__AttemptPuzzle` (Python only) **R.** if any other code included	1
10	2	**Mark is for AO2 (analyse)** It would not match with some valid patterns (if the iteration structures were not outside the exception handling structure) // it would not find patterns that start in the first/top two rows of the grid (if the iteration structures were not outside the exception handling code) // it would stop looking for patterns as soon as a location outside of the bounds of the array was checked (if the iteration structures were not outside the exception handling structure);	1
10	3	**All marks for AO2 (analyse)** To prevent the program crashing / terminating; when a cell outside the grid/list is checked; because it would be within a 3x3 grid with the cell the user has just entered a symbol in // because the user has just entered a symbol in a cell near the top or bottom of the grid; // To simplify the code structure; by avoiding the need to use many if statements/conditions; to avoid checking cells that are outside the grid/list; **Note for examiners** While the two answers are shown as alternatives, students could get marks by making valid points from both answers.	3
10	4	**Mark is for AO2 (analyse)** The grid is stored in a one-dimensional list // the top-left symbol of the 3x3 pattern is in the last two columns of the grid;	1

Question			Marks
11	1	**All marks for AO3 (programming)** 1. Selection structure with correct condition for one boundary; 2. Second correct condition and correct connecting logic; **A.** second selection structure 3. `Valid` set to true if the conditions for allowed values are met; **A.** any equivalent method that would ensure that iteration takes place in these circumstances 4. Selection structure(s) added in correct location in code; **Alternative answer** 1. Iteration structure modified with correct condition for one boundary; 2. Iteration structure modified with second correct condition; 3. Correct connecting logic for three conditions; 4. `Column` initialised to value that ensures code in loop executed at least once; **Max 3** if code contains errors **Max 3** if used value `8` instead of `GridSize`	4
11	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **11.1**.* *Code for **11.1** must be sensible.* Screen capture showing `1` is entered followed by `10` then `4` when asked to enter column number again, with `4` being accepted:  	1

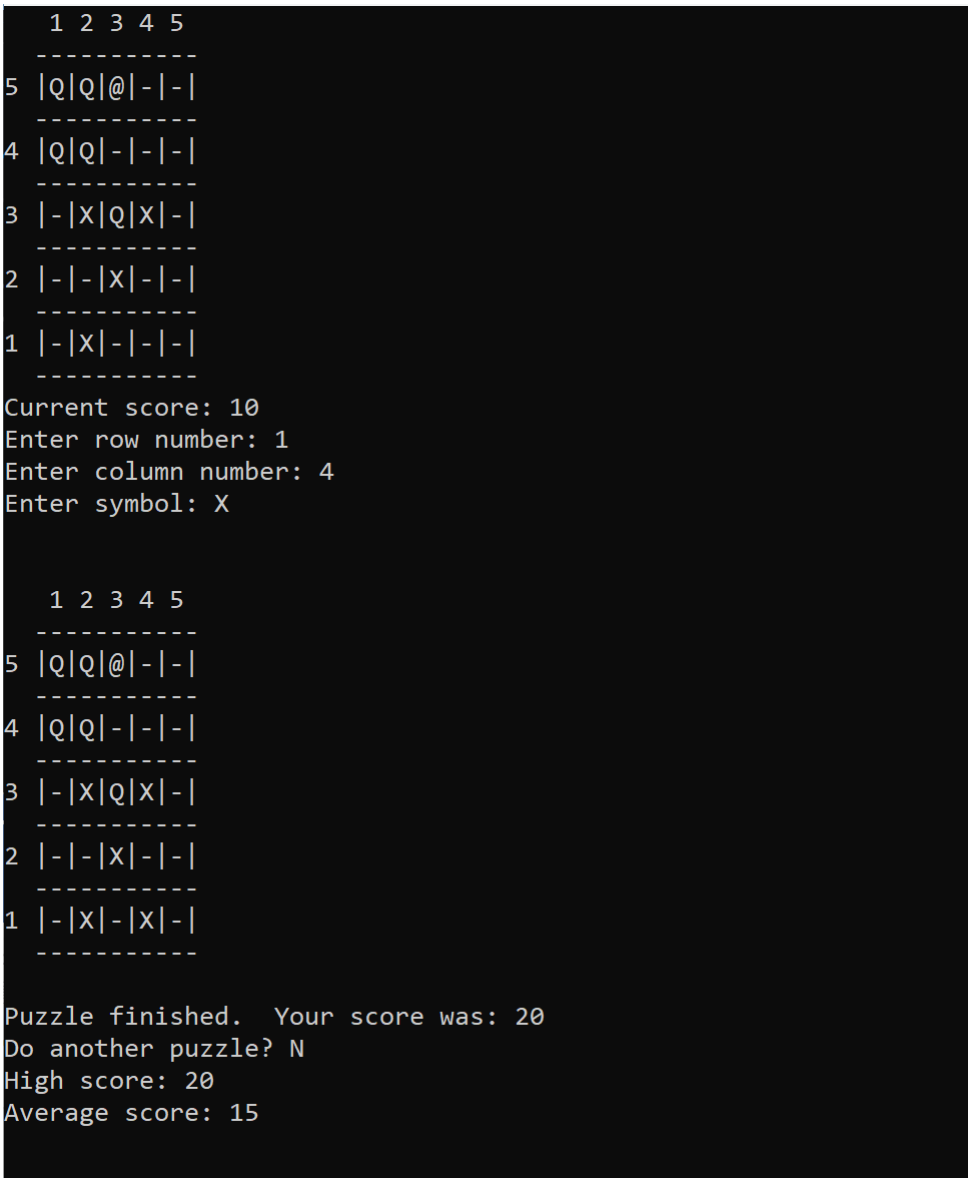
Question			Marks
12	1	**All marks for AO3 (programming)** 1. Create variables for average/total and highest, set to appropriate start values // creates a list to store all the scores for completed puzzles; **R.** if not in appropriate location **A.** equivalent variables 2. Compare final score for puzzle to highest score so far; 3. Change highest score found so far if condition for selection structure is met; **A.** incorrect condition 4. Calculate new average; **A.** calculate new total as long as there is an attempt to calculate average later in the code. 5. Display average and highest scores after iteration structure that checks if the user wants to do another puzzle; **A.** inside iterative stucture if also inside a selection structure that checks that the user does not want to do another puzzle **Max 4 marks** if code contains errors	**5**
12	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **12.1**.* *Code for **12.1** must be sensible.*	**1**

```
Press Enter to start a standard puzzle or enter name of file to load: puzzle1

   1 2 3 4 5
  -----------
5 |Q|Q|@|-|-|
  -----------
4 |Q|Q|-|-|-|
  -----------
3 |-|X|Q|X|-|
  -----------
2 |-|-|X|-|-|
  -----------
1 |-|X|-|-|-|
  -----------
Current score: 10
Enter row number: 5
Enter column number: 5
Enter symbol: X


   1 2 3 4 5
  -----------
5 |Q|Q|@|-|X|
  -----------
4 |Q|Q|-|-|-|
  -----------
3 |-|X|Q|X|-|
  -----------
2 |-|-|X|-|-|
  -----------
1 |-|X|-|-|-|
  -----------

Puzzle finished.  Your score was: 10
Do another puzzle? Y
Press Enter to start a standard puzzle or enter name of file to load: puzzle1
```



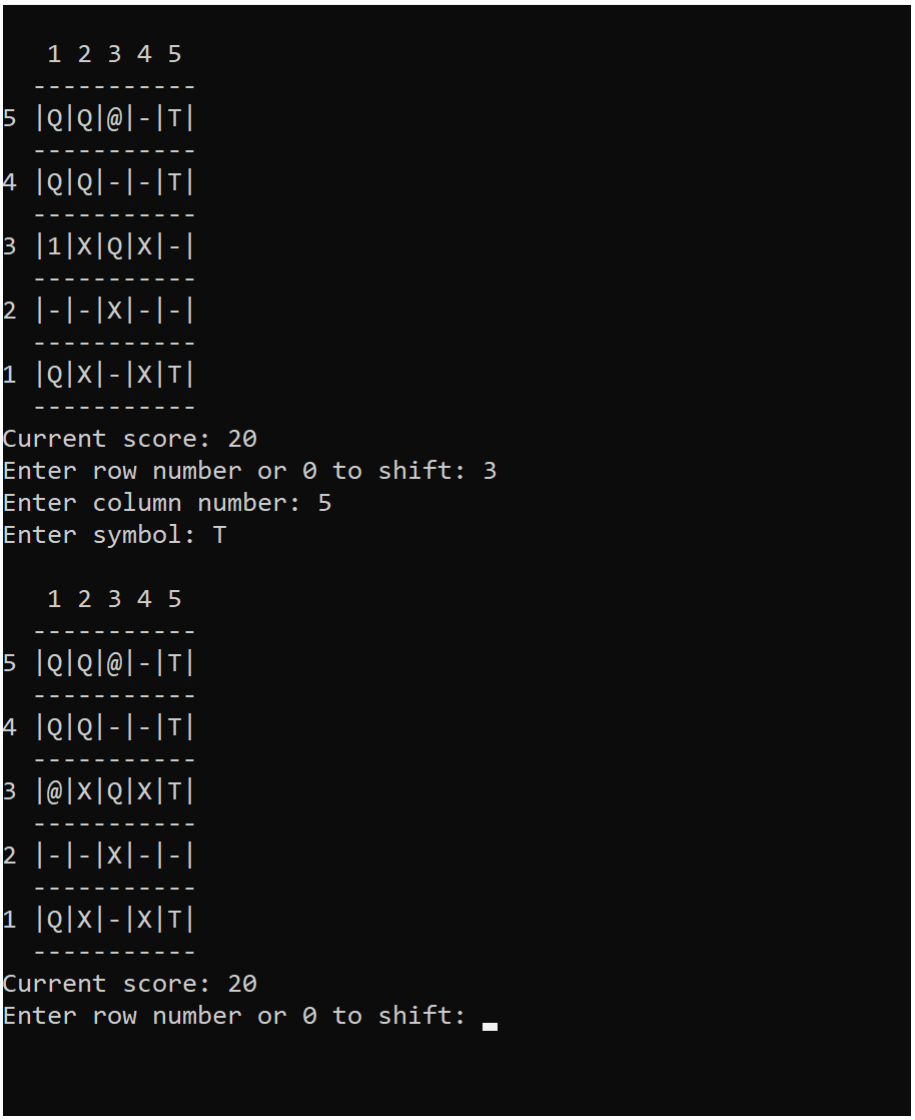
Screen capture(s) showing puzzle1 was used followed by correct high score and average score;

Question			Marks
13	1	**All marks for AO3 (programming)** Mark points 1 to 6 refer to the new method `ShiftCellsInRowLeft`; mark points 7 to 11 refer to `AttemptPuzzle`. 1. Create a new method called `ShiftCellsInRowLeft` with an integer parameter; **R.** other names for method; **I.** case and minor typos 2. Calculate the index of one cell in the specified row eg for the first cell in a row `(GridSize - Row) * GridSize`; **A.** correct use of `GetCell` 3. Store a cell in a temporary variable; 4. Iteration structure that repeats based on number of cells in a row (must result in attempting to inspect all but one cell in a row or attempting to inspect all cells in a row); 5. Move one cell in `Grid` one place to the left; 6. Moves the leftmost cell in the row to the end of the row; 7. Modified message about entering 0 to shift and selection structure that checks if 0 entered; **R.** if not in iterative structure that gets row from user 8. Gets the row to shift from the user; 9. Call to new method from `AttemptPuzzle` with correct parameter value; 10. Decrease score by 20; **R.** if not in selection structure for shifting cells option 11. Display new grid and new current score; **R.** if before score decreased **Max 2 marks** for mark points 2, 4, 6 if actual numeric value used instead of `GridSize` **Max 1 mark** for mark points 5, 6 if any cell would be lost during the shifting process **Max 10 marks** if code contains errors	11

13	**2**	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **13.1**.* *Code for **13.1** must be sensible.* Press Enter to start a standard puzzle or enter name of file to load: puzzle2 1 2 3 4 5 5 |Q|Q|@|-|-| 4 |Q|Q|-|-|-| 3 |-|X|Q|X|-| 2 |-|-|X|-|-| 1 |Q|X|-|-|T| Current score: 10 Enter row number or 0 to shift: 0 Enter row number to shift: 1 1 2 3 4 5 5 |Q|Q|@|-|-| 4 |Q|Q|-|-|-| 3 |-|X|Q|X|-| 2 |-|-|X|-|-| 1 |X|-|-|T|Q| Current score: -10 Enter row number or 0 to shift: 1 Enter column number: Screen capture(s) showing the correct final grid state and score for the user;	**1**
13	**3**	**Mark is for AO2 (apply)** (–1, 0);	**1**
13	**4**	**Mark is for AO2 (apply)** (GridSize - 1, 0);	**1**

Question			Marks
14	1	**All marks for AO3 (programming)** Mark points 1 to 7 relate to the `CountdownCell` class; mark points 8 to 13 relate to the `AttemptPuzzle` method. 1. Creating a new class called `CountdownCell`; **R.** other method identifiers **I.** case and minor typos 2. `CountdownCell` inherits from `BlockedCell` and has a constructor; 3. Constructor gives appropriate initial values for timer attribute and `Symbol`; ***A.*** Constructor gives appropriate initial values for timer attribute only, if *`GetSymbol`* overridden **A.** using `Symbol` as the timer if there is evidence of the numeric value in `Symbol` being decremented 4. Overrides `UpdateCell` method; 5. Decrease value of timer by 1 and update `Symbol` to match; 6. Selection structure that checks if value of timer is now 0 and if so changes `Symbol` to @; **A.** overriding `GetSymbol` to return `@` when timer is 0 7. `Symbol` doesn't change <u>again</u> after becoming `@` (eg if timer decreases to –1 it will stay the same); **R.** if inside selection structure that checks if timer value is equal to 0, unless other code prevents symbol changing on further iterations **R.** if no attempt to update the symbol with the timer values/use the symbol as a timer 8. Indefinite iteration structure that contains attempt to find an empty cell; 9. Generate random number in correct range; **R.** if range is based on a specific size for the grid 10. Checks if cell at random number position is empty; 11. Replaces cell in selected position with a `CountdownCell`; **R.** if more than one cell changed 12. Iteration structure that repeats a number of times equal to the number of cells in `Grid`; **R.** if only works with one size of `Grid` 13. Call to `UpdateCell` selected cell; **R.** if not in iteration structure for mark point 12 **Max 12** if code contains any errors	**13**

14	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		*Must match code from **14.1**, including prompts on screen capture matching those in code.*
		*Code for **14.1** must be sensible.*
		
		`Press Enter to start a standard puzzle or enter name of file to load: puzzle3`
		
		`  1 2 3 4 5`
		`  -----------`
		`5 |Q|Q|@|-|-|`
		`  -----------`
		`4 |Q|Q|-|-|-|`
		`  -----------`
		`3 |-|X|Q|X|-|`
		`  -----------`
		`2 |-|-|X|-|-|`
		`  -----------`
		`1 |Q|X|-|-|T|`
		`  -----------`
		`Current score: 10`
		`Enter row number or 0 to shift: 1`
		`Enter column number: 4`
		`Enter symbol: X`
		
		`  1 2 3 4 5`
		`  -----------`
		`5 |Q|Q|@|-|-|`
		`  -----------`
		`4 |Q|Q|-|-|-|`
		`  -----------`
		`3 |3|X|Q|X|-|`
		`  -----------`
		`2 |-|-|X|-|-|`
		`  -----------`
		`1 |Q|X|-|X|T|`
		`  -----------`
		`Current score: 20`
		`Enter row number or 0 to shift: 5`
		`Enter column number: 5`
		`Enter symbol: T`
		
		`  1 2 3 4 5`
		`  -----------`
		`5 |Q|Q|@|-|T|`
		`  -----------`
		`4 |Q|Q|-|-|-|`
		`  -----------`
		`3 |2|X|Q|X|-|`
		`  -----------`
		`2 |-|-|X|-|-|`
		`  -----------`
		`1 |Q|X|-|X|T|`
		`  -----------`
		`Current score: 20`
		`Enter row number or 0 to shift: 4`
		`Enter column number: 5`
		`Enter symbol: T`
			1



Screen capture(s) showing that a number 3 appeared in a cell, changed to 2, then 1, then @;

Note for examiners: the location of the cell that shows a 3 (then 2, then 1, then @) and the three cells containing Ts (apart from row 1, column 5) are likely to be different to those shown here.

VB.Net

Question			Marks

07	1	`Console.Write("Enter a number: ")`
		`Dim n As Integer = Console.ReadLine()`
		`While n <= 0`
		`  Console.Write("Enter a number: ")`
		`  n = Console.ReadLine()`
		`End While`
		`Dim Decreasing As Boolean = True`
		`Dim Increasing As Boolean = True`
		`Dim NoOfDecreasing As Integer = 0`
		`Dim NoOfIncreasing As Integer = 0`
		`Dim NumberAsString As String = n.ToString()`
		`For Count = 0 To NumberAsString.Length - 2`
		`  If NumberAsString(Count) < NumberAsString(Count + 1) Then`
		`    Decreasing = False`
		`    NoOfIncreasing += 1`
		`  ElseIf NumberAsString(Count) > NumberAsString(Count + 1) Then`
		`    Increasing = False`
		`    NoOfDecreasing += 1`
		`  End If`
		`Next`
		`If Not Decreasing And Not Increasing Then`
		`  If NoOfDecreasing = NoOfIncreasing Then`
		`    Console.WriteLine("perfectly bouncy")`
		`  Else`
		`    Console.WriteLine("bouncy")`
		`  End If`
		`Else`
		`  Console.WriteLine("not bouncy")`
		`End If`
			12

11	1	`Public Function AttemptPuzzle() As Integer`
		`  Dim Finished As Boolean = False`
		`  While Not Finished`
		`    ...`
		`    Dim Column As Integer`
		`    While Not Valid`
		`      Console.Write("Enter column number: ")`
		`      Try`
		`        Column = Console.ReadLine()`
		`        Valid = True`
		`        **If Column < 1 Or Column > GridSize Then**`
		`          **Valid = False**`
		`        **End If**`
		`      Catch`
		`      End Try`
		`    End While`
		`    ...`
			4

12	**1**	See below	**5**

```
Sub Main()
  Dim Again As String = "y"
  Dim Score As Integer
  Dim HighScore As Integer = 0
  Dim AverageScore As Single = 0
  Dim NumberOfPuzzles As Integer = 0
  While Again = "y"
    Console.Write("Press Enter to start a standard puzzle or enter
name of file to load: ")
    Dim Filename As String = Console.ReadLine()
    Dim MyPuzzle As Puzzle
    If Filename.Length > 0 Then
      MyPuzzle = New Puzzle(Filename & ".txt")
    Else
      MyPuzzle = New Puzzle(8, Int(8 * 8 * 0.6))
    End If
    Score = MyPuzzle.AttemptPuzzle()
    If Score > HighScore Then
      HighScore = Score
    End If
    AverageScore = AverageScore * NumberOfPuzzles + Score
    NumberOfPuzzles = NumberOfPuzzles + 1
    AverageScore = AverageScore / NumberOfPuzzles
    Console.WriteLine("Puzzle finished.  Your score was: " & Score)
    Console.WriteLine("Do another puzzle? ")
    Again = Console.ReadLine().ToLower()
  End While
  Console.WriteLine("High score: " & HighScore)
  Console.WriteLine("Average score: " & AverageScore)
  Console.ReadLine()
End Sub
```

13	**1**	**Public Sub ShiftCellsInRowLeft(ByVal Row As Integer)**
		** Dim FirstCell As Integer = (GridSize - Row) * GridSize**
		** Dim Temp As Cell = Grid(FirstCell)**
		** For Count = 0 To GridSize - 2**
		** Grid(FirstCell + Count) = Grid(FirstCell + Count + 1)**
		** Next**
		** Grid(FirstCell + GridSize - 1) = Temp**
		End Sub
		
		Public Function AttemptPuzzle() As Integer
		  ...
		  While Not Valid
		  Console.Write("Enter row number **or 0 to shift:** ")
		  Try
		  Row = Console.ReadLine()
		  Valid = True
		  Catch
		  End Try
		  If Row = 0 Then
		  Console.Write("Enter row number to shift: ")
		  Dim RowToShift As Integer = Console.ReadLine()
		  ShiftCellsInRowLeft(RowToShift)
		  Score = Score - 20
		  DisplayPuzzle()
		  Console.WriteLine("Current score: " & Score)
		  Valid = False
		  End If
		  End While
		  Dim Column As Integer
		  Valid = False
		...
			11

14	1	**Class CountdownCell**
		** Inherits BlockedCell**
		
		** Private Timer As Integer**
		
		** Sub New()**
		** Timer = 4**
		** Symbol = Timer**
		** End Sub**
		
		** Public Overrides Sub UpdateCell()**
		** Timer = Timer - 1**
		** If Timer < 1 Then**
		** Symbol = "@"**
		** Else**
		** Symbol = Timer**
		** End If**
		** End Sub**
		End Class
		
		Public Function AttemptPuzzle() As Integer
		  ...
		  Dim AmountToAddToScore As Integer =
		CheckForMatchWithPattern(RowNo, ColumnNo)
		  If AmountToAddToScore > 0 Then
		  Score += AmountToAddToScore
		  Dim GridPosition As Integer = -1
		  While GridPosition = -1
		  GridPosition = Rng.Next(Grid.Count)
		  If Grid(GridPosition).GetSymbol() <> "-" Then
		  GridPosition = -1
		  End If
		  End While
		  Grid(GridPosition) = New CountdownCell()
		  End If
		  End If
		  For Each c In Grid
		  c.UpdateCell()
		  Next
		  If SymbolsLeft = 0 Then
		  Finished = True
		  End If
		  End While
		  ...
			13

Python 3

Question			Marks

07	1	`n = int(input("Enter a number: "))`
		`while n <= 0:`
		`    n = int(input("Enter a number: "))`
		`Decreasing = True`
		`NoOfDecreasing = 0`
		`NoOfIncreasing = 0`
		`Increasing = True`
		`n = str(n)`
		`for count in range(len(n) - 1):`
		`    if n[count] < n[count + 1]:`
		`        Decreasing = False`
		`        NoOfIncreasing += 1`
		`    elif n[count] > n[count + 1]:`
		`        Increasing = False`
		`        NoOfDecreasing += 1`
		`if not Decreasing and not Increasing:`
		`    if NoOfDecreasing == NoOfIncreasing:`
		`        print("perfectly bouncy")`
		`    else:`
		`        print("bouncy")`
		`else:`
		`    print("not bouncy")`
			12

11	1	`def AttemptPuzzle(self):`
		`  Finished = False`
		`  while not Finished:`
		`    ...`
		`    while not Valid:`
		`      ...`
		`    Valid = False`
		`    while not Valid:`
		`      try:`
		`        Column = int(input("Enter column number: "))`
		`        Valid = True`
		`        **if Column < 1 or Column > self.__GridSize:**`
		`          **Valid = False**`
		`      except:`
		`        pass`
		`    Symbol = self.__GetSymbolFromUser()`
			4

12	**1**	`def Main():`
		`  Again = "y"`
		`  Score = 0`
		`  **HighScore = 0**`
		`  **AverageScore = 0**`
		`  **NumberOfPuzzles = 0**`
		`  while Again == "y":`
		`  FileName = input("Press Enter to start a standard puzzle or enter name of file to load: ")`
		`  if len(FileName) > 0:`
		`  MyPuzzle = Puzzle(FileName + ".txt")`
		`  else:`
		`  MyPuzzle = Puzzle(8, int(8 * 8 * 0.6))`
		`  Score = MyPuzzle.AttemptPuzzle()`
		`  **if Score > HighScore:**`
		`  **HighScore = Score**`
		`  **AverageScore = AverageScore * NumberOfPuzzles + Score**`
		`  **NumberOfPuzzles += 1**`
		`  **AverageScore = AverageScore / NumberOfPuzzles**`
		`  print("Puzzle finished. Your score was: " + str(Score))`
		`  Again = input("Do another puzzle? ").lower()`
		`  **print("High score: " + str(HighScore))**`
		`  **print("Average score: " + str(AverageScore))**`
			5

13	**1**	**`def __ShiftCellsInRowLeft(self, Row):`**
		`  FirstCell = (self.__GridSize - Row) * self.__GridSize`
		`  Temp = self.__Grid[FirstCell]`
		`  for Count in range (0, self.__GridSize - 1):`
		`  self.__Grid[FirstCell + Count] = self.__Grid[FirstCell + Count + 1]`
		`  self.__Grid[FirstCell + self.__GridSize - 1] = Temp`
		
		`def AttemptPuzzle(self):`
		`  Finished = False`
		`  while not Finished:`
		`  ...`
		`  while not Valid:`
		`  try:`
		`  Row = int(input("Enter row number **or 0 to shift**: "))`
		`  Valid = True`
		`  except:`
		`  pass`
		`  **if Row == 0:**`
		`  **RowToShift = int(input("Enter row number to shift: "))**`
		`  **self.__ShiftCellsInRowLeft(RowToShift)**`
		`  **self.__Score = self.__Score - 20**`
		`  **self.DisplayPuzzle()**`
		`  **print("Current score:", self.__Score)**`
		`  **Valid = False**`
		`  Valid = False`
		`  while not Valid:`
		`  ...`
			11

14	**1**	**class CountdownCell(BlockedCell):**
		** def __init__(self):**
		** super(CountdownCell, self).__init__()**
		** self.__Timer = 4**
		** self._Symbol = str(self.__Timer)**
		
		** def UpdateCell(self):**
		** self.__Timer -= 1**
		** if self.__Timer <= 0:**
		** self._Symbol = "@"**
		** else:**
		** self._Symbol = str(self.__Timer)**
		
		def AttemptPuzzle(self):
		  ...
		  if AmountToAddToScore > 0:
		  self.__Score += AmountToAddToScore
		  GridPosition = -1
		  while GridPosition == -1:
		  GridPosition = random.randrange(0, len(self.__Grid))
		  if self.__Grid[GridPosition].GetSymbol() != "-":
		  GridPosition = -1
		  self.__Grid[GridPosition] = CountdownCell()
		  for c in self.__Grid:
		  c.UpdateCell()
		  if self.__SymbolsLeft == 0:
		  Finished = True
			13

Python 2

Question			Marks

07	1	`n = int(raw_input("Enter a number: "))`
		`while n <= 0:`
		`    n = int(raw_input("Enter a number: "))`
		`Decreasing = True`
		`NoOfDecreasing = 0`
		`NoOfIncreasing = 0`
		`Increasing = True`
		`n = str(n)`
		`for count in range(len(n) - 1):`
		`    if n[count] < n[count + 1]:`
		`        Decreasing = False`
		`        NoOfIncreasing += 1`
		`    elif n[count] > n[count + 1]:`
		`        Increasing = False`
		`        NoOfDecreasing += 1`
		`if not Decreasing and not Increasing:`
		`    if NoOfDecreasing == NoOfIncreasing:`
		`        print "perfectly bouncy"`
		`    else:`
		`        print "bouncy"`
		`else:`
		`    print "not bouncy"`
			12

11	1	`def AttemptPuzzle(self):`
		`  Finished = False`
		`  while not Finished:`
		`    ...`
		`    while not Valid:`
		`      ...`
		`    Valid = False`
		`    while not Valid:`
		`      try:`
		`        Column = int(raw_input("Enter column number: "))`
		`        Valid = True`
		`        **if Column < 1 or Column > self.__GridSize:**`
		`          **Valid = False**`
		`      except:`
		`        pass`
		`    Symbol = self.__GetSymbolFromUser()`
			4

12	**1**	`def Main():`
		`  Again = "y"`
		`  Score = 0`
		`  HighScore = 0`
		`  AverageScore = 0`
		`  NumberOfPuzzles = 0`
		`  while Again == "y":`
		`  FileName = raw_input("Press Enter to start a standard puzzle or enter name of file to load: ")`
		`  if len(FileName) > 0:`
		`  MyPuzzle = Puzzle(FileName + ".txt")`
		`  else:`
		`  MyPuzzle = Puzzle(8, int(8 * 8 * 0.6))`
		`  Score = MyPuzzle.AttemptPuzzle()`
		`  if Score > HighScore:`
		`  HighScore = Score`
		`  AverageScore = AverageScore * NumberOfPuzzles + Score`
		`  NumberOfPuzzles += 1`
		`  AverageScore = AverageScore / NumberOfPuzzles`
		`  Print "Puzzle finished. Your score was: " + str(Score)`
		`  Again = raw_input("Do another puzzle? ").lower()`
		`  Print "High score: " + str(HighScore)`
		`  Print "Average score: " + str(AverageScore)`
			5

13	**1**	`def __ShiftCellsInRowLeft(self, Row):`
		`  FirstCell = (self.__GridSize - Row) * self.__GridSize`
		`  Temp = self.__Grid[FirstCell]`
		`  for Count in range (0, self.__GridSize - 1):`
		`  self.__Grid[FirstCell + Count] = self.__Grid[FirstCell + Count + 1]`
		`  self.__Grid[FirstCell + self.__GridSize - 1] = Temp`
		
		`def AttemptPuzzle(self):`
		`  Finished = False`
		`  while not Finished:`
		`  ...`
		`  while not Valid:`
		`  try:`
		`  Row = int(raw_input("Enter row number or 0 to shift: "))`
		`  Valid = True`
		`  except:`
		`  pass`
		`  if Row == 0:`
		`  RowToShift = int(raw_input("Enter row number to shift: "))`
		`  self.__ShiftCellsInRowLeft(RowToShift)`
		`  self.__Score = self.__Score - 20`
		`  self.DisplayPuzzle()`
		`  print "Current score: " + str(self.__Score)`
		`  Valid = False`
		`  Valid = False`
		`  while not Valid:`
		`  ...`
			11

14	1	**class CountdownCell(BlockedCell):**
		** def __init__(self):**
		** super(CountdownCell, self).__init__()**
		** self.__Timer = 4**
		** self._Symbol = str(self.__Timer)**
		
		** def UpdateCell(self):**
		** self.__Timer -= 1**
		** if self.__Timer <= 0:**
		** self._Symbol = "@"**
		** else:**
		** self._Symbol = str(self.__Timer)**
		
		def AttemptPuzzle(self):
		  ...
		  if AmountToAddToScore > 0:
		  self.__Score += AmountToAddToScore
		  GridPosition = -1
		  while GridPosition == -1:
		  GridPosition = random.randrange(0, len(self.__Grid))
		  if self.__Grid[GridPosition].GetSymbol() != "-":
		  GridPosition = -1
		  self.__Grid[GridPosition] = CountdownCell()
		  for c in self.__Grid:
		  c.UpdateCell()
		  if self.__SymbolsLeft == 0:
		  Finished = True
			13

C#

Question			Marks

07	1	`int n;`
		`do`
		`{`
		`    Console.WriteLine("Enter a number");`
		`    n = Convert.ToInt32(Console.ReadLine());`
		
		`} while (n <= 0);`
		`bool decreasing = true;`
		`bool increasing = true;`
		`int noDecreasing = 0;`
		`int noIncreasing = 0;`
		`string nStr = n.ToString();`
		`for (int i = 0; i < nStr.Length - 1; i++)`
		`{`
		`    if (nStr[i] < nStr[i + 1])`
		`    {`
		`        decreasing = false;`
		`        noIncreasing++;`
		`    }`
		`    else if (nStr[i] > nStr[i + 1])`
		`    {`
		`        increasing = false;`
		`        noDecreasing++;`
		`    }`
		`}`
		`if (!decreasing && !increasing)`
		`{`
		`    if (noIncreasing == noDecreasing)`
		`    {`
		`        Console.WriteLine("perfectly bouncy");`
		`    }`
		`    else`
		`    {`
		`        Console.WriteLine("bouncy");`
		`    }`
		`}`
		`else`
		`{`
		`    Console.WriteLine("not bouncy");`
		`}`
		`Console.ReadLine();`
			12

11	**1**	`public virtual int AttemptPuzzle()`
		`{`
		`  bool Finished = false;`
		`  while (!Finished)`
		`  {`
		`  ...`
		`  int Column = 0;`
		`  Valid = false;`
		`  while (!Valid)`
		`  {`
		`  Console.Write("Enter column number: ");`
		`  try`
		`  {`
		`  Column = Convert.ToInt32(Console.ReadLine());`
		`  Valid = true;`
		`  **if (Column < 1 || Column > GridSize)**`
		`  **{**`
		`  **Valid = false;**`
		`  **}**`
		`  }`
		`  catch (Exception)`
		`  {`
		`  }`
		`  }`
		`  ...`
			4

12	**1**		**5**

```
static void Main(string[] args)
{
    string Again = "y";
    int Score;
    int HighScore = 0;
    double AverageScore = 0;
    int NumberOfPuzzles = 0;
    while (Again == "y")
    {
        Console.Write("Press Enter to start a standard puzzle or
enter name of file to load: ");
        string Filename = Console.ReadLine();
        Puzzle MyPuzzle;
        if (Filename.Length > 0)
        {
            MyPuzzle = new Puzzle(Filename + ".txt");
        }
        else
        {
            MyPuzzle = new Puzzle(8, Convert.ToInt32(8 * 8 * 0.6));
        }
        Score = MyPuzzle.AttemptPuzzle();
        if (Score > HighScore)
        {
            HighScore = Score;
        }
        AverageScore = AverageScore * NumberOfPuzzles + Score;
        NumberOfPuzzles = NumberOfPuzzles + 1;
        AverageScore = AverageScore / NumberOfPuzzles;
        Console.WriteLine("Puzzle finished. Your score was: " +
Score);
        Console.Write("Do another puzzle? ");
        Again = Console.ReadLine().ToLower();
    }
    Console.WriteLine("High score: " + HighScore);
    Console.WriteLine("Average score: " + AverageScore);
    Console.ReadLine();
}
```

13	1	**public void ShiftCellsInRowLeft(int Row)**
		{
		int FirstCell = (GridSize - Row) * GridSize;
		Cell Temp = Grid[FirstCell];
		for (var Count = 0; Count <= GridSize - 2; Count++)
		{
		Grid[FirstCell + Count] = Grid[FirstCell + Count + 1];
		}
		Grid[FirstCell + GridSize - 1] = Temp;
		}
		
		public virtual int AttemptPuzzle()
		{
		...
		while (!Valid)
		{
		Console.Write("Enter row number **or 0 to shift:** ");
		try
		{
		Row = Convert.ToInt32(Console.ReadLine());
		Valid = true;
		}
		catch (Exception)
		{
		}
		if (Row == 0)
		{
		Console.Write("Enter row number to shift: ");
		int RowToShift =
		Convert.ToInt32(Console.ReadLine());
		ShiftCellsInRowLeft(RowToShift);
		Score = Score - 20;
		DisplayPuzzle();
		Console.WriteLine("Current score: " + Score);
		Valid = false;
		}
		}
		int Column = 0;
		Valid = false;
		...
			11

14	1	**class CountdownCell : BlockedCell**
		{
		private int Timer;
		
		public CountdownCell()
		{
		Timer = 4;
		Symbol = Timer.ToString();
		}
		
		public override void UpdateCell()
		{
		Timer = Timer - 1;
		if (Timer <= 0)
		Symbol = "@";
		else
		Symbol = Timer.ToString();
		}
		}
		
		public virtual int AttemptPuzzle()
		{
		...
		int AmountToAddToScore = CheckForMatchWithPattern(Row, Column);
		if (AmountToAddToScore > 0)
		{
		Score += AmountToAddToScore;
		int GridPosition = -1;
		while (GridPosition == -1)
		{
		GridPosition = Rng.Next(Grid.Count);
		if (Grid[GridPosition].GetSymbol() != "-")
		{
		GridPosition = -1;
		}
		}
		Grid[GridPosition] = new CountdownCell();
		}
		}
		foreach (var c in Grid)
		{
		c.UpdateCell();
		}
		if (SymbolsLeft == 0)
		Finished = true;
		}
		...
			13

Pascal/Delphi

Question			Marks

07	1	program Q07;
		
		{$APPTYPE CONSOLE}
		
		Uses SysUtils;
		
		var
		  n: integer;
		  digits: string;
		  increases, decreases: integer;
		  i: integer;
		
		begin
		  repeat
		  write('Enter a number: ');
		  readln(n);
		  until n > 0;
		  digits := IntToStr(n);
		  increases := 0;
		  decreases := 0;
		  for i := 2 to digits.Length do
		  begin
		  if digits[i] >= digits[i - 1] then
		  begin
		  inc(increases);
		  end;
		  if digits[i] <= digits[i - 1] then
		  begin
		  inc(decreases);
		  end;
		  end;
		  if (increases <> digits.Length - 1) and (decreases <>
		digits.Length - 1) then
		  begin
		  if increases = decreases then
		  begin
		  writeln(digits + ' is perfectly bouncy');
		  end
		  else
		  begin
		  writeln(digits + ' is bouncy');
		  end;
		  end
		  else
		  begin
		  writeln(digits + ' is not bouncy');
		  end;
		  readln;
		end.
			12

11 **1** **4**

```
function Puzzle.AttemptPuzzle(): integer;
...
    while not Valid do
    begin
      write('Enter column number: ');
      readln(ValueRead);
      Valid := integer.TryParse(ValueRead, Column);
      if Valid then
      begin
        Valid := (Column >= 1) and (Column <= GridSize);
      end;
    end;
...
```

12 **1** **5**

```
procedure Main();
var
  Again: string;
  Score: integer;
  Filename: string;
  MyPuzzle: Puzzle;
  HighestScore, TotalScore, PuzzlesAttempted: integer;

begin
  Randomize();
  Again := 'y';
  HighestScore := -1;
  TotalScore := 0;
  PuzzlesAttempted := 0;
  while Again = 'y' do
  begin
    write('Press Enter to start a standard puzzle or enter name of
file to load: ');
    readln(Filename);
    if Filename.Length > 0 then
    begin
      MyPuzzle := Puzzle.Create(Filename + '.txt');
    end
    else
    begin
      MyPuzzle := Puzzle.Create(8, Trunc(8 * 8 * 0.6));
    end;
    Score := MyPuzzle.AttemptPuzzle();
    inc(PuzzlesAttempted);
    if Score > HighestScore then
    begin
      HighestScore := Score;
    end;
    inc(TotalScore, Score);
    writeln('Puzzle finished. Your score was ' + IntToStr(Score));
    write('Do another puzzle? ');
    readln(Again);
    Again := LowerCase(Again);
  end;
  writeln('Highest score: ', HighestScore, ', average score:
', TotalScore / PuzzlesAttempted:3:1);
  readln;
end;
```

13	**1**		**11**

```
Puzzle = class
  private
    Score: integer;
    SymbolsLeft: integer;
    GridSize: integer;
    Grid: TList<Cell>;
    AllowedPatterns: TList<Pattern>;
    AllowedSymbols: TList<string>;
    procedure LoadPuzzle(FileName: string);
    function GetCell(Row: integer; Column: integer): integer;
    function CheckForMatchWithPattern(Row: integer; Column:
integer): integer; virtual;
    function GetSymbolFromUser(): string;
    function CreateHorizontalLine(): string;
    procedure ShiftCellsInRowLeft(Row: integer);
  public
    constructor Create(Filename: string); overload;
    constructor Create(Size: integer; StartSymbols: integer);
overload;
    function AttemptPuzzle(): integer; virtual;
    procedure DisplayPuzzle(); virtual;
end;

procedure Puzzle.ShiftCellsInRowLeft(Row: integer);
var
  RowStart: integer;
  LeftmostCell: Cell;
  i: integer;
begin
  RowStart := (GridSize - Row) * GridSize;
  LeftmostCell := Grid[RowStart];
  for i := RowStart to RowStart + GridSize - 2 do
  begin
    Grid[i] := Grid[i + 1];
  end;
  Grid[RowStart + GridSize - 1] := LeftmostCell;
end;

function Puzzle.AttemptPuzzle(): integer;
...
    while not Valid do
    begin
      write('Enter row number (0 to shift a row left): ');
      readln(ValueRead);
      Valid := integer.TryParse(ValueRead, Row);
      if Valid and (Row = 0) then
      begin
        write('Row number to shift left: ');
        readln(Row);
        ShiftCellsInRowLeft(Row);
        dec(Score, 20);
        DisplayPuzzle();
        writeln('Current score: ', Score);
        Valid := false;
      end;
    end;
    Valid := false;
    ...
```

14	1	**CountdownCell = Class(BlockedCell)**
		private
		** Timer: integer;**
		public
		** constructor Create();**
		** function GetSymbol(): string; override;**
		** procedure UpdateCell(); override;**
		end;
		
		constructor CountdownCell.Create();
		begin
		** Timer := 4;**
		end;
		
		function CountdownCell.GetSymbol(): string;
		begin
		** if Timer > 0 then**
		** begin**
		** exit(IntToStr(Timer));**
		** end**
		** else**
		** begin**
		** exit('@');**
		** end;**
		end;
		
		procedure CountdownCell.UpdateCell();
		begin
		** if Timer > 0 then**
		** begin**
		** dec(Timer);**
		** end;**
		end;
		
		procedure AddCountdownCell();
		var
		** i: integer;**
		begin
		** repeat**
		** i := random(Grid.Count);**
		** until Grid[i].IsEmpty();**
		** Grid[i] := CountdownCell.Create();**
		end;
		
		function Puzzle.AttemptPuzzle(): integer;
		var
		** c: Cell;**
		...
		  if Grid[ListIndex].CheckSymbolAllowed(Symbol) then
		  begin
		  Grid[ListIndex].ChangeSymbolInCell(Symbol);
		  AmountToAddToScore := CheckForMatchWithPattern(Row, Column);
		  if AmountToAddToScore > 0 then
		  begin
		  AddCountdownCell();
			13

```
          inc(Score, AmountToAddToScore);
        end;
      end;
      for c in Grid do
        begin
          c.UpdateCell();
        end;
      if SymbolsLeft = 0 then
        begin
          Finished := true;
        end;
    end;
...
```

Java

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>07</td><td>1</td><td>

```
int number = 0;
while (number <= 0) {
    Console.write("Enter a number: ");
    number = Integer.parseInt(Console.readLine());
}
boolean increasing = true;
boolean decreasing = true;
int noDecreasing = 0;
int noIncreasing = 0;
String numberStr = Integer.toString(number);
for (int i = 0; i < numberStr.length() - 1; i++) {
    if (numberStr.charAt(i) < numberStr.charAt(i + 1)) {
        decreasing = false;
        noIncreasing++;
    } else if (numberStr.charAt(i) > numberStr.charAt(i + 1)) {
        increasing = false;
        noDecreasing++;
    }
}
if (!decreasing && !increasing) {
    if (noIncreasing == noDecreasing) {
        Console.writeLine("perfectly bouncy");
    } else {
        Console.writeLine("bouncy");
    }
} else {
    Console.writeLine("not bouncy");
}
```

</td><td>12</td></tr>
<tr><td>11</td><td>1</td><td>

```
public int attemptPuzzle() {
      ...
      boolean finished = false;
      ...
      while (!finished) {
            ...
            while (!valid) {
                Console.write("Enter column number: ");
                valueRead = Console.writeLine();
                try {
                    column = Integer.parseInt(valueRead);
                    valid = true;
                    if (column < 1 || column > gridSize) {
                        valid = false;
                    }
                } catch (Exception e) {
                  valid = false;
                }
            }
            ...
      }
      ...
}
```

</td><td>4</td></tr>
</table>

12	**1**	`public static void main(String[] args) {`
		`    String again = "y";`
		`    int score;`
		`    **int highScore = 0;**`
		`    **float totalScore = 0;**`
		`    **int numberOfPuzzles = 0;**`
		`    while (again.equals("y")) {`
		`        Console.write("Press Enter to start a standard puzzle or enter name of file to load: ");`
		`        String filename = Console.readLine();`
		`        Puzzle myPuzzle;`
		`        if (filename.length() > 0) {`
		`            myPuzzle = new Puzzle(filename + ".txt");`
		`        } else {`
		`            myPuzzle = new Puzzle(8, (int)(8 * 8 * 0.6));`
		`        }`
		`        score = myPuzzle.attemptPuzzle();`
		`        **if (score > highScore) {**`
		`            **highScore = score;**`
		`        **}**`
		`        **totalScore += score;;**`
		`        **numberOfPuzzles += 1;**`
		`        Console.writeLine("Puzzle finished. Your score was: " + score);`
		`        Console.write("Do another puzzle? ");`
		`        again = Console.readLine().toLowerCase();`
		`    }`
		`    **Console.writeLine("High score: " + highScore);**`
		`    **Console.writeLine("Average score: " + (totalScore / numberOfPuzzles));**`
		`    Console.readLine();`
		`}`
			5

13	1	**public void shiftCellsInRowLeft(int row) {**
		int firstCell = (gridSize - row) * gridSize;
		Cell temp = grid.get(firstCell);
		for (int count = 0; count < gridSize - 2; count++) {
		grid.set(firstCell + count, grid.get(firstCell + count + 1));
		}
		grid.set(firstCell + gridSize - 1, temp);
		}
		
		public int attemptPuzzle() {
		...
		while (!valid) {
		Console.write("Enter row number **or 0 to shift:** ");
		valueRead = Console.readLine();
		try {
		row = Integer.parseInt(valueRead);
		valid = true;
		} catch (Exception e) {
		valid = false;
		}
		if (valid && row == 0) {
		Console.write("Enter row number to shift: ");
		int rowToShift = Integer.parseInt(Console.readLine());
		shiftCellsInRowLeft(rowToShift);
		score -= 20;
		displayPuzzle();
		Console.writeLine("Current score: " + score);
		valid = false;
		}
		}
		...
		}
			11

14	1	**class CountdownCell extends Cell {**
		private int timer;
		
		public CountdownCell() {
		timer = 4;
		symbol = "4";
		}
		
		@Override
		public void updateCell() {
		timer -= 1;
		if (timer <= 0) {
		symbol = "@";
		} else {
		symbol = Integer.toString(timer);
		}
		}
		}
		
		public int attemptPuzzle() {
		...
		int amountToAddToScore =
		checkForMatchWithPattern(row, column);
		if (amountToAddToScore > 0) {
		score += amountToAddToScore;
		int gridPosition = -1;
		while (gridPosition == -1) {
		gridPosition = getRandomInt(0, grid.size());
		if
		(!grid.get(gridPosition).getSymbol().equals("-")) {
		gridPosition = -1;
		}
		}
		grid.set(gridPosition, new CountdownCell());
		}
		}
		for (Cell c : grid) {
		c.updateCell();
		}
		if (symbolsLeft == 0) {
		finished = true;
		}
		}
		Console.writeLine();
		displayPuzzle();
		Console.writeLine();
		return score;
		}
			13