



---

AS  
**COMPUTER SCIENCE**  
**7516/1**

Paper 1

---

**Mark scheme**  
June 2019

---

Version: 1.0 Final

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 Assessment Writer.

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.

Further copies of this mark scheme are available from [aqa.org.uk](http://aqa.org.uk)

The following annotation is used in the mark scheme:

- ;** - means a single mark
- //** - means alternative response
- /** - means an alternative word or sub-phrase
- A** - means 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.

Page 5 – 18 contain the generic mark scheme.

Pages 19 to 39 contain the 'Program Source Codes' specific to the programming languages for questions 03.1, 14.1, 15.1, 16.1 and 17.2

- pages 20 to 23 – VB.NET
- pages 24 to 26 – PYTHON 2
- pages 27 to 29 – PYTHON 3
- pages 30 to 34 – PASCAL/Delphi
- pages 35 to 39 – C#
- pages 40 to 43 – JAVA

## 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.

Examiners are required to assign each of the candidates' responses to the most appropriate level according to **its overall quality**, 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 17.1, the marks available for the AO3 elements are as follows:

AO3 (design) – 2 marks

AO3 (programming) – 7 marks

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

Qu		Marks																																																																																														
01	1	<p><b>All marks for AO1 (knowledge)</b></p> <p>Difference:                      global variables accessible to all parts of the program                      // declared in main program block                      // local variables declared in subroutine                      // accessible only in the program block/subroutine in which it was declared;</p> <p>Reason:                      memory allocated to local variables can be reused when subroutine not in use;                      local variable only exists while the program block/subroutine is executing;                      using local variables makes subroutines self-contained;  <b>A prevents accidental changes;</b>  <b>A easier debugging/maintenance;</b>  <b>Max 2</b></p>	3																																																																																													
02	1	<p><b>All marks for AO2 (apply)</b></p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th rowspan="2">x</th> <th rowspan="2">MyValue</th> <th rowspan="2">y</th> <th rowspan="2">y &gt; -1 ? (True/False)</th> <th rowspan="2">Numbers [y]</th> <th rowspan="2">Numbers [y] &lt; MyValue ? (True/False)</th> <th colspan="3">Numbers</th> </tr> <tr> <th>[0]</th> <th>[1]</th> <th>[2]</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>43</td> <td>17</td> <td>85</td> </tr> <tr> <td>1</td> <td>17</td> <td>0</td> <td>True</td> <td>43</td> <td>False</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>(17)</td> <td></td> </tr> <tr> <td>2</td> <td>85</td> <td>1</td> <td>True</td> <td>17</td> <td>True</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>17</td> </tr> <tr> <td></td> <td></td> <td>0</td> <td>True</td> <td>43</td> <td>True</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>43</td> <td></td> </tr> <tr> <td></td> <td></td> <td>-1</td> <td>False</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>85</td> <td></td> <td></td> </tr> </tbody> </table> <p><b>1 mark</b> for correct x column and MyValue column;  <b>1 mark</b> for correct y column (0, 1, 0, -1);  <b>1 mark</b> for correct Boolean values in columns 4 and 6;  <b>A. TRUE&gt;true, FALSE/false, Yes/No, Y/N and any other suitable indicators</b>  <b>1 mark</b> for final contents of Numbers correct;</p>	x	MyValue	y	y > -1 ? (True/False)	Numbers [y]	Numbers [y] < MyValue ? (True/False)	Numbers			[0]	[1]	[2]							43	17	85	1	17	0	True	43	False											(17)		2	85	1	True	17	True												17			0	True	43	True											43				-1	False												85			4
x	MyValue	y							y > -1 ? (True/False)	Numbers [y]	Numbers [y] < MyValue ? (True/False)	Numbers																																																																																				
			[0]	[1]	[2]																																																																																											
						43	17	85																																																																																								
1	17	0	True	43	False																																																																																											
							(17)																																																																																									
2	85	1	True	17	True																																																																																											
								17																																																																																								
		0	True	43	True																																																																																											
							43																																																																																									
		-1	False																																																																																													
						85																																																																																										
02	2	<p><b>Mark is for AO2 (analyse)</b></p> <p>sort from largest to smallest;  <b>NE</b> Sort on its own  <b>A</b> bubble sort;</p>	1																																																																																													

03	1	<p><b>All marks for AO3 (programming)</b></p> <p><b>Mark as follows:</b></p> <ol style="list-style-type: none"> <li>1) Correct variable declarations for NumberIn, NumberOut, Count, PartValue;</li> </ol> <p><b>Note to examiners</b> If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the correct variables exist in the program code and the first value they are assigned is of the correct data type.</p> <ol style="list-style-type: none"> <li>2) Correct prompt "Enter a positive whole number: " and NumberIn assigned value entered by user;</li> <li>3) Correct initialisation of NumberOut and Count;</li> <li>4) WHILE loop with syntax allowed by the programming language and correct condition for termination of the loop;</li> <li>5) Correct incrementation of Count within WHILE loop;</li> <li>6) Correct assignment to PartValue within WHILE loop but before FOR loop;</li> <li>7) Correct updating of NumberIn within WHILE loop but before FOR loop;</li> <li>8) FOR loop with syntax allowed by the programming language over correct range;</li> <li>9) Correct assignment to PartValue inside FOR loop;</li> <li>10) Correct calculation of NumberOut after FOR loop but within WHILE loop;</li> <li>11) Output statement giving correct output after WHILE loop;</li> </ol> <p>I. Ignore minor differences in case and spelling</p> <p><b>Max 10</b> if code does not function correctly</p>	11
03	2	<p><b>Mark is for AO3 (evaluate)</b></p> <p>**** SCREEN CAPTURE ****</p> <p>Must match code from 03.1, including prompts on screen capture matching those in code. Code for 03.1 must be sensible.</p> <p>Screen capture showing: '22' being entered and the message 'The result is: 10110' displayed '29' being entered and the message 'The result is: 11101' displayed '-1' being entered and the message 'The result is: 0' displayed</p> <pre> Enter a positive whole number: 22 The result is: 10110 &gt;&gt;&gt; Enter a positive whole number: 29 The result is: 11101 &gt;&gt;&gt; Enter a positive whole number: -1 The result is: 0 &gt;&gt;&gt; </pre>	1
03	3	<p><b>Mark is for AO2 (analyse)</b></p> <p>converts from (positive) decimal/denary to binary;</p>	1

04	1	<p><b>Mark is for AO1 (understand)</b></p> <pre>Valid /ValidPiece /ValidMove /Found /EndOfList /Jumping /GameOver /FileFound; A CanJump;</pre> <p>R. if any additional code R. if spelt incorrectly I. case &amp; spacing</p>	1
04	2	<p><b>Mark is for AO1 (understand)</b></p> <pre>ValidMove /ValidJump /ListOfMoves;</pre> <p>A. setUpBoard (for Java only) R. if any additional code R. if spelt incorrectly I. case &amp; spacing</p>	1
05		<p><b>Mark is for AO1 (understand)</b></p> <pre>MoveRecord /ListOfMoves;</pre> <p>R. if any additional code R. if spelt incorrectly I. case &amp; spacing</p>	1
06		<p><b>Mark is for AO1 (understand)</b></p> <p>catch any <u>file</u> errors // stop program crashing if <u>file doesn't exist</u>;</p>	1
07		<p><b>Mark is for AO2 (analyse)</b></p> <p>positions of player A's pieces; A the contents of (the data structure/variable) A // pointer/address to A // A;</p>	1

08		<p><b>All marks for AO1 (understand)</b></p> <table border="1" data-bbox="347 282 1283 788"> <thead> <tr> <th data-bbox="347 282 459 336">Label</th> <th data-bbox="459 282 1283 336">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="347 336 459 450">(a)</td> <td data-bbox="459 336 1283 450">no move possible (for player A)</td> </tr> <tr> <td data-bbox="347 450 459 564">(b)</td> <td data-bbox="459 450 1283 564">Player B moves</td> </tr> <tr> <td data-bbox="347 564 459 678">(c)</td> <td data-bbox="459 564 1283 678">Player B's turn</td> </tr> <tr> <td data-bbox="347 678 459 788">(d)</td> <td data-bbox="459 678 1283 788">no move possible (for player B)</td> </tr> </tbody> </table> <p><b>1 mark</b> for 2 correct labels  <b>2 marks</b> for 4 correct labels</p>	Label	Description	(a)	no move possible (for player A)	(b)	Player B moves	(c)	Player B's turn	(d)	no move possible (for player B)	2
Label	Description												
(a)	no move possible (for player A)												
(b)	Player B moves												
(c)	Player B's turn												
(d)	no move possible (for player B)												
09	1	<p><b>Mark is for AO2 (analyse)</b></p> <p>DisplayBoard;</p> <p><b>R.</b> if any additional code  <b>R.</b> if spelt incorrectly  <b>I.</b> case &amp; spacing</p>	1										
09	2	<p><b>Mark is for AO2 (analyse)</b></p> <p>PrintResult;</p> <p><b>R.</b> if any additional code  <b>R.</b> if spelt incorrectly  <b>I.</b> case &amp; spacing</p>	1										
09	3	<p><b>Mark is for AO2 (analyse)</b></p> <p>PrintLine;  <b>A.</b> PrintRow / PrintMiddleRow;</p> <p><b>Max 1</b>  <b>R.</b> if any additional code  <b>R.</b> if spelt incorrectly  <b>I.</b> case &amp; spacing</p>	1										



10	1	<p><b>All marks for AO2 (analyse)</b></p> <p>(row 0 column 0) is used to store the number of moves;                  (row 0 column 1) is used to store the number of pieces promoted to dames;</p>	2
10	2	<p><b>Mark for AO2 (analyse)</b></p> <p>There are (a maximum of) 12 pieces per player // each row stores data for each piece;</p>	1
10	3	<p><b>All marks for AO2 (analyse)</b></p> <p>rows 1 to 12 (in columns 0 and 1) store the coordinates/<b>location</b> of the pieces on the board;                  if coordinates are -1 then indicates no piece;                  (column 2) indicates if the piece is a dame // <b>indicates state of each piece</b>;  <b>Max 2</b></p>	2
11		<p><b>1 mark is for AO1 (understand)</b></p> <p>it checks whether the sum of row and column are an even number;</p> <p><b>2 marks for AO2(analyse)</b></p> <p>to blank out a square if it can't be used;                  to store a space if it can be used;  <b>A for 1 mark: creates the checker board pattern;</b></p>	3
12		<p><b>All marks for AO2 (analyse)</b></p> <p>it counts the number of moves that are possible at the current state of play;                  it acts as the index for the data structure <code>ListOfMoves</code>;</p>	2
13		<p><b>All marks for AO2 (analyse)</b></p> <ol style="list-style-type: none"> <li>1) User is asked to enter a Piece ID;</li> <li>2) the <code>ListOfMoves</code> is searched (sequentially) // linear search of <code>ListOfMoves</code> // <code>ListOfMoves</code> is stepped through;</li> <li>3) for an occurrence of the piece ID entered;</li> <li>4) until either the piece ID is found or the end of <code>ListOfMoves</code> is encountered;</li> <li>5) if end of list is encountered user is asked again to enter the Piece ID;</li> </ol>	5

14	1	<p><b>All marks for AO3 (programming)</b></p> <p><b>Mark as follows:</b>  <b>1 mark</b> for error codes 1 to 3 tested (using IF, nested IF or CASE)  <b>A Error messages in a data structure and accessed via error code as index</b>  <b>1 mark</b> for appropriate error messages (<b>A</b> similar wording but same meaning as):  'Error code 1 - Not a valid piece'  'Error code 2 - Not a valid move'  'Error code 3 - Not a number'  <b>1 mark</b> <b>outputting error code (1, 2, 3 or 4)</b></p> <p><b>Note:</b>  Messages such as “Error Code 1 – not valid” are not detailed enough and are not creditworthy.</p>	3
14	2	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from 14.1, including prompts on screen capture matching those in code.  Code for 14.1 must be sensible.</p> <p>Screen capture showing:</p> <pre> Next Player: a a5 can jump to 3 , 2 a6 can jump to 3 , 0 a6 can jump to 3 , 4 a7 can jump to 3 , 2 a7 can jump to 3 , 6 a8 can jump to 3 , 4 a9 can move to 3 , 0 a9 can move to 3 , 2 a10 can move to 3 , 2 a10 can move to 3 , 4 a11 can move to 3 , 4 a11 can move to 3 , 6 a12 can move to 3 , 6 There are 13 possible moves Which piece do you want to move? a4 Error code 1 - not a valid piece Which piece do you want to move? a9 Which row do you want to move to? 3 Which column do you want to move to? 4 Error code 2 - not a valid move Which row do you want to move to? a Which column do you want to move to? 9 Error code 3 - not a number Which row do you want to move to? 3 Which column do you want to move to? 0                     </pre>	1

<p>15</p>	<p>1</p>	<p><b>1 mark for AO3 (design) and 1 mark for AO3 (programming)</b></p> <p><b>Mark as follows:</b></p> <p><b>AO3 (design) – 1 mark:</b></p> <p>1) choosing the final if statement to amend;</p> <p><b>AO3 (programming) – 1 mark:</b></p> <p>2) correct logic statement;</p>	<p>2</p>
<p>15</p>	<p>2</p>	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from 15.1, including prompts on screen capture matching those in code. Code for 15.1 must be sensible.</p> <p>Screen capture showing:</p> <pre>Next Player: a a1 can move to 1 , 0 a1 can move to 1 , 2 a2 can move to 7 , 0 a3 can move to 3 , 6 a5 can move to 4 , 3 a5 can jump to 5 , 0 a6 can jump to 5 , 2 a7 can move to 3 , 4 a7 can move to 3 , 6 There are 9 possible moves Which piece do you want to move? a5 Which row do you want to move to? 5 Which column do you want to move to? 0 jumped over b1  Player A: [[9, 0, 0], [0, 1, 0], [6, 1, 0], [2, 7, 0], [0, 7, 0], [5, 0, 0], [3, 0, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]] Player B: [[8, 0, 0], [4, 1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [1, 4, 0], [6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]        0      1      2      3      4      5      6      7 -----  XXXXXX        XXXXXX        XXXXXX        XXXXXX         0  XXXXXX  a1  XXXXXX        XXXXXX        XXXXXX  a4    XXXXXX        XXXXXX        XXXXXX        XXXXXX         -----         XXXXXX        XXXXXX        XXXXXX        XXXXXX  1         XXXXXX        XXXXXX  b5  XXXXXX  a8  XXXXXX          XXXXXX        XXXXXX        XXXXXX        XXXXXX  -----</pre>	<p>1</p>

2	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	XXXXX	XXXXX	a7	XXXXX
	XXXXX	XXXXX	XXXXX		a3
	XXXXX	XXXXX	XXXXX	XXXXX	
-----					
3	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
	a6	XXXXX	XXXXX	XXXXX	XXXXX
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
-----					
4	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	b1	XXXXX	XXXXX	XXXXX
	XXXXX	XXXXX	XXXXX	XXXXX	
-----					
5	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
	a5	XXXXX	XXXXX	b4	XXXXX
	XXXXX	XXXXX	XXXXX	XXXXX	b3
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
-----					
6	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	a2	XXXXX	b6	XXXXX
	XXXXX	XXXXX	XXXXX	b7	XXXXX
	XXXXX	XXXXX	XXXXX	XXXXX	b8
-----					
7	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
	XXXXX	b2	XXXXX	XXXXX	XXXXX
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
-----					

16	1	<p><b>2 marks for AO3 (design) and 7 marks for AO3 (programming)</b></p> <table border="1" data-bbox="268 282 1310 931"> <thead> <tr> <th data-bbox="268 282 392 353">Level</th> <th data-bbox="392 282 1158 353">Description</th> <th data-bbox="1158 282 1310 353">Mark Range</th> </tr> </thead> <tbody> <tr> <td data-bbox="268 353 392 488">3</td> <td data-bbox="392 353 1158 488">A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.</td> <td data-bbox="1158 353 1310 488">7–9</td> </tr> <tr> <td data-bbox="268 488 392 589">2</td> <td data-bbox="392 488 1158 589">There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.</td> <td data-bbox="1158 488 1310 589">4–6</td> </tr> <tr> <td data-bbox="268 589 392 931">1</td> <td data-bbox="392 589 1158 931">An attempt has been made to write and amend the subroutine <code>PrintResult</code>. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.</td> <td data-bbox="1158 589 1310 931">1–3</td> </tr> </tbody> </table> <p><b>Marking guidance:</b></p> <p><b>Evidence of AO3 design – 2 points:</b></p> <p>Evidence of design to look for in response:</p> <ol style="list-style-type: none"> <li>1) subroutine <code>CountNumberOfPieces</code> with interface so can be used for both A and B</li> <li>2) A method for checking piece exists on board</li> </ol> <p><b>Evidence of AO3 programming – 7 points:</b></p> <p>Evidence of programming to look for in response:</p> <ol style="list-style-type: none"> <li>3) in <code>CountNumberOfPieces</code> count variable initialised, updated and returned correctly <b>A</b> counting non-dames only</li> <li>4) in <code>CountNumberOfPieces</code> loop through <code>A/B/PlayersPieces</code></li> <li>5) use value stored in <code>A/B [0,1]</code> as the number of dames</li> <li>6) formula given in Q correctly programmed</li> <li>7) comparing the two players' scores <b>and output winner correctly</b></li> <li>8) <b>output calculated scores</b></li> <li>9) sensible output in case of a draw</li> </ol> <p><b>Note:</b> output is the same whether or not Question 15 has been attempted.</p>	Level	Description	Mark Range	3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.	7–9	2	There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.	4–6	1	An attempt has been made to write and amend the subroutine <code>PrintResult</code> . Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1–3	9
Level	Description	Mark Range													
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.	7–9													
2	There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.	4–6													
1	An attempt has been made to write and amend the subroutine <code>PrintResult</code> . Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1–3													
16	2	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from 16.1, including prompts on screen capture matching those</p>	1												

in code.

Code for 16.1 must be sensible.

Screen capture showing:

Enter the filename: game4.txt

Player A:

```
[[15, 2, 0], [1, 2, 0], [0, 3, 0], [0, 5, 0], [1, 6, 0], [0,
1, 1], [1, 0, 1], [1, 4, 0], [2, 7, 0], [2, 1, 0], [2, 3, 0],
[2, 5, 0], [3, 6, 0]]
```

Player B:

```
[[15, 0, 0], [4, 3, 0], [5, 0, 0], [5, 6, 0], [5, 4, 0], [4,
1, 0], [3, 2, 0], [6, 5, 0], [6, 7, 0], [3, 0, 0], [3, 4, 0],
[4, 5, 0], [4, 7, 0]]
```

```

      0      1      2      3      4      5      6      7
-----
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
0 |XXXXX|  A5 |XXXXX|  a2 |XXXXX|  a3 |XXXXX|      |
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
-----
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
1 |  A6 |XXXXX|  a1 |XXXXX|  a7 |XXXXX|  a4 |XXXXX|
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
-----
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
2 |XXXXX|  a9 |XXXXX|  a10 |XXXXX|  a11 |XXXXX|  a8 |
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
-----
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
3 |  b9 |XXXXX|  b6 |XXXXX|  b10 |XXXXX|  a12 |XXXXX|
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
-----
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
4 |XXXXX|  b5 |XXXXX|  b1 |XXXXX|  b11 |XXXXX|  b12 |
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
-----
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
5 |  b2 |XXXXX|      |XXXXX|  b4 |XXXXX|  b3 |XXXXX|
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
-----
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
6 |XXXXX|      |XXXXX|      |XXXXX|  b7 |XXXXX|  b8 |
|XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|      |
-----
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
7 |      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
|      |XXXXX|      |XXXXX|      |XXXXX|      |XXXXX|
-----

```

Next Player: a

There are 0 possible moves

Game ended

A won this game with a score of -17

B got a score of 3

17	1	<p><b>Mark is for AO2 (analyse)</b></p> <p>OpponentsPieces;</p> <p>R. if any additional code R. if spelt incorrectly I. case &amp; spacing</p>	1												
17	2	<p><b>2 marks for AO3 (design) and 7 marks for AO3 (programming)</b></p> <table border="1" data-bbox="268 539 1350 1155"> <thead> <tr> <th data-bbox="268 539 392 607">Level</th> <th data-bbox="392 539 1217 607">Description</th> <th data-bbox="1217 539 1350 607">Mark Range</th> </tr> </thead> <tbody> <tr> <td data-bbox="268 607 392 745">3</td> <td data-bbox="392 607 1217 745">A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.</td> <td data-bbox="1217 607 1350 745">7–9</td> </tr> <tr> <td data-bbox="268 745 392 846">2</td> <td data-bbox="392 745 1217 846">There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.</td> <td data-bbox="1217 745 1350 846">4–6</td> </tr> <tr> <td data-bbox="268 846 392 1155">1</td> <td data-bbox="392 846 1217 1155">An attempt has been made to amend the subroutine MoveDame. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.</td> <td data-bbox="1217 846 1350 1155">1–3</td> </tr> </tbody> </table> <p><b>Marking guidance:</b></p> <p><b>Evidence of AO3 design – 2 points:</b></p> <p>Evidence of design to look for in response:</p> <ol style="list-style-type: none"> <li>1) validate that chosen piece is an opponent’s existing piece</li> <li>2) return updated OpponentsPieces from subroutine MoveDame (parameter by reference)</li> </ol> <p><b>Evidence of AO3 programming – 7 points:</b></p> <p>Evidence of programming to look for in response:</p> <ol style="list-style-type: none"> <li>3) user prompt for which piece to take</li> <li>4) extracting player letter from chosen piece</li> <li>5) extracting index from chosen piece</li> <li>6) retrieving coodinates from OpponentsPieces</li> <li>7) set opponent’s piece coordinates to -1</li> <li>8) new dame’s coordinates set to taken piece’s coordinates</li> <li>9) update parameters in calls to MovePiece in subroutine MakeMove (parameter by reference)</li> </ol> <p><b>A.</b> solutions that ask the user to input the row and column of the piece to be removed.</p>	Level	Description	Mark Range	3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.	7–9	2	There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.	4–6	1	An attempt has been made to amend the subroutine MoveDame. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1–3	9
Level	Description	Mark Range													
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.	7–9													
2	There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work.	4–6													
1	An attempt has been made to amend the subroutine MoveDame. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1–3													

17	3	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from 17.2, including prompts on screen capture matching those in code. Code for 17.2 must be sensible.</p> <p>Screen capture showing: Do you want to load a saved game? (Y/N): y Enter the filename: game3.txt</p> <p>Player A: [[8, 0, 0], [0, 1, 0], [6, 1, 0], [2, 7, 0], [0, 7, 0], [3, 2, 0], [3, 0, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]</p> <p>Player B: [[8, 0, 0], [4, 1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [1, 4, 0], [6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]</p> <pre>       0      1      2      3      4      5      6      7 -----  XXXXX        XXXXX        XXXXX        XXXXX         0  XXXXX  a1  XXXXX        XXXXX        XXXXX  a4    XXXXX        XXXXX        XXXXX        XXXXX         -----         XXXXX        XXXXX        XXXXX        XXXXX  1         XXXXX        XXXXX  b5  XXXXX  a8  XXXXX          XXXXX        XXXXX        XXXXX        XXXXX  -----  XXXXX        XXXXX        XXXXX        XXXXX         2  XXXXX        XXXXX        XXXXX  a7  XXXXX  a3    XXXXX        XXXXX        XXXXX        XXXXX         -----         XXXXX        XXXXX        XXXXX        XXXXX  3   a6  XXXXX  a5  XXXXX        XXXXX        XXXXX          XXXXX        XXXXX        XXXXX        XXXXX  -----  XXXXX        XXXXX        XXXXX        XXXXX         4  XXXXX  b1  XXXXX        XXXXX        XXXXX          XXXXX        XXXXX        XXXXX        XXXXX         -----         XXXXX        XXXXX        XXXXX        XXXXX  5         XXXXX        XXXXX  b4  XXXXX  b3  XXXXX          XXXXX        XXXXX        XXXXX        XXXXX  -----  XXXXX        XXXXX        XXXXX        XXXXX         6  XXXXX  a2  XXXXX  b6  XXXXX  b7  XXXXX  b8    XXXXX        XXXXX        XXXXX        XXXXX         -----         XXXXX        XXXXX        XXXXX        XXXXX  7         XXXXX  b2  XXXXX        XXXXX        XXXXX          XXXXX        XXXXX        XXXXX        XXXXX  ----- Next Player: a a1 can move to 1 , 0 </pre>	1
----	---	---	---



```

a1 can move to 1 , 2
a2 can move to 7 , 0
a3 can move to 3 , 6
a5 can move to 4 , 3
a7 can move to 3 , 4
a7 can move to 3 , 6
a8 can jump to 3 , 4
There are 8 possible moves
Which piece do you want to move? a2
Which row do you want to move to? 7
Which column do you want to move to? 0
Which piece do you want to take? b1

```

```

Player A:
[[9, 1, 0], [0, 1, 0], [4, 1, 1], [2, 7, 0], [0, 7, 0], [3, 2, 0],
[3, 0, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]
Player B:
[[8, 0, 0], [-1, -1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [1, 4, 0],
[6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]

```

	0	1	2	3	4	5	6	7
0	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	a1  XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	a4
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
1		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	b5  XXXXX	XXXXX	a8  XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
2	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	XXXXX	XXXXX	XXXXX	a7  XXXXX	XXXXX	XXXXX	a3
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
3		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
	a6  XXXXX	a5  XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
4	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	A2  XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
5		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	b4  XXXXX	XXXXX	b3  XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
6	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
	XXXXX	XXXXX	b6  XXXXX	XXXXX	b7  XXXXX	XXXXX	XXXXX	b8
	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	
7		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	b2  XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX

17	4	<p><b>Mark is for AO3 (evaluate)</b></p> <p><b>**** SCREEN CAPTURE ****</b></p> <p>Must match code from 17.2, including prompts on screen capture matching those in code. Code for 17.2 must be sensible.</p> <p>Screen capture showing:</p> <pre> Next Player:  b b2  can move to  6  ,  1 b3  can move to  4  ,  5 b3  can move to  4  ,  7 b4  can move to  4  ,  3 b4  can move to  4  ,  5 b5  can move to  0  ,  3 b5  can move to  0  ,  5 b6  can move to  5  ,  2 b6  can jump to  4  ,  5 b7  can jump to  4  ,  3 b7  can jump to  4  ,  7 b8  can jump to  4  ,  5 There are  12  possible moves Which piece do you want to move? b5 Which row do you want to move to? 0 Which column do you want to move to? 3 Which piece do you want to take? a6  Player A: [[9, 1, 0], [0, 1, 0], [4, 1, 1], [2, 7, 0], [0, 7, 0], [3, 2, 0], [-1, -1, 0], [2, 5, 0], [1, 6, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]] Player B: [[9, 1, 0], [-1, -1, 0], [7, 2, 0], [5, 6, 0], [5, 4, 0], [3, 0, 1], [6, 3, 0], [6, 5, 0], [6, 7, 0], [-1, -1, 0], [-1, -1, 0], [-1, -1, 0]]        0      1      2      3      4      5      6      7 -----  XXXXX        XXXXX        XXXXX        XXXXX         0  XXXXX   a1  XXXXX        XXXXX        XXXXX   a4    XXXXX        XXXXX        XXXXX        XXXXX         -----         XXXXX        XXXXX        XXXXX        XXXXX  1         XXXXX        XXXXX        XXXXX   a8  XXXXX          XXXXX        XXXXX        XXXXX        XXXXX  -----  XXXXX        XXXXX        XXXXX        XXXXX         2  XXXXX        XXXXX        XXXXX   a7  XXXXX   a3    XXXXX        XXXXX        XXXXX        XXXXX         -----         XXXXX        XXXXX        XXXXX        XXXXX  3    B5  XXXXX   a5  XXXXX        XXXXX        XXXXX          XXXXX        XXXXX        XXXXX        XXXXX  -----  XXXXX        XXXXX        XXXXX        XXXXX         4  XXXXX   A2  XXXXX        XXXXX        XXXXX         </pre>	1
----	---	--	---

		XXXXX	XXXXX	XXXXX	XXXXX	
		-----				
5		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	b4	XXXXX	b3  XXXXX
		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		-----				
6		XXXXX	XXXXX	XXXXX	XXXXX	
		XXXXX	XXXXX	b6	XXXXX	b7  XXXXX  b8
		XXXXX	XXXXX	XXXXX	XXXXX	
		-----				
7		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		XXXXX	b2	XXXXX	XXXXX	XXXXX
		XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
		-----				

**VB.Net**

03	1	<pre> Dim NumberIn, NumberOut, Count, PartValue As Integer Console.WriteLine("Enter a positive whole number: ") NumberIn = Console.ReadLine NumberOut = 0 Count = 0 While NumberIn &gt; 0     Count += 1     PartValue = NumberIn Mod 2     NumberIn \= 2     For i = 1 To Count - 1         PartValue *= 10     Next     NumberOut += PartValue End While Console.WriteLine("The result is: " &amp; NumberOut) Console.ReadLine()                 </pre>	<b>11</b>
14	1	<pre> Sub DispayErrorCode(ByVal ErrorNumber As Integer)     Console.WriteLine("Error Code " &amp; ErrorNumber)     If ErrorNumber = 1 Then         Console.WriteLine("not a valid piece")     ElseIf ErrorNumber = 2 Then         Console.WriteLine("not a valid move")     ElseIf ErrorNumber = 3 Then         Console.WriteLine("not a number")     ElseIf ErrorNumber = 4 Then         Console.WriteLine("file error")     End If End Sub                 </pre>	<b>3</b>
15	1	<pre> Function ValidJump(ByVal Board(,) As String, ByVal PlayersPieces(,) As Integer, ByVal Piece As String, ByVal NewRow As Integer, ByVal NewColumn As Integer) As Boolean     Dim Valid As Boolean     Dim MiddlePiece, Player, OppositePiecePlayer, MiddlePiecePlayer As String     Dim Index, CurrentRow, CurrentColumn, MiddlePieceRow, MiddlePieceColumn As Integer     Valid = False     MiddlePiece = ""     Player = Left(Piece, 1).ToLower()     If Len(Piece) = 2 Then         Index = CInt(Right(Piece, 1))     Else         Index = CInt(Right(Piece, 2))     End If     If Player = "a" Then         OppositePiecePlayer = "b"     Else         OppositePiecePlayer = "a"     End If     If NewRow &gt;= 0 And NewRow &lt; BoardSize And NewColumn     &gt;= 0 And NewColumn &lt; BoardSize Then         If Board(NewRow, NewColumn) = Space Then                 </pre>	<b>2</b>

		<pre> CurrentRow = PlayersPieces(Index, Row) CurrentColumn = PlayersPieces(Index, Column) MiddlePieceRow = (CurrentRow + NewRow) \ 2 MiddlePieceColumn = (CurrentColumn + NewColumn) \ 2 MiddlePiece = Board(MiddlePieceRow, MiddlePieceColumn) MiddlePiecePlayer = Left(MiddlePiece, 1).ToLower() If MiddlePiecePlayer = <b>OppositePiecePlayer</b> Then     Valid = True End If End If End If Return Valid End Function </pre> <p><b>Alternative logic statement:</b></p> <p>MiddlePiecePlayer = <b>OppositePiecePlayer</b> and  MiddlePiecePlayer != ' ':</p>	
16	1	<pre> <b>Function</b> CountNumberOfPieces(ByVal PlayersPieces(,) As Integer) As Integer     Dim Count As Integer = 0     For Index = 1 To NumberOfPieces         If PlayersPieces(Index, Row) &gt; -1 Then             Count += 1         End If     Next     Return Count <b>End Function</b> </pre> <pre> Sub PrintResult(ByVal A(,) As Integer, ByVal B(,) As Integer, ByVal NextPlayer As String)     Console.WriteLine("Game ended")     Dim TotalA As Integer = CountNumberOfPieces(A)     Dim TotalB As Integer = CountNumberOfPieces(B)     TotalA = A(0, 0) - TotalA - 10 * A(0, 1)     TotalB = B(0, 0) - TotalB - 10 * B(0, 1)     If TotalA &lt; TotalB Then         Console.WriteLine("A won this game with a score of " &amp; TotalA)         Console.WriteLine("B got a score of " &amp; TotalB)     ElseIf TotalB &lt; TotalA Then         Console.WriteLine("B won this game with a score of " &amp; TotalB)         Console.WriteLine("A got a score of ", TotalA)     Else         Console.WriteLine("It was a draw. Both players got a score of " &amp; TotalA)     End If     PrintPlayerPieces(A, B) End Sub </pre>	9

17	2	<pre> Sub MoveDame (ByRef OpponentsPieces(,) As Integer, ByRef NewRow As Integer, ByRef NewColumn As Integer, ByVal Player As String)     Dim Opponent As String = ""     Dim ChosenPiece As String     Dim Index As Integer     NewRow = -1     While Player = Opponent Or NewRow = -1         Console.Write("Which piece do you want to take?")         ChosenPiece = Console.ReadLine         Opponent = ChosenPiece.Substring(0, 1).ToLower         Index = CInt(ChosenPiece.Substring(1, ChosenPiece.Length - 1))         NewRow = OpponentsPieces(Index, Row)         NewColumn = OpponentsPieces(Index, Column)     End While     OpponentsPieces(Index, Row) = -1     OpponentsPieces(Index, Column) = -1 End Sub  Sub MakeMove(ByRef Board(,) As String, ByRef PlayersPieces(,) As Integer, ByRef OpponentsPieces(,) As Integer, ByVal ListOfMoves() As MoveRecord, ByVal PieceIndex As Integer)     Dim Piece, MiddlePiece As String     Dim NewRow, NewColumn, PlayersPieceIndex, CurrentRow, CurrentColumn, MiddlePieceRow, MiddlePieceColumn As Integer     Dim Jumping As Boolean     PlayersPieces(0, 0) = PlayersPieces(0, 0) + 1     If PieceIndex &gt; 0 Then         Piece = ListOfMoves(PieceIndex).Piece         NewRow = ListOfMoves(PieceIndex).NewRow         NewColumn = ListOfMoves(PieceIndex).NewColumn         If Len(Piece) = 2 Then             PlayersPieceIndex = CInt(Right(Piece, 1))         Else             PlayersPieceIndex = CInt(Right(Piece, 2))         End If         CurrentRow = PlayersPieces(PlayersPieceIndex, Row)         CurrentColumn = PlayersPieces(PlayersPieceIndex, Column)         Jumping = ListOfMoves(PieceIndex).CanJump         MovePiece(Board, PlayersPieces, OpponentsPieces, Piece, NewRow, NewColumn)         If Jumping Then             MiddlePieceRow = (CurrentRow + NewRow) \ 2             MiddlePieceColumn = (CurrentColumn + NewColumn) \ 2             MiddlePiece = Board(MiddlePieceRow, MiddlePieceColumn)             Console.WriteLine("jumped over " &amp; MiddlePiece)         End If     End If End Sub </pre>	9
----	---	--	---

```
Sub MovePiece(ByRef Board(,) As String, ByRef
PlayersPieces(,) As Integer, ByRef OpponentsPieces(,) As
Integer, ByVal ChosenPiece As String, ByVal NewRow As
Integer, ByVal NewColumn As Integer)
    Dim Index, CurrentRow, CurrentColumn As Integer
    Dim Player As String
    If Len(ChosenPiece) = 2 Then
        Index = CInt(Right(ChosenPiece, 1))
    Else
        Index = CInt(Right(ChosenPiece, 2))
    End If
    CurrentRow = PlayersPieces(Index, Row)
    CurrentColumn = PlayersPieces(Index, Column)
    Board(CurrentRow, CurrentColumn) = Space
    If NewRow = BoardSize - 1 And PlayersPieces(Index,
Dame) = 0 Then
        Player = "a"
        PlayersPieces(0, 1) = PlayersPieces(0, 1) + 1
        PlayersPieces(Index, Dame) = 1
        ChosenPiece = ChosenPiece.ToUpper()
        MoveDame(OpponentsPieces, NewRow, NewColumn,
Player)
    Else
        If NewRow = 0 And PlayersPieces(Index, Dame) = 0
Then
            Player = "b"
            PlayersPieces(0, 1) = PlayersPieces(0, 1) + 1
            PlayersPieces(Index, Dame) = 1
            ChosenPiece = ChosenPiece.ToUpper()
            MoveDame(OpponentsPieces, NewRow, NewColumn,
Player)
        End If
    End If
    PlayersPieces(Index, Row) = NewRow
    PlayersPieces(Index, Column) = NewColumn
    Board(NewRow, NewColumn) = ChosenPiece
End Sub
```

## Python 3

03	1	<pre> NumberIn = int(input('Enter a positive whole number: ')) NumberOut = 0 Count = 0 while NumberIn &gt; 0:     Count += 1     PartValue = NumberIn % 2     NumberIn = NumberIn // 2     for i in range(1, Count):         PartValue = PartValue * 10     NumberOut = NumberOut + PartValue print('The result is: ', NumberOut)                 </pre>	11
14	1	<pre> def DisplayErrorCode(ErrorNumber):     print('Error Code ', ErrorNumber, ' - ', end='')     if ErrorNumber == 1:         print('not a valid piece')     elif ErrorNumber == 2:         print('not a valid move')     elif ErrorNumber == 3:         print('not a number')     elif ErrorNumber == 4:         print('file error')                 </pre>	3
15	1	<pre> def ValidJump(Board, PlayersPieces, Piece, NewRow, NewColumn):     Valid = False     MiddlePiece = ''     Player = Piece[0].lower()     Index = int(Piece[1:])     if Player == 'a':         OppositePiecePlayer = 'b'     else:         OppositePiecePlayer = 'a'     if NewRow in range(BOARD_SIZE) and NewColumn in range(BOARD_SIZE):         if Board[NewRow][NewColumn] == SPACE:             CurrentRow = PlayersPieces[Index][ROW]             CurrentColumn = PlayersPieces[Index][COLUMN]             MiddlePieceRow = (CurrentRow + NewRow) // 2             MiddlePieceColumn = (CurrentColumn + NewColumn) // 2             MiddlePiece = Board[MiddlePieceRow][MiddlePieceColumn]             MiddlePiecePlayer = MiddlePiece[0].lower()             if <b>MiddlePiecePlayer == OppositePiecePlayer:</b>                 Valid = True     return Valid  <b>Alternative logic statement:</b>  <b>MiddlePiecePlayer == OppositePiecePlayer and</b> <b>MiddlePiecePlayer != ' ':</b>                 </pre>	2
16	1	<pre> def CountNumberOfPieces(PlayersPieces):     Count = 0     for Index in range(1, NUMBER_OF_PIECES + 1):         if PlayersPieces[Index][ROW] &gt; -1:                 </pre>	9



		<pre> # allow COLUMN instead of ROW     Count += 1     return Count  def PrintResult(A, B, NextPlayer):     print('Game ended')     TotalA = CountNumberOfPieces(A)     TotalB = CountNumberOfPieces(B)     TotalA = A[0][0] - TotalA - 10 * A[0][1]     TotalB = B[0][0] - TotalB - 10 * B[0][1]     if TotalA &lt; TotalB:         print('A won this game with a score of ', TotalA)         print('B got a score of ', TotalB)     elif TotalB &lt; TotalA:         print('B won this game with a score of ', TotalB)         print('A got a score of ', TotalA)     else:         print('it was a draw. Both players got a score of ', TotalA)     PrintPlayerPieces(A, B) </pre>	
17	2	<pre> def MoveDame(Player, OpponentsPieces):     NewRow = -1     Opponent = ''     while Player == Opponent or NewRow == -1:         ChosenPiece = input('Which piece do you want to take? ')         Opponent = ChosenPiece[0].lower()         Index = int(ChosenPiece[1:])         NewRow = OpponentsPieces[Index][ROW]         NewColumn = OpponentsPieces[Index][COLUMN]         OpponentsPieces[Index][ROW] = -1         OpponentsPieces[Index][COLUMN] = -1     return NewRow, NewColumn, OpponentsPieces  def MovePiece(Board, PlayersPieces, OpponentsPieces, ChosenPiece, NewRow, NewColumn):     Index = int(ChosenPiece[1:])     CurrentRow = PlayersPieces[Index][ROW]     CurrentColumn = PlayersPieces[Index][COLUMN]     Board[CurrentRow][CurrentColumn] = SPACE      if NewRow == BOARD_SIZE - 1 and PlayersPieces[Index][DAME] == 0:         Player = 'a'         PlayersPieces[0][1] += 1         PlayersPieces[Index][DAME] = 1         ChosenPiece = ChosenPiece.upper()         NewRow, NewColumn, OpponentsPieces = MoveDame(Player, OpponentsPieces)     elif NewRow == 0 and PlayersPieces[Index][DAME] == 0:         Player = 'b'         PlayersPieces[0][1] += 1         PlayersPieces[Index][DAME] = 1         ChosenPiece = ChosenPiece.upper()         NewRow, NewColumn, OpponentsPieces = MoveDame(Player, OpponentsPieces) </pre>	9

```
PlayersPieces[Index][ROW] = NewRow
PlayersPieces[Index][COLUMN] = NewColumn
Board[NewRow][NewColumn] = ChosenPiece
return Board, PlayersPieces, OpponentsPieces

def MakeMove(Board, PlayersPieces, OpponentsPieces,
ListOfMoves, PieceIndex):
    PlayersPieces[0][0] += 1
    if PieceIndex > 0:
        Piece = ListOfMoves[PieceIndex].Piece
        NewRow = ListOfMoves[PieceIndex].NewRow
        NewColumn = ListOfMoves[PieceIndex].NewColumn
        PlayersPieceIndex = int(Piece[1:])
        CurrentRow = PlayersPieces[PlayersPieceIndex][ROW]
        CurrentColumn =
PlayersPieces[PlayersPieceIndex][COLUMN]
        Jumping = ListOfMoves[PieceIndex].CanJump
        Board, PlayersPieces, OpponentsPieces =
MovePiece(Board, PlayersPieces, OpponentsPieces, Piece,
NewRow, NewColumn)
        if Jumping:
            MiddlePieceRow = (CurrentRow + NewRow) // 2
            MiddlePieceColumn = (CurrentColumn + NewColumn) // 2
            MiddlePiece =
Board[MiddlePieceRow][MiddlePieceColumn]
            print('jumped over ', MiddlePiece)
    return Board, PlayersPieces, OpponentsPieces
```

**Python 2**

03	1	<pre> NumberIn = int(raw_input('Enter a positive whole number: ')) NumberOut = 0 Count = 0 while NumberIn &gt; 0:     Count += 1     PartValue = NumberIn % 2     NumberIn = NumberIn // 2     for i in range(1, Count):         PartValue = PartValue * 10     NumberOut = NumberOut + PartValue print 'The result is: ', NumberOut                     </pre>	<b>11</b>
14	1	<pre> def DisplayErrorCode(ErrorNumber):     print 'Error Code ', ErrorNumber     if ErrorNumber == 1:         print 'not a valid piece'     elif ErrorNumber == 2:         print 'not a valid move'     elif ErrorNumber == 3:         print 'not a number'     elif ErrorNumber == 4:         print 'file error'                     </pre>	<b>3</b>
15	1	<pre> def ValidJump(Board, PlayersPieces, Piece, NewRow, NewColumn):     Valid = False     MiddlePiece = ''     Player = Piece[0].lower()     Index = int(Piece[1:])     if Player == 'a':         OppositePiecePlayer = 'b'     else:         OppositePiecePlayer = 'a'     if NewRow in range(BOARD_SIZE) and NewColumn in range(BOARD_SIZE):         if Board[NewRow][NewColumn] == SPACE:             CurrentRow = PlayersPieces[Index][ROW]             CurrentColumn = PlayersPieces[Index][COLUMN]             MiddlePieceRow = (CurrentRow + NewRow) // 2             MiddlePieceColumn = (CurrentColumn + NewColumn) // 2             MiddlePiece = Board[MiddlePieceRow][MiddlePieceColumn]             MiddlePiecePlayer = MiddlePiece[0].lower()             if <b>MiddlePiecePlayer == OppositePiecePlayer</b> and MiddlePiecePlayer != ' ':                 Valid = True     return Valid                     </pre>	<b>2</b>
16	1	<pre> def CountNumberOfPieces(PlayersPieces):     Count = 0     for Index in range(1, NUMBER_OF_PIECES + 1):         if PlayersPieces[Index][ROW] &gt; -1: <b># allow COLUMN instead of ROW</b>             Count += 1     return Count                     </pre>	<b>9</b>

		<pre>def PrintResult(A, B, NextPlayer):     print 'Game ended'     TotalA = CountNumberOfPieces(A)     TotalB = CountNumberOfPieces(B)     TotalA = A[0][0] - TotalA - 10 * A[0][1]     TotalB = B[0][0] - TotalB - 10 * B[0][1]     if TotalA &lt; TotalB:         print 'A won this game with a score of ', TotalA         print 'B got a score of ', TotalB     elif TotalB &lt; TotalA:         print 'B won this game with a score of ', TotalB         print 'A got a score of ', TotalB     else:         print 'it was a draw. Both players got a score of ', TotalA PrintPlayerPieces(A, B)</pre>	
17	2	<pre>def MoveDame(Player, OpponentsPieces):     NewRow = -1     Opponent = ''     while Player == Opponent or NewRow == -1:         ChosenPiece = raw_input('Which piece do you want to take? ')         Opponent = ChosenPiece[0].lower()         Index = int(ChosenPiece[1:])         NewRow = OpponentsPieces[Index][ROW]         NewColumn = OpponentsPieces[Index][COLUMN]         OpponentsPieces[Index][ROW] = -1         OpponentsPieces[Index][COLUMN] = -1     return NewRow, NewColumn, OpponentsPieces  def MovePiece(Board, PlayersPieces, OpponentsPieces, ChosenPiece, NewRow, NewColumn):     Index = int(ChosenPiece[1:])     CurrentRow = PlayersPieces[Index][ROW]     CurrentColumn = PlayersPieces[Index][COLUMN]     Board[CurrentRow][CurrentColumn] = SPACE      if NewRow == BOARD_SIZE - 1 and PlayersPieces[Index][DAME] == 0:         Player = 'a'         PlayersPieces[0][1] += 1         PlayersPieces[Index][DAME] = 1         ChosenPiece = ChosenPiece.upper()         NewRow, NewColumn, OpponentsPieces = MoveDame(Player, OpponentsPieces)     elif NewRow == 0 and PlayersPieces[Index][DAME] == 0:         Player = 'b'         PlayersPieces[0][1] += 1         PlayersPieces[Index][DAME] = 1         ChosenPiece = ChosenPiece.upper()         NewRow, NewColumn, OpponentsPieces = MoveDame(Player, OpponentsPieces)     PlayersPieces[Index][ROW] = NewRow     PlayersPieces[Index][COLUMN] = NewColumn     Board[NewRow][NewColumn] = ChosenPiece     return Board, PlayersPieces, OpponentsPieces</pre>	9

	<pre>def MakeMove(Board, PlayersPieces, OpponentsPieces, ListOfMoves, PieceIndex):     PlayersPieces[0][0] += 1     if PieceIndex &gt; 0:         Piece = ListOfMoves[PieceIndex].Piece         NewRow = ListOfMoves[PieceIndex].NewRow         NewColumn = ListOfMoves[PieceIndex].NewColumn         PlayersPieceIndex = int(Piece[1:])         CurrentRow = PlayersPieces[PlayersPieceIndex][ROW]         CurrentColumn = PlayersPieces[PlayersPieceIndex][COLUMN]         Jumping = ListOfMoves[PieceIndex].CanJump         Board, PlayersPieces, <b>OpponentsPieces</b> = MovePiece(Board, PlayersPieces, <b>OpponentsPieces</b>, Piece, NewRow, NewColumn)         if Jumping:             MiddlePieceRow = (CurrentRow + NewRow) // 2             MiddlePieceColumn = (CurrentColumn + NewColumn) // 2             MiddlePiece = Board[MiddlePieceRow][MiddlePieceColumn]             print 'jumped over ', MiddlePiece     return Board, PlayersPieces, OpponentsPieces</pre>	
--	--	--

**Pascal**

03	1	<pre> var     NumberIn, NumberOut, Count, PartValue, i: integer; begin     write('Enter a positive whole number: ');     readln(NumberIn);     NumberOut := 0;     Count := 0;     while NumberIn &gt; 0 do         begin             Count := Count + 1;             PartValue := NumberIn mod 2;             NumberIn := NumberIn div 2;             for i := 1 to Count - 1 do                 PartValue := PartValue * 10;             NumberOut := NumberOut + PartValue;             end;             writeln('The result is: ', NumberOut);         end;     end; </pre>	11
14	1	<pre> procedure DisplayErrorCode(ErrorNumber: integer); begin     write('Error Code ', ErrorNumber, ' - ');     case ErrorNumber of         1 : writeln('not a valid piece');         2 : writeln('not a valid move');         3 : writeln('not a number');         4 : writeln('file error');     end; end; </pre>	3
15	1	<pre> function ValidJump(Board: TBoard; PlayersPieces: TPieces; Piece: string; NewRow, NewColumn: integer): boolean; var     Valid: boolean;     MiddlePiece: string;     Player, OppositePiecePlayer, MiddlePiecePlayer: string;     Index, CurrentRow, CurrentColumn, MiddlePieceRow, MiddlePieceColumn: integer; begin     Valid := false;     MiddlePiece := '';     Player := LowerCase(LeftStr(Piece, 1));     if Length(Piece) = 2 then         Index := StrToInt(RightStr(Piece, 1))     else         Index := StrToInt(RightStr(Piece, 2));     if Player = 'a' then         OppositePiecePlayer := 'b'     else         OppositePiecePlayer := 'a';     if (NewRow &gt;= 0) and (NewRow &lt; BoardSize)     and (NewColumn &gt;= 0) and (NewColumn &lt; BoardSize) then         if Board[NewRow, NewColumn] = Space then             begin                 CurrentRow := PlayersPieces[Index, Row];                 CurrentColumn := PlayersPieces[Index, Column];             end; </pre>	2

		<pre> MiddlePieceRow := (CurrentRow + NewRow) div 2; MiddlePieceColumn := (CurrentColumn + NewColumn) div 2; MiddlePiece := Board[MiddlePieceRow, MiddlePieceColumn]; MiddlePiecePlayer := LowerCase(LeftStr(MiddlePiece, 1)); if (MiddlePiecePlayer = OppositePiecePlayer) then Valid := true; end; ValidJump := Valid; end;  <b>Alternative logic statement:</b>  (MiddlePiecePlayer = OppositePiecePlayer) and (MiddlePiecePlayer &lt;&gt; ' ') </pre>	
16	1	<pre> <b>function</b> CountNumberOfPieces(PlayersPieces: TPieces): <b>integer</b>; <b>var</b> Count, Index: <b>integer</b>; <b>begin</b>     Count := 0;     <b>for</b> Index := 1 <b>to</b> NumberOfPieces <b>do</b>         <b>if</b> PlayersPieces[Index, ROW] &gt; -1 <b>then</b> // allow Column instead of Row             Count := Count + 1;         CountNumberOfPieces := Count;     <b>end</b>;  <b>procedure</b> PrintResult(A, B: TPieces; NextPlayer: <b>string</b>); <b>var</b> TotalA, TotalB: <b>integer</b>; <b>begin</b>     writeln('Game ended');     TotalA := CountNumberOfPieces(A);     TotalB := CountNumberOfPieces(B);     TotalA := A[0, 0] - TotalA - 10 * A[0, 1];     TotalB := B[0, 0] - TotalB - 10 * B[0, 1];     <b>if</b> TotalA &lt; TotalB <b>then</b>         <b>begin</b>             writeln('A won this game with a score of ', TotalA);             writeln('B got a score of ', TotalB);         <b>end</b>     <b>else</b>         <b>if</b> TotalB &lt; TotalA <b>then</b>             <b>begin</b>                 writeln('B won this game with a score of ', TotalB);                 writeln('A got a score of ', TotalA);             <b>end</b>         <b>else</b>             writeln('it was a draw. Both players got a score of ', TotalA);             PrintPlayerPieces(A, B);     <b>end</b>; </pre>	9

17	2	<pre> procedure MoveDame(Player: string; var OpponentsPieces: <b>TPieces</b>; var NewRow, NewColumn: integer); var     Opponent, ChosenPiece: string;     Index: integer; begin     NewRow := -1;     Opponent := '';     while (Player = Opponent) or (NewRow = -1) do         begin             write('Which piece do you want to take? ');             readln(ChosenPiece);             Opponent := LowerCase(LeftStr(ChosenPiece,1));             if Length(ChosenPiece) = 2 then                 Index := StrToInt(RightStr(ChosenPiece,1))             else                 Index := StrToInt(RightStr(ChosenPiece,2));             NewRow := OpponentsPieces[Index, Row];             NewColumn := OpponentsPieces[Index][Column];             end;             OpponentsPieces[Index, Row] := -1;             OpponentsPieces[Index, Column] := -1;         end;  procedure MakeMove(var Board: TBoard; var PlayersPieces,     OpponentsPieces: TPieces; ListOfMoves: TList;     PieceIndex: integer); var     Piece, MiddlePiece: string;     NewRow, NewColumn, PlayersPieceIndex, CurrentRow,     CurrentColumn: integer;     MiddlePieceRow, MiddlePieceColumn: integer;     Jumping: boolean; begin     PlayersPieces[0, 0] := PlayersPieces[0, 0] + 1;     if PieceIndex &gt; 0 then         begin             Piece := ListOfMoves[PieceIndex].Piece;             NewRow := ListOfMoves[PieceIndex].NewRow;             NewColumn := ListOfMoves[PieceIndex].NewColumn;             if Length(Piece) = 2 then                 PlayersPieceIndex := StrToInt(RightStr(Piece, 1))             else                 PlayersPieceIndex := StrToInt(RightStr(Piece, 2));             CurrentRow := PlayersPieces[PlayersPieceIndex, Row];             CurrentColumn := PlayersPieces[PlayersPieceIndex,             Column];             Jumping := ListOfMoves[PieceIndex].CanJump;             MovePiece(Board, PlayersPieces, <b>OpponentsPieces</b>,             Piece, NewRow, NewColumn);             if Jumping then                 begin                     MiddlePieceRow := (CurrentRow + NewRow) div 2;                     MiddlePieceColumn := (CurrentColumn + NewColumn)                     div 2;                     MiddlePiece := Board[MiddlePieceRow, </pre>	9
----	---	--	---



	<pre> MiddlePieceColumn];         end;     end; end;  procedure MovePiece(var Board: TBoard; var PlayersPieces, <b>OpponentsPieces</b>: TPieces;     ChosenPiece: string; NewRow, NewColumn: integer); var     Index, CurrentRow, CurrentColumn: integer;     Player: string; begin     if Length(ChosenPiece) = 2 then         Index := StrToInt(RightStr(ChosenPiece,1))     else         Index := StrToInt(RightStr(ChosenPiece,2));     CurrentRow := PlayersPieces[Index, Row];     CurrentColumn := PlayersPieces[Index, Column];     Board[CurrentRow, CurrentColumn] := Space;      if (NewRow = BoardSize-1) and (PlayersPieces[Index, Dame] = 0) then         begin             Player := 'a';             PlayersPieces[0,1] := PlayersPieces[0,1] + 1;             PlayersPieces[Index, Dame] := 1;             ChosenPiece := UpperCase(ChosenPiece);             MoveDame(Player, <b>OpponentsPieces</b>, NewRow, NewColumn);         end     else         if (NewRow = 0) and (PlayersPieces[Index, Dame] = 0) then             begin                 Player := 'b';                 PlayersPieces[0, 1] := PlayersPieces[0, 1] + 1;                 PlayersPieces[Index, Dame] := 1;                 ChosenPiece := UpperCase(ChosenPiece);                 MoveDame(Player, <b>OpponentsPieces</b>, NewRow, NewColumn);             end;         PlayersPieces[Index, Row] := NewRow;         PlayersPieces[Index, Column] := NewColumn;         Board[NewRow, NewColumn] := ChosenPiece;     end;  procedure MakeMove(var Board: TBoard; var PlayersPieces,     OpponentsPieces: TPieces; ListOfMoves: TList; PieceIndex: integer); var     Piece, MiddlePiece: string;     NewRow, NewColumn, PlayersPieceIndex, CurrentRow, CurrentColumn: integer;     MiddlePieceRow, MiddlePieceColumn: integer;     Jumping: boolean; begin </pre>	
--	--	--

	<pre>PlayersPieces[0, 0] := PlayersPieces[0, 0] + 1; if PieceIndex &gt; 0 then   begin     Piece := ListOfMoves[PieceIndex].Piece;     NewRow := ListOfMoves[PieceIndex].NewRow;     NewColumn := ListOfMoves[PieceIndex].NewColumn;     if Length(Piece) = 2 then       PlayersPieceIndex := StrToInt(RightStr(Piece, 1))     else       PlayersPieceIndex := StrToInt(RightStr(Piece, 2));     CurrentRow := PlayersPieces[PlayersPieceIndex, Row];     CurrentColumn := PlayersPieces[PlayersPieceIndex, Column];     Jumping := ListOfMoves[PieceIndex].CanJump;     MovePiece(Board, PlayersPieces, OpponentsPieces, Piece, NewRow, NewColumn);     if Jumping then       begin         MiddlePieceRow := (CurrentRow + NewRow) div 2;         MiddlePieceColumn := (CurrentColumn + NewColumn) div 2;         MiddlePiece := Board[MiddlePieceRow, MiddlePieceColumn];       end;     end;   end; end;</pre>	
--	--	--

**C#**

03	1	<pre>int count = 0, partValue, numberIn, numberOut = 0; Console.Write("Enter a positive whole number: "); numberIn = Convert.ToInt32(Console.ReadLine()); while (numberIn &gt; 0) {     count++;     partValue = numberIn % 2;     numberIn = numberIn / 2;     for (int i = 1; i &lt; count; i++)     {         partValue = partValue * 10;     }     numberOut = numberOut + partValue; } Console.WriteLine("The result is: " + numberOut ); Console.ReadLine();</pre>	11
14	1	<pre>private static void DisplayErrorCode(int errorNumber) {     Console.WriteLine("Error Code " + errorNumber);     if (errorNumber == 1)     {         Console.WriteLine("not a valid piece");     }     else if (errorNumber == 2)     {         Console.WriteLine("not a valid move");     }     else if (errorNumber == 3)     {         Console.WriteLine("not a number");     }     else if (errorNumber == 4)     {         Console.WriteLine("file error");     } }</pre>	3
15	1	<pre>private static bool ValidJump(string[,] board, int[,] playersPieces, string piece, int newRow, int newColumn) {     string middlePiece = "";     string player, oppositePiecePlayer, middlePiecePlayer;     int index, currentRow, currentColumn, middlePieceRow, middlePieceColumn;     player = piece[0].ToString().ToLower();     if (piece.Length == 2)     {         index = Convert.ToInt32(piece[1].ToString());     }     else     {         index = Convert.ToInt32(piece.Substring(1));     }     if (player == "a")</pre>	2

		<pre> {     oppositePiecePlayer = "b"; } else {     oppositePiecePlayer = "a"; } if (newRow &gt;= 0 &amp;&amp; newRow &lt; BoardSize &amp;&amp;     newColumn &gt;= 0 &amp;&amp; newColumn &lt; BoardSize) {     if (board[newRow, newColumn] == Space)     {         currentRow = playersPieces[index, Row];         currentColumn = playersPieces[index, Column];         middlePieceRow = (currentRow + newRow) / 2;         middlePieceColumn = (currentColumn + newColumn) / 2;         middlePiece = board[middlePieceRow, middlePieceColumn];         middlePiecePlayer = middlePiece[0].ToString().ToLower();         if (middlePiecePlayer == oppositePiecePlayer)         {             return true;         }     }     return false; } </pre> <p><b>Alternative logic statement:</b></p> <pre> (middlePiecePlayer == oppositePiecePlayer) &amp;&amp; middlePiecePlayer != " " </pre>	
16	1	<pre> private static int CountNumberOfPieces(int[,], playersPieces) {     int count = 0;     for (int index = 1; index &lt; NumberOfPieces + 1; index++)     {         if (playersPieces[index,Row] &gt; -1) // allow Column instead of Row         {             count++;         }     }     return count; }  private static void PrintResult(int[,], a, int[,], b, string nextPlayer) {     int totalA, totalB;     Console.WriteLine("Game ended"); } </pre>	9

		<pre> totalA = CountNumberOfPieces(a); totalB = CountNumberOfPieces(b); totalA = a[0, 0] - totalA - 10 * a[0, 1]; totalB = b[0, 0] - totalB - 10 * b[0, 1]; if (totalA &lt; totalB) {     Console.WriteLine("A won this game with a score of " + totalA);     Console.WriteLine("B got a score of " + totalB); } else if (totalB &lt; totalA) {     Console.WriteLine("B won this game with a score of " + totalB);     Console.WriteLine("A got a score of " + totalA); } else {     Console.WriteLine("it was a draw. Both players got a score of " + totalA); } PrintPlayerPieces(a, b); } </pre>	
17	2	<pre> private static void MoveDame(string[,] board, string player, ref int newRow, ref int newColumn, int[,] opponentsPieces) {     string opponent, chosenPiece;     int index = 0;     newRow = -1;     opponent = "";     while ((player == opponent)    (newRow == -1))     {         Console.Write("Which piece do you want to take? ");         chosenPiece = Console.ReadLine();         opponent = chosenPiece[0].ToString().ToLower();         index = Convert.ToInt32(chosenPiece.Substring(1));         newRow = opponentsPieces[index, Row];         newColumn = opponentsPieces[index, Column];     }     opponentsPieces[index, Row] = -1;     opponentsPieces[index, Column] = -1; } private static void MovePiece(string[,] board, int[,] playersPieces, string chosenPiece, int newRow, int newColumn, int[,] opponentsPieces) {     int index, currentRow, currentColumn;     string player;     if (chosenPiece.Length == 2)     {         index = Convert.ToInt32(chosenPiece[1].ToString());     } } </pre>	9

```

else
{
    index = Convert.ToInt32(chosenPiece.Substring(1));
}
currentRow = playersPieces[index, Row];
currentColumn = playersPieces[index, Column];
board[currentRow, currentColumn] = Space;
if (newRow == BoardSize - 1 && playersPieces[index,
Dame] == 0)
{
    player = "a";
    playersPieces[0, 1] = playersPieces[0, 1] + 1;
    playersPieces[index, Dame] = 1;
    chosenPiece = chosenPiece.ToUpper();
    MoveDame(board, player, ref newRow, ref newColumn,
opponentsPieces);
}
else if (newRow == 0 && playersPieces[index, Dame] ==
0)
{
    player = "b";
    playersPieces[0, 1] = playersPieces[0, 1] + 1;
    playersPieces[index, Dame] = 1;
    chosenPiece = chosenPiece.ToUpper();
    MoveDame(board, player, ref newRow, ref newColumn,
opponentsPieces);
}
playersPieces[index, Row] = newRow;
playersPieces[index, Column] = newColumn;
board[newRow, newColumn] = chosenPiece;
}

private static void MakeMove(string[,] board, int[,]
playersPieces, int[,] opponentsPieces, MoveRecord[]
listOfMoves, int pieceIndex)
{
    string piece, middlePiece;
    int newRow, newColumn, playersPieceIndex,
currentRow, currentColumn;
    int middlePieceRow, middlePieceColumn;
    bool jumping;
    playersPieces[0, 0] = playersPieces[0, 0] + 1;
    if (pieceIndex > 0)
    {
        piece = listOfMoves[pieceIndex].Piece;
        newRow = listOfMoves[pieceIndex].NewRow;
        newColumn = listOfMoves[pieceIndex].NewColumn;
        playersPieceIndex =
Convert.ToInt32(piece.Substring(1));
        currentRow = playersPieces[playersPieceIndex,
Row];
        currentColumn = playersPieces[playersPieceIndex,
Column];
        jumping = listOfMoves[pieceIndex].CanJump;
        MovePiece(board, playersPieces, piece, newRow,
newColumn, opponentsPieces);
    }
}

```

		<pre>        if (jumping)         {             middlePieceRow = (currentRow + newRow) / 2;             middlePieceColumn = (currentColumn + newColumn) / 2;             middlePiece = board[middlePieceRow, middlePieceColumn];             Console.WriteLine("jumped over " + middlePiece);         }     } }</pre>	
--	--	---	--

## Java

03	1	<pre> Console.WriteLine("Enter a positive whole number: "); int numberIn = Integer.parseInt(Console.ReadLine()); int numberOut = 0; int count = 0; int partValue; while (numberIn &gt; 0) {     count++;     partValue = numberIn % 2;     numberIn = numberIn / 2;     for (int i = 1; i &lt; count; i++) {         partValue = partValue * 10;     }     numberOut = numberOut + partValue; } Console.WriteLine("The result is: " + numberOut);                 </pre>	11
14	1	<pre> void displayErrorCode(int errorNumber) {     Console.write("Error Code " + errorNumber + " - ");     if (errorNumber == 1) {         Console.WriteLine("not a valid piece");     } else if (errorNumber == 2) {         Console.WriteLine("not a valid move");     } else if (errorNumber == 3) {         Console.WriteLine("not a number");     } else if (errorNumber == 4) {         Console.WriteLine("file error");     } }  Alternative Example  void displayErrorCode(int errorNumber) {     Console.write("Error " + errorNumber + " - ");     switch (errorNumber) {         case 1:             Console.WriteLine("not a valid piece.");             break;         case 2:             Console.WriteLine("not a valid move");             break;         case 3:             Console.WriteLine("not a number");             break;         case 4:             Console.WriteLine("file error");             break;     } }                 </pre>	3
15	1	<pre> boolean validJump(String[][] board, int[][] playersPieces, String piece, int newRow, int newColumn) {     boolean valid = false;     String oppositePiecePlayer, middlePiecePlayer, player, middlePiece;     int index, currentRow, currentColumn, middlePieceRow,                 </pre>	2



		<pre> middlePieceColumn; player = (piece.charAt(0) + "").toLowerCase(); index = Integer.parseInt(piece.substring(1)); if (player.equals("a")) {     oppositePiecePlayer = "b"; } else {     oppositePiecePlayer = "a"; } if (newRow &gt;= 0 &amp;&amp; newRow &lt; BOARD_SIZE     &amp;&amp; newColumn &gt;= 0 &amp;&amp; newColumn &lt; BOARD_SIZE) {     if (board[newRow][newColumn].equals(SPACE)) {         currentRow = playersPieces[index][ROW];         currentColumn = playersPieces[index][COLUMN];         middlePieceRow = (currentRow + newRow) / 2;         middlePieceColumn = (currentColumn + newColumn) / 2;         middlePiece = board[middlePieceRow][middlePieceColumn];         middlePiecePlayer = (middlePiece.charAt(0) + "".toLowerCase());         if <b>(middlePiecePlayer.equals(oppositePiecePlayer))</b> {             valid = true;         }     } } return valid; } </pre>	
16	1	<pre> <b>int countNumberOfPieces(int[][] playerPieces) {</b>     <b>int count = 0;</b>     <b>for (int index = 1; index &lt; NUMBER_OF_PIECES + 1;</b> <b>index++) {</b>         <b>if (playerPieces[index][ROW] &gt; -1) {</b>             <b>count++;</b>         <b>}</b>     <b>}</b>     <b>return count;</b> <b>}</b>  <b>void printResult(int[][] a, int[][] b, String nextPlayer)</b> <b>{</b>     <b>Console.WriteLine("Game ended");</b>     <b>int totalA = countNumberOfPieces(a);</b>     <b>int totalB = countNumberOfPieces(b);</b>     <b>totalA = a[0][0] - totalA - 10 * a[0][1];</b>     <b>totalB = b[0][0] - totalB - 10 * b[0][1];</b>     <b>if (totalA &lt; totalB) {</b>         <b>Console.WriteLine("A won with a score of " +</b> <b>totalA);</b>         <b>Console.WriteLine("B got a score of " + totalB);</b>     <b>} else if (totalB &lt; totalA) {</b>         <b>Console.WriteLine("B won with a score of " +</b> <b>totalB);</b>         <b>Console.WriteLine("A got a score of " + totalA);</b>     <b>} else {</b>         <b>Console.WriteLine("it was a draw. Both players got</b> </pre>	9

		<pre> a score of " + totalA);     }     printPlayerPieces(a, b); } </pre>	
17	2	<pre> int[] moveDame(String player, int [][] opponentsPieces) {     int newRow = -1;     int newColumn = 0;     String opponent = "";     int index = 0;     while (player.equals(opponent)    newRow == -1) {         Console.WriteLine("Which piece do you want to take?");         String chosenPiece = Console.readLine();         opponent = chosenPiece.substring(0, 1).toLowerCase();         index = Integer.parseInt(chosenPiece.substring(1));         newRow = opponentsPieces[index][ROW];         newColumn = opponentsPieces[index][COLUMN];     }     opponentsPieces[index][ROW] = -1;     opponentsPieces[index][COLUMN] = -1;     return new int[]{newRow, newColumn}; }  void movePiece(String[][] board, int[][] playersPieces, int[][] opponentsPieces, String chosenPiece, int newRow, int newColumn) {     int index = Integer.parseInt(chosenPiece.substring(1));     int currentRow = playersPieces[index][ROW];     int currentColumn = playersPieces[index][COLUMN];     board[currentRow][currentColumn] = SPACE;     String player;      if (newRow == BOARD_SIZE - 1 &amp;&amp; playersPieces[index][DAME] == 0) {         player = "a";         playersPieces[0][1] += 1;         playersPieces[index][DAME] = 1;         chosenPiece = chosenPiece.toUpperCase();         int[] rtnInts = moveDame(player, opponentsPieces);         newRow = rtnInts[0];         newColumn = rtnInts[1];     } else if (newRow == 0 &amp;&amp; playersPieces[index][DAME] == 0) {         player = "b";         playersPieces[0][1] += 1;         playersPieces[index][DAME] = 1;         chosenPiece = chosenPiece.toUpperCase();         int[] rtnInts = moveDame(player, opponentsPieces);         newRow = rtnInts[0];         newColumn = rtnInts[1];     }     playersPieces[index][ROW] = newRow;     playersPieces[index][COLUMN] = newColumn; } </pre>	9

	<pre>        board[newRow][newColumn] = chosenPiece;     }      void makeMove(String[][] board, int[][] playersPieces,         int[][] opponentsPieces, MoveRecord[] listOfMoves,     int pieceIndex) {         playersPieces[0][0] += 1;         if (pieceIndex &gt; 0) {             String piece = listOfMoves[pieceIndex].piece;             int newRow = listOfMoves[pieceIndex].newRow;             int newColumn = listOfMoves[pieceIndex].newColumn;             int playersPieceIndex = Integer.parseInt(piece.substring(1));             int currentRow = playersPieces[playersPieceIndex][ROW];             int currentColumn = playersPieces[playersPieceIndex][COLUMN];             boolean jumping = listOfMoves[pieceIndex].canJump;             movePiece(board, playersPieces, <b>opponentsPieces</b>, piece, newRow, newColumn);             if (jumping) {                 int middlePieceRow = (currentRow + newRow) / 2;                 int middlePieceColumn = (currentColumn + newColumn) / 2;                 String middlePiece = board[middlePieceRow][middlePieceColumn];                 Console.WriteLine("jumped over " + middlePiece);             }         }     } }</pre>	
--	--	--