AQA

A-level
Computer Science

Paper 1 (7517/1)

Mark scheme (applicable for **all** programming languages A, B, C, D and E)

7517
June 2017

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

A-level Computer Science

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

June 2017

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.

Pages 4 to 5 contain 'Level of Response' marking instructions.

Pages 6 to 17 contain the generic mark scheme.

Pages 18 to 47 contain the 'Program Source Code' specific to the programming languages for questions 7.1, 9.1, 10.1, 11.1, 11.2, 11.3, 11.4, 12.1, 12.2;

pages 18 to 22 – VB.NET
pages 23 to 26 – PYTHON 2
pages 27 to 30 – PYTHON 3
pages 31 to 36 – C#
pages 37 to 41 – PASCAL/Delphi
pages 42 to 47 – 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 **7.1**, 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.

<table>
<tr><td>01</td><td>1</td><td>Marks are for AO1 (understanding)

<table><tr><th>Real number</th><th>Valid? (Yes/No)</th></tr><tr><td>87.000</td><td>Yes</td></tr><tr><td>97+12</td><td>No</td></tr><tr><td>12.31E+12</td><td>Yes</td></tr></table>
A. alternative indicators for Yes/No eg Y/N.

Mark as follows:
One mark per correct row</td><td>3</td></tr>
<tr><td>01</td><td>2</td><td>Marks are for AO2 (apply)

<code><natural> ::= <digit> | <digit> <natural></code>

A. alternative names for <code><natural></code>
A. recursive and non-recursive cases swapped around

Mark as follows:

1 mark: correct recursive case
1 mark: correct non-recursive case
MAX 1 if any errors in answer eg missing |</td><td>2</td></tr>
<tr><td>02</td><td>1</td><td>Mark is for AO2 (analyse)

Input string is a (valid) postcode followed by additional characters // the input string is not a valid (UK) postcode // the mail will not be put in any of the three vans;

NE. the input string is not a valid IP postcode
A. Postcode has additional characters at the end
A. Postcode is too long</td><td>1</td></tr>
<tr><td>02</td><td>2</td><td>Mark is for AO2 (analyse)

(The string represents) an IP postcode that is not for a location in the town of Ipswich //
(The string represents) an IP postcode that is for a location near Ipswich //
(The string represents) a postcode for a letter that needs to go in Van B;

NE. valid postcode</td><td>1</td></tr>
<tr><td>02</td><td>3</td><td>Mark is for AO2 (analyse)

(IP / two letters) followed by number, letter, (number, letter, letter) //
(IP / two letters) followed by number between 5 and 9, number, (number, letter, letter) //
IP followed by 0;

A. postcodes that only have one letter at the start</td><td>1</td></tr>
</table>

02	**4**	**Marks are for AO2 (apply)** `\a?\a;\d;(\a|\d)?;\d\a\a;` **//** `\a\a?;\d;(\a|\d)?;\d\a\a;` **//** `\a?\a;\d;(\d|\a)?;\d\a\a;` **//** `\a\a?;\d;(\d|\a)?;\d\a\a;` **Mark as follows:** **1 mark:** 1. regular expression can start with either one or two letters **R.** if more than two letters allowed **1 mark:** 2. regular expression has a numeric digit after the initial letters **A.** if more than the correct number of letters allowed // regular expression has a numeric digit before it allows a single, optional letter or numeric digit **1 mark:** 3. regular expression allows a single, optional letter or numeric digit after the first numeric digit in the expression **//** regular expression allows a single, optional letter or numeric digit before the numeric digit followed by exactly two letters at the end of the expression **1 mark:** 4. regular expression ends with a numeric digit followed by exactly two letters **MAX 3** if final answer is not correct **R.** any mark points after 2nd use of | metacharacter **A.** suitable alternatives to `\a` and `\d` e.g. use of `[A-Z]`, `[a-z]` or `[A-Za-z]` instead of `\a` and `[0-9]` instead of `\d` **DPT.** / instead of \	**4**
03	**1**	**Mark is for AO1 (knowledge)** Merge sort;	**1**
03	**2**	**Mark is for AO1 (understanding)** 4;	**1**
03	**3**	**Mark is for AO1 (knowledge)** n^2 // $O(n^2)$; **A.** other ways of indicating n^2 e.g. `n^2` **A.** On^2	**1**

<table>
<tr><td>03</td><td>4</td><td>Marks are for AO1 (understanding)

In each pass through the list <code>n</code> items will be examined;
There will be (at most) <code>n</code> passes through the list;</td><td>2</td></tr>
<tr><td>04</td><td>1</td><td>Mark is for AO1 (knowledge)

A subroutine that calls itself;</td><td>1</td></tr>
<tr><td>04</td><td>2</td><td>Mark is for AO1 (understanding)

When target equals node // (When target does not equal node and) node is a leaf //
<code>node = target</code>;</td><td>1</td></tr>
<tr><td>04</td><td>3</td><td>Marks are for AO2 (apply)

<table><tr><th>Function Call</th><th>Output</th></tr><tr><td><code>TreeSearch(Olivia, Norbert)</code></td><td>(Visited) Norbert;</td></tr><tr><td><code>TreeSearch(Olivia, Phil);</code></td><td>(Visited) Phil;</td></tr><tr><td></td><td></td></tr></table>
MAX 2 if any errors eg additional outputs / function calls after output of Phil

I. minor spelling and punctuation errors</td><td>3</td></tr>
<tr><td>05</td><td>1</td><td>Mark is for AO2 (apply)

-2;</td><td>1</td></tr>
<tr><td>05</td><td>2</td><td>Mark is for AO2 (apply)

[8, 3];

I. missing brackets
I. wrong type of brackets</td><td>1</td></tr>
<tr><td>05</td><td>3</td><td>Marks are for AO2 (apply)

<table><tr><th>Calculation</th><th>Result</th></tr><tr><td><code>U</code></td><td>[1, 1]</td></tr><tr><td><code>v = [position of hero] - [position of enemy]</code></td><td>[6, -4];</td></tr><tr><td><code>u.v</code></td><td>2;</td></tr><tr><td><code>EnemyCanSee</code></td><td>True;</td></tr></table>
A. different answers that have been correctly calculated based on an incorrect answer for 5.2</td><td>3</td></tr>
</table>

05	**4**	**1 mark for AO1 (knowledge)** a heuristic approach employs a method of finding a solution that might not be the best; **1 mark for AO1 (understanding)** algorithm might need to consider visiting less/fewer cells/co-ordinates // algorithm might use knowledge of the domain to cut-down the search space // algorithm might consider visiting certain cells/coordinates first;	**2**
05	**5**	**Marks are for AO1 (understanding)** static data structures have storage size determined at compile-time / before program is run / when program code is translated; dynamic data structures can grow/shrink during execution / at run-time; // Static data structures can waste storage space/memory if the number of data items stored is small relative to the size of the structure; whereas dynamic data structures only take up the amount of storage space required for the actual data; // Static data structures have fixed (maximum) size; whereas size of dynamic data structures can change; // Dynamic data structures (typically) require memory to store pointer(s) to the next item(s); which static data structures (typically) do not need; **NE.** Dynamic data structures use pointers // Static data structures (typically) store data in consecutive memory locations; which dynamic data structures (typically) do not;	**2**
06	**1**	**Marks are for AO2 (analyse)** 1. Stack / data structure is used to store the (user's) actions; **A.** by implication 2. Each time an action is completed it is pushed/added onto the <u>top</u> of the stack; 3. unless it is an undo (or repeat) action; 4. When repeat action is used the top item from the stack is used to indicate the action to complete // when repeat action is used the result of peek function is used to indicate the action to complete; **R.** implication that top item of stack is popped/deleted from stack – unless it is clear it is subsequently pushed/added back to the stack **A.** when repeat action is used a copy of the top item from the stack is pushed/added to the top of the stack 5. When undo action is used the top item is popped/removed from the stack of actions;	**5**
06	**2**	**Mark is for AO1 (understanding)** Stack empty (error) // (stack) underflow;	**1**

07 1 **4 marks for AO3 (design) and 8 marks for AO3 (programming)** **12**

Mark scheme

Level	Description	Mark Range
4	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 of **Task 1**. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met.	10-12
3	There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays a prompt, inputs the string value and includes a loop. An attempt has been made to count the number of consecutive instances of a character and to output a character followed by the count of that character, although some of this may not work. The solution demonstrates good design work as most of the correct design decisions have been taken.	7-9
2	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.	4-6
1	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.	1-3

Guidance

Evidence of AO3 (design) – 4 points:

Evidence of design to look for in responses:

1. Identifying that a method that looks at each character in text entered is needed
2. Identifying that a comparison is needed to check if the current character is the same as the previous character or not
3. Mechanism that "remembers" value of previous character in the string // mechanism that "remembers" character at start of the run
4. Identifying that the first character in the string can't be compared to a previous

		character // the last character in the string can't be compared to the next character **NOTE:** award mark based on method attempted in answer provided 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. Suitable prompt displayed before any loop structures 6. Text input by user and stored into a variable with a suitable name, after prompt is displayed and before any loop structures 7. Loop structure coded with correct termination condition 8. Selection structure coded with correct condition, selection structure must be inside loop **A.** second loop structure with correct condition that is nested in first loop structure 9. One added to count of character under the correct circumstances 10. Count of character reset to one under the correct circumstances 11. Character and correct count of character displayed for some characters from beginning of text input by user 12. Character and correct count of character displayed for all characters of any text entered by the user Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code. **Information for examiner:** Refer answers that use alternative methods to produce the RLE to team leader.	
07	**2**	**Mark is for AO3 (evaluate)** *****SCREEN CAPTURE(S)***** *Info for examiner: Must match code from 7.1, including prompts on screen capture matching those in code. Code for 7.1 must be sensible.* Display of suitable prompt and user input of `AAARRRRGGGHH` followed by output of `A 3 R 4 G 3 H 2`; **A.** Each output on its own line, no spaces, other delimiter used instead of space	**1**
07	**3**	**Mark is for AO3 (evaluate)** *****SCREEN CAPTURE(S)***** *Info for examiner: Must match code from 7.1, including prompts on screen capture matching those in code. Code for 7.1 must be sensible.* Display of suitable prompt and user input of `A` followed by output of `A 1`; **A.** no space between `A` and `1`, other delimiter used instead of space	**1**

<table>
<tr><td>08</td><td>1</td><td>Marks are for AO2 (analyse)

<table><tr><th>Feature</th><th>Is present in Figure 11? (Yes/No)</th></tr><tr><td>Inheritance</td><td>No</td></tr><tr><td>Protected method</td><td>No</td></tr><tr><td>Private attribute</td><td>Yes</td></tr></table>
A. alternative indicators instead of Yes/No eg Y/N.

Mark as follows:
One mark per correct row</td><td>3</td></tr>
<tr><td>8</td><td>2</td><td>Mark is for AO2 (analyse)

<code>Rabbit</code> // <code>Fox</code>;

R. if spelt incorrectly
R. if any additional code
I. case</td><td>1</td></tr>
<tr><td>8</td><td>3</td><td>Marks are for AO1 (understanding)

A protected attribute can be accessed (within its class and) by derived class instances / subclasses;

A private attribute can only be accessed within its class;

A. private attribute can only be accessed within its file (Java only)</td><td>2</td></tr>
<tr><td>8</td><td>4</td><td>1 mark for AO2 (analyse)

MAX 1 from:
<code>RabbitCount</code> (is a private attribute and) is not accessible outside of the <code>Warren</code> class;
<code>GetRabbitCount</code> (is a public method and) is accessible outside of the <code>Warren</code> class;

1 mark for AO1 (understanding)

means the way <code>RabbitCount</code> is represented can be modified without having to change any other objects that interact with <code>Warren</code> NE. without having to change other code // makes it easier to reuse / inherit from the <code>Warren</code> class (as there is a well-defined interface) ;

A. this allows data/properties to be modified in a controlled way</td><td>2</td></tr>
</table>

8	**5**	**Marks are for AO2 (analyse)** when a rabbit dies it is replaced by null/none; **A.** when rabbits die they are not removed from the list `CompressRabbitList` makes sure that the space used for dead rabbits in the list is made available for new rabbits // `CompressRabbitList` makes sure that the fixed size array does not fill up with dead rabbits; `CompressRabbitList` moves live rabbits to the start of the list **A.** `CompressRabbitList` moves null objects / dead rabbits to the end of the list // other sections of the code assume that the live rabbits are in continuous locations in the array (so would not work correctly without a call to `CompressRabbitList`); **MAX 2**	**2**

8	**6**	**Marks are for AO2 (apply)**
		
		`HDRabbit = Class(Rabbit)`
		`  Private:`
		`  InfectionRate: Real`
		`  Generation: Integer`
		`  Public:`
		`  Procedure Inspect()  (Override)`
		`  Function IsInfertile()`
		`  Function GetGeneration()`
		`  Function GetInfectionRate()`
		`End Class`
		
		Information for examiner:
		Accept answers that use different notations, so long as meaning is clear.
		
		Mark as follows:
		
		1 mark: 1. for correct header including name of class and parent class
		1 mark: 2. for redefining the `Inspect` method **A.** Override not stated
		1 mark: 3. for defining the two additional attributes, with appropriate data types and identified as private **R.** if other attributes included
		1 mark: 4. for defining methods needed to read the two additional attributes, and an `IsFertile` method, all identified as being public **R.** if other methods included
		
		I. missing brackets
		I. additional Get/Set methods
		I. constructor method
		A. any suitable alternatives used instead of `Function` or `Procedure` keywords
		A. any suitable alternatives for data types eg float or double instead of real
		R. do not award mark for declaring new methods if any of the functions have the same name as the variables
			4

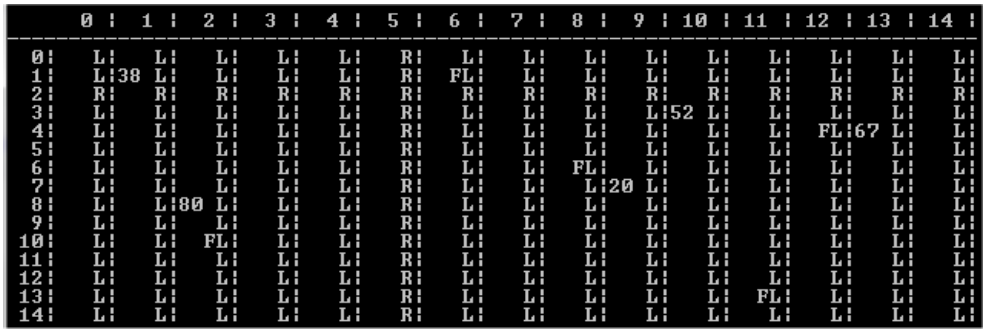
09	**1**	**Marks are for AO3 (programming)** **1 mark:** 1. tests for lower bound and displays error message if below **1 mark:** 2. tests for upper bound and displays error message if above **1 mark:** 3. Upper bound test uses `LandscapeSize` instead of data value of 14/15 **A.** in use of incorrect condition **1 mark:** 4. 1-3 happen repeatedly until valid input (for the upper and lower bounds used in the code provided) and forces re-entry of data each time **A.** use of pre or post-conditioned loop **MAX 3** if error message is not `Coordinate is outside of landscape, please try again` **A.** minor typos in error message **I.** case **I.** spacing **I.** minor punctuation differences **MAX 2** if new code has been added to `Simulation` constructor instead of `InputCoordinate` method	**4**

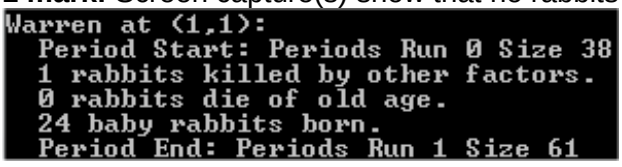
09	**2**	**Mark is for AO3 (evaluate)**
		
		****SCREEN CAPTURE(S)****
		Must match code from 09.1, including error message. Code for 09.1 must be sensible.
		
		1 mark: Screen capture(s) showing the required sequence of inputs (`-1`, `15`, `0`), the correct error message being displayed for `-1` and `15`, and that `0` has been accepted as the program has displayed the prompt for the y coordinate to be input.
		
		`Select option: 3`
		`  Input x coordinate: -1`
		`Coordinate is outside of landscape, please try again.`
		`  Input x coordinate: 15`
		`Coordinate is outside of landscape, please try again.`
		`  Input x coordinate: 0`
		`  Input y coordinate: _`
		
		A. alternative error messages if match code for 09.1
			1

10	**1**	**Marks are for AO3 (programming)** **1 mark:** 1. New subroutine created, with correct name, that overrides the subroutine in the `Animal` class **I.** private, protected, public modifiers **1 mark:** 2. `CalculateNewAge` subroutine in `Animal` class is always called **1 mark:** 3. Check made on gender of rabbit, and calculations done differently for each gender **I.** incorrect calculations **1 mark:** 4. Probability of death by other causes calculated correctly for male rabbits **1 mark:** 5. Probability of death by other causes calculated correctly for female rabbits	**5**

10	**2**	**Mark is for AO3 (evaluate)**
		
		****SCREEN CAPTURE(S)****
		Must match code from 10.1. Code for 10.1 must be sensible.
		
		1 mark: Any screen capture(s) showing the correct probability of death by other causes for a male rabbit (0.11 to 2dp) and a female rabbit (0.1);
		
		Example:
		`ID 3 Age 2 LS 4 Pr dth 0.1 Rep rate 1.2 Gender Female`
		`ID 4 Age 2 LS 4 Pr dth 0.11 Rep rate 1.2 Gender Male`
			1

11	**1**	**Marks are for AO3 (programming)** **1 mark:** Structure set-up to store the representation of terrain for a location **1 mark:** Type of terrain is passed to constructor as parameter **1 mark:** Type of terrain stored into attribute by constructor **A.** default value, that makes type of terrain for location clear, instead of value from a parameter	**3**
11	**2**	**Marks are for AO3 (programming)** **1 mark:** 1. An indicator for type of terrain will be stored for every location **I.** wrong type of terrain in a location **R.** if indicators other than R or L used **I.** case of indicators **1 mark:** 2. Vertical river created in column 5 **1 mark:** 3. Horizontal river created in row 2 **MAX 1 FOR 2 & 3** if only creates a river when foxes & warrens are in default locations **MAX 2** if creates any rivers in incorrect locations	**3**
11	**3**	**Marks are for AO3 (programming)** **1 mark:** R/L, or other indicator as long as it is clear what the type of terrain is, displayed in each location (could be different letters, use of different colours) **A.** type of terrain not displayed if location contains a fox **1 mark:** Row containing column indices matches new display of landscape **I.** number of dashes not adjusted to match new width **R.** if terrain indicators not displayed **A.** no adjustment made if indicators for terrain used mean no adjustment to width of display for terrain was needed	**2**
11	**4**	**Marks are for AO3 (programming)** **1 mark:** Warren/fox will not be placed in a river **1 mark:** Warren will not be placed where there is a warren // fox will not be placed where there is a fox **R.** if no sensible attempt at preventing warren/fox from being placed in a river **1 mark:** Fully correct logic in second subroutine	**3**

11	**5**	**Mark is for AO3 (evaluate)** ****SCREEN CAPTURE(S)**** *Must match code from 11.1 to 11.4. Code for these parts must be sensible* **1 mark:** Screen capture(s) indicating which locations are land and which are rivers **A.** incorrect location of rivers if these match those set in 11.2  	**1**
12	**1**	**Marks are for AO3 (programming)** **Structure of subroutine:** 1) **1 mark:** Subroutine created with correct name `CheckIfPathCrossesRiver` **I.** private/public/protected modifiers 2) **1 mark:** Subroutine has four parameters of appropriate data type, which are the coordinates of the two locations to check the path between **I.** `self` parameter in Python answers **I.** additional parameters 3) **1 mark:** Subroutine returns a Boolean value **Horizontal or vertical:** 4) **1 mark:** Repetition structure created that has start and end points that correspond to one coordinate of the locations that need to be checked on the column/row **A.** if start and end points include the columns/rows that contain the fox and warren, even though this is not necessary 5) **1 mark:** Repetition structure will work regardless of whether or not the fox is to the left/right of or above/below the warren (depending on which direction is being checked) **A.** use of separate repetition structures to achieve this 6) **1 mark:** Within repetition structure a check is made of the type of terrain at the appropriate coordinate 7) **1 mark:** If a section of river is detected, subroutine will return true **R.** if subroutine would return true when the path does not cross a river **Other of vertical or horizontal:** 8) **1 mark:** Correct cells are checked regardless of whether or not the fox is to the left/right of or above/below the warren **A.** if start and/or end points include the columns/rows that contain the fox and warren 9) **1 mark:** If a river is detected, subroutine will return true; **R.** if subroutine would return true when the path does not cross a river **MAX 7** if 2 and 5 are used instead of checking terrain type **MAX 5** if code does not use each of the relevant coordinates between fox and warren	**9**

12	**2**	**Marks are for AO3 (programming)** **1 mark:** `CheckIfPathCrossesRiver` subroutine is called within the two repetition structures, with the coordinates of the warren and fox as parameters **1 mark:** If the subroutine returns true, the fox will not eat any rabbits in the warren, otherwise it will eat rabbits if the warren is near enough	**2**
12	**3**	**Mark is for AO3 (evaluate)** ****SCREEN CAPTURE(S)**** *Must match code from 12.1 to 12.2. Code for these parts must be sensible* **1 mark:** Screen capture(s) show that no rabbits are eaten in the warren at (1, 1)  **Note:** Exact rabbit numbers killed/born do not need to match screenshot, but the start and end periods should be 0 and 1.	**1**

VB.NET

07 | **1** | **12**

Example Solution

```
Sub Main()
  Dim Text As String
  Dim LastChar As String
  Dim CountOfLastChar As Integer
  Console.Write("Enter the text to compress: ")
  Text = Console.ReadLine()
  Console.Write("The compressed text is: ")
  LastChar = ""
  CountOfLastChar = 0
  For Count = 0 To Len(Text) - 1
    If Text(Count) = LastChar Then
      CountOfLastChar += 1
    Else
      If LastChar <> "" Then
        Console.Write(LastChar & " " & CountOfLastChar & " ")
      End If
      LastChar = Text(Count)
      CountOfLastChar = 1
    End If
  Next
  Console.Write(LastChar & " " & CountOfLastChar & " ")
  Console.ReadLine()
End Sub
```

09 | **1** | **4**

```
Do
  Console.Write("  Input " & CoordinateName & " coordinate: ")
  Coordinate = CInt(Console.ReadLine())
  If Coordinate < 0 Or Coordinate >= LandscapeSize Then
    Console.WriteLine("Coordinate is outside of landscape, please
try again.")
  End If
Loop While Coordinate < 0 Or Coordinate >= LandscapeSize
```

Alternative answer

```
Do
  Console.Write("  Input " & CoordinateName & " coordinate: ")
  Coordinate = CInt(Console.ReadLine())
  If Coordinate < 0 Or Coordinate >= LandscapeSize Then
    Console.WriteLine("Coordinate is outside of landscape, please
try again.")
  End If
Loop Until Coordinate >= 0 And Coordinate < LandscapeSize
```

10 | **1** | **5**

```
Public Overrides Sub CalculateNewAge()
  MyBase.CalculateNewAge()
  If Gender = Genders.Male Then
```

		ProbabilityOfDeathOtherCauses = ProbabilityOfDeathOtherCauses * 1.5
		Else
		If Age >= 2 Then
		ProbabilityOfDeathOtherCauses = ProbabilityOfDeathOtherCauses + 0.05
		End If
		End If
		End Sub
		
		A. If Age > 1 Then instead of If Age >= 2 Then	

11	**1**	Class Location Public Fox As Fox Public Warren As Warren **Public Terrain As Char** Public Sub New(**ByVal TerrainType As Char**) Fox = Nothing Warren = Nothing **Terrain = TerrainType** End Sub End Class	**3**

11	**2**	For x = 0 To LandscapeSize - 1
		For y = 0 To LandscapeSize - 1
		If x = 5 Or y = 2 Then
		Landscape(x, y) = New Location("R")
		Else
		Landscape(x, y) = New Location("L")
		End If
		Next
		Next
			3

11	**3**	Private Sub DrawLandscape()
		Console.WriteLine()
		Console.WriteLine("TIME PERIOD: " & TimePeriod)
		Console.WriteLine()
		Console.Write(" ")
		For x = 0 To LandscapeSize - 1
		Console.Write(" ")
		If x < 10 Then
		Console.Write(" ")
		End If
		Console.Write(x & " |")
		Next
		Console.WriteLine()
		For x = 0 To LandscapeSize * **5** + 3 **'CHANGE MADE HERE**
		Console.Write("-")
		Next
			2

```
  Console.WriteLine()
  For y = 0 To LandscapeSize - 1
    If y < 10 Then
      Console.Write(" ")
    End If
    Console.Write(" " & y & "|")
    For x = 0 To LandscapeSize - 1
      If Not Me.Landscape(x, y).Warren Is Nothing Then
        If Me.Landscape(x, y).Warren.GetRabbitCount() < 10 Then
          Console.Write(" ")
        End If
        Console.Write(Landscape(x, y).Warren.GetRabbitCount())
      Else
        Console.Write("  ")
      End If
      If Not Me.Landscape(x, y).Fox Is Nothing Then
        Console.Write("F")
      Else
        Console.Write(" ")
      End If
      Console.Write(Landscape(x, y).Terrain)
      Console.Write("|")
    Next
    Console.WriteLine()
  Next
End Sub
```

11 **4**

```
Private Sub CreateNewWarren()
  Dim x As Integer
  Dim y As Integer
  Do
    x = Rnd.Next(0, LandscapeSize)
    y = Rnd.Next(0, LandscapeSize)
  Loop While Not Landscape(x, y).Warren Is Nothing Or Landscape(x,
y).Terrain = "R"
  If ShowDetail Then
    Console.WriteLine("New Warren at (" & x & "," & y & ")")
  End If
  Landscape(x, y).Warren = New Warren(Variability)
  WarrenCount += 1
End Sub

Private Sub CreateNewFox()
  Dim x As Integer
  Dim y As Integer
  Do
    x = Rnd.Next(0, LandscapeSize)
    y = Rnd.Next(0, LandscapeSize)
  Loop While Not Landscape(x, y).Fox Is Nothing Or Landscape(x,
y).Terrain = "R"
  If ShowDetail Then
    Console.WriteLine("  New Fox at (" & x & "," & y & ")")
```

3

		End If Landscape(x, y).Fox = New Fox(Variability) FoxCount += 1 End Sub	

12	**1**	Private Function CheckIfPathCrossesRiver(ByVal FoxX As Integer, ByVal FoxY As Integer, ByVal WarrenX As Integer, ByVal WarrenY As Integer) As Boolean
		Dim xChange As Integer
		Dim yChange As Integer
		Dim x As Integer
		Dim y As Integer
		If FoxX - WarrenX > 0 Then
		xChange = 1
		Else
		xChange = -1
		End If
		If WarrenX <> FoxX Then
		x = WarrenX + xChange
		While x <> FoxX
		If Landscape(x, FoxY).Terrain = "R" Then
		Return True
		End If
		x += xChange
		End While
		End If
		If FoxY - WarrenY > 0 Then
		yChange = 1
		Else
		yChange = -1
		End If
		If WarrenY <> FoxY Then
		y = WarrenY + yChange
		While y <> FoxY
		If Landscape(FoxX, y).Terrain = "R" Then
		Return True
		End If
		y += yChange
		End While
		End If
		Return False
		End Function
			9

12	**2**	Private Sub FoxesEatRabbitsInWarren(ByVal WarrenX As Integer, ByVal WarrenY As Integer) Dim FoodConsumed As Integer Dim PercentToEat As Integer Dim Dist As Double Dim RabbitsToEat As Integer Dim RabbitCountAtStartOfPeriod As Integer = Landscape(WarrenX, WarrenY).Warren.GetRabbitCount()	**2**

```
  For FoxX = 0 To LandscapeSize - 1
    For FoxY = 0 To LandscapeSize - 1
      If Not Landscape(FoxX, FoxY).Fox Is Nothing Then
        If Not CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY) Then
          Dist = DistanceBetween(FoxX, FoxY, WarrenX, WarrenY)
          If Dist <= 3.5 Then
            PercentToEat = 20
          ElseIf Dist <= 7 Then
            PercentToEat = 10
          Else
            PercentToEat = 0
          End If
          RabbitsToEat = CInt(Math.Round(CDbl(PercentToEat *
RabbitCountAtStartOfPeriod / 100)))
          FoodConsumed = Landscape(WarrenX,
WarrenY).Warren.EatRabbits(RabbitsToEat)
          Landscape(FoxX, FoxY).Fox.GiveFood(FoodConsumed)
          If ShowDetail Then
            Console.WriteLine("  " & FoodConsumed & " rabbits
eaten by fox at (" & FoxX & "," & FoxY & ").")
          End If
        End If
      End If
    Next
  Next
End Sub
```

Python 2

07	**1**	`text = raw_input("Enter the text to compress: ")`
		`print "The compressed text is:",`
		`LastChar = ""`
		`CountOfLastChar = 0`
		`for Count in range(0, len(text)):`
		`  if text[Count] == LastChar:`
		`  CountOfLastChar += 1`
		`  else:`
		`  if LastChar != "":`
		`  print LastChar,  CountOfLastChar,`
		`  LastChar = text[Count]`
		`  CountOfLastChar = 1`
		`print LastChar,CountOfLastChar`
			12

09	**1**	`def __InputCoordinate(self, CoordinateName):`
		`  Coordinate = int(raw_input("  Input " + CoordinateName + "`
		`coordinate:"))`
		`  while Coordinate < 0 or Coordinate >= self.__LandscapeSize:`
		`  Coordinate = int(raw_input("Coordinate is outside of`
		`landscape, please try again."))`
		`  return Coordinate`
			4

10	**1**	`def CalculateNewAge(self):`
		`  super(Rabbit, self).CalculateNewAge()`
		`  if self.__Gender == Genders.Male:`
		`  self._ProbabilityOfDeathOtherCauses =`
		`self._ProbabilityOfDeathOtherCauses * 1.5`
		`  else:`
		`  if self._Age >= 2:`
		`  self._ProbabilityOfDeathOtherCauses =`
		`self._ProbabilityOfDeathOtherCauses + 0.05`
			5

11	**1**	`class Location:`
		`  def __init__(self, `**`TerrainType`**`):`
		`  self.Fox = None`
		`  self.Warren = None`
		`  self.Terrain = TerrainType`
			3

11	**2**	`def __CreateLandscapeAndAnimals(self, InitialWarrenCount,`
		`InitialFoxCount, FixedInitialLocations):`
		`  for x in range (0, self.__LandscapeSize):`
		`  for y in range (0, self.__LandscapeSize):`
		`  if x == 5 or y == 2:`
		`  self.__Landscape[x][y] = Location("R")`
		`  else:`
		`  self.__Landscape[x][y] = Location("L")`
		`  if FixedInitialLocations:`
			3

		...	

11	**3**	def __DrawLandscape(self):
		  print
		  print "TIME PERIOD:", str(self.__TimePeriod)
		  print
		  sys.stdout.write(" ")
		  for x in range (0, self.__LandscapeSize):
		  sys.stdout.write(" ")
		  if x < 10:
		  sys.stdout.write(" ")
		  sys.stdout.write(str(x) + " |")
		  print
		  for x in range (0, self.__LandscapeSize * **5** + 3): #CHANGED 4 TO 5
		  sys.stdout.write("-")
		  print
		  for y in range (0, self.__LandscapeSize):
		  if y < 10:
		  sys.stdout.write(" ")
		  sys.stdout.write(str(y) + "|")
		  for x in range (0, self.__LandscapeSize):
		  if not self.__Landscape[x][y].Warren is None:
		  if self.__Landscape[x][y].Warren.GetRabbitCount() < 10:
		  sys.stdout.write(" ")
		
		sys.stdout.write(self.__Landscape[x][y].Warren.GetRabbitCount())
		  else:
		  sys.stdout.write(" ")
		  if not self.__Landscape[x][y].Fox is None:
		  sys.stdout.write("F")
		  else:
		  sys.stdout.write(" ")
		  sys.stdout.write(self.__Landscape[x][y].Terrain)
		  sys.stdout.write("|")
		  print
			2

11	**4**	def __CreateNewWarren(self):
		  x = random.randint(0, self.__LandscapeSize - 1)
		  y = random.randint(0, self.__LandscapeSize - 1)
		  while not self.__Landscape[x][y].Warren is None **or self.__Landscape[x][y].Terrain == "R"**:
		  x = random.randint(0, self.__LandscapeSize - 1)
		  y = random.randint(0, self.__LandscapeSize - 1)
		  if self.__ShowDetail:
		  sys.stdout.write("New Warren at (" + str(x) + "," + str(y) + ")")
		  self.__Landscape[x][y].Warren = Warren(self.__Variability)
		  self.__WarrenCount += 1
		
		def __CreateNewFox(self):
			3

		x = random.randint(0, self.__LandscapeSize - 1) y = random.randint(0, self.__LandscapeSize - 1) while not self.__Landscape[x][y].Fox is None **or** **self.__Landscape[x][y].Terrain == "R":** x = random.randint(0, self.__LandscapeSize - 1) y = random.randint(0, self.__LandscapeSize - 1) if self.__ShowDetail: sys.stdout.write(" New Fox at (" + str(x) + "," + str(y) + ")") self.__Landscape[x][y].Fox = Fox(self.__Variability) self.__FoxCount += 1	

12	**1**	def CheckIfPathCrossesRiver(self, FoxX, FoxY, WarrenX, WarrenY):
		if FoxX - WarrenX > 0:
		xChange = 1
		else:
		xChange = -1
		if WarrenX != FoxX:
		x = WarrenX + xChange
		while x != FoxX:
		if self.__Landscape[x][FoxY].Terrain == "R":
		return True
		x += xChange
		if FoxY - WarrenY > 0:
		yChange = 1
		else:
		yChange = -1
		if WarrenY != FoxY:
		y = WarrenY + yChange
		while y != FoxY:
		if self.__Landscape[FoxX][y].Terrain == "R":
		return True
		y += yChange
		return False
			9

12	**2**	def __FoxesEatRabbitsInWarren(self, WarrenX, WarrenY):
		RabbitCountAtStartOfPeriod = self.__Landscape[WarrenX][WarrenY].Warren.GetRabbitCount()
		for FoxX in range(0, self.__LandscapeSize):
		for FoxY in range (0, self.__LandscapeSize):
		if not self.__Landscape[FoxX][FoxY].Fox is None:
		if not self.CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX, WarrenY): #INDENTATION CHANGED AFTER THIS LINE
		Dist = self.__DistanceBetween(FoxX, FoxY, WarrenX, WarrenY)
		if Dist <= 3.5:
		PercentToEat = 20
		elif Dist <= 7:
		PercentToEat = 10
		else:
		PercentToEat = 0
			2

```
          RabbitsToEat = int(round(float(PercentToEat *
RabbitCountAtStartOfPeriod / 100)))
          FoodConsumed =
self.__Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)
          self.__Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsumed)
          if self.__ShowDetail:
            sys.stdout.write("  " + str(FoodConsumed) + " rabbits
eaten by fox at (" + str(FoxX) + "," + str(FoxY) + ")."  + "\n")
```

Python 3

07	**1**	**Example Solution**
		
		`text = input("Enter the text to compress: ")`
		`print ("The compressed text is: ", end="")`
		`LastChar = ""`
		`CountOfLastChar = 0`
		`for Count in range(0, len(text)):`
		`  if text[Count] == LastChar:`
		`  CountOfLastChar += 1`
		`  else:`
		`  if LastChar != "":`
		`  print (LastChar, " " , CountOfLastChar, " ",end="")`
		`  LastChar = text[Count]`
		`  CountOfLastChar = 1`
		`print (LastChar, " " , CountOfLastChar, " ")`
			12

09	**1**	`def __InputCoordinate(self, CoordinateName):`
		`  Coordinate = int(input("  Input " + CoordinateName + "`
		`coordinate:"))`
		`  **while Coordinate < 0 or Coordinate >= self.__LandscapeSize:**`
		`  **Coordinate = int(input("Coordinate is outside of landscape,`**
		`please try again."))`
		`  return Coordinate`
			4

10	**1**	`def CalculateNewAge(self):`
		`  super(Rabbit, self).CalculateNewAge()`
		`  if self.__Gender == Genders.Male:`
		`  self._ProbabilityOfDeathOtherCauses =`
		`self._ProbabilityOfDeathOtherCauses * 1.5`
		`  else:`
		`  if self._Age >= 2:`
		`  self._ProbabilityOfDeathOtherCauses =`
		`self._ProbabilityOfDeathOtherCauses + 0.05`
			5

11	**1**	`class Location:`
		`  def __init__(self, **TerrainType**):`
		`  self.Fox = None`
		`  self.Warren = None`
		` **self.Terrain = TerrainType**
			3

11	**2**	`def __CreateLandscapeAndAnimals(self, InitialWarrenCount,` `InitialFoxCount, FixedInitialLocations):` `  for x in range (0, self.__LandscapeSize):` `  for y in range (0, self.__LandscapeSize):` `  **if x == 5 or y == 2:** ` **self.__Landscape[x][y] = Location("R")** ` **else:**	**3**

		self.__Landscape[x][y] = Location("L") if FixedInitialLocations: ...	

11	**3**	def __DrawLandscape(self):
		print()
		print("TIME PERIOD:", self.__TimePeriod)
		print()
		print(" ", end = "")
		for x in range (0, self.__LandscapeSize):
		print(" ", end = "")
		if x < 10:
		print(" ", end = "")
		print(x, "|", end = "")
		print()
		for x in range (0, self.__LandscapeSize * **5** + 3): #CHANGE
		print("-", end = "")
		print()
		for y in range (0, self.__LandscapeSize):
		if y < 10:
		print(" ", end = "")
		print("", y, "|", sep = "", end = "")
		for x in range (0, self.__LandscapeSize):
		if not self.__Landscape[x][y].Warren is None:
		if self.__Landscape[x][y].Warren.GetRabbitCount() < 10:
		print(" ", end = "")
		print(self.__Landscape[x][y].Warren.GetRabbitCount(), end = "")
		else:
		print(" ", end = "")
		if not self.__Landscape[x][y].Fox is None:
		print("F", end = "")
		else:
		print(" ", end = "")
		print(self.__Landscape[x][y].Terrain, end = "")
		print("|", end = "")
		print()
			2

11	**4**	def __CreateNewWarren(self):
		x = random.randint(0, self.__LandscapeSize - 1)
		y = random.randint(0, self.__LandscapeSize - 1)
		while not self.__Landscape[x][y].Warren is None **or** **self.__Landscape[x][y].Terrain == "R":**
		x = random.randint(0, self.__LandscapeSize - 1)
		y = random.randint(0, self.__LandscapeSize - 1)
		if self.__ShowDetail:
		print("New Warren at (", x, ",", y, ")", sep = "")
		self.__Landscape[x][y].Warren = Warren(self.__Variability)
		self.__WarrenCount += 1
		
		def __CreateNewFox(self):
			3

		x = random.randint(0, self.__LandscapeSize - 1) y = random.randint(0, self.__LandscapeSize - 1) while not self.__Landscape[x][y].Fox is None **or** **self.__Landscape[x][y].Terrain == "R":** x = random.randint(0, self.__LandscapeSize - 1) y = random.randint(0, self.__LandscapeSize - 1) if self.__ShowDetail: print(" New Fox at (", x, ",", y, ")", sep = "") self.__Landscape[x][y].Fox = Fox(self.__Variability) self.__FoxCount += 1	

12	**1**	def CheckIfPathCrossesRiver(self, FoxX, FoxY, WarrenX, WarrenY):
		if FoxX - WarrenX > 0:
		xChange = 1
		else:
		xChange = -1
		if WarrenX != FoxX:
		x = WarrenX + xChange
		while x != FoxX:
		if self.__Landscape[x][FoxY].Terrain == "R":
		return True
		x += xChange
		if FoxY - WarrenY > 0:
		yChange = 1
		else:
		yChange = -1
		if WarrenY != FoxY:
		y = WarrenY + yChange
		while y != FoxY:
		if self.__Landscape[FoxX][y].Terrain == "R":
		return True
		y += yChange
		return False
			9

12	**2**	def __FoxesEatRabbitsInWarren(self, WarrenX, WarrenY):
		RabbitCountAtStartOfPeriod = self.__Landscape[WarrenX][WarrenY].Warren.GetRabbitCount()
		for FoxX in range(0, self.__LandscapeSize):
		for FoxY in range (0, self.__LandscapeSize):
		if not self.__Landscape[FoxX][FoxY].Fox is None:
		if not self.CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX, WarrenY): #INDENTATION CHANGED AFTER THIS LINE
		Dist = self.__DistanceBetween(FoxX, FoxY, WarrenX, WarrenY)
		if Dist <= 3.5:
		PercentToEat = 20
		elif Dist <= 7:
		PercentToEat = 10
		else:
		PercentToEat = 0
		RabbitsToEat = int(round(float(PercentToEat * RabbitCountAtStartOfPeriod / 100)))
			2

```
            FoodConsumed =
self.__Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)

self.__Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsumed)
            if self.__ShowDetail:
              print("  ", FoodConsumed, " rabbits eaten by fox at
(", FoxX, ",", FoxY, ").", sep = "")
```

C#

07	**1**	`string Text = "";`
		`string LastChar = "";`
		`int CountOfLastChar = 0;`
		`Console.Write("Enter the text to compress: ");`
		`Text = Console.ReadLine();`
		`Console.Write("The compressed text is: ");`
		`for (int Count = 0; Count < Text.Length ; Count++)`
		`{`
		`  if (Text[Count].ToString() == LastChar )`
		`  {`
		`  CountOfLastChar++;`
		`  }`
		`  else`
		`  {`
		`  if (LastChar != "")`
		`  {`
		`  Console.Write(LastChar + " " + CountOfLastChar + " ");`
		`  }`
		`  LastChar = Text[Count].ToString();`
		`  CountOfLastChar = 1;`
		`  }`
		`}`
		`Console.Write(LastChar + " " + CountOfLastChar + " ");`
		`Console.ReadLine();`
			12

09	**1**	**`do`**
		`{`
		`  Console.Write("  Input " + Coordinatename + " coordinate: ");`
		`  Coordinate = Convert.ToInt32(Console.ReadLine());`
		`  if ((Coordinate < 0) || (Coordinate >= LandscapeSize))`
		`  {`
		`  Console.WriteLine("Coordinate is outside of landscape, please try again.");`
		`  }`
		`} while ((Coordinate < 0) || (Coordinate >= LandscapeSize));`
			4

10	**1**	`public override void CalculateNewAge()`
		`{`
		`  base.CalculateNewAge();`
		`  if (Gender == Genders.Male)`
		`  {`
		`  ProbabilityOfDeathOtherCauses = ProbabilityOfDeathOtherCauses * 1.5;`
		`  }`
		`  else`
		`  {`
		`  if (Age >= 2)`
		`  {`
		` ProbabilityOfDeathOtherCauses =
			5

		ProbabilityOfDeathOtherCauses + 0.05; } } }	

11	**1**	class Location
		{
		public Fox Fox;
		public Warren Warren;
		public char Terrain;
		
		public Location(**char Terraintype**)
		{
		Fox = null;
		Warren = null;
		Terrain = Terraintype;
		}
		}
			3

11	**2**	for (int x = 0; x < LandscapeSize; x++)
		{
		for (int y = 0; y < LandscapeSize; y++)
		{
		if ((x == 5) || (y == 2))
		{
		Landscape[x, y] = new Location('R');
		}
		else
		{
		Landscape[x, y] = new Location('L');
		}
		}
		}
			3

11	**3**	private void DrawLandscape()
		{
		Console.WriteLine();
		Console.WriteLine("TIME PERIOD: "+TimePeriod);
		Console.WriteLine();
		Console.Write(" ");
		for (int x = 0; x < LandscapeSize; x++)
		{
		Console.Write(" ");
		if (x < 10) { Console.Write(" "); }
		Console.Write(x + " |");
		}
		Console.WriteLine();
		for (int x = 0; x <= LandscapeSize * **5** + 3; x++)
		{
		Console.Write("-");
			2

```
    }
    Console.WriteLine();
    for (int y = 0; y < LandscapeSize; y++)
    {
        if (y < 10) { Console.Write(" "); }
        Console.Write(" " + y + "|");
        for (int x = 0; x < LandscapeSize; x++)
        {
            if (Landscape[x, y].Warren != null)
            {
                if (Landscape[x, y].Warren.GetRabbitCount() < 10)
                {
                  Console.Write(" ");
                }
                Console.Write(Landscape[x,
y].Warren.GetRabbitCount());
            }
            else { Console.Write("  ");  }
            if (Landscape[x, y].Fox != null)
            {
              Console.Write("F");
            }
            else
            {
              Console.Write(" ");
            }
            Console.Write(Landscape[x, y].Terrain);
            Console.Write("|");
        }
        Console.WriteLine();
    }
}
```

11 **4**

```
private void CreateNewWarren()
{
    int x, y;
    do
    {
        x = Rnd.Next(0, LandscapeSize);
        y = Rnd.Next(0, LandscapeSize);
    }  while ((Landscape[x, y].Warren != null) || (Landscape[x,
y].Terrain == 'R'));
;
    if (ShowDetail)
    {
        Console.WriteLine("New Warren at (" + x + "," + y + ")");
    }
    Landscape[x, y].Warren = new Warren(Variability);
    WarrenCount++;
}

private void CreateNewFox()
```

3

```
{
    int x, y;
    do
    {
        x = Rnd.Next(0, LandscapeSize);
        y = Rnd.Next(0, LandscapeSize);
    } while ((Landscape[x, y].Fox != null) || (Landscape[x,
y].Terrain == 'R'));
    if (ShowDetail) { Console.WriteLine("  New Fox at (" + x + ","
+ y + ")"); }
    Landscape[x, y].Fox = new Fox(Variability);
    FoxCount++;
}
```

12 **1**

```
private bool CheckIfPathCrossesRiver(int FoxX, int FoxY, int
WarrenX, int WarrenY)
{
    int xChange, yChange, x, y;
    if (FoxX - WarrenX > 0)
    {
        xChange = 1;
    }
    else
    {
        xChange = -1;
    }
    if (WarrenX != FoxX)
    {
        x = WarrenX + xChange;
        while(x != FoxX)
        {
            if (Landscape[x, FoxY].Terrain == 'R')
            {
                return true;
            }
            x += xChange;
        }
    }
    if (FoxY - WarrenY > 0)
    {
        yChange = 1;
    }
    else
    {
        yChange = -1;
    }
    if (WarrenY != FoxY)
    {
        y = WarrenY + yChange;
        while(y != FoxY)
        {
            if (Landscape[FoxX, y].Terrain == 'R')
```

9

```
            {
                return true;
            }
            y += yChange;
        }
    }
    return false;
}
```

12 **2**

```
private void FoxesEatRabbitsInWarren(int WarrenX, int WarrenY)
{
    int FoodConsumed;
    int PercentToEat;
    double Dist;
    int RabbitsToEat;
    int RabbitCountAtStartOfPeriod = Landscape[WarrenX,
WarrenY].Warren.GetRabbitCount();
    for (int FoxX = 0; FoxX < LandscapeSize; FoxX++)
    {
        for (int FoxY = 0; FoxY < LandscapeSize; FoxY++)
        {
            if (Landscape[FoxX, FoxY].Fox != null)
            {
                if (!CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY))
                {
                    Dist = DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
                    if (Dist <= 3.5)
                    {
                        PercentToEat = 20;
                    }
                    else if (Dist <= 7)
                    {
                        PercentToEat = 10;
                    }
                    else
                    {
                        PercentToEat = 0;
                    }
                    RabbitsToEat =
(int)Math.Round((double)(PercentToEat * RabbitCountAtStartOfPeriod
/ 100.0));
                    FoodConsumed = Landscape[WarrenX,
WarrenY].Warren.EatRabbits(RabbitsToEat);
                    Landscape[FoxX,
FoxY].Fox.GiveFood(FoodConsumed);
                    if (ShowDetail)
                    {
                        Console.WriteLine("  " + FoodConsumed + "
rabbits eaten by fox at (" + FoxX + "," + FoxY + ").");
                    }
```

2

		} } } } }	

Pascal

07 **1** **Example solution** **12**

```
var
  Text : string;
  LastChar : string;
  CountOfLastChar : integer;
  Count : integer;
begin
  write('Enter the text to compress: ');
  readln(Text);
  write('The compressed text is: ');
  LastChar := '';
  CountOfLastChar := 0;
  for Count := 1 to Length(Text) do
    begin
      if Text[Count] = LastChar then
        inc(CountOfLastChar)
      else
        begin
          if LastChar <> '' then
            write(LastChar, ' ', CountOfLastChar, ' ');
          LastChar := Text[Count];
          CountOfLastChar := 1;
        end;
    end;
  write(LastChar, ' ', CountOfLastChar, ' ');
  readln;
end.
```

09 **1** **4**

```
repeat
  write('  Input ' , CoordinateName, ' coordinate: ');
  readln(Coordinate);
  if (Coordinate < 0) or (Coordinate >= LandscapeSize) then
    writeln('Coordinate is outside of landscape, please try
again.');
until (Coordinate >= 0) and (Coordinate < LandscapeSize);
```

10 **1** **5**

```
Procedure Rabbit.CalculateNewAge();
  begin
    inherited;
    if Gender = Male then
      ProbabilityOfDeathOtherCauses :=
ProbabilityOfDeathOtherCauses * 1.5
    else
      if Age >= 2 then
        ProbabilityOfDeathOtherCauses :=
ProbabilityOfDeathOtherCauses + 0.05;
  end;
```

11	**1**	type
		  Location = class
		  public
		  Fox : Fox;
		  Warren : Warren;
		  Terrain : char;
		  constructor New(**TerrainType : char**);
		  end;
		
		constructor Location.New(**TerrainType : char**);
		  begin
		  Fox := nil;
		  Warren := nil;
		  Terrain := TerrainType;
		  end;
			3

11	**2**	for x := 0 to LandscapeSize - 1 do
		  for y := 0 to LandscapeSize - 1 do
		  if (x = 5) or (y = 2) then
		  Landscape[x][y] := Location.New('R')
		  else
		  Landscape[x][y] := Location.New('L');
			3

11	**3**	procedure Simulation.DrawLandscape();
		  var
		  x : integer;
		  y : integer;
		  begin
		  writeln;
		  writeln('TIME PERIOD: ', TimePeriod);
		  writeln;
		  write(' ');
		  for x := 0 to LandscapeSize - 1 do
		  begin
		  write(' ');
		  if x < 10 then
		  write(' ');
		  write(x, ' |');
		  end;
		  writeln;
		  for x:=0 to LandscapeSize * **5** + 3 do //CHANGE MADE HERE
		  write('-');
		  writeln;
		  for y := 0 to LandscapeSize - 1 do
		  begin
		  if y < 10 then
		  write(' ');
		  write(' ', y, '|');
		  for x:= 0 to LandscapeSize - 1 do
		  begin
			2

```
            if not(self.Landscape[x][y].Warren = nil) then
              begin
                  if self.Landscape[x][y].Warren.GetRabbitCount()
< 10 then
                    write(' ');
                  write(Landscape[x][y].Warren.GetRabbitCount());
              end
            else
              write('  ');
            if not(self.Landscape[x][y].fox = nil) then
              write('F')
            else
              write(' ');
            write(Landscape[x][y].Terrain);
            write('|');
          end;
          writeln;
      end;
  end;
```

11 | **4**

```
procedure Simulation.CreateNewWarren();
  var
    x : integer;
    y : integer;
  begin
    repeat
      x := random(LandscapeSize);
      y := random(LandscapeSize);
    until (Landscape[x][y].Warren = Nil) and
(not(Landscape[x][y].Terrain = 'R'));
    if ShowDetail then
      writeln('New Warren at (', x, ',', y, ')');
    Landscape[x][y].Warren := Warren.New(Variability);
    inc(WarrenCount);
  end;

procedure Simulation.CreateNewFox();
  var
    x : integer;
    y : integer;
  begin
    randomize();
    repeat
      x := Random(LandscapeSize);
      y := Random(LandscapeSize);
    until (Landscape[x][y].fox = Nil) and
(not(Landscape[x][y].Terrain = 'R'));
    if ShowDetail then
      writeln('  New Fox at (',x, ',',y, ')');
    Landscape[x][y].Fox := Fox.New(Variability);
    inc(FoxCount);
  end;
```

3

12	**1**	`function Simulation.CheckIfPathCrossesRiver(FoxX : integer; Foxy : integer; WarrenX : integer; WarrenY : integer) : boolean;`
		`  var`
		`  xChange : integer;`
		`  yChange : integer;`
		`  x : integer;`
		`  y : integer;`
		`  Answer : boolean;`
		`  begin`
		`  Answer := False;`
		`  if (FoxX - WarrenX) > 0 then`
		`  xChange := 1`
		`  else`
		`  xChange := -1;`
		`  if WarrenX <> FoxX then`
		`  begin`
		`  x := warrenX + xChange;`
		`  if x <> FoxX then`
		`  repeat`
		`  if Landscape[x][FoxY].Terrain = 'R' then`
		`  Answer := True;`
		`  x := x + xChange;`
		`  until x = FoxX;`
		`  end;`
		`  if (FoxY - WarrenY) > 0 then`
		`  yChange := 1`
		`  else`
		`  yChange := -1;`
		`  if WarrenY <> FoxY then`
		`  begin`
		`  y := WarrenY + yChange;`
		`  if y <> FoxY then`
		`  repeat`
		`  if Landscape[FoxX][y].Terrain = 'R' then`
		`  Answer := True;`
		`  y := y + yChange;`
		`  until y = FoxY;`
		`  end;`
		`  CheckIfPathCrossesRiver := Answer;`
		`  end;`
			9

12	**2**	`procedure Simulation.FoxesEatRabbitsInWarren(WarrenX : integer; WarrenY : integer);`
		`  var`
		`  FoodConsumed : integer;`
		`  PercentToEat : integer;`
		`  Dist : double;`
		`  RabbitsToEat : integer;`
		`  RabbitCountAtStartOfPeriod : integer;`
		`  FoxX : integer;`
		`  FoxY : integer;`
			2

```
  begin
    RabbitCountAtStartOfPeriod :=
Landscape[WarrenX][WarrenY].Warren.GetRabbitCount();
    for FoxX := 0 to LandscapeSize - 1 do
      for FoxY := 0 to LandscapeSize - 1 do
        if not(Landscape[FoxX][FoxY].fox = nil) then
          if not(CheckIfPathCrossesRiver(FoxX, Foxy, WarrenX,
WarrenY)) then
            begin
              Dist := DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
              if Dist <= 3.5 then
                PercentToEat := 20
              else if Dist <= 7 then
                PercentToEat := 10
              else
                PercentToEat := 0;
              RabbitsToEat := round(PercentToEat *
RabbitCountAtStartOfPeriod / 100);
              FoodConsumed :=
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat);
              Landscape[FoxX][FoxY].fox.GiveFood(FoodConsumed);
              if ShowDetail then
                writeln('  ', FoodConsumed, ' rabbits eaten by fox
at (', FoxX, ',', FoxY, ')');
            end;
  end;
```

Java

07 **1**

```
public static void main(String[] args)
{
    String Text;
    char LastChar;
    int CountOfLastChar;
    Console.print("Enter the text to compress: ");
    Text = Console.readLine();
    Console.print("The compressed text is: ");
    LastChar = ' ';
    CountOfLastChar = 0;
    for (int Count = 0; Count < Text.length(); Count++)
    {
        char CurrentChar = Text.charAt(Count);
        if(CurrentChar == LastChar)
        {
            CountOfLastChar += 1;
        }
        else
        {
            if (LastChar !=' ')
            {
              Console.print(LastChar + " " + CountOfLastChar + "
");
            }
            LastChar = CurrentChar;
            CountOfLastChar = 1;
        }
    }
    Console.print(LastChar + " " + CountOfLastChar + " ");
    Console.readLine();
}
```

12

09 **1**

```
private int InputCoordinate(char CoordinateName)
{
    int Coordinate;
    do
    {
        Coordinate = Console.readInteger("  Input " +
CoordinateName + " coordinate: ");
        if (Coordinate >= LandscapeSize || Coordinate < 0)
        {
            Console.println("Coordinate is outside of landscape,
please try again.");
        }
    }while (Coordinate >= LandscapeSize || Coordinate < 0);
    return Coordinate;
}
```

4

10	**1**	**@Override** **public void CalculateNewAge()** **{** **super.CalculateNewAge();** **if (Gender == Genders.Male)** **{** **ProbabilityOfDeathOtherCauses *= 1.5;** **}** **else if(Age >= 2)** **{** **ProbabilityOfDeathOtherCauses += 0.05;** **}** **}**	**5**

11	**1**	class Location
		{
		public Fox Fox;
		public Warren Warren;
		public char Terrain;
		
		public Location(**char Terrain**)
		{
		Fox = null;
		Warren = null;
		this.Terrain = Terrain;
		}
		}
			3

11	**2**	for(int x = 0 ; x < LandscapeSize; x++)
		{
		for(int y = 0; y < LandscapeSize; y++)
		{
		if(x==5||y==2)
		{
		Landscape[x][y] = new Location('R');
		}
		else
		{
		Landscape[x][y] = new Location('L');
		}
		}
		}
			3

11	**3**	private void DrawLandscape()
		{
		Console.println();
		Console.println("TIME PERIOD: " + TimePeriod);
		Console.println();
		Console.print(" ");
		for(int x = 0; x < LandscapeSize; x++)
			2

```
    {
        Console.print(" ");
        if (x < 10)
        {
            Console.print(" ");
        }
        Console.print(x + " |");
    }
    Console.println();
    for(int x = 0; x < LandscapeSize * 5 + 4; x++) //Change made
here
    {
        Console.print("-");
    }
    Console.println();
    for(int y = 0; y < LandscapeSize; y++)
    {
        if(y < 10 )
        {
            Console.print(" ");
        }
        Console.print(" " + y + "|");
        for(int x = 0; x < LandscapeSize; x++)
        {
            if ( Landscape[x][y].Warren != null )
            {
                if ( Landscape[x][y].Warren.GetRabbitCount() < 10
)
                {
                    Console.print(" ");
                }

Console.print(Landscape[x][y].Warren.GetRabbitCount());
            }
            else
            {
                Console.print("  ");
            }
            if ( Landscape[x][y].Fox != null)
            {
                Console.print("F");
            }
            else
            {
                Console.print(" ");
            }
            Console.print(Landscape[x][y].Terrain);
            Console.print("|");
        }
        Console.println();
    }
}
```

11	**4**	`private void CreateNewWarren()`
		`{`
		`  int x;`
		`  int y;`
		`  do`
		`  {`
		`  x = Rnd.nextInt( LandscapeSize);`
		`  y = Rnd.nextInt( LandscapeSize);`
		`  } while (Landscape[x][y].Warren != null ||`
		`**Landscape[x][y].Terrain == 'R'**);`
		`  if (ShowDetail)`
		`  {`
		`  Console.println("New Warren at (" + x + "," + y + ")");`
		`  }`
		`  Landscape[x][y].Warren = new Warren(Variability);`
		`  WarrenCount += 1;`
		`}`
		
		`private void CreateNewFox()`
		`{`
		`  int x;`
		`  int y;`
		`  do`
		`  {`
		`  x = Rnd.nextInt( LandscapeSize);`
		`  y = Rnd.nextInt( LandscapeSize);`
		`  }while (Landscape[x][y].Fox != null **|| Landscape[x][y].Terrain`
		`== 'R'**);`
		`  if (ShowDetail)`
		`  {`
		`  Console.println("  New Fox at (" + x + "," + y + ")");`
		`  }`
		`  Landscape[x][y].Fox = new Fox(Variability);`
		`  FoxCount += 1;`
		`}`
			3

12	**1**	`private boolean CheckIfPathCrossesRiver(int FoxX, int FoxY, int`
		`WarrenX, int WarrenY)`
		`{`
		`  int xChange, yChange;`
		`  if (FoxX-WarrenX > 0)`
		`  {`
		`  xChange = 1;`
		`  }`
		`  else`
		`  {`
		`  xChange = -1;`
		`  }`
		`  if (WarrenX != FoxX)`
		`  {`
		`  for (int x = WarrenX + xChange; x != FoxX; x = x +`
		`xChange)`
			9

```
        {
            if (Landscape[x][FoxY].Terrain == 'R')
            {
                return true;
            }
        }
    }
    if (FoxY - WarrenY > 0)
    {
        yChange = 1;
    }
    else
    {
        yChange = -1;
    }
    if (WarrenY != FoxY)
    {
        for (int y = WarrenY + yChange; y != FoxY; y = y +
yChange)
        {
            if (Landscape[FoxX][y].Terrain == 'R')
            {
                return true;
            }
        }
    }
    return false;
}
```

12 **2**

```
private void FoxesEatRabbitsInWarren(int WarrenX, int WarrenY)
{
    int FoodConsumed;
    int PercentToEat;
    double Dist;
    int RabbitsToEat;
    int RabbitCountAtStartOfPeriod =
Landscape[WarrenX][WarrenY].Warren.GetRabbitCount();
    for(int FoxX = 0; FoxX < LandscapeSize; FoxX++)
    {
        for(int FoxY = 0; FoxY < LandscapeSize; FoxY++)
        {
            if (Landscape[FoxX][FoxY].Fox != null)
            {
                if (!CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY))
                {
                    Dist = DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
                    if ( Dist <= 3.5 )
                    {
                        PercentToEat = 20;
                    }
```

2

```
                    else if ( Dist <= 7 )
                    {
                        PercentToEat = 10;
                    }
                    else
                    {
                        PercentToEat = 0;
                    }
                    RabbitsToEat =
(int)(Math.round((double)(PercentToEat *
RabbitCountAtStartOfPeriod / 100)));
                    FoodConsumed =
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat);

Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsumed);
                    if ( ShowDetail )
                    {
                        Console.println("  " + FoodConsumed + "
rabbits eaten by fox at (" + FoxX + "," + FoxY + ").");
                    }
                }
            }
        }
    }
}
```