A-level
COMPUTER SCIENCE
7517/1

Paper 1

Mark scheme

June 2022

Version: 1.0 Final

226A7517/1/MS

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

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

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

Level of response marking instructions

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

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

Step 1 Determine a level

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

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

Step 2 Determine a mark

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

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

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

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

A-level Computer Science

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

June 2022

The following annotation is used in the mark scheme:

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

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

eg

In question **05.1**, 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><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>01</td><td></td><td>
All marks AO1 (knowledge)
<table>
<tr><th>Algorithm</th><th>Time Complexity</th></tr>
<tr><td>Binary tree search</td><td>O(log n)</td></tr>
<tr><td>Bubble sort</td><td>O(n^2)</td></tr>
<tr><td>Linear search</td><td>O(n)</td></tr>
<tr><td>Merge sort</td><td>O(n log n)</td></tr>
</table>
A. O(n x log n) NE. O(log n)

I. missing brackets
I. missing O

Mark as follows:

1 mark: 1st row correct
1 mark: 2nd row correct
1 mark: 3rd row correct
</td><td>3</td></tr>
</table>

Question			Marks
02	1	**All marks for AO1 (understanding)** 1. Check the queue is (not already) empty; 2. Compare the value of the front pointer with the maximum size of the array; 3. If equal then front pointer becomes one; **A.** index of the first position in the array instead of one 4. Otherwise, add one to the front pointer; **Alternative answer 1** 1. Check the queue is (not already) empty; 2. Compare the value of the front pointer with the maximum size of the array minus one; 3. If equal then front pointer becomes zero; **A.** index of the first position in the array instead of zero 4. Otherwise, add one to the front pointer; **Alternative answer 2** 1. Check the queue is (not already) empty; 2. Add one to the front pointer; 3. Compare the value of the front pointer with the maximum size of the array; 4. If equal then front pointer becomes zero; **A.** index of the first position in the array instead of zero **Alternative answer 3** 1. Check the queue is (not already) empty; 2. Add one to the front pointer; 3. Compare the value of the front pointer with the maximum size of the array plus one; 4. If equal then front pointer becomes one; **A.** index of the first position in the array instead of one **Alternative answer 4** 1. Check the queue is (not already) empty; 2. Add one to the front pointer; 3. Use modulus/modulo operator/function with new value of front pointer; 4. Use modulus/modulo operator/function with maximum size of array; **Max 3** if any errors	4

Question			Marks
02	2	**All marks for AO1 (understanding)** Static data structures have storage size determined at compile-time / before program is run / when program code is translated / before the data structure is first used // dynamic data structures can grow / shrink during execution / at run-time // static data structures have fixed (maximum) size // size of dynamic data structures can change; Static data structures can waste storage space / memory if the number of data items stored is small relative to the size of the structure // dynamic data structures only take up the amount of storage space required for the actual data; Dynamic data structures require (memory to store) pointers to the next item(s) // static data structures (typically) do not need (memory to store) pointers; Static data structures (typically) store data in consecutive memory locations // dynamic data structures (typically) do not store data in consecutive memory locations; **Max 3**	3
02	3	**Mark is for AO2 (apply)** Jib;	1
02	4	**Mark is for AO2 (apply)** Jib;	1
02	5	**All marks for AO1 (understanding)** (Until the queue is empty) repeatedly remove / delete (the front item) from the queue and push it on to the stack; (Until the stack is empty) repeatedly pop items from the stack and add them to the (rear of the) queue;	2

Question			Marks
03	1	**Mark is for AO2 (analyse)** Statement 1 can't be correct because it means Statement 5 / Statement 6 is true which means Statement 1 is false; Statement 1 can't be correct because it would mean Statement 2 is correct which would mean all of the other statements have to be both correct and incorrect; Statement 1 can't be correct because it would mean Statement 4 is correct which means that Statements 2 and 3 have to be both correct and incorrect; Questions says only one of the statements is true so Statement 1 can't be true as that means more than one statement would be true; **Max 1**	1
03	2	**Mark is for AO2 (analyse)** (Statement) 5;	1
03	3	**All marks AO2 (analyse)** Statement 3 can't be correct because Statement 1 is false; Statement 3 can't be correct because the question says only one of the statements is correct; Statement 3 can't be correct because that would mean Statement 2 would be a contradiction as this would mean Statement 3 would have to be incorrect; If Statement 2 is true then Statement 4 has to be false. As Statements 1 and 3 are false for Statement 4 to be false Statement 2 has to be false as well (otherwise one of the above would be true). This is a contradiction so Statement 2 can't be true; Statements 1, 2 and 3 are false so Statement 4 is false; If Statement 6 is true then 5 has to be false implying at least one of Statements 1 to 4 have to be true but they are all false so Statement 6 has to be false; **Max 2**	2

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>04</td><td>1</td><td>
All marks AO1 (understanding)
<table>
<tr><th></th><th>True or False?</th></tr>
<tr><td>Calculates the shortest path between a node and other nodes in a graph</td><td>True</td></tr>
<tr><td>Can be used to prove the Halting Problem cannot be solved</td><td>False</td></tr>
<tr><td>Can be used with both directed and undirected graphs</td><td>True</td></tr>
<tr><td>Can be used with both weighted and unweighted graphs</td><td>False</td></tr>
</table>
Mark as follows:

1 mark: three rows correct
2 marks: all rows correct.
</td><td>2</td></tr>
<tr><td>04</td><td>2</td><td>
Mark is for AO1 (knowledge)

A subroutine that calls itself;
</td><td>1</td></tr>
<tr><td>04</td><td>3</td><td>
All marks AO2 (apply)
<table>
<tr><th><code>Count</code></th><th>Value returned</th></tr>
<tr><td><code>0</code></td><td></td></tr>
<tr><td><code>1</code></td><td><code>False</code></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
</table>
Mark as follows:

1. Column for <code>Count</code> is correct I. repeated consecutive instances of values
I. blank cells
2. Value returned is <code>False</code>

Max 1 if any errors
</td><td>2</td></tr>
</table>

04	**4**	**Mark is for AO2 (analyse)** **A.** any suitable indicators used instead of 0 and 1 **A.** blank cell instead of 0 **R.** if usage inconsistent	**1**

	0	1	2	3
0	0	1	1	1
1		0	0	1
2			0	0
3				0

04	5	**All marks AO2 (apply)**	6

			Visited				
Subroutine call	**V**	**P**	**[0]**	**[1]**	**[2]**	**[3]**	**N**
			False	False	False	False	
G(0, -1)	0	-1	True				1
G(1, 0)	1	0		True			0
							3
G(3, 1)	3	1				True	0
G(1, 0)							
G(0, -1)							
Final value returned:	True						

Mark as follows:

1. `Visited[0]` set to `True` and then not changed
2. `Visited[1]` set to `True` and then not changed, `Visited[3]` set to `True` and then not changed, `Visited[2]` always has value of `False`
3. Second subroutine call is `G(1, 0)` **I.** repeated consecutive instances of this call
4. Third and final subroutine call is `G(3, 1)` **I.** repeated consecutive instances of this call **I.** missing calls `G(1, 0)` and `G(0, -1)`
5. Value returned is `True`
6. `N` column contains correct values **A.** values of 3 in 2nd last cell for `N` and value of 1 in last cell for `N`, instead of the two blank cells

Max 5 if any errors

04	**6**	**Mark is for AO2 (analyse)** Determine if a graph contains a cycle or not;	**1**
04	**7**	**Mark is for AO2 (analyse)** Depth-first search;	**1**
04	**8**	**Mark is for AO2 (analyse)** The graph is a tree;	**1**

Question			Marks
05	1	**4 marks for AO3 (design) and 8 marks for AO3 (programming)** **Mark Scheme**	12

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. 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 relevant prompts, inputs the required string, has at least one iterative structure and at least one selection structure and uses appropriate variables to store most of the needed data. An attempt has been made to swap the positions of vowels in the string, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made.	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 string concatenation is needed when swapping vowels in the string // identifying that swapping items in a list of characters is needed. 2. Identifying that a loop is needed that repeats a number of times determined by the word entered by the user // identifying that a loop is needed that repeats a number of times determined by the number of vowels in the word entered by the user. 3. Identifying that two integer variables are needed to store positions of characters in the string // identifying that an ordered list of vowels in the string needs to be created // identifying one integer variable is needed to show the distance from the start and end of the string (**R.** if no attempt to use this integer with the start and end positions of the string). 4. Selection structure that checks if a character is a vowel **A.** more than one selection structure used **R.** if no attempt at comparing with each of the five vowels. 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 asking user to enter a string followed by user input being assigned to appropriate variable. 6. Iterative structure that repeats a number of times that is sufficient to check all the characters in the string. 7. Correctly checks if a character is a vowel. 8. Correctly checks all characters in the string to see if they are vowels. 9. Swaps/moves the position of two characters in the string. 10. Program only moves/changes the position of vowels. 11. Program works correctly if a string contains one vowel and works correctly if a string contains no vowels. **R.** if program does not attempt to swap positions of vowels or identify that there are less than two vowels. 12. Program works correctly under all circumstances. **I.** additional loop to get program to repeat multiple times. **DPT.** mark points 7 and 8 if only checks for some vowels or includes at most one non-vowel character. **Max 11** if any errors	

Question			Marks
05	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **05.1**, including prompts on screen capture matching those in code.* *Code for **05.1** must be sensible.* Screen capture showing the string `persepolis` being entered and then the string `pirsopeles` being displayed and screen capture showing the string `darius` being entered and then the string `durias` being displayed and screen capture showing the string `xerxes` being entered and then the string `xerxes` being displayed; **I.** order of tests   	1

Question			Marks
06	1	**Mark is for AO2 (analyse)** Inheritance;	1
06	2	**Mark is for AO2 (analyse)** `Card`; **R.** if spelt incorrectly **R.** if any additional code **I.** case	1
06	3	**Marks are for AO1 (understanding)** Public means it can be accessed / seen outside of the class it is in; Protected means it can be accessed / seen in the class it is in and in any subclasses // protected means it can be accessed / seen in the class it is in and in any classes derived / inheriting from it; **A.** Protected means it can be accessed / seen by any class in the same package (and by subclasses in any package) (**Java only**)	2
06	4	**Mark is for AO1 (knowledge)** When a derived class / subclass has a different implementation for a method / function / subroutine to the class it inherits from / from the base class;	1

Question			Marks
07	1	**Mark is for AO2 (analyse)** There will end up being two copies of the second card (**A.** a card); The first card (**A.** a card) will be overwritten; **Max 1**	1
07	2	**Mark is for AO2 (apply)** Because sets are unordered // because the cards have an order; **R.** because sets only allow one instance of a value to be stored	1
07	3	**One mark for AO2 (apply)** Hash algorithm / function applied to `CardNumber` (**NE.** primary key); **Two marks for AO1 (understanding)** Result indicates location that card should be stored in; If there is already a card in that location a method is needed to deal with collisions; **A.** description of any suitable method for dealing with collisions	3
07	4	**Mark is for AO1 (understanding)** Allows direct (**A.** faster) access to the value being looked-up // No need to search through the list to find a value (assuming a good choice of hash function);	1

Question			Marks
08	1	**Mark is for AO2 (analyse)** `ProcessLockSolved`; **R.** if spelt incorrectly **R.** if any additional code **I.** case and spacing	1
08	2	**Mark is for AO2 (apply)** `MaxHandSize`; **A.** `HandSize` **A.** any suitable identifier that makes it clear that the constant represents the hand size	1
08	3	**Mark is for AO2 (analyse)** `game1.txt`; **I.** quotation marks around `game1.txt`	1
08	4	**Mark is for AO2 (analyse)** To make sure no difficulty cards are put into the player's hand;	1

Question			Marks
09		**Mark is for AO2 (analyse)** 2;	1

Question			Marks

10		**Mark is for AO2 (apply)**
		
		`D|P`
		//
		`[DP]`
		//
		`P|D`
		//
		`[PD];`
		
		I. use of quotes around each character
		A. use of `^` and/or `$` in expression as long as done correctly
			1

Question			Marks
11	1	**All marks for AO3 (programming)** 1. Iterative structure contains code that gets the choice from the player; 2. One correct condition; 3. Both correct conditions and correct logic; 4. Displays error message under all correct circumstances and only under correct circumstances; **Max 3** if code contains errors	4
11	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **11.1**.* *Code for **11.1** must be sensible.* Screen capture showing same message as code for 11.1 displayed when `L` is entered followed by `D` being entered and accepted; (D)iscard inspect, (U)se card:> U Enter a number between 1 and 5 to specify card to use:> 2 (D)iscard or (P)lay?:> L Not a valid choice. (D)iscard or (P)lay?:> D Current score: 0 CURRENT LOCK ------------ Not met: F c, P c, F c SEQUENCE: empty HAND: ----------------------------- | P b | P a | K b | F b | P c | ----------------------------- (D)iscard inspect, (U)se card:> **Notes for examiners:** ignore contents of the hand and the current score. (B)lasting cap might be shown in list of choices or might not be. Output after entering `D` will be different if a difficulty card was drawn.	1

Question			Marks
12	1	**All marks for AO3 (programming)** 1. Creating a new subroutine called `GetNumberOfToolCards`; **R.** other identifiers **I.** case **I.** minor spelling mistakes 2. New subroutine has mechanism to return an integer and returns an integer value; **I.** incorrect value **A.** other numeric data types 3. Iterative structure that repeats a number of times based on the size of `Cards`; 4. Gets type of card inside iterative structure; 5. Selection structure inside iterative structure that compares (their attempt at) type of card with at least one of P, F or K; 6. Selection structure with correct conditions and value to return incremented by one inside selection structure; The following mark points relate to the `PlayGame` subroutine: 7. Valid calls to `GetNumberOfToolCards` or `GetNumberOfCards` and value returned by this call is displayed; **A**. alternative identifier for subroutine if matches identifier used for mark point 1 8. Appropriate messages displayed along with values returned by calls to `GetNumberOfToolCards` and `GetNumberOfCards`; **R.** if before display of current score **R.** if after display of player's hand **Alternative answer for mark points 5 and 6** 5. Selection structure inside iterative structure that compares (their attempt at) type of card with `Dif`; 6. Selection structure with correct condition and value incremented by one inside selection structure, this value is subtracted from the total number of cards before being returned to the calling routine; **Alternative answer for mark points 4 and 5** 4. Gets score for card inside iterative structure; 5. Selection structure inside iterative structure that compares (their attempt at) score for card with zero; **Max 7** if code contains errors	**8**

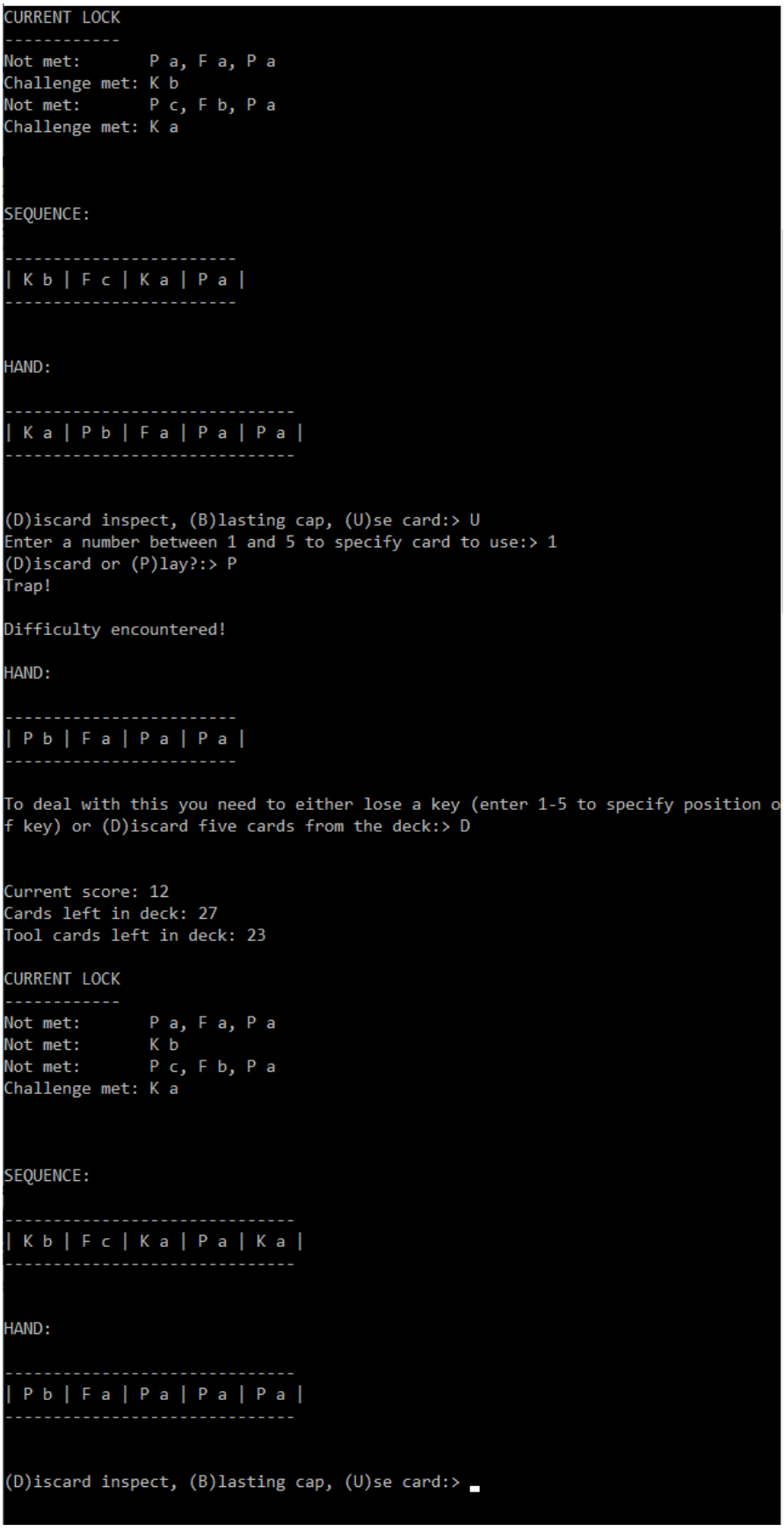
Question			Marks

12	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		*Must match code from **12.1**.*
		*Code for **12.1** must be sensible.*
		
		Screen capture(s) showing the values of 33 and 28 followed by the values of 32 and 27; (**A.** alternative values for the second set of numbers if there is evidence that a difficulty card was drawn from the deck)
		
		`Enter L to load a game from a file, anything else to play a new game:>`
		
		`Current score: 0`
		`Cards left in deck: 33`
		`Tool cards left in deck: 28`
		
		`CURRENT LOCK`
		`------------`
		`Not met:  K a`
		`Not met:  K b`
		`Not met:  K c`
		
		`SEQUENCE: empty`
		
		`HAND:`
		
		`-----------------------------`
		`| P c | F c | P c | P c | P b |`
		`-----------------------------`
		
		`(D)iscard inspect, (U)se card:> U`
		`Enter a number between 1 and 5 to specify card to use:> 2`
		`(D)iscard or (P)lay?:> D`
		
		`Current score: 0`
		`Cards left in deck: 32`
		`Tool cards left in deck: 27`
		
		`CURRENT LOCK`
		`------------`
		`Not met:  K a`
		`Not met:  K b`
		`Not met:  K c`
		
		`SEQUENCE: empty`
		
		`HAND:`
		
		`-----------------------------`
		`| P c | P c | P c | P b | P a |`
		`-----------------------------`
		
		`(D)iscard inspect, (U)se card:>`
			1

Question			Marks
13	1	**All marks for AO3 (programming)** 1. Create a variable, with an appropriate name and data type, to use to keep track if there is a blasting cap available and give variable a value of `True` (or equivalent) // create a variable, with an appropriate name and data type, to use to keep track if blasting cap has been used and give variable a value of `False` (or equivalent); **R.** if inside iterative structure that repeats until game is over 2. Selection structure, in appropriate place that checks for the player's choice being `B` (or suitable alternative) and appropriate modified message in `GetChoice` subroutine; 3. Selection structure that checks if the player has a blasting cap (or does not have a blasting cap); 4. If there is a blasting cap (**A.** incorrect condition) gets the player's choice of challenge; **R.** if value is not of integer data type, unless it is converted to be an integer before it is used 5. If they chose to use a blasting cap (**A.** incorrect condition) sets the value of variable used to indicate if there is a blasting cap to `False` (or equivalent); 6. Selection structure that checks player's choice of challenge is less than or equal to the number of challenges; 7. Selection structure that checks player's choice of challenge has not already been met; **R.** if checks the wrong challenge 8. If conditions for both mark points 6 and 7 are met displays message saying blasting cap has been used; **I.** incorrect logic for selection structure(s) 9. Changes the met status of the challenge specified by the player inside selection structure(s) for mark points 6 and 7; **I.** incorrect logic for selection structure(s) **R.** if changes the wrong challenge **Max 8 marks** if code contains errors	9
13	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **13.1**.* *Code for **13.1** must be sensible.* Screen capture(s) showing that the third condition is met after use of blasting cap and that use of a second blasting cap is not permitted; **Notes for examiners:** ignore messages about number of cards in deck and number of tool cards in deck.	1

```
(D)iscard inspect, (B)lasting cap, (U)se card:> B
Challenge to blast? 3
Blasting cap used.

CURRENT LOCK
------------
Not met:       P a, F a, P a
Challenge met: K b
Challenge met: P c, F b, P a
Challenge met: K a



Current score: 8
Cards left in deck: 30
Tool cards left in deck: 25

CURRENT LOCK
------------
Not met:       P a, F a, P a
Challenge met: K b
Challenge met: P c, F b, P a
Challenge met: K a



SEQUENCE:

------------------
| K b | F c | K a |
------------------


HAND:

------------------------------
| P a | K a | P b | F a | P a |
------------------------------


(D)iscard inspect, (B)lasting cap, (U)se card:> B

Current score: 8
Cards left in deck: 30
Tool cards left in deck: 25

CURRENT LOCK
------------
Not met:       P a, F a, P a
Challenge met: K b
Challenge met: P c, F b, P a
Challenge met: K a



SEQUENCE:

------------------
| K b | F c | K a |
------------------


HAND:

------------------------------
| P a | K a | P b | F a | P a |
------------------------------


(D)iscard inspect, (B)lasting cap, (U)se card:>
```

Question			Marks
14	1	**All marks for AO3 (programming)** Mark points 1 to 9 relate to the `TrapCard` class. 1. Creating a new class called `TrapCard` that inherits from `DifficultyCard`; **R.** other names for class **I.** case and minor typos 2. Constructor calls parent class constructor and then sets `Type` to `Trp` // Constructor sets `Type` to `Trp` and sets `CardNumber` to value of parameter; 3. Subroutine called `Process` created that overrides parent class method and contains a call to parent class method (or equivalent); 4. Iteration structure that repeats based on the number of challenges on the current lock; 5. Selection structure, inside iteration structure, that compares the value of a challenge's status with either `True` or `False`; 6. Adds challenge / index of challenge to a list if challenge has been met // after ascertaining that at least one challenge has been met repeatedly selects a random challenge; 7. Selects a random challenge that has been met; **R.** if could select a challenge that has not been met under some circumstances 8. Changes the status of the selected challenge to not met (`False`); **R.** if multiple challenges changed 9. If no challenges have been met then a call is made to the parent class method (or equivalent); **R.** if could also set a met challenge's status to `False` 10. Modified `GetCardFromDeck` so that trap cards are processed in the same way as difficulty cards; **R.** other messages **I.** case and minor typos 11. Modified `GetCardFromDeck` so it displays the message `Trap!` if a trap card is drawn; **R.** other messages **R.** if message displayed when non-trap card is drawn **I.** case and minor typos 12. Modified `SetupCardCollectionFromGameFile` so that it creates trap cards instead of difficulty cards; **Max 11** if code contains errors or if other parts of the subroutines `GetCardFromDeck` or `SetupCardCollectionFromGameFile` no longer work correctly	**12**
14	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **14.1**, including prompts on screen capture matching those in code.* *Code for **14.1** must be sensible.* Screen capture(s) showing that for the game from `game1.txt` one of the two challenges that was met is now shown as not met;	**1**



VB.Net

Question			Marks
05	1		12

```
Console.Write("Enter a string: ")
Dim UserString As String = Console.ReadLine()
Dim EndPos As Integer = UserString.Length - 1
Dim StartPos As Integer = 0
While EndPos > StartPos
  If "aeiou".Contains(UserString(EndPos)) Then
    If "aeiou".Contains(UserString(StartPos)) Then
      Dim NewString As String = UserString.Substring(0, StartPos) +
UserString(EndPos) + UserString.Substring(StartPos + 1, EndPos -
StartPos - 1) + UserString(StartPos) + UserString.Substring(EndPos +
1, UserString.Length - 1 - EndPos)
      UserString = NewString
      StartPos += 1
      EndPos -= 1
    Else
      StartPos += 1
    End If
  Else
    EndPos -= 1
    If Not "aeiou".Contains(UserString(StartPos)) Then
      StartPos += 1
    End If
  End If
End While
Console.WriteLine(UserString)
Console.ReadLine()
```

Alternative answer

```
Console.Write("Enter a string: ")
Dim UserString As String = Console.ReadLine()
Dim EndPos As Integer = UserString.Length - 1
Dim StartPos As Integer = 0
Dim StartNewWord As String = ""
Dim EndNewWord As String = ""
While EndPos > StartPos
  If "aeiou".Contains(UserString(EndPos)) Then
    If "aeiou".Contains(UserString(StartPos)) Then
      EndNewWord = UserString(StartPos) + EndNewWord
      StartNewWord += UserString(EndPos)
      StartPos += 1
      EndPos -= 1
    Else
      StartNewWord += UserString(StartPos)
      StartPos += 1
    End If
  Else
    EndNewWord = UserString(EndPos) + EndNewWord
    EndPos -= 1
    If Not "aeiou".Contains(UserString(StartPos)) Then
      StartNewWord += UserString(StartPos)
      StartPos += 1
    End If
  End If
End While
If StartPos = EndPos Then
```

		Console.WriteLine(StartNewWord + UserString(StartPos) + EndNewWord)
		Else
		  Console.WriteLine(StartNewWord + EndNewWord)
		End If
		Console.ReadLine()	

11	**1**	Function GetDiscardOrPlayChoice() As String
		  Dim Choice As String
		  Do
		  Console.Write("(D)iscard or (P)lay?:> ")
		  Choice = Console.ReadLine().ToUpper()
		  If Choice <> "D" And Choice <> "P" Then
		  Console.WriteLine("Not a valid choice.")
		  End If
		  Loop While Choice <> "D" And Choice <> "P"
		  Return Choice
		End Function
			4

12	**1**	Public Sub PlayGame()
		  ...
		  While Not LockSolved And Not GameOver
		  Console.WriteLine()
		  Console.WriteLine("Current score: " & Score)
		  Console.WriteLine("Cards left in deck: " & Deck.GetNumberOfCards())
		  Console.WriteLine("Tool cards left in deck: " & Deck.GetNumberOfToolCards())
		  Console.WriteLine(CurrentLock.GetLockDetails())
		  ...
		
		Public Function GetNumberOfToolCards() As Integer
		  Dim Count As Integer = 0
		  For Each C In Cards
		  If {"K", "F", "P"}.Contains(C.GetDescription()(0)) Then
		  Count += 1
		  End If
		  Next
		  Return Count
		End Function
			8

13	**1**	Function GetChoice() As String
		  Console.WriteLine()
		  Console.Write("(D)iscard inspect, **(B)lasting cap**, (U)se card:> ")
		  Dim Choice As String = Console.ReadLine().ToUpper()
		  Return Choice
		End Function
		
		Public Sub PlayGame()
		  ...
		  If Locks.Count > 0 Then
		  GameOver = False
		  CurrentLock = New Lock()
		  Dim BlastingCap As Boolean = True
		  SetupGame()
		  ...
			9

```
    MenuChoice = GetChoice()
    Select Case MenuChoice
      Case "B"
        If BlastingCap Then
          Console.Write("Challenge to blast? ")
          Dim BlastChoice As Integer = Console.ReadLine() - 1
          BlastingCap = False
          If BlastChoice < CurrentLock.GetNumberOfChallenges() Then
            If Not CurrentLock.GetChallengeMet(BlastChoice) Then
              CurrentLock.SetChallengeMet(BlastChoice, True)
              Console.WriteLine("Blasting cap used.")
              Console.WriteLine(CurrentLock.GetLockDetails())
            End If
          End If
        End If
      Case "D"
        Console.WriteLine(Discard.GetCardDisplay())
  ...
```

14 **1**

```
Private Sub GetCardFromDeck(ByVal CardChoice As Integer)
  If Deck.GetNumberOfCards() > 0 Then
    If Deck.GetCardDescriptionAt(0) = "Dif" Or
Deck.GetCardDescriptionAt(0) = "Trp" Then
      Dim CurrentCard As Card =
Deck.RemoveCard(Deck.GetCardNumberAt(0))
      Console.WriteLine()
      If CurrentCard.GetDescription() = "Trp" Then
        Console.WriteLine("Trap!")
      End If
      Console.WriteLine("Difficulty encountered!”)
      Console.WriteLine(Hand.GetCardDisplay())
      Console.Write("To deal with this you need to either lose a
key")
      Console.Write("(enter 1-5 to specify position of key) or
(D)iscard five cards from the deck:> ")
      Dim Choice As String = Console.ReadLine()
      Discard.AddCard(CurrentCard)
      CurrentCard.Process(Deck, Discard, Hand, Sequence,
CurrentLock, Choice, CardChoice)
    End If
  End If
  While Hand.GetNumberOfCards() < 5 And Deck.GetNumberOfCards > 0
    If Deck.GetCardDescriptionAt(0) = "Dif" Or
Deck.GetCardDescriptionAt(0) = "Trp" Then
   ...


Private Sub SetupCardCollectionFromGameFile(ByVal LineFromFile As
String, ByVal CardCol As CardCollection)
  ...
  For Each Item In SplitLine
    If Item.Length = 5 Then
      CardNumber = ToInt32(Item(4))
    Else
      CardNumber = ToInt32(Item.Substring(4, 2))
    End If
    If Item.Substring(0, 3) = "Dif" Then
      Dim CurrentCard As New TrapCard(CardNumber)
      CardCol.AddCard(CurrentCard)
    Else
```

12

```
      Dim CurrentCard As New ToolCard(Item(0), Item(2), CardNumber)
      CardCol.AddCard(CurrentCard)
    End If
  Next
  ...

Class TrapCard
  Inherits DifficultyCard

  Sub New(ByVal CardNo As Integer)
    CardType = "Trp"
    CardNumber = CardNo
  End Sub

  Public Overrides Sub Process(ByVal Deck As CardCollection, ByVal
Discard As CardCollection, ByVal Hand As CardCollection, ByVal
Sequence As CardCollection, ByVal CurrentLock As Lock, ByVal Choice
As String, ByVal CardChoice As Integer)
    Dim MetChallenges As New List(Of Integer)
    For Count = 0 To CurrentLock.GetNumberOfChallenges() - 1
      If CurrentLock.GetChallengeMet(Count) Then
        MetChallenges.Add(Count)
      End If
    Next
    If MetChallenges.Count = 0 Then
      MyBase.Process(Deck, Discard, Hand, Sequence, CurrentLock,
Choice, CardChoice)
    Else
      Dim RNo As Integer = RNoGen.Next(0, MetChallenges.Count)
      CurrentLock.SetChallengeMet(MetChallenges(RNo), False)
    End If
  End Sub
End Class
```

Python 3

Question			Marks

05	1	`UserString = input("Enter a string: ")`
		`EndPos = len(UserString) - 1`
		`StartPos = 0`
		`while EndPos > StartPos:`
		`  if UserString[EndPos] in "aeiou":`
		`  if UserString[StartPos] in "aeiou":`
		`  UserString = UserString[0: StartPos] + UserString[EndPos] + UserString[StartPos+1: EndPos] + UserString[StartPos] + UserString[EndPos+1:len(UserString)]`
		`  StartPos += 1`
		`  EndPos -= 1`
		`  else:`
		`  StartPos += 1`
		`  else:`
		`  EndPos -= 1`
		`  if UserString[StartPos] not in "aeiou":`
		`  StartPos += 1`
		`print(UserString)`
		
		Alternative answer
		
		`UserString = input("Enter a string: ")`
		`EndPos = len(UserString) - 1`
		`StartPos = 0`
		`StartNewString = ""`
		`EndNewString = ""`
		`while EndPos > StartPos:`
		`  if UserString[EndPos] in "aeiou":`
		`  if UserString[StartPos] in "aeiou":`
		`  EndNewString = UserString[StartPos] + EndNewString`
		`  StartNewString += UserString[EndPos]`
		`  StartPos += 1`
		`  EndPos -= 1`
		`  else:`
		`  StartNewString += UserString[StartPos]`
		`  StartPos += 1`
		`  else:`
		`  EndNewString = UserString[EndPos] + EndNewString`
		`  EndPos -= 1`
		`  if UserString[StartPos] not in "aeiou":`
		`  StartNewString += UserString[StartPos]`
		`  StartPos += 1`
		`if StartPos == EndPos:`
		`  print(StartNewString + UserString[StartPos] + EndNewString)`
		`else:`
		`  print(StartNewString + EndNewString)`
			12

11	1	`def __GetDiscardOrPlayChoice(self):`
		`  Choice = input("(D)iscard or (P)lay?:> ").upper()`
		`  **while Choice != "D" and Choice != "P":**`
		`  **print("Not a valid choice")**`
		`  **Choice = input("(D)iscard or (P)lay?:> ").upper()**`
		`  return Choice`
			4

12 **1**

```
def PlayGame(self):
  if len(self.__Locks) > 0:
    self.__SetupGame()
    while not self.__GameOver:
        self.__LockSolved = False
        while not self.__LockSolved and not self.__GameOver:
          print()
          print("Current score:", self.__Score)
          print("Cards left in deck:",
self.__Deck.GetNumberOfCards())
          print("Tool cards left in deck:",
self.__Deck.GetNumberOfToolCards())
          print(self.__CurrentLock.GetLockDetails())
          ...

def GetNumberOfToolCards(self):
  Count = 0
  for C in self._Cards:
    if C.GetDescription()[0] in ["K", "F", "P"]:
      Count += 1
  return Count
```

8

13 **1**

```
def __GetChoice(self):
  print()
  Choice = input("(D)iscard inspect, (B)lasting cap, (U)se card:>
").upper()
  return Choice

def PlayGame(self):
  if len(self.__Locks) > 0:
    self.__SetupGame()
    BlastingCap = True
    while not self.__GameOver:
      self.__LockSolved = False
      while not self.__LockSolved and not self.__GameOver:
        ...
        MenuChoice = self.__GetChoice()
        if MenuChoice == "D":
          print(self.__Discard.GetCardDisplay())
        elif MenuChoice == "B":
          if BlastingCap:
            BlastChoice = int(input("Challenge to blast? ")) - 1
            BlastingCap = False
            if BlastChoice <
self.__CurrentLock.GetNumberOfChallenges():
              if not
self.__CurrentLock.GetChallengeMet(BlastChoice):
                self.__CurrentLock.SetChallengeMet(BlastChoice,
True)
                print("Blasting cap used")
                print(self.__CurrentLock.GetLockDetails())
        elif MenuChoice == "U":
          ...
```

9

14	**1**	`def __GetCardFromDeck(self, CardChoice):`
		`  if self.__Deck.GetNumberOfCards() > 0:`
		`  if self.__Deck.GetCardDescriptionAt(0) in ["Dif", "Trp"]:`
		`  CurrentCard =`
		`self.__Deck.RemoveCard(self.__Deck.GetCardNumberAt(0))`
		`  print()`
		`  if CurrentCard.GetDescription() == "Trp":`
		`  print("Trap!")`
		`  print("Difficulty encountered!")`
		`  print(self.__Hand.GetCardDisplay())`
		`  ...`
		`  while self.__Hand.GetNumberOfCards() < 5 and`
		`self.__Deck.GetNumberOfCards() > 0:`
		`  if self.__Deck.GetCardDescriptionAt(0) in ["Dif", "Trp"]:`
		`  self.__MoveCard(self.__Deck, self.__Discard,`
		`self.__Deck.GetCardNumberAt(0))`
		`  print("A difficulty card was discarded from the deck when`
		`refilling the hand.")`
		`  ...`
		
		`def __SetupCardCollectionFromGameFile(self, LineFromFile, CardCol):`
		`  ...`
		`  if Item[0: 3] == "Dif":`
		`  CurrentCard = TrapCard(CardNumber)`
		`  CardCol.AddCard(CurrentCard)`
		`  else:`
		`  CurrentCard = ToolCard(Item[0], Item[2], CardNumber)`
		`  CardCol.AddCard(CurrentCard)`
		
		`class TrapCard(DifficultyCard):`
		`  def __init__(self, CardNo):`
		`  self._CardType = "Trp"`
		`  self._CardNumber = CardNo`
		
		`  def Process(self, Deck, Discard, Hand, Sequence, CurrentLock,`
		`Choice, CardChoice):`
		`  MetChallenges = []`
		`  for Count in range (CurrentLock.GetNumberOfChallenges()):`
		`  if CurrentLock.GetChallengeMet(Count):`
		`  MetChallenges.append(Count)`
		`  if len(MetChallenges) == 0:`
		`  super(TrapCard, self).Process(Deck, Discard, Hand, Sequence,`
		`CurrentLock, Choice, CardChoice)`
		`  else:`
		`  RNo = random.randint(0, len(MetChallenges) - 1)`
		`  CurrentLock.SetChallengeMet(MetChallenges[RNo], False)`
			12

Python 2

Question			Marks

05	1	`UserString = raw_input("Enter a string: ")`
		`EndPos = len(UserString) - 1`
		`StartPos = 0`
		`while EndPos > StartPos:`
		`  if UserString[EndPos] in "aeiou":`
		`  if UserString[StartPos] in "aeiou":`
		`  UserString = UserString[0: StartPos] + UserString[EndPos] +`
		`UserString[StartPos+1: EndPos] + UserString[StartPos] +`
		`UserString[EndPos+1:len(UserString)]`
		`  StartPos += 1`
		`  EndPos -= 1`
		`  else:`
		`  StartPos += 1`
		`  else:`
		`  EndPos -= 1`
		`  if UserString[StartPos] not in "aeiou":`
		`  StartPos += 1`
		`print UserString`
		
		Alternative answer
		
		`UserString = raw_input("Enter a string: ")`
		`EndPos = len(UserString) - 1`
		`StartPos = 0`
		`StartNewString = ""`
		`EndNewString = ""`
		`while EndPos > StartPos:`
		`  if UserString[EndPos] in "aeiou":`
		`  if UserString[StartPos] in "aeiou":`
		`  EndNewString = UserString[StartPos] + EndNewString`
		`  StartNewString += UserString[EndPos]`
		`  StartPos += 1`
		`  EndPos -= 1`
		`  else:`
		`  StartNewString += UserString[StartPos]`
		`  StartPos += 1`
		`  else:`
		`  EndNewString = UserString[EndPos] + EndNewString`
		`  EndPos -= 1`
		`  if UserString[StartPos] not in "aeiou":`
		`  StartNewString += UserString[StartPos]`
		`  StartPos += 1`
		`if StartPos == EndPos:`
		`  print StartNewString + UserString[StartPos] + EndNewString`
		`else:`
		`  print StartNewString + EndNewString`
			12

11	1	`def __GetDiscardOrPlayChoice(self):`
		`  Choice = raw_input("(D)iscard or (P)lay?:> ").upper()`
		`  while Choice != "D" and Choice != "P":`
		`  print "Not a valid choice"`
		`  Choice = raw_input("(D)iscard or (P)lay?:> ").upper()`
		`  return Choice`
			4

<table>
<tr>
<td>12</td>
<td>1</td>
<td>

```
def PlayGame(self):
  if len(self.__Locks) > 0:
    self.__SetupGame()
    while not self.__GameOver:
        self.__LockSolved = False
        while not self.__LockSolved and not self.__GameOver:
          print
          print "Current score: " + str(self.__Score)
          print "Cards left in deck:" +
str(self.__Deck.GetNumberOfCards())
          print "Tool cards left in deck:" +
str(self.__Deck.GetNumberOfToolCards())
          print self.__CurrentLock.GetLockDetails()
          ...

def GetNumberOfToolCards(self):
  Count = 0
  for C in self._Cards:
    if C.GetDescription()[0] in ["K", "F", "P"]:
      Count += 1
  return Count
```

</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>

```
def __GetChoice(self):
  print
  Choice = raw_input("(D)iscard inspect, (B)lasting cap, (U)se
card:> ").upper()
  return Choice

def PlayGame(self):
  if len(self.__Locks) > 0:
    self.__SetupGame()
    BlastingCap = True
    while not self.__GameOver:
      self.__LockSolved = False
      while not self.__LockSolved and not self.__GameOver:
        ...
        MenuChoice = self.__GetChoice()
        if MenuChoice == "D":
          print self.__Discard.GetCardDisplay()
        elif MenuChoice == "B":
          if BlastingCap:
            BlastChoice = int(raw_input("Challenge to blast? ")) - 1
            BlastingCap = False
            if BlastChoice <
self.__CurrentLock.GetNumberOfChallenges():
              if not
self.__CurrentLock.GetChallengeMet(BlastChoice):
                self.__CurrentLock.SetChallengeMet(BlastChoice,
True)
                print "Blasting cap used"
                print self.__CurrentLock.GetLockDetails()
        elif MenuChoice == "U":
          ...
```

</td>
<td>9</td>
</tr>
</table>

14	**1**	`def __GetCardFromDeck(self, CardChoice):`
		`  if self.__Deck.GetNumberOfCards() > 0:`
		`  if self.__Deck.GetCardDescriptionAt(0) in ["Dif", "Trp"]:`
		`  CurrentCard =`
		`self.__Deck.RemoveCard(self.__Deck.GetCardNumberAt(0))`
		`  print()`
		`  if CurrentCard.GetDescription() == "Trp":`
		`  print "Trap!"`
		`  print "Difficulty encountered!"`
		`  print self.__Hand.GetCardDisplay()`
		`  ...`
		`  while self.__Hand.GetNumberOfCards() < 5 and`
		`self.__Deck.GetNumberOfCards() > 0:`
		`  if self.__Deck.GetCardDescriptionAt(0) in ["Dif", "Trp"]:`
		`  self.__MoveCard(self.__Deck, self.__Discard,`
		`self.__Deck.GetCardNumberAt(0))`
		`  print "A difficulty card was discarded from the deck when`
		`refilling the hand."`
		`  ...`
		
		`def __SetupCardCollectionFromGameFile(self, LineFromFile, CardCol):`
		`  ...`
		`  if Item[0: 3] == "Dif":`
		`  CurrentCard = TrapCard(CardNumber)`
		`  CardCol.AddCard(CurrentCard)`
		`  else:`
		`  CurrentCard = ToolCard(Item[0], Item[2], CardNumber)`
		`  CardCol.AddCard(CurrentCard)`
		
		`class TrapCard(DifficultyCard):`
		`  def __init__(self, CardNo):`
		`  self._CardType = "Trp"`
		`  self._CardNumber = CardNo`
		
		`  def Process(self, Deck, Discard, Hand, Sequence, CurrentLock,`
		`Choice, CardChoice):`
		`  MetChallenges = []`
		`  for Count in range (CurrentLock.GetNumberOfChallenges()):`
		`  if CurrentLock.GetChallengeMet(Count):`
		`  MetChallenges.append(Count)`
		`  if len(MetChallenges) == 0:`
		`  super(TrapCard, self).Process(Deck, Discard, Hand, Sequence,`
		`CurrentLock, Choice, CardChoice)`
		`  else:`
		`  RNo = random.randint(0, len(MetChallenges) - 1)`
		`  CurrentLock.SetChallengeMet(MetChallenges[RNo], False)`
			12

C#

Question			Marks
05	1		12

```
Console.Write("Enter a string: ");
string userString = Console.ReadLine();
int endPos = (userString.Length - 1);
int startPos = 0;
while (endPos > startPos)
{
    if ("aeiou".Contains(userString[endPos]))
    {
        if ("aeiou".Contains(userString[startPos]))
        {
            string newString = userString.Substring(0, startPos)
                        + userString[endPos]
                        + userString.Substring(startPos + 1, endPos
- startPos - 1)
                        + userString[startPos]
                        + userString.Substring(endPos + 1,
userString.Length - 1 - endPos);
            userString = newString;
            startPos++;
            endPos--;
        }
        else
        {
            startPos++;
        }
    }
    else
    {
        endPos--;
        if (!"aeiou".Contains(userString[startPos]))
        {
            startPos++;
        }
    }
}
Console.WriteLine(userString);
```

Alternative answer

```
Console.Write("Enter a string: ");
string userString = Console.ReadLine();
int endPos = (userString.Length - 1);
int startPos = 0;
string startNewWord = "";
string endNewWord = "";
while ((endPos > startPos))
{
    if ("aeiou".Contains(userString[endPos]))
    {
```

```
        if ("aeiou".Contains(userString[startPos]))
        {
            endNewWord = (userString[startPos] + endNewWord);
            startNewWord = (startNewWord + userString[endPos]);
            startPos++;
            endPos--;
        }
        else
        {
            startNewWord = (startNewWord + userString[startPos]);
            startPos++;
        }
    }
    else
    {
        endNewWord = userString[endPos] + endNewWord;
        endPos--;
        if (!"aeiou".Contains(userString[startPos]))
        {
            startNewWord = (startNewWord + userString[startPos]);
            startPos++;
        }
    }
}
if ((startPos == endPos))
{
    Console.WriteLine((startNewWord + (userString[startPos] +
endNewWord)));
}
else
{
    Console.WriteLine((startNewWord + endNewWord));
}
```

Alternative answer

```
Console.Write("Enter a string: ");
string userString = Console.ReadLine();
int endPos = userString.Length - 1;
int startPos = 0;
char [] charArray = userString.ToCharArray(0,userString.Length);
while (endPos > startPos)
{
    if (IsaVowel(charArray[startPos]))
    {
        while (!IsaVowel(charArray[endPos]) && endPos > startPos)
        {
            endPos--;
        }
        if (endPos > startPos)
        {
            char temp = charArray[startPos];
```

		charArray[startPos] = charArray[endPos];
		charArray[endPos] = temp;
		endPos--;
		}
		}
		startPos++;
		}
		userString = new string (charArray);
		Console.WriteLine(userString);
		
		private static bool IsaVowel(char v)
		{
		if (v == 'a' || v == 'e' || v == 'i' || v == 'o' || v == 'u')
		{
		return true;
		}
		return false;
		}	

11	**1**	private string GetDiscardOrPlayChoice()
		{
		string Choice;
		do
		{
		Console.Write("(D)iscard or (P)lay?:> ");
		Choice = Console.ReadLine().ToUpper();
		if (Choice != "D" && Choice != "P")
		{
		Console.WriteLine("Not a valid choice.");
		}
		} while (Choice != "D" && Choice != "P");
		
		return Choice;
		}
			4

12	**1**	public void PlayGame()
		...
		while (!GameOver)
		{
		LockSolved = false;
		while (!LockSolved && !GameOver)
		{
		Console.WriteLine();
		Console.WriteLine("Current score: " + Score);
		Console.WriteLine("Cards left in deck: " + Deck.GetNumberOfCards());
		Console.WriteLine("Tool cards left in deck: " + Deck.GetNumberOfToolCards());
		Console.WriteLine(CurrentLock.GetLockDetails());
		Console.WriteLine(Sequence.GetCardDisplay());
		...
			8

```
public int GetNumberOfToolCards()
{
    int Count = 0;
    foreach (var C in cards)
    {
        if ("KFP".Contains(C.GetDescription()[0]))
        {
            Count++;
        }
    }
    return Count;
}
```

13 **1**

```
private string GetChoice()
{
    Console.WriteLine();
    Console.Write("(D)iscard inspect, (B)lasting cap, (U)se card:>
");
    string Choice = Console.ReadLine().ToUpper();
    return Choice;
}

public void PlayGame()
{
    string MenuChoice;
    bool BlastingCap = true;
    if (Locks.Count > 0)
    {
        GameOver = false;
        CurrentLock = new Lock();
        SetupGame();
        ...
                MenuChoice = GetChoice();
                switch (MenuChoice)
                {
 case "B":
            {
                if (BlastingCap)
                {
                    Console.Write("Challenge to blast? ");
                    int BlastChoice =
Convert.ToInt32(Console.ReadLine()) - 1;
                    BlastingCap = false;
                    if (BlastChoice <
CurrentLock.GetNumberOfChallenges())
                    {
                        if (!
CurrentLock.GetChallengeMet(BlastChoice))
                        {
                            CurrentLock.SetChallengeMet(BlastChoice,
true);
                            Console.WriteLine("Blasting cap used.");
```

9

```
Console.WriteLine(CurrentLock.GetLockDetails());
                        }
                    }
                }
                break;
            }
            case "D":
                {
```

14 **1**

```
private void GetCardFromDeck(int cardChoice)
{
    if (Deck.GetNumberOfCards() > 0)
    {
        if (Deck.GetCardDescriptionAt(0) == "Dif" ||
Deck.GetCardDescriptionAt(0) == "Trp")
        {
            Card CurrentCard =
Deck.RemoveCard(Deck.GetCardNumberAt(0));
            Console.WriteLine();
            if (CurrentCard.GetDescription() == "Trp")
            {
                Console.WriteLine("Trap!");
            }
            Console.WriteLine("Difficulty encountered!");
            Console.WriteLine(Hand.GetCardDisplay());
            Console.Write("To deal with this you need to either lose
a key ");
            Console.Write("(enter 1-5 to specify position of key) or
(D)iscard five cards from the deck:> ");
            string Choice = Console.ReadLine();
            Console.WriteLine();
            Discard.AddCard(currentCard);
            CurrentCard.Process(Deck, Discard, Hand, Sequence,
CurrentLock, Choice, cardChoice);
        }
    }
    while (Hand.GetNumberOfCards() < 5 && Deck.GetNumberOfCards() >
0)
    {
        if (Deck.GetCardDescriptionAt(0) == "Dif" ||
Deck.GetCardDescriptionAt(0) == "Trp")
        {
...

private void SetupCardCollectionFromGameFile(string lineFromFile,
CardCollection cardCol)
{
    List<string> SplitLine;
    int CardNumber;
    if (lineFromFile.Length > 0)
    {
        SplitLine = lineFromFile.Split(',').ToList();
```

12

```
        foreach (var Item in SplitLine)
        {
            if (Item.Length == 5)
                CardNumber = Convert.ToInt32(Item[4]);
            else
                CardNumber = Convert.ToInt32(Item.Substring(4, 2));
            if (Item.Substring(0, 3) == "Dif")
            {
                DifficultyCard CurrentCard = new
TrapCard(CardNumber);
                cardCol.AddCard(CurrentCard);
            }
            else
            {
                ToolCard CurrentCard = new
ToolCard(Item[0].ToString(), Item[2].ToString(), CardNumber);
                cardCol.AddCard(CurrentCard);
            }
        }
...

class TrapCard : DifficultyCard
{
    static Random rNoGen = new Random();
    public TrapCard(int cardNo)
    {
        CardType = "Trp";
        CardNumber = cardNo;
    }
    public override void Process(CardCollection deck, CardCollection
discard, CardCollection hand, CardCollection sequence, Lock
currentLock, string choice, int cardChoice)
    {
        List<int> MetChallenges = new List<int>();
        for (int Count = 0; Count <=
currentLock.GetNumberOfChallenges() - 1; Count++)
        {
            if (currentLock.GetChallengeMet(Count))
                MetChallenges.Add(Count);
        }
        if (MetChallenges.Count == 0)
            base.Process(deck, discard, hand, sequence, currentLock,
choice, cardChoice);
        else
        {
            int rNo = rNoGen.Next(0, metChallenges.Count);
            currentLock.SetChallengeMet(MetChallenges[rNo], false);
        }
    }
}
```

Pascal/Delphi

Question			Marks
05	1		12

```
var
  UserString, Vowels, NewString : string;
  EndPos, StartPos : integer;
begin
  Vowels := 'aeiou';
  write('Enter a string: ');
  readln(UserString);
  EndPos := length(UserString);
  StartPos := 1;
  while EndPos > StartPos do
  begin
    if pos(UserString[EndPos], Vowels) > 0 then
    begin
      if pos(UserString[StartPos], Vowels) > 0 then
      begin
        NewString := copy(UserString, 1, StartPos - 1) +
UserString[EndPos] + copy(UserString, StartPos + 1, EndPos -
StartPos - 1) + UserString[StartPos] + copy(UserString, EndPos + 1,
length(UserString) - EndPos);
        UserString := NewString;
        inc(StartPos);
        dec(EndPos);
      end
      else
        inc(StartPos);
    end
    else
    begin
      dec(EndPos);
      if pos(UserString[StartPos], Vowels) = 0 then
        inc(StartPos);
    end;
  end;
  writeln(UserString);
  readln;
end.
```

Alternative

```
var
  UserString, Vowels, StartNewWord, EndNewWord : string;
  EndPos, StartPos : integer;
begin
  Vowels := 'aeiou';
  write('Enter a string: ');
  readln(UserString);
  EndPos := length(UserString);
  StartPos := 1;
  StartNewWord := '';
```

```
  EndNewWord := '';
  while EndPos > StartPos do
  begin
    if pos(UserString[EndPos], Vowels) > 0 then
    begin
      if pos(UserString[StartPos], Vowels) > 0 then
      begin
        EndNewWord := UserString[StartPos] + EndNewWord;
        StartNewWord := StartNewWord + UserString[EndPos];
        inc(StartPos);
        dec(EndPos);
      end
      else
      begin
        StartNewWord := StartNewWord + UserString[StartPos];
        inc(StartPos);
      end;
    end
    else
    begin
      EndNewWord := UserString[EndPos] + EndNewWord;
      dec(EndPos);
      if pos(UserString[StartPos], Vowels) = 0 then
      begin
        StartNewWord := StartNewWord + UserString[StartPos];
        inc(StartPos);
      end;
    end;
  end;
  if StartPos = EndPos then
    writeln(StartNewWord + UserString[StartPos] + EndNewWord)
  else
    writeln(StartNewWord + EndNewWord);
  readln;
end.
```

11 **1**

```
function Breakthrough.GetDiscardOrPlayChoice() : string;
var
  Choice : string;
begin
  repeat
    write('(D)iscard or (P)lay?:> ');
    readln(Choice);
    Choice := UpperCase(Choice);
    if (Choice <> 'D') and (Choice <> 'P') then
      writeln('Not a valid choice.');
  until (Choice = 'D') or (Choice = 'P');
  GetDiscardOrPlayChoice := Choice;
end;
```

4

12 **1**

```
while (not(LockSolved)) and (not(GameOver)) do
begin
```

8

```
  writeln;
  writeln('Current score: ' + inttostr(Score));
  writeln('Cards left in deck: ' +
inttostr(Deck.GetNumberOfCards()));
  writeln('Tool cards left in deck: ' +
inttostr(Deck.GetNumberOfToolCards()));
  writeln(CurrentLock.GetLockDetails());
  writeln(Sequence.GetCardDisplay());
  writeln(Hand.GetCardDisplay());
  MenuChoice := GetChoice();
  ...

function CardCollection.GetNumberOfToolCards() : integer;
var
  Count, index : integer;
  C : Card;
  Codes : string;
begin
  Count := 0;
  Codes := 'KFP';
  for index := 0 to High(Cards) do
  begin
    C := Cards[index];
    if pos(C.GetDescription()[1], Codes) > 0 then
      inc(Count);
  end;
  GetNumberOfToolCards := Count;
end;
```

13 **1**

```
function Breakthrough.GetChoice() : string;
var
  Choice : string;
begin
  writeln;
  write('(D)iscard inspect, (B)lasting cap, (U)se card:> ');
  readln(Choice);
  Choice := UpperCase(Choice);
  GetChoice := Choice;
end;

procedure Breakthrough.PlayGame();
var
  MenuChoice, DiscardOrPlay : string;
  CardChoice, BlastChoice : integer;
  BlastingCap : boolean;
begin
  if length(Locks) > 0 then
  begin
    GameOver := False;
    CurrentLock := Lock.New();
    BlastingCap := True;
    SetupGame();
    while not(GameOver) do
```

9

```
    begin
      LockSolved := False;
      while (not(LockSolved)) and (not(GameOver)) do
      begin
        writeln;
        writeln('Current score: ' + inttostr(Score));
        writeln(CurrentLock.GetLockDetails());
        writeln(Sequence.GetCardDisplay());
        writeln(Hand.GetCardDisplay());
        MenuChoice := GetChoice();
        case MenuChoice[1] of
          'B' :
           begin
            if BlastingCap then
            begin
              write('Challenge to blast? ');
              readln(MenuChoice);
              BlastChoice := strtoint(MenuChoice) - 1;
              BlastingCap := False;
              if BlastChoice < CurrentLock.GetNumberOfChallenges()
then
              begin
                if not(CurrentLock.GetChallengeMet(BlastChoice))
then
                begin
                  CurrentLock.SetChallengeMet(BlastChoice, True);
                  writeln('Blasting cap used.');
                  writeln(CurrentLock.GetLockDetails());
                end;
              end;
            end;
          end;
          'D' : writeln(Discard.GetCardDisplay());
          ...
```

14 **1**

```
TrapCard = class(DifficultyCard)
  public
    constructor New(CardNo : integer); overload;
    procedure Process(Deck : CardCollection; Discard :
CardCollection; Hand : CardCollection; Sequence : CardCollection;
CurrentLock : Lock; Choice : string; CardChoice : integer);
override;
  end;
...
  TIntegerArray = array of integer;
...

constructor TrapCard.New(CardNo: Integer);
begin
  CardType := 'Trp';
  CardNumber := CardNo;
end;
```

12

```
procedure TrapCard.Process(Deck: CardCollection; Discard:
CardCollection; Hand: CardCollection; Sequence: CardCollection;
CurrentLock: Lock; Choice: string; CardChoice: Integer);
var
  MetChallenges : TIntegerArray;
  RNo, Count : integer;
begin
  for Count := 0 to CurrentLock.GetNumberOfChallenges() - 1 do
    if CurrentLock.GetChallengeMet(Count) then
    begin
      setLength(MetChallenges, length(MetChallenges) + 1);
      MetChallenges[High(MetChallenges)] := Count;
    end;
  if length(MetChallenges) = 0 then
    inherited
  else
  begin
    RNo := random(length(MetChallenges));
    CurrentLock.SetChallengeMet(MetChallenges[RNo], False);
  end;
end;

procedure Breakthrough.GetCardFromDeck(CardChoice : integer);
var
  CurrentCard : Card;
  Choice : string;
begin
  if Deck.GetNumberOfCards() > 0 then
  begin
    if (Deck.GetCardDescriptionAt(0) = 'Dif') or
(Deck.GetCardDescriptionAt(0) = 'Trp') then
    begin
      CurrentCard := Deck.RemoveCard(Deck.GetCardNumberAt(0));
      writeln;
      if CurrentCard.GetDescription() = 'Trp' then
        writeln('Trap!')
      ...
  while (Hand.GetNumberOfCards() < 5) and (Deck.GetNumberOfCards() >
0) do
  begin
    if (Deck.GetCardDescriptionAt(0) = 'Dif') or
(Deck.GetCardDescriptionAt(0) = 'Trp') then
...

procedure Breakthrough.SetupCardCollectionFromGameFile(LineFromFile
: string; CardCol : CardCollection);
var
  SplitLine : TStringArray;
  CardNumber, index : integer;
  Item : String;
  CurrentCard : Card;
begin
  if length(LineFromFile) > 0 then
  ...
```

```
      if copy(Item, 1, 3) = 'Dif' then
      begin
        CurrentCard := TrapCard.New(CardNumber);
        CardCol.AddCard(CurrentCard);
      end
      else
      begin
        CurrentCard := ToolCard.New(Item[1], Item[3], CardNumber);
        CardCol.AddCard(CurrentCard);
      end;
    end;
  end;
end;
```

Java

Question			Marks

05	1	`Console.writeLine("Enter a string: ");`
		`String input = Console.readLine();`
		`int startPos = 0;`
		`int endPos = input.length()-1;`
		`String startNewWord = "", endNewWord = "";`
		`while (endPos>startPos) {`
		`  if (input.charAt(endPos) == 'a' || input.charAt(endPos) == 'e' || input.charAt(endPos) == 'i' || input.charAt(endPos) == 'o' || input.charAt(endPos) == 'u') {`
		`  if (input.charAt(startPos) == 'a' || input.charAt(startPos) == 'e' || input.charAt(startPos) == 'i' || input.charAt(startPos) == 'o' || input.charAt(startPos) == 'u') {`
		`  endNewWord = input.charAt(startPos) + endNewWord;`
		`  startNewWord += input.charAt(endPos);`
		`  startPos++;`
		`  endPos--;`
		`  }`
		`  else {`
		`  startNewWord += input.charAt(startPos);`
		`  startPos++;`
		`  }`
		`  }`
		`  else {`
		`  endNewWord = input.charAt(endPos) + endNewWord;`
		`  endPos--;`
		`  if (input.charAt(startPos) == 'a' || input.charAt(startPos) == 'e' || input.charAt(startPos) == 'i' || input.charAt(startPos) == 'o' || input.charAt(startPos) == 'u') {`
		`  startNewWord += input.charAt(startPos);`
		`  startPos++;`
		`  }`
		`  }`
		`}`
		`if (startPos == endPos) {`
		`  Console.writeLine(startNewWord + input.charAt(startPos) + endNewWord);`
		`}`
		`else {`
		`  Console.writeLine(startNewWord + endNewWord);`
		`}`
			12

11	1	`private String getdiscardOrPlayChoice() {`
		`  String choice;`
		`  do {`
		`  Console.write("(D)iscard or (P)lay?:> ");`
		`  choice = Console.readLine().toUpperCase();`
			4

```
        if (!choice.equals("D") && !choice.equals("P")) {
            Console.writeLine("Error only choices D or P are
permitted.");
        }
    } while (!choice.equals("D") && !choice.equals("P"));
    return choice;
}
```

12 **1** **8**

```
public void playGame() {
    …
          while (! lockSolved && ! gameOver) {
              Console.writeLine();
              Console.writeLine("Current score: " + score);
              Console.writeLine("Cards left in deck: " +
deck.getNumberOfCards());
              Console.writeLine("Tools left in deck: " +
deck.getNumberOfToolCards());
              Console.writeLine(currentLock.getLockDetails());
              Console.writeLine(sequence.getCardDisplay());
              Console.writeLine(hand.getCardDisplay());
              menuChoice = getChoice();
    …

public int getNumberOfToolCards()
{
    int toolCardNo = 0;
    for (Card card : cards) {
        if ("PFK".contains(card.getDescription().charAt(0)+"")) {
            toolCardNo++;
        }
    }
    return toolCardNo;
}
```

13 **1** **9**

```
private String getChoice() {
    Console.writeLine();
    Console.write("(D)iscard inspect, (U)se card, (B)lasting Cap:>
");
    String choice = Console.readLine().toUpperCase();
    return choice;
}

public void playGame() {
...
    Console.writeLine(hand.getCardDisplay());
    menuChoice = getChoice();
    switch (menuChoice) {
        case "B":
            if (!blastingCapUsed) {
                blastingCapUsed = true;
                Console.write("Enter the position of the challenge:>
");
```

```
                int challengePos =
Integer.parseInt(Console.readLine()) - 1;
                if (challengePos <
currentLock.getNumberOfChallenges() &&
!currentLock.getChallengeMet(challengePos)) {
                    currentLock.setChallengeMet(challengePos, true);
                    Console.writeLine("Blasting cap used
successfully.");
                    Console.writeLine(currentLock.getLockDetails());
                }
            }
            break;
        case "D":
            Console.writeLine(discard.getCardDisplay());
            break;
     ...
```

14 **1**

```
private void getCardFromDeck(int cardChoice) {
    if (deck.getNumberOfCards() > 0) {
        if (deck.getCardDescriptionAt(0).equals("Dif") ||
deck.getCardDescriptionAt(0).equals("Trp")) {
            if (deck.getCardDescriptionAt(0).equals("Trp")) {
                Console.writeLine("Trap!");
            }
            Card currentCard =
deck.removeCard(deck.getCardNumberAt(0));
            Console.writeLine();
            Console.writeLine("Difficulty encountered!");
...
    while (hand.getNumberOfCards() < 5 && deck.getNumberOfCards() >
0) {
        if
(deck.getCardDescriptionAt(0).equals("Dif")||deck.getCardDescription
At(0).equals("Trp")) {
            moveCard(deck, discard, deck.getCardNumberAt(0));
            Console.writeLine("A difficulty card was discarded from
the deck when refilling the hand.");
        } else {
            moveCard(deck, hand, deck.getCardNumberAt(0));
        }
    }
    if (deck.getNumberOfCards() == 0 && hand.getNumberOfCards() < 5)
{
        gameOver = true;
    }
}

private void setupCardCollectionFromGameFile(String lineFromFile,
CardCollection cardCol) {
    List<String> splitLine;
    int cardNumber;
    if (lineFromFile.length() > 0) {
...
```

12

```
            if (item.substring(0, 3).equals("Dif")) {
                DifficultyCard currentCard = new
TrapCard(cardNumber);
                cardCol.addCard(currentCard);
            } else {
                ToolCard currentCard = new ToolCard(rNoGen,
item.charAt(0)+"", item.charAt(2)+"", cardNumber);
                cardCol.addCard(currentCard);
            }
        }
    }
}

class TrapCard extends DifficultyCard{

    public TrapCard(int cardNo) {
        cardType = "Trp";
        cardNumber = cardNo;
    }

    @Override
    public void process(CardCollection deck, CardCollection discard,
CardCollection hand, CardCollection sequence, Lock currentLock,
String choice, int cardChoice) {
        List<Integer> challengesMet = new ArrayList<>();
        for (int i = 0; i < currentLock.getNumberOfChallenges();
i++) {
            if (currentLock.getChallengeMet(i)) {
                challengesMet.add(i);
            }
        }
        if (challengesMet.isEmpty()) {
            super.process(deck, discard, hand, sequence,
currentLock, choice, cardChoice);
        }
        else {
            int rNo = rNoGen.nextInt(challengesMet.size());
            currentLock.setChallengeMet(rNo, false);
        }
    }
}
```