AQA

A-LEVEL
Computer Science

7517/1 - Paper 1

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

June 2018

Version/Stage: 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 2018

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.

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

Pages 6 to 21 contain the generic mark scheme.

Pages 21 to 43 contain the 'Program Source Code' specific to the programming languages for questions 5, 7, 8, 9, 10 and 11.

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 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 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.

01	1	**All marks AO2 (analyse)** Take a vegetable from the box labelled "onions and carrots"; If it is an onion then the box labelled "onions" contains carrots and the box labelled "carrots" contains onions and carrots. If it is a carrot then the box labelled "carrots" contains onions and the box labelled "onions" contains carrots and onions;	2

02	1	**Mark is for AO2 (apply)** 5 3 -	1
02	2	**All marks AO2 (apply)** 3 4 2 * + 1 – **Mark as follows:** **1 mark**: correct order for values and + and – either side of the 1 **1 mark**: * directly after 4 2 **Max 1 if any errors**	2
02	3	**All marks AO1 (understanding)** Simpler for a machine/computer to evaluate; **A.** easier **R.** to understand simpler to code algorithm; Do not need brackets (to show correct order of evaluation/calculation); **A.** RPN expressions cannot be ambiguous as **BOD** Operators appear in the order required for computation; No need for order of precedence of operators; No need to backtrack when evaluating; **Max 2**	2
02	4	**All marks AO1 (understanding)** (Starting at LHS of expression) push values/operands onto stack; **R.** if operators are also pushed onto stack Each time operator reached pop top two values off stack (and apply operator to them); Add result (of applying operator) to stack; **Max 2** if any errors **Max 2** if more than one stack used **Note for examiners:** award 0 marks if description is not about a stack / LIFO structure even if the word “stack” has been used	3

<table>
<tr><td>02</td><td>5</td><td>All marks AO1 (knowledge)

local variables;
return address;
parameters;
register values; A. example of register that would be in stack frame

Max 2</td><td>2</td></tr>
<tr><td>03</td><td>1</td><td>All marks AO2 (analyse)

<table>
<tr><th></th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><th>1</th><td>0</td><td>2</td><td>5</td><td>3</td><td>0</td><td>8</td></tr>
<tr><th>2</th><td>2</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><th>3</th><td>5</td><td>1</td><td>0</td><td>0</td><td>0</td><td>4</td></tr>
<tr><th>4</th><td>3</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><th>5</th><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>5</td></tr>
<tr><th>6</th><td>8</td><td>0</td><td>4</td><td>0</td><td>5</td><td>0</td></tr>
</table>

Alternative answer
<table>
<tr><th></th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><th>1</th><td>0</td><td>2</td><td>5</td><td>3</td><td>0</td><td>8</td></tr>
<tr><th>2</th><td></td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><th>3</th><td></td><td></td><td>0</td><td>0</td><td>0</td><td>4</td></tr>
<tr><th>4</th><td></td><td></td><td></td><td>0</td><td>1</td><td>0</td></tr>
<tr><th>5</th><td></td><td></td><td></td><td></td><td>0</td><td>5</td></tr>
<tr><th>6</th><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
</table>
</td><td>2</td></tr>
</table>

		Alternative answer |	**1**	**2**	**3**	**4**	**5**	**6**
---	---	---	---	---	---	---
1	0					
2	2	0				
3	5	1	0			
4	3	0	0	0		
5	0	0	0	1	0	
6	8	0	4	0	5	0	 **Mark as follows:** **1 mark**: 0s in correct places **1 mark:** all other values correct **I.** non-zero symbols used to denote no edge but only for showing no edge going from a node to itself	
03	2	**All marks for AO1 (understanding)** Adjacency list appropriate when there are few edges between vertices // when graph/matrix is sparse; **NE.** few edges Adjacency list appropriate when edges rarely changed; Adjacency list appropriate when presence/absence of specific edges does not need to be tested (frequently); **A.** Alternative words which describe edge, eg connection, line, arc **Max 2**	2

03	3	**Mark is for AO2 (apply)** It contains a cycle / cycles;	1
03	4	**Mark for AO1 (knowledge)** A graph where each edge has a weight/value associated with it;	1
03	5	**All marks AO2 (apply)** **Mark as follows:** **I.** output column **1 mark:** first value of `A` is 2 **1 mark:** second value of `A` is 5 and third value is 3 **1 mark:** fourth and subsequent values of `A` are 8, 3, 7, 4, 9 with no more values after this **1 mark:** `D[2]` is set to 2 and then does not change	7

				D						P					
U	**Q**	**V**	**A**	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

1 mark: `D[3]` is set to 5 and then changes to 3 and does not change again
1 mark: correct final values for each position of array `P`

				D						P					
U	**Q**	**V**	**A**	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

1 mark: correct final values for `D[1], D[4], D[5], D[6]`

				D						P					
U	**Q**	**V**	**A**	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

Max 6 marks if any errors

03 | 6 | **Mark is for AO2 (analyse)**

The shortest distance / time between locations/nodes 1 and 6;

NE distance/time between locations/nodes 1 and 6
R. shortest route/path

1

03	7	**All marks AO2 (analyse)** Used to store the previous node/location in the path (to this node); Allows the path (from node/location 1 to any other node/location) to be recreated // stores the path (from node/location 1 to any other node/location); **Max 1** if not clear that the values represent the shortest path **Alternative answer** Used to store the nodes that should be traversed; And the order that they should be traversed; **Max 1** if not clear that the values represent the shortest path	2
04	1	**All marks AO1 (knowledge)** Determining if a program will halt; **Max 1 for the following points, but only award mark if 1st mark was awarded:** without running the program; for a particular input;	2
04	2	**Mark is for AO1 (understanding)** The Halting problem is non-computable / undecideable // there is no algorithm that solves the Halting problem; **A.** it is not computable In general, inspection alone cannot always determine whether any given algorithm will halt for its given inputs // a program cannot be written that can determine whether any given algorithm will halt for its given inputs; **Max 1 mark**	1
04	3	**All marks AO1 (knowledge)** Finite set of states (in a state transition diagram); A set of transition rules; A (sensing) read-write head (that can move along the tape one square at a time); Start state; (Set of) accepting / halting states; State register // current state;	2

04	4	**All marks AO1 (knowledge)** A Turing machine that can execute/simulate the behaviour of any other Turing machine // can compute any computable sequence; Faithfully executes operations on the data precisely as the simulated TM does; (Note: must have idea of same process) Description of/Instructions for TM (and the TM's input) are stored on the (Universal Turing machine's) tape // The UTM acts as an interpreter; **A.** take any other TM and data as input *Alternative definition:* A UTM, U, is an interpreter that reads the description <M> of any arbitrary Turing machine M; and faithfully executes operations on data D precisely as M does.; The description <M> is written at the beginning of the tape, followed by D.; **Max 2 marks**	2
04	5	**Mark is for AO1 (understanding)** Because it has an infinite amount of memory/tape;	1

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 number value and includes two iterative structures. An attempt has been made to check for factors of the number entered, 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 a selection structure is needed to compare user's input with the number 1
2. Identifying that a loop is needed that repeats from 2 to the square root of the number entered **A.** half the value of the number entered **A.** to the number 1 less than the number entered
3. Identifying that use of remainder operator needed **A.** alternative methods to using the remainder operator that calculate if there is a remainder
4. Boolean variable (or equivalent) used to indicate if a number is prime or not

Alternative AO3 design marks:

1. Identifying that a selection structure is needed to compare user's input with the number 1
2. Using nested loops that generate pairs of potential factors
3. Identifying that a test is needed to compare the multiplied factor pairs with the number being checked
4. Boolean (or equivalent) variable used to indicate if a number is prime or not

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. Correct termination condition on iterative structure that repeats until the user does not want to enter another number
6. Suitable prompt, inside iterative structure that asks the user to enter a number and number entered by user is stored in a suitable-named variable
7. Iterative structure that checks for factors has correct syntax and start/end conditions
8. Correct test to see if a potential factor is a factor of the number entered, must be inside the iterative structure for checking factors and the potential factor must change each iteration
9. If an output message saying "Is prime" or "Is not prime" is shown for every integer (greater than 1) **A.** any suitable message
10. Outputs correct message "Is not prime" or "Is prime" under all correct circumstances **A.** any suitable message
11. Outputs message "Not greater than 1" under the correct circumstances **A.** any suitable message

		12. In an appropriate location in the code asks the user if they want to enter another number **R.** if message will not be displayed after each time the user has entered a number **Note for examiners:** if a candidate produces an unusual answer for this question which seems to work but does not match this mark scheme then this answer should be referred to team leader for guidance on how it should be marked.	

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 captures showing the number 1 being entered with the message "Not greater than 1" displayed, then the number 5 being entered with the message "Is prime" displayed and then the number 8 being entered with the message "Is not prime" being displayed and program stops after user input stating they do not want to enter another number;
		
		A. alternative messages being displayed if they match code from 05.1
		
		`Enter a number: 1`
		`Not greater than 1`
		`Again (y or n)? y`
		`Enter a number: 5`
		`Is prime`
		`Again (y or n)? y`
		`Enter a number: 8`
		`Is not prime`
		`Again (y or n)? n`
		`>>>`
			1

06	1	**Mark is for AO2 (analyse)** `Len` (Python/VB only); `Length` (Pascal/Java only); `IndexOf` (C#/VB only); **I.** case **I.** spacing **R.** if any additional code	1
06	2	**Mark is for AO2 (analyse)** `Item` // `RandNo` // `Count`; `Rnd`; (Java only) **A.** `MaxSize` **I.** case **I.** spacing **R.** if any additional code **R.** if spelt incorrectly	1
06	3	**All marks AO1 (understanding)** **Mark as follows** • Check for 1st mark point from either solution 1 or solution 2. • 2nd mark point for Solution 1 only to be awarded if 1st mark point for Solution 1 has been awarded. • 2nd mark point for Solution 2 only to be awarded if 1st mark point for Solution 2 has been awarded **Solution 1** **1st mark:** With a linear queue there could be locations available that are not able to be used **A.**there could be wasted space (where there is space available in the data structure but it is unusable as it is in front of the data items in the queue); **2nd mark:** (To avoid this issue) items in the queue are all shuffled forward when an item is deleted from (the front of the) queue; // Circular lists "wrap round" so (avoid this problem as) the front of the queue does not have to be in the first position in the data structure;	2

		Solution 2 **Alternative answer** **1st mark:** Items in a linear queue are all shuffled forward when an item is deleted from (the front); // No need to shuffle items forward after deleting an item in a circular queue; **2nd mark:** this makes (deleting from) (large) linear lists time inefficient; // meaning circular queues are more time efficient (when deleting);	
06	4	**Mark is for AO2 (analyse)** The queue is small in size (so the time inefficiency is not significant);	1
06	5	**Mark is for AO1 (understanding)** Front // pointer to the front of the queue;	1
06	6	**All marks for AO2 (analyse)** Change the `Add` method; Generate a random number between 1 and 2; **NE.** so there is a 50% chance **Note for examiners:** needs to be clear how a 50% chance is created If it is a 1 then generate a random number from 0, 4, 8, 13, 14, 17, 18, 19 // if it is a 1 then generate a random number from those equivalent to 1-point tiles; Otherwise generate a random number from the other numbers between 0 and 25 // otherwise generate a random number from those equivalent to non 1-point tiles; **A.** equivalent methods to the one described **Note for examiners:** refer unusual answers that would work to team leader	4
06	7	**All marks for AO2 (analyse)** Iterate over the characters in the string; Get the character code for the current character; Subtract 32 from the character code // AND the character code with the bit pattern 1011111 / 11011111 // AND the character code with (the decimal value) 95 / 223; **A.** Hexadecimal equivalents Convert that value back into a character and replace the current character with the new character; **A.** answers that create a new string instead of replace characters in the existing string	4

		Alternative answer Iterate over the characters in the string; Using a list of the lowercase letters and a list of the uppercase letters; Find the index of the lowercase letter in the list of lowercase letters; Get the character in the corresponding position in the uppercase list and replace the current character with the new character; **A.** answers that create a new string instead of replace characters in the existing string	

07	1	**Mark is for AO3 (programming)** Selection structure with correct condition(s) (9, 23) added in suitable place and value of 4 assigned to two tiles in the dictionary; **R.** if any other tile values changed	1
07	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from 07.1, including prompts on screen capture matching those in code.* *Code for 07.1 must be sensible.* Screen captures showing the requested test being performed and the correct points values for J, X, Z and Q are shown; **I.** order of letters TILE VALUES Points for X: 4 Points for R: 1 Points for Q: 5 Points for Z: 5 Points for M: 2 Points for K: 3 Points for A: 1 Points for Y: 3 Points for L: 2 Points for I: 1 Points for F: 3 Points for H: 3 Points for D: 2 Points for U: 2 Points for N: 1 Points for V: 3 Points for T: 1 Points for E: 1 Points for W: 3 Points for C: 2 Points for G: 2 Points for P: 2 Points for J: 4 Points for O: 1 Points for B: 2 Points for S: 1 Either: enter the word you would like to play OR press 1 to display the letter values OR press 4 to view the tile queue OR press 7 to view your tiles again OR press 0 to fill hand and stop the game.	1

08	1	**All marks for AO3 (programming)** Iterative structure with one correct condition added in suitable place; Iterative structure with second correct condition and logical connective; Suitable prompt displayed inside iterative structure or in appropriate place before iterative structure; **A.** any suitable prompt `StartHandSize` assigned user-entered value inside iterative structure; **Max 3** if code contains errors	4

08	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 08.1, including prompts on screen capture matching those in code.
		Code for 08.1 must be sensible.
		
		Screen capture(s) showing that after the values 0 and 21 are entered the user is asked to enter the start hand size again and then the menu is displayed;
		
		`++++++++++++++++++++++++++++++++++++`
		`+ Welcome to the WORDS WITH AQA game +`
		`++++++++++++++++++++++++++++++++++++`
		
		`Enter start hand size: 0`
		`Enter start hand size: 21`
		`Enter start hand size: 5`
		
		`=========`
		`MAIN MENU`
		`=========`
		
		`1. Play game with random start hand`
		`2. Play game with training start hand`
		`9. Quit`
		
		`Enter your choice: 1`
		
		`Player One it is your turn.`
			1

09	1	**All marks for AO3 (programming)** 1) Create variables to store the current start, mid and end points; **A.** no variable for midpoint if midpoint is calculated each time it is needed in the code 2) Setting correct initial values for start and end variables; 3) Iterative structure with one correct condition (either word is valid or start is greater than end); **R.** if code is a linear search 4) Iterative structure with 2nd correct condition and correct logic; 5) Inside iterative structure, correctly calculate midpoint between start and end; **A.** mid-point being either the position before or the position after the exact middle if calculated midpoint is not a whole number **R.** if midpoint is sometimes the position before and sometimes the position after the exact middle **R.** if not calculated under all circumstances when it should be 6) Inside iterative structure there is a selection structure that compares word at midpoint position in list with word being searched for; 7) Values of start and end changed correctly under correct circumstances; 8) True is returned if match with midpoint word found and True is not returned under any other circumstances; **I.** missing statement to display current word **Max 7** if code contains errors	8

Alternative answer using recursion

1) Create variable to store the current midpoint, start and end points passed as parameters to subroutine; **A.** no variable for midpoint if midpoint is calculated each time it is needed in the code **A.** midpoint as parameter instead of as local variable
2) Initial subroutine call has values of 0 for startpoint parameter and number of words in `AllowedWords` for endpoint parameter;
3) Selection structure which contains recursive call if word being searched for is after word at midpoint;
4) Selection structure which contains recursive call if word being searched for is before word at midpoint;
5) Correctly calculate midpoint between start and end;
   A. midpoint being either the position before or the position after the exact middle if calculated midpoint is not a whole number **R.** if midpoint is sometimes the position before and sometimes the position after the exact middle **R.** if not calculated under all circumstances when it should be
6) There is a selection structure that compares word at midpoint position in list with word being searched for and there is no recursive call if they are equal with a value of True being returned;
7) In recusive calls the parameters for start and end points have correct values;
8) There is a selection structure that results in no recursive call and False being returned if it is now known that the word being searched for is not in the list;

Note for examiners: mark points 1, 2, 7 could be replaced by recursive calls that appropriately half the number of items in the list of words passed as a parameter – this would mean no need for start and end points. In this case award one mark for each of the two recursive calls if they contain the correctly reduced lists and one mark for the correct use of the length function to find the number of items in the list. These marks should not be awarded if the list is passed by reference resulting in the original list of words being modified.

I. missing statement to display current word
Max 7 if code contains errors

Note for examiners: refer unusual solutions to team leader

09	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from 09.1, including prompts on screen capture matching those in code.* *Code for 09.1 must be sensible.* **R.** if comparison words not shown in screen capture r Screen capture(s) showing that the word "jars" was entered and the words "MALEFICIAL", "DONGLES", "HAEMAGOGUE", "INTERMINGLE", "LAGGER", "JOULED", "ISOCLINAL", "JAUKING", "JACARANDA", "JAMBEUX", "JAPONICA", "JAROVIZE", "JASPER", "JARTA", "JARRAH", "JARRINGLY", "JARS" are displayed in that order; **A.** "MALEFICIAL", "DONGOLA", "HAEMAGOGUES", "INTERMINGLED", "LAGGERS", "JOULING", "ISOCLINE", "JAUNCE", "JACARE", "JAMBING", "JAPPING", "JAROVIZING", "JASPERISES", "JARVEY", "JARRINGLY", "JARTA", "JARS" being displayed if alternative answer for mark point 5 in 9.1 used **ALTERNATIVE ANSWERS (for different versions of text file)** Screen capture(s) showing that the word "jars" was entered and the words "MALEATE", "DONDER", "HADST", "INTERMENDIS", "LAGAN", "JOTTERS", "ISOCHROMATIC", "JASPERS", "JABBING", "JALOUSIE", "JAPANISES", "JARGOONS", "JARRED", "JASIES", "JARUL", "JARS" are displayed in that order; **A.** "MALEATE", "DONDERED", "HAE", "INTERMEDIUM", "LAGANS", "JOTTING", "ISOCHROMOSONES", "JASPERWARES", "JABBLED", "JALOUSING", "JAPANIZED", "JARINA", "JARRINGS", "JASMINES", "JARVEYS", "JARTAS", "JARSFUL", "JARS" being displayed if alternative answer for mark point 5 in 9.1 used Screen capture(s) showing that the word "jars" was entered and the words "LAMP", "DESK", "GAGE", "IDEAS", "INVITATION", "JOURNALS", "JAMAICA", "JEWELLERY", "JEAN", "JAR", "JAY", "JASON", "JARS" are displayed in that order; **A.** "LAMP", "DESK", "GAGE", "IDEAS", "INVITATIONS", "JOURNEY", "JAMIE", "JEWISH", "JEEP", "JAVA", "JAPAN", "JARS" being displayed if alternative answer for mark point 5 in 9.1 used Either: enter the word you would like to play OR press 1 to display the letter values OR press 4 to view the tile queue OR press 7 to view your tiles again OR press 0 to fill hand and stop the game. >jars MALEFICIAL	1

```
DONGLES
HAEMAGOGUE
INTERMINGLE
LAGGER
JOULED
ISOCLINAL
JAUKING
JACARANDA
JAMBEUX
JAPONICA
JAROVIZE
JASPER
JARTA
JARRAH
JARRINGLY
JARS

Valid word

Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get three extra tiles (3)
OR
     get no new tiles (4)?
>
```

10	1	**All marks for AO3 (programming)** 1) Creating new subroutine called `CalculateFrequencies` with appropriate interface; **R.** if spelt incorrectly **I.** case 2) Iterative structure that repeats 26 times (once for each letter in the alphabet); 3) Iterative structure that looks at each word in `AllowedWords`; 4) Iterative structure that looks at each letter in a word and suitable nesting for iterative structures; 5) Selection structure, inside iterative structure, that compares two letters; **A.** use of built-in functions that result in same functionality as mark points 4 and 5;; 6) Inside iterative structure increases variable used to count instances of a letter; 7) Displays a numeric count (even if incorrect) and the letter for each letter in the alphabet; **A.** is done in sensible place in `DisplayTileValues` 8) Syntactically correct call to new subroutine from `DisplayTileValues`; **A.** any suitable place for subroutine call **Alternative answer** If answer looks at each letter in AllowedWords in turn and maintains a count (eg in array/list) for the number of each letter found then mark points 2 and 5 should be: 2) Creation of suitable data structure to store 26 counts. 5) Appropriate method to select count that corresponds to current letter. **Max 7** if code contains errors	8

10	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 10.1, including prompts on screen capture matching those in code.
		Code for 10.1 must be sensible.
		
		Screen capture(s) showing correct list of letter frequencies are displayed;
		
		I. Ignore order of letter frequency pairs
		I. any additional output eg headings like “Letter” and “Count”
		
		`Letter frequencies in the allowed words are:`
		`A  188704`
		`B  44953`
		`C  98231`
		`D  81731`
		`E  275582`
		`F  28931`
		`G  67910`
			1

```
H   60702
I   220483
J   4010
K   22076
L   127865
M   70700
N   163637
O   161752
P   73286
Q   4104
R   170522
S   234673
T   159471
U   80636
V   22521
W   18393
X   6852
Y   39772
Z   11772


Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>
```

ALTERNATIVE ANSWERS (for different versions of text file)

```
Letter frequencies in the allowed words are:
A   188627
B   44923
C   98187
D   81686
E   275478
F   28899
G   67795
H   60627
I   220331
J   4007
K   22028
L   127814
M   70679
N   163547
O   161720
P   73267
Q   4104
R   170461
S   234473
T   159351
U   80579
V   22509
W   18377
X   6852
```

```
Y   39760
Z   11765


Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>



Letter frequencies in the allowed words are:
A   5299
B   1105
C   2980
D   2482
E   7523
F   909
G   1692
H   1399
I   5391
J   178
K   569
L   3180
M   1871
N   4762
O   4177
P   1992
Q   122
R   4812
S   4999
T   4695
U   1898
V   835
W   607
X   246
Y   999
Z   128


Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
>
```

11	1	**All marks for AO3 (programming)** **Modifying subroutine `UpdateAfterAllowedWord`:** 1) Correct subroutine call to `GetScoreForWordAndPrefix` added in `UpdateAfterAllowedWord`; 2) Result returned by `GetScoreForWordAndPrefix` added to `PlayerScore`; **A.** alternative names for subroutine `GetScoreForWordAndPrefix` if match name of subroutine created **Creating new subroutine:** 3) Subroutine `GetScoreForWordAndPrefix` created; **R.** if spelt incorrectly **I.** case 4) All data needed (`Word`, `TileDictionary`, `AllowedWords`) is passed into subroutine via interface; 5) Integer value always returned by subroutine; **Base case in subroutine:** 6) Selection structure for differentiating base case and recursive case with suitable condition (word length of 0 // 1 // 2); **R.** if base case will result in recursion 7) If base case is word length is 0 then value of 0 is returned by subroutine and there is no recursive call // if base case is word length is 1 then value of 0 is returned by subroutine and there is no recursive call // if base case is word length is 2 the the subroutine returns 0 if the two-letter word is not a valid word and returns the score for the two-letter word if it is a valid word; **Recursive case in subroutine:** 8) Selection structure that contains code that adds value returned by call to `GetScoreForWord` to score if word is valid; **A.** no call to subroutine `GetScoreForWord` if correct code to calculate score included in sensible place in `GetScoreForWordAndPrefix` subroutine **R.** if no check for word being valid 9) Call to `GetScoreForWordAndPrefix`; 10) Result from recursive call added to score; 11) Recursion will eventually reach base case as recursive call has a parameter that is word with last letter removed;	11

		How to mark question if no attempt to use recursion: Mark points 1-5 same as for recursive attempt. No marks awarded for mark points 6-11, instead award marks as appropriate for mark points 12-14. 12) Adds the score for the original word to the score once // sets the initial score to be the score for the original word; **A.** no call to subroutine `GetScoreForWord` if correct code to calculate score included in sensible place in `GetScoreForWordAndPrefix` subroutine. **Note for examiners:** there is no need for the answer to check if the original word is valid 13) Iterative structure that will repeat *n - 1* times where *n* is the length of the word; **A.** *n – 2* **A.** n 14) Inside iterative structure adds score for current prefix word, if it is a valid word, to score once; **A.** no call to `GetScoreForWord` if own code to calculate score is correct **Max 10** if code contains errors **Max 8** if recursion not used in an appropriate way	

11	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 11.1, including prompts on screen capture matching those in code.
		Code for 11.1 must be sensible.
		
		Screen capture(s) showing that the word abandon was entered and the new score of 78 is displayed;
		
		`Do you want to:`
		`  replace the tiles you used (1) OR`
		`  get three extra tiles (2) OR`
		`  replace the tiles you used and get three extra tiles (3) OR`
		`  get no new tiles (4)?`
		`>4`
		
		`Your word was: ABANDON`
		`Your new score is: 78`
		`You have played 7 tiles so far in this game.`
		
		`Press Enter to continue`
			1

Python 2

05	1	`import math`
		`again = "y"`
		`while again == "y":`
		`  num = int(raw_input("Enter a number: "))`
		`  if num > 1:`
		`  prime = True`
		`  for count in range(2, int(math.sqrt(num)) + 1):`
		`  if num % count == 0:`
		`  prime = False`
		`  if prime == True:`
		`  print "Is prime"`
		`  else:`
		`  print "Is not prime"`
		`  else:`
		`  print "Not greater than 1"`
		`  again = raw_input("Again (y or n)? ")`
			12

07	1	`def CreateTileDictionary():`
		`  TileDictionary = dict()`
		`  for Count in range(26):`
		`  if Count in [0, 4, 8, 13, 14, 17, 18, 19]:`
		`  TileDictionary[chr(65 + Count)] = 1`
		`  elif Count in [1, 2, 3, 6, 11, 12, 15, 20]:`
		`  TileDictionary[chr(65 + Count)] = 2`
		`  elif Count in [5, 7, 10, 21, 22, 24]:`
		`  TileDictionary[chr(65 + Count)] = 3`
		`  elif Count in [9, 23]:`
		`  TileDictionary[chr(65 + Count)] = 4`
		`  else:`
		`  TileDictionary[chr(65 + Count)] = 5`
		`  return TileDictionary`
			2

08	1	…
		`  StartHandSize = int(raw_input("Enter start hand size: "))`
		`  while StartHandSize < 1 or StartHandSize > 20:`
		`  StartHandSize = int(raw_input("Enter start hand size: "))`
		…
			4

09	1	`def CheckWordIsValid(Word, AllowedWords):`
		`  ValidWord = False`
		`  Start = 0`
		`  End = len(AllowedWords) - 1`
		`  while not ValidWord and Start <= End:`
		`  Mid = (Start + End) // 2`
		`  print AllowedWords[Mid]`
		`  if AllowedWords[Mid] == Word:`
		`  ValidWord = True`
		`  elif Word > AllowedWords[Mid]:`
		`  Start = Mid + 1`
		`  else:`
		`  End = Mid - 1`
		`  return ValidWord`
			8

10	1	**def CalculateFrequencies(AllowedWords):**
		** print "Letter frequencies in the allowed words are:"**
		** for Code in range (26):**
		** LetterCount = 0**
		** LetterToFind = chr(Code + 65)**
		** for Word in AllowedWords:**
		** for Letter in Word:**
		** if Letter == LetterToFind:**
		** LetterCount += 1**
		** sys.stdout.write(LetterToFind + " " + LetterCount)**
		
		def DisplayTileValues(TileDictionary, AllowedWords):
		  print()
		  print("TILE VALUES")
		  print()
		  for Letter, Points in TileDictionary.items():
		  sys.stdout.write("Points for " + Letter + ": " + str(Points) + "\n")
		  print()
		** CalculateFrequencies(AllowedWords)**
		
		Alternative answer
		
		def CalculateFrequencies(AllowedWords):
		** for Letter in "ABCDEFGHIJKLNOPQRSTYVWXYZ":**
		** Count=0**
		** for Word in AllowedWords:**
		** NumberOfTimes = Word.count(Letter)**
		** Count = Count + NumberOfTimes**
		** sys.stdout.write(Letter + " " + str(Count))**
		
		Alternative answer
		
		def CalculateFrequencies(AllowedWords):
		** Counts = []**
		** for a in range(26):**
		** Counts.append(0)**
		** for Word in AllowedWords:**
		** for Letter in Word:**
		** Counts[ord(Letter) - 65] += 1**
		** for a in range(26):**
		** sys.stdout.write(chr(a + 65) + " " + str(Counts[a]))**
			8

11	1	def UpdateAfterAllowedWord(Word, PlayerTiles, PlayerScore, PlayerTilesPlayed, TileDictionary, AllowedWords):
		  PlayerTilesPlayed += len(Word)
		  for Letter in Word:
		  PlayerTiles = PlayerTiles.replace(Letter, "", 1)
		** PlayerScore += GetScoreForWordAndPrefix(Word, TileDictionary, AllowedWords)**
		  return PlayerTiles, PlayerScore, PlayerTilesPlayed
			11

```
def GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords):
  if len(Word) <= 1:
    return 0
  else:
    Score = 0
    if CheckWordIsValid(Word, AllowedWords):
      Score += GetScoreForWord(Word, TileDictionary)
    Score += GetScoreForWordAndPrefix(Word[0:len(Word) -
1], TileDictionary, AllowedWords)
    return Score
```

Alternative answer

```
def GetScoreForWordAndPrefix(Word,TileDictionary,
AllowedWords):
  Score = 0
  if CheckWordIsValid(Word,AllowedWords):
      Score += GetScoreForWord(Word, TileDictionary)
  if len(Word[:-1]) > 0:
      Score +=GetScoreForWordAndPrefix(Word[:-1],
TileDictionary,AllowedWords)
  return Score
```

Python 3

05	1	`import math`
		`again = "y"`
		`while again == "y":`
		`  num = int(input("Enter a number: "))`
		`  if num > 1:`
		`  prime = True`
		`  for count in range(2, int(math.sqrt(num)) + 1):`
		`  if num % count == 0:`
		`  prime = False`
		`  if prime == True:`
		`  print("Is prime")`
		`  else:`
		`  print("Is not prime")`
		`  else:`
		`  print("Not greater than 1")`
		`  again = input("Again (y or n)? ")`
			12

07	1	`def CreateTileDictionary():`
		`  TileDictionary = dict()`
		`  for Count in range(26):`
		`  if Count in [0, 4, 8, 13, 14, 17, 18, 19]:`
		`  TileDictionary[chr(65 + Count)] = 1`
		`  elif Count in [1, 2, 3, 6, 11, 12, 15, 20]:`
		`  TileDictionary[chr(65 + Count)] = 2`
		`  elif Count in [5, 7, 10, 21, 22, 24]:`
		`  TileDictionary[chr(65 + Count)] = 3`
		`  elif Count in [9, 23]:`
		`  TileDictionary[chr(65 + Count)] = 4`
		`  else:`
		`  TileDictionary[chr(65 + Count)] = 5`
		`  return TileDictionary`
			2

08	1	…
		`  StartHandSize = int(input("Enter start hand size: "))`
		`  while StartHandSize < 1 or StartHandSize > 20:`
		`  StartHandSize = int(input("Enter start hand size: "))`
		…
			4

09	1	`def CheckWordIsValid(Word, AllowedWords):`
		`  ValidWord = False`
		`  Start = 0`
		`  End = len(AllowedWords) - 1`
		`  while not ValidWord and Start <= End:`
		`  Mid = (Start + End) // 2`
		`  print(AllowedWords[Mid])`
		`  if AllowedWords[Mid] == Word:`
		`  ValidWord = True`
		`  elif Word > AllowedWords[Mid]:`
		`  Start = Mid + 1`
		`  else:`
		`  End = Mid - 1`
		`  return ValidWord`
			8

10	1	**def CalculateFrequencies(AllowedWords):**
		** print("Letter frequencies in the allowed words are:")**
		** for Code in range (26):**
		** LetterCount = 0**
		** LetterToFind = chr(Code + 65)**
		** for Word in AllowedWords:**
		** for Letter in Word:**
		** if Letter == LetterToFind:**
		** LetterCount += 1**
		** print(LetterToFind, " ", LetterCount)**
		
		def DisplayTileValues(TileDictionary, AllowedWords):
		  print()
		  print("TILE VALUES")
		  print()
		  for Letter, Points in TileDictionary.items():
		  print("Points for " + Letter + ": " + str(Points))
		  print()
		  CalculateFrequencies(AllowedWords)
		
		Alternative answer
		
		def CalculateFrequencies(AllowedWords):
		** for Letter in "ABCDEFGHIJKLNOPQRSTYVWXYZ":**
		** Count=0**
		** for Word in AllowedWords:**
		** NumberOfTimes = Word.count(Letter)**
		** Count = Count + NumberOfTimes**
		** print(Letter,Count)**
		
		Alternative answer
		
		def CalculateFrequencies(AllowedWords):
		** Counts = []**
		** for a in range(26):**
		** Counts.append(0)**
		** for Word in AllowedWords:**
		** for Letter in Word:**
		** Counts[ord(Letter) - 65] += 1**
		** for a in range(26):**
		** print(chr(a + 65), Counts[a])**
			8

11	1	def UpdateAfterAllowedWord(Word, PlayerTiles, PlayerScore, PlayerTilesPlayed, TileDictionary, AllowedWords):
		  PlayerTilesPlayed += len(Word)
		  for Letter in Word:
		  PlayerTiles = PlayerTiles.replace(Letter, "", 1)
		  PlayerScore += GetScoreForWordAndPrefix(Word, TileDictionary, AllowedWords)
		  return PlayerTiles, PlayerScore, PlayerTilesPlayed
			11

```
def GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords):
  if len(Word) <= 1:
    return 0
  else:
    Score = 0
    if CheckWordIsValid(Word, AllowedWords):
      Score += GetScoreForWord(Word, TileDictionary)
    Score += GetScoreForWordAndPrefix(Word[0:len(Word) -
1], TileDictionary, AllowedWords)
    return Score
```

Alternative answer

```
def GetScoreForWordAndPrefix(Word,TileDictionary,
AllowedWords):
  Score = 0
  if CheckWordIsValid(Word,AllowedWords):
      Score += GetScoreForWord(Word, TileDictionary)
  if len(Word[:-1]) > 0:
      Score +=GetScoreForWordAndPrefix(Word[:-1],
TileDictionary,AllowedWords)
  return Score
```

Visual Basic

05	1	`Sub Main()`
		`    Dim Again As Char = "y"`
		`    Dim Num As Integer`
		`    Dim Prime As Boolean`
		`    While Again = "y"`
		`        Console.Write("Enter a number: ")`
		`        Num = Console.ReadLine()`
		`        If Num > 1 Then`
		`            Prime = True`
		`            For Count = 2 To System.Math.Sqrt(Num)`
		`                If Num Mod Count = 0 Then`
		`                    Prime = False`
		`                End If`
		`            Next`
		`            If Prime Then`
		`                Console.WriteLine("Is prime")`
		`            Else`
		`                Console.WriteLine("Is not prime")`
		`            End If`
		`        Else`
		`            Console.WriteLine("Not greater than 1")`
		`        End If`
		`        Console.Write("Again (y or n)? ")`
		`        Again = Console.ReadLine()`
		`    End While`
		`End Sub`
			12

07	1	`Function CreateTileDictionary() As Dictionary(Of Char, Integer)`
		`    Dim TileDictionary As New Dictionary(Of Char, Integer)()`
		`    For Count = 0 To 25`
		`        If Array.IndexOf({0, 4, 8, 13, 14, 17, 18, 19}, Count) > -1 Then`
		`            TileDictionary.Add(Chr(65 + Count), 1)`
		`        ElseIf Array.IndexOf({1, 2, 3, 6, 11, 12, 15, 20}, Count) > -1 Then`
		`            TileDictionary.Add(Chr(65 + Count), 2)`
		`        ElseIf Array.IndexOf({5, 7, 10, 21, 22, 24}, Count) > -1 Then`
		`            TileDictionary.Add(Chr(65 + Count), 3)`
		`        **ElseIf Array.IndexOf({9, 23}, Count) > -1 Then`**
		`            **TileDictionary.Add(Chr(65 + Count), 4)`**
		`        Else`
		`            TileDictionary.Add(Chr(65 + Count), 5)`
		`        End If`
		`    Next`
		`    Return TileDictionary`
		`End Function`
			2

08	1	… **Do** **Console.Write("Enter start hand size: ")** **StartHandSize = Console.ReadLine()** **Loop Until StartHandSize >= 1 And StartHandSize <= 20** …	4

09	1	Function CheckWordIsValid(ByVal Word As String, ByRef AllowedWords As List(Of String)) As Boolean
		Dim ValidWord As Boolean = False
		Dim LStart As Integer = 0
		Dim LMid As Integer
		Dim LEnd As Integer = Len(AllowedWords) - 1
		While Not ValidWord And LStart <= LEnd
		LMid = (LStart + LEnd) \ 2
		Console.WriteLine(AllowedWords(LMid))
		If AllowedWords(LMid) = Word Then
		ValidWord = True
		ElseIf Word > AllowedWords(LMid) Then
		LStart = LMid + 1
		Else
		LEnd = LMid - 1
		End If
		End While
		Return ValidWord
		End Function
			8

10	1	**Sub CalculateFrequencies(ByRef AllowedWords As List(Of String))**
		Dim LetterCount As Integer
		Dim LetterToFind As Char
		Console.WriteLine("Letter frequencies in the allowed words are:")
		For Code = 0 To 25
		LetterCount = 0
		LetterToFind = Chr(Code + 65)
		For Each Word In AllowedWords
		For Each Letter In Word
		If Letter = LetterToFind Then
		LetterCount += 1
		End If
		Next
		Next
		Console.WriteLine(LetterToFind & " " & LetterCount)
		Next
		End Sub
		
		Sub DisplayTileValues(ByVal TileDictionary As Dictionary(Of Char, Integer), ByRef AllowedWords As List(Of String))
		Console.WriteLine()
		Console.WriteLine("TILE VALUES")
		Console.WriteLine()
		For Each Tile As KeyValuePair(Of Char, Integer) In
			8

```
TileDictionary
        Console.WriteLine("Points for " & Tile.Key & ": "
& Tile.Value)
    Next
    Console.WriteLine()
    CalculateFrequencies(AllowedWords)
End Sub
```

Alternative answer

```
Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
    Dim NumberOfTimes, Count As Integer
    Console.WriteLine("Letter frequencies in the allowed
words are:")
    For Each Letter In "ABCDEFGHIJKLMOPQRSTUVWXYZ"
        Count = 0
        For Each Word In AllowedWords
            NumberOfTimes = Word.Split(Letter).Length - 1
            Count += NumberOfTimes
        Next
        Console.WriteLine(Letter & " " & Count)
    Next
End Sub
```

Alternative answer

```
Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
    Dim Counts(25) As Integer
    For Count = 0 To 25
        Counts(Count) = 0
    Next
    Console.WriteLine("Letter frequencies in the allowed
words are:")
    For Each Word In AllowedWords
        For Each Letter In Word
            Counts(Asc(Letter) - 65) += 1
        Next
    Next
    For count = 0 To 25
        Console.WriteLine(Chr(count + 65) & " " &
Counts(count))
    Next
End Sub
```

11	1	Sub UpdateAfterAllowedWord(ByVal Word As String, ByRef PlayerTiles As String, ByRef PlayerScore As Integer, ByRef PlayerTilesPlayed As Integer, ByVal TileDictionary As Dictionary(Of Char, Integer), ByRef AllowedWords As List(Of String))
		  PlayerTilesPlayed += Len(Word)
		  For Each Letter In Word
		  PlayerTiles = Replace(PlayerTiles, Letter, "", , 1)
		  Next
		  PlayerScore += GetScoreForWordAndPrefix(Word, TileDictionary, AllowedWords)
		End Sub
		
		Function GetScoreForWordAndPrefix(ByVal Word As String, ByVal TileDictionary As Dictionary(Of Char, Integer), ByRef AllowedWords As List(Of String)) As Integer
		  Dim Score As Integer
		  If Len(Word) <= 1 Then
		  Return 0
		  Else
		  Score = 0
		  If CheckWordIsValid(Word, AllowedWords) Then
		  Score += GetScoreForWord(Word, TileDictionary)
		  End If
		  Score += GetScoreForWordAndPrefix(Mid(Word, 1, Len(Word) - 1), TileDictionary, AllowedWords)
		  End If
		  Return Score
		End Function
		
		Alternative answer
		
		Function GetScoreForWordAndPrefix(ByVal Word As String, ByVal TileDictionary As Dictionary(Of Char, Integer), ByRef AllowedWords As List(Of String)) As Integer
		  Dim Score As Integer = 0
		  If CheckWordIsValid(Word, AllowedWords) Then
		  Score += GetScoreForWord(Word, TileDictionary)
		  End If
		  If Len(Word) - 1 > 0 Then
		  Score += GetScoreForWordAndPrefix(Mid(Word, 1, Len(Word) - 1), TileDictionary, AllowedWords)
		  End If
		  Return Score
		End Function
			11

C#

05	1	{
		  string Again = "Y";
		  int Num = 0;
		  bool Prime = true;
		  while (Again == "Y")
		  {
		  Console.Write("Enter a number: ");
		  Num = Convert.ToInt32(Console.ReadLine());
		  if (Num > 1)
		  {
		  for (int Count = 2; Count < Convert.ToInt32(Math.Sqrt(Num)) + 1; Count++)
		  {
		  if (Num % Count == 0)
		  {
		  Prime = false;
		  }
		  }
		  if (Prime == true)
		  {
		  Console.WriteLine("Is prime");
		  }
		  else
		  {
		  Console.WriteLine("Is not prime");
		  }
		  }
		  else
		  {
		  Console.WriteLine("Not greater than 1");
		  }
		  Console.Write("Again (y or n)? ");
		  Again = Console.ReadLine().ToUpper();
		  }
		}
			12

07	1	private static void CreateTileDictionary(ref Dictionary<char, int> TileDictionary)
		{
		  int[] Value1 = { 0, 4, 8, 13, 14, 17, 18, 19 };
		  int[] Value2 = { 1, 2, 3, 6, 11, 12, 15, 20 };
		  int[] Value3 = { 5, 7, 10, 21, 22, 24 };
		  int[] Value4 = { 9, 23 };
		
		  for (int Count = 0; Count < 26; Count++)
		  {
		  if (Value1.Contains(Count))
		  {
		  TileDictionary.Add((char)(65 + Count), 1);
		  }
		  else if (Value2.Contains(Count))
		  {
		  TileDictionary.Add((char)(65 + Count), 2);
		  }
			2

		else if (Value3.Contains(Count))
		{
		TileDictionary.Add((char)(65 + Count), 3);
		}
		else if (Value4.Contains(Count))
		{
		TileDictionary.Add((char)(65 + Count), 4);
		}
		else
		{
		TileDictionary.Add((char)(65 + Count), 5);
		}
		}
		}	

08	1	… **do** **{** **Console.Write("Enter start hand size: ");** **StartHandSize = Convert.ToInt32(Console.ReadLine());** **} while (StartHandSize < 1 || StartHandSize > 20);** …	4

09	1	private static bool CheckWordIsValid(string Word,
		List<string> AllowedWords)
		{
		bool ValidWord = false;
		int Start = 0;
		int End = AllowedWords.Count - 1;
		int Mid = 0;
		while (!ValidWord && Start <= End)
		{
		Mid = (Start + End) / 2;
		Console.WriteLine(AllowedWords[Mid]);
		if (AllowedWords[Mid] == Word)
		{
		ValidWord = true;
		}
		else if (string.Compare(Word, AllowedWords[Mid]) > 0)
		{
		Start = Mid + 1;
		}
		else
		{
		End = Mid -1;
		}
		}
		return ValidWord;
		}
			8

10	1	**private static void CalculateFrequencies(List<string> AllowedWords)**
		{
		** Console.WriteLine("Letter frequencies in the allowed words are:");**
		** int LetterCount = 0;**
		** char LetterToFind;**
		** for (int Code = 0; Code < 26; Code++)**
		** {**
		** LetterCount = 0;**
		** LetterToFind = (char)(Code + 65);**
		** foreach (var Word in AllowedWords)**
		** {**
		** foreach (var Letter in Word)**
		** {**
		** if (Letter == LetterToFind)**
		** {**
		** LetterCount++;**
		** }**
		** }**
		** }**
		** Console.WriteLine(LetterToFind + " " + LetterCount);**
		** }**
		}
		
		private static void DisplayTileValues(Dictionary<char, int> TileDictionary, List<string> AllowedWords)
		{
		  Console.WriteLine();
		  Console.WriteLine("TILE VALUES");
		  Console.WriteLine();
		  char Letter;
		  int Points;
		  foreach (var Pair in TileDictionary)
		  {
		  Letter = Pair.Key;
		  Points = Pair.Value;
		  Console.WriteLine("Points for " + Letter + ": " + Points);
		  }
		  CalculateFrequencies(AllowedWords);
		  Console.WriteLine();
		}
		
		Alternative answer
		
		private static void CalculateFrequencies(List<string> AllowedWords)
		{
		** Console.WriteLine("Letter frequencies in the allowed words are:");**
		** int LetterCount = 0;**
		** string Alphabet = "ABCDEFGHIJKLNOPQRSTYVWXYZ";**
		** foreach (var Letter in Alphabet)**
		** {**
			8

```
    LetterCount = 0;
    foreach (var Words in AllowedWords)
    {
      LetterCount = LetterCount +
(Words.Split(Letter).Length - 1);
    }
    Console.WriteLine(Letter + " " + LetterCount);
  }
}
```

Alternative answer

```
private static void CalculateFrequencies(List<string>
AllowedWords)
{
  List<int> Counts = new List<int>() ;
  for (int i = 0; i < 26; i++)
  {
    Counts.Add(0);
  }
  foreach (var Words in AllowedWords)
  {
    foreach (var Letter in Words)
    {
      Counts[(int)Letter - 65]++;
    }
  }
  for (int a = 0; a < 26; a++)
  {
    char Alpha =Convert.ToChar( a + 65);
    Console.WriteLine(Alpha + " " + Counts[a] );
  }
}
```

11 | 1 | 11

```
private static void UpdateAfterAllowedWord(string Word,
ref string PlayerTiles, ref int PlayerScore, ref int
PlayerTilesPlayed, Dictionary<char, int> TileDictionary,
List<string> AllowedWords)
{
  PlayerTilesPlayed = PlayerTilesPlayed + Word.Length;
  foreach (var Letter in Word)
  {
    PlayerTiles =
PlayerTiles.Remove(PlayerTiles.IndexOf(Letter), 1);
  }
  PlayerScore = PlayerScore +
GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords);
}

private static int GetScoreForWordAndPrefix(string Word,
Dictionary<char, int> TileDictionary, List<string>
AllowedWords)
{
  int Score = 0;
  if (Word.Length <= 1)
```

```
  {
    return 0;
  }
  else
  {
    Score = 0;
    if (CheckWordIsValid(Word, AllowedWords))
    {
      Score = Score + GetScoreForWord(Word,
TileDictionary);
    }
    Score = Score +
GetScoreForWordAndPrefix(Word.Remove(Word.Length - 1),
TileDictionary, AllowedWords);
    return Score;
  }
}
```

Alternative answer

```
private static int GetScoreForWordAndPrefix(string Word,
Dictionary<char, int> TileDictionary, List<string>
AllowedWords)
{
  int Score = 0;
  if (CheckWordIsValid(Word, AllowedWords))
  {
    Score = Score + GetScoreForWord(Word, TileDictionary);
  }
  if (Word.Remove(Word.Length - 1).Length > 0)
  {
    Score = Score +
GetScoreForWordAndPrefix(Word.Remove(Word.Length - 1),
TileDictionary, AllowedWords);
  }
  return Score;
}
```

Java

05	1	`public static void main(String[] args)`
		`{`
		`    String again;`
		`    do {`
		`        Console.println("Enter a number:");`
		`        int number = Integer.parseInt(Console.readLine());`
		`        if(number <= 1)`
		`        {`
		`            Console.println("Not greater than 1");`
		`        }`
		`        else`
		`        {`
		`            boolean prime = true;`
		`            int count = number - 1;`
		`            while (prime && count > 1)`
		`            {`
		`                if(number%count == 0)`
		`                {`
		`                    prime = false;`
		`                }`
		`                count--;`
		`            }`
		`            if(prime)`
		`            {`
		`                Console.println("Is prime");`
		`            }`
		`            else`
		`            {`
		`                Console.println("Is not prime");`
		`            }`
		`        }`
		`        Console.println("Would you like to enter another number? YES/NO");`
		`        again = Console.readLine();`
		`    } while (again.equals("YES"));`
		`}`
			12

07	1	`Map createTileDictionary()`
		`{`
		`    Map<Character,Integer> tileDictionary = new HashMap<Character,Integer>();`
		`    for (int count = 0; count < 26; count++)`
		`    {`
		`        switch (count) {`
		`            case 0:`
		`            case 4:`
		`            case 8:`
		`            case 13:`
		`            case 14:`
		`            case 17:`
		`            case 18:`
		`            case 19:`
		`                tileDictionary.put((char)(65 + count), 1);`
		`                break;`
			2

		case 1:
		case 2:
		case 3:
		case 6:
		case 11:
		case 12:
		case 15:
		case 20:
		tileDictionary.put((char)(65 + count), 2);
		break;
		case 5:
		case 7:
		case 10:
		case 21:
		case 22:
		case 24:
		tileDictionary.put((char)(65 + count), 3);
		break;
		case 9:
		case 23:
		tileDictionary.put((char)(65 + count), 4);
		break;
		default:
		tileDictionary.put((char)(65 + count), 5);
		break;
		}
		}
		return tileDictionary;
		}	

08	1	... **do {** **Console.println("Enter start hand size: ");** **startHandSize = Integer.parseInt(Console.readLine());** **} while (startHandSize < 1 || startHandSize > 20);** ...	4

09	1	boolean checkWordIsValid(String word, String[] allowedWords)
		{
		boolean validWord = false;
		int start = 0;
		int end = allowedWords.length - 1;
		int mid = 0;
		while (!validWord && start <= end)
		{
		mid = (start + end) / 2;
		Console.println(allowedWords[mid]);
		if (allowedWords[mid].equals(word))
		{
		validWord = true;
		}
		else if (word.compareTo(allowedWords[mid]) > 0)
		{
		start = mid + 1;
			8

		`}`
		`else`
		`{`
		`end = mid -1;`
		`}`
		`}`
		`return validWord;`
		`}`	

10	1	`void calculateFrequencies(String[] allowedWords)`
		`{`
		`int letterCount;`
		`char letterToFind;`
		`for (int count = 0; count < 26; count++)`
		`{`
		`letterCount = 0;`
		`letterToFind = (char)(65 + count);`
		`for(String word:allowedWords)`
		`{`
		`for(char letter : word.toCharArray())`
		`{`
		`if(letterToFind == letter)`
		`{`
		`letterCount++;`
		`}`
		`}`
		`}`
		`Console.println(letterToFind + ", Frequency: " + letterCount);`
		`}`
		`}`
		
		`void displayTileValues(Map tileDictionary, String[] allowedWords)`
		`{`
		`Console.println();`
		`Console.println("TILE VALUES");`
		`Console.println();`
		`for (Object letter : tileDictionary.keySet())`
		`{`
		`int points = (int)tileDictionary.get(letter);`
		`Console.println("Points for " + letter + ": " + points);`
		`}`
		`calculateFrequencies(allowedWords);`
		`Console.println();`
		`}`
		
		Alternative answer
		
		`void calculateFrequencies(String[] allowedWords)`
		`{`
		`int letterCount;`
		`String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";`
		`for(char letter: alphabet.toCharArray())`
			8

```
    {
        letterCount = 0;
        for(String word: allowedWords)
        {
            letterCount += word.split(letter + "").length
- 1;
        }
        Console.println(letter + ", Frequency: " +
letterCount);
    }
}
```

Alternative answer

```
void calculateFrequencies(String[] allowedWords)
{
    int[] counts = new int[26];
    for(String word: allowedWords)
    {
        for(char letter: word.toCharArray())
        {
            int letterPostion = (int)letter - 65;
            counts[letterPostion]++;
        }
    }
    for (int count = 0; count < 26; count++)
    {
        char letter = (char)(65 + count);
        Console.println(letter + ", Frequency: " +
counts[count]);
    }
}
```

11 | 1

```
int getScoreForWordAndPrefix(String word, Map
tileDictionary, String[] allowedWords)
{
    int score = 0;
    if(word.length() < 2)
    {
        return 0;
    }
    else
    {
        if(checkWordIsValid(word, allowedWords))
        {
            score = getScoreForWord(word, tileDictionary);
        }
        word = word.substring(0, word.length()-1);
        return score + getScoreForWordAndPrefix(word,
tileDictionary, allowedWords);
    }
}

void updateAfterAllowedWord(String word, Tiles
```

11

```
playerTiles,
        Score playerScore, TileCount playerTilesPlayed,
Map tileDictionary,
        String[] allowedWords)
{
    playerTilesPlayed.numberOfTiles += word.length();
    for(char letter : word.toCharArray())
    {
        playerTiles.playerTiles =
playerTiles.playerTiles.replaceFirst(letter+"", "");
    }
    playerScore.score += getScoreForWordAndPrefix(word,
tileDictionary, allowedWords);
}
```

Alternative answer

```
int getScoreForWordAndPrefix(String word, Map tileDictionary, String[]
allowedWords)
{
   int score = 0;
   if(checkWordIsValid(word, allowedWords))
   {
      score += getScoreForWord(word, tileDictionary);
   }
   word = word.substring(0, word.length()-1);
   if(word.length()>1)
   {
      score += getScoreForWordAndPrefix(word, tileDictionary,
allowedWords);
   }
   return score;
}
```

Pascal/Delphi

05	1	`var`
		`  again : string;`
		`  num, count : integer;`
		`  prime : boolean;`
		
		`begin`
		`  again := 'y';`
		`  while again = 'y' do`
		`  begin`
		`  write('Enter a number: ');`
		`  readln(num);`
		`  if num > 1 then`
		`  begin`
		`  prime := True;`
		`  for count := 2 to round(sqrt(num)) do`
		`  if num mod count = 0 then`
		`  prime := False;`
		`  if prime = true then`
		`  writeln('Is prime')`
		`  else`
		`  writeln('Is not prime');`
		`  end`
		`  else`
		`  writeln('Not greater than 1');`
		`  write('Again (y or n)? ');`
		`  readln(again);`
		`  end;`
		`  readln;`
		`end.`
			12

07	1	`function CreateTileDictionary() : TTileDictionary;`
		`  var`
		`  TileDictionary : TTileDictionary;`
		`  Count : integer;`
		`  begin`
		`  TileDictionary := TTileDictionary.Create();`
		`  for Count := 0 to 25 do`
		`  begin`
		`  case count of`
		`  0, 4, 8, 13, 14, 17, 18, 19:`
		`TileDictionary.Add(chr(65 + count), 1);`
		`  1, 2, 3, 6, 11, 12, 15, 20:`
		`TileDictionary.Add(chr(65 + count), 2);`
		`  5, 7, 10, 21, 22, 24: TileDictionary.Add(chr(65`
		`+ count), 3);`
		`  **9, 23: TileDictionary.Add(chr(65 + count), 4);**`
		`  else TileDictionary.Add(chr(65 + count), 5);`
		`  end;`
		`  end;`
		`  CreateTileDictionary := TileDictionary;`
		`  end;`
			2

08	1	…
		StartHandSize := 0;
		Choice := '';
		while (StartHandSize < 1) or (StartHandSize > 20) do
		  begin
		  write('Enter start hand size: ');
		  readln(StartHandSize);
		  end;
		…
			4

09	1	function CheckWordIsValid(Word : string; AllowedWords : array of string) : boolean;
		  var
		  ValidWord : boolean;
		  Start, Mid, EndValue : integer;
		  begin
		  ValidWord := False;
		  Start := 0;
		  EndValue := length(AllowedWords) - 1;
		  while (not(ValidWord)) and (Start <= EndValue) do
		  begin
		  Mid := (Start + EndValue) div 2;
		  writeln(AllowedWords[Mid]);
		  if AllowedWords[Mid] = Word then
		  ValidWord := True
		  else if Word > AllowedWords[Mid] then
		  Start := Mid + 1
		  else
		  EndValue := Mid - 1;
		  end;
		  CheckWordIsValid := ValidWord;
		  end;
			8

10	1	**procedure CalculateFrequencies(AllowedWords : array of string);**
		  var
		  Code, LetterCount : integer;
		  LetterToFind, Letter : char;
		  Word : string;
		  begin
		  writeln('Letter frequencies in the allowed words are:');
		  for Code := 0 to 25 do
		  begin
		  LetterCount := 0;
		  LetterToFind := chr(65 + Code);
		  for Word in AllowedWords do
		  begin
		  for Letter in Word do
		  begin
		  if Letter = LetterToFind then
		  LetterCount := LetterCount + 1;
		  end;
		  end;
		  writeln(LetterToFind, ' ', LetterCount);
			8

		end; **end;**	

11	1	**function GetScoreForWordAndPrefix(Word : string; TileDictionary : TTileDictionary; AllowedWords : array of string) : integer;**
		var
		Score : integer;
		begin
		if length(word) <= 1 then
		Score := 0
		else
		begin
		Score := 0;
		if CheckWordIsValid(Word, AllowedWords) then
		Score := Score + GetScoreForWord(Word, TileDictionary);
		Delete(Word,length(Word),1);
		Score := Score + GetScoreForWordAndPrefix(Word, TileDictionary, AllowedWords);
		end;
		GetScoreForWordAndPrefix := Score;
		end;
		
		procedure UpdateAfterAllowedWord(Word : string; var PlayerTiles : string; var PlayerScore : integer; var PlayerTilesPlayed : integer; TileDictionary : TTileDictionary; var AllowedWords : array of string);
		var
		Letter : Char;
		begin
		PlayerTilesPlayed := PlayerTilesPlayed + length(Word);
		for Letter in Word do
		Delete(PlayerTiles,pos(letter, PlayerTiles),1);
		PlayerScore := PlayerScore + GetScoreForWordAndPrefix(Word, TileDictionary, AllowedWords);
		end;
			11