



2.7 Dictionaries Mark Scheme

Mark schemes

Q1.

- (a) (i) **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

Python 2

```
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
```

Python 3

```
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
```

Visual Basic

```
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

```

C#

```

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);
        }
        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);
        }
    }
}

```

Java

```

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;
            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;
}

```

Pascal / Delphi

```

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;

```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

Must match code from part (a)(i), including prompts on screen capture matching those in code.

Code for part (a)(i) 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

(b) (i) **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

Python 2

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

Python 3

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

Visual Basic

```

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

```

C#

```

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

```

Java

```

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

```

Pascal / Delphi

```

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

```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

*Must match code from part (b)(i), including prompts on screen capture matching those in code.
Code for part (b)(i) 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

(c) (i) **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

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 recursive 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

8

Python 2

```
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
```

Python 3

```
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
```

Visual Basic

```
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

```

C#

```

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;
}

```

Java

```

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;
        }
        else
        {
            end = mid - 1;
        }
    }
}

```

```

    }
    return validWord;
}

```

Pascal / Delphi

```

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;

```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

*Must match code from part (c)(i), including prompts on screen capture matching those in code.
Code for part (c)(i) 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", "JAU KING", "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 part (c)(i) 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 part (c)(i) 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 part (c)(i) 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
DONGLES
HAEMAGOGUE
INTERMINGLE
LAGGER
JOULED
ISOCLINAL
JAUING
JACARANDA
JAMBEUX
JAPONICA
JAROVIZE
JASPER
JARTA
JARRAH
JARRINGLY
JARS
```



Valid word **EXAM PAPERS PRACTICE**

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)?
```

>

1

(d) (i) **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

Python 2

```
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 "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        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]))
```

Python 3

```
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 "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        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])
```

Visual Basic

```
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
        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 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        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

```

C#

```

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 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    foreach (var Letter in Alphabet)
    {
        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] );
}
}

```

Java

```

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())
    {
        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]);
    }
}
```

Pascal / Delphi

```
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);
    end;
end;
```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

Must match code from part (d)(i), including prompts on screen capture matching those in code.

Code for part (d)(i) 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
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.
```

>

1

(e) (i) **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 word length is 0 then value of 0 is returned by subroutine and there is no recursive call // if base case word length is 1 then value of 0 is returned by subroutine and there is no recursive call // if base case word length is 2 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;

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

Python 2

```
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

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

```

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

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

```

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

```

C#

```

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

```

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
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

```
function GetScoreForWordAndPrefix(Word : string;
TileDictionary : TileDictionary; 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;
end;

procedure UpdateAfterAllowedWord(Word : string; var
PlayerTiles : string; var PlayerScore : integer; var
PlayerTilesPlayed : integer; TileDictionary : TileDictionary;
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;
```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

Must match code from part (e)(i), including prompts on screen capture matching those in code.

Code for part (e)(i) 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

[37]



Examiner reports

Q1.

- (a) This was the first of the questions that required modifying the Skeleton Program. It was a simple question that over 80% of students were able to answer correctly. When mistakes were made this was normally because tiles other than just J and X were also changed to be worth 4 points.
- (b) Like question (a), this question was normally well-answered with almost all student getting some marks and about 75% obtaining full marks. Where students didn't get full marks this was normally due to the conditions on the loop being incorrect which prevented the values of 1 and / or 20 from being valid.
- (c) For this question students had to replace the linear search algorithm used to check if a word is in the list of allowed words with a binary search algorithm. An example of how a binary search algorithm works was included on the question paper but if a similar question is asked in the future that may not be done. A mixture of iterative and recursive solutions were seen. The most common error made by students who didn't get full marks but made a good attempt at answering the question was to miss out the condition that terminates the loop if it is now known that the word is **not** in the list.
- (d) Students found question (d) easier than questions (c) and (e). Better answers made good use of iteration and arrays / lists, less efficient answers which used 26 variables to store the different letter counts could also get full marks. Some students added code in their new subroutine to read the contents of the text file rather than pass the list as a parameter to the subroutine; this was not necessary but was not penalised.
- (e) Question (e) asked students to create a recursive subroutine. If students answered the question without using recursion they could still get 9 out of the 12 marks available.

It was disappointing that many students did not include any evidence of their attempt to answer the question. Good exam technique would be to include some program code that answers some part or parts of the question. For instance, in question (e) students could get marks for creating a subroutine with the specified name and calling that subroutine – even if the subroutine didn't do anything. There are many examples of subroutines and subroutine calls in the Skeleton Program that students could have used to help them obtain some marks on this question.

A number of very well-written subroutines were seen that made appropriate use of recursion and string handling. Some good recursive answers did not get full marks because they did not include a check that the word / prefix passed as a parameter was valid before the tile points included in the word were used to modify the score, this meant that all prefixes would be included in the score and not just the valid prefixes. Another frequent mistake came when students wrote their own code to calculate the score for a prefix rather than use the existing subroutine included in the Skeleton Program that calculated the score for a word – if done correctly full marks could be obtained by doing this but a number of students made mistakes when writing their own score-calculating code.