1.2 Programming paradigms 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”, “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 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
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)?
>
```


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:
          b>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;

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]

Q2.

(a) **Marks are for AO2 (analyse)**

Feature	Is present in Figure 11? (Yes/No)
Inheritance	No
Protected method	No
Private attribute	Yes

A. alternative indicators instead of Yes/No eg Y/N.

Mark as follows:
One mark per correct row

3

(b) **Mark is for AO2 (analyse)**

```
Rabbit // Fox;
```

R. if spelt incorrectly
R. if any additional code
I. case

1

(c) **Marks are for AO1 (understanding)**

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

A private attribute can only be accessed within its class;
A. private attribute can only be accessed within its file (**Java only**)

2

(d) **1 mark for AO2 (analyse)**

MAX 1 from:

`RabbitCount` (is a private attribute and) is not accessible outside of the `Warren` class;

`GetRabbitCount` (is a public method and) is accessible outside of the `Warren` class;

1 mark for AO1 (understanding)

Means the way `RabbitCount` is represented can be modified without having to change any other objects that interact with `Warren` **NE**. without having to change other code // makes it easier to reuse / inherit from the `Warren` class (as there is a well-defined interface) ;
A. this allows data/properties to be modified in a controlled way

2

(e) **Marks are for AO2 (analyse)**

When a rabbit dies it is replaced by null/none; **A**. when rabbits die they are not removed from the list

`CompressRabbitList` makes sure that the space used for dead rabbits in the list is made available for new rabbits // `CompressRabbitList` makes sure that the fixed size array does not fill up with dead rabbits;

`CompressRabbitList` moves live rabbits to the start of the list

A. `CompressRabbitList` moves null objects / dead rabbits to the end of the list // other sections of the code assume that the live rabbits are in continuous locations in the array (so would not work correctly without a call to `CompressRabbitList`);

Max 2

(f) **Marks are for AO2 (apply)**

```
HDRabbit = Class(Rabbit)
  Private:
    InfectionRate: Real
    Generation: Integer
  Public:
    Procedure Inspect() (Override)
    Function IsInfertile()
    Function GetGeneration()
    Function GetInfectionRate()
End Class
```


Information for examiner:
Accept answers that use different notations, so long as meaning is clear.

Mark as follows:

1 mark: 1. for correct header including name of class and parent class
1 mark: 2. for redefining the `Inspect` method **A**. Override not stated
1 mark: 3. for defining the two additional attributes, with appropriate data types and identified as private **R**. if other attributes included
1 mark: 4. for defining methods needed to read the two additional attributes, and an `IsFertile` method, all identified as being public **R**. if other methods included

I. missing brackets
I. additional Get/Set methods
I. constructor method

A. any suitable alternatives used instead of `Function` or `Procedure` keywords
A. any suitable alternatives for data types eg float or double instead of real
R. do not award mark for declaring new methods if any of the functions have the same name as the variables

4

[14]

Q3.

(a) (i) **Marks are for AO3 (programming)**

1 mark: 1. tests for lower bound and displays error message if below
1 mark: 2. tests for upper bound and displays error message if above
1 mark: 3. Upper bound test uses `LandscapeSize` instead of data value of 14/15 **A**. in use of incorrect condition
1 mark: 4. 1-3 happen repeatedly until valid input (for the upper and lower bounds used in the code provided) and forces re-entry of data each time

A. use of pre or post-conditioned loop

MAX 3 if error message is not `Coordinate is outside of landscape, please try again` **A**. minor typos in error message **I**. case **I**. spacing **I**. minor punctuation differences

MAX 2 if new code has been added to `Simulation` constructor instead of `InputCoordinate` method

4

VB.NET

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


Alternative answer

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

PYTHON 2

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

PYTHON 3

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

C#

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

PASCAL

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

JAVA

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


(ii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****

Must match code from part (a)(i), including error message. Code for part (a)(i) must be sensible.

1 mark: Screen capture(s) showing the required sequence of inputs `(-1, 15, 0)`, the correct error message being displayed for `-1` and `15`, and that `0` has been accepted as the program has displayed the prompt for the y coordinate to be input.

```
Select option: 3
  Input x coordinate: -1
Coordinate is outside of landscape, please try again.
  Input x coordinate: 15
Coordinate is outside of landscape, please try again.
  Input x coordinate: 0
  Input y coordinate: _
```

A. alternative error messages if match code for part (a)(i)

1

(b) (i) **Marks are for AO3 (programming)**

1 mark: New subroutine created, with correct name, that overrides the subroutine in the `Animal` class
I. private, protected, public modifiers

1 mark: 2. `CalculateNewAge` subroutine in `Animal` class is always called
1 mark: 3. Check made on gender of rabbit, and calculations done differently for each gender
I. incorrect calculations

1 mark: 4. Probability of death by other causes calculated correctly for male rabbits
1 mark: 5. Probability of death by other causes calculated correctly for female rabbits

5

VB.NET

```
Public Overrides Sub CalculateNewAge()
  MyBase.CalculateNewAge()
  If Gender = Genders.Male Then
    ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses * 1.5
  Else
    If Age >= 2 Then
      ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses + 0.05
    End If
  End If
End Sub
```

A. `If Age > 1 Then` **instead of** `If Age >= 2 Then`

PYTHON 2

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

PYTHON 3

```
def CalculateNewAge(self):
  super(Rabbit, self).CalculateNewAge()
  if self.__Gender == Genders.Male:
    self._ProbabilityOfDeathOtherCauses =
self._ProbabilityOfDeathOtherCauses * 1.5
```

```
  else:
    if self._Age >= 2:
      self._ProbabilityOfDeathOtherCauses =
self._ProbabilityOfDeathOtherCauses + 0.05
```

C#

```
public override void CalculateNewAge()
{
  base.CalculateNewAge();
  if (Gender == Genders.Male)
  {
    ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses * 1.5;
  }
  else
  {
    if (Age >= 2)
    {
      ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses + 0.5;
    }
  }
}
```

PASCAL

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

JAVA

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


(ii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Must match code from part (b)(i). Code for part (b)(i) must be sensible.

1 mark: Any screen capture(s) showing the correct probability of death by other causes for a male rabbit (0.11 to 2dp) and a female rabbit (0.1);

Example:

```
ID 3 Age 2 LS 4 Pr dth 0.1 Rep rate 1.2 Gender Female
ID 4 Age 2 LS 4 Pr dth 0.11 Rep rate 1.2 Gender Male
```

1

(c) (i) **Marks are for AO3 (programming)**

1 mark: Structure set-up to store the representation of terrain for a location
1 mark: Type of terrain is passed to constructor as parameter
1 mark: Type of terrain stored into attribute by constructor **A**. default value, that makes type of terrain for location clear, instead of value from a parameter

3

VB.NET

```
Class Location
  Public Fox As Fox
  Public Warren As Warren
  Public Terrain As Char

  Public Sub New(ByVal TerrainType As Char)
    Fox = Nothing
    Warren = Nothing
    Terrain = TerrainType

  End Sub
End Class
```

PYTHON 2

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

PYTHON 3

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

C#

```
class Location
{
  public Fox Fox;
  public Warren Warren;
  public char Terrain;

  public Location(char Terraintype)
  {
    Fox = null;
    Warren = null;
    Terrain = Terraintype;
  }
}
```

PASCAL

```
type
  Location = class
    Fox : Fox;
    Warren : Warren;
```

```
    Terrain : char;
    constructor New(TerrainType : char);
  end;

constructor Location.New(TerrainType : char);
  begin
    Fox := nil;
    Warren := nil;
    Terrain := TerrainType;
  end;
```

JAVA

```
class Location
{
  public Fox Fox;
  public Warren Warren;
  public char Terrain;

  public Location(char Terrain)
  {
    Fox = null;
    Warren = null;
    this.Terrain = Terrain;
  }
}
```

(ii) **Marks are for AO3 (programming)**

1 mark: 1. An indicator for type of terrain will be stored for every location
I. wrong type of terrain in a location
R. if indicators other than R or L used
I. case of indicators

1 mark: 2. Vertical river created in column 5
1 mark: 3. Horizontal river created in row 2
MAX 1 FOR 2 & 3 if only creates a river when foxes & warrens are in default locations
MAX 2 if creates any rivers in incorrect locations

3

VB.NET

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

PYTHON 2

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

PYTHON 3

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

C#

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

PASCAL

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

JAVA

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


(iii) **Marks are for AO3 (programming)**

1 mark: R/L, or other indicator as long as it is clear what the type of terrain is, displayed in each location (could be different letters, use of different colours) **A**. type of terrain not displayed if location contains a fox

1 mark: Row containing column indices matches new display of landscape **I**. number of dashes not adjusted to match new width R. if terrain indicators not displayed **A**. no adjustment made if indicators for

terrain used mean no adjustment to width of display for terrain was needed

2

VB.NET

```
Private Sub DrawLandscape()
  Console.WriteLine()
  Console.WriteLine("TIME PERIOD: " & TimePeriod)
  Console.WriteLine()
  Console.Write("  ")
  For x = 0 To LandscapeSize - 1
    Console.Write(" ")
    If x < 10 Then
      Console.Write(" ")
    End If
    Console.Write(x & " |")
  Next
  Console.WriteLine()
  For x = 0 To LandscapeSize * 5 + 3 'CHANGE MADE HERE
    Console.Write("-")
  Next
  Console.WriteLine()
  For y = 0 To LandscapeSize - 1
    If y < 10 Then
      Console.Write(" ")
    End If
    Console.Write(" " & y & "|")
    For x = 0 To LandscapeSize - 1
      If Not Me.Landscape(x, y).Warren Is Nothing Then
        If Me.Landscape(x, y).Warren.GetRabbitCount() < 10
Then
          Console.Write(" ")
        End If
        Console.Write(Landscape(x,
y).Warren.GetRabbitCount())
      Else
       Console.Write(" ")
      End If
      If Not Me.Landscape(x, y).Fox Is Nothing Then
        Console.Write("F")
      Else
        Console.Write(" ")
      End If
      Console.Write(Landscape(x, y).Terrain)
      Console.Write("|")
    Next
    Console.WriteLine()
  Next
End Sub
```

PYTHON 2

```
def __DrawLandscape(self):
  print
  print "TIME PERIOD:", str(self.__TimePeriod)
  print
  sys.stdout.write("  ")
  for x in range (0, self.__LandscapeSize):
    sys.stdout.write(" ")
    if x < 10:
      sys.stdout.write(" ")
    sys.stdout.write(str(x) + " |")
  print
  for x in range (0, self.__LandscapeSize * 5 + 3): #CHANGED
```

```
4 TO 5
    sys.stdout.write("-")
  print
  for y in range (0, self.__LandscapeSize):
    if y < 10:
      sys.stdout.write(" ")
    sys.stdout.write(str(y) + "|")
    for x in range (0, self.__LandscapeSize):
      if not self.__Landscape[x][y].Warren is None:
        if self.__Landscape[x][y].Warren.GetRabbitCount() <
10:
          sys.stdout.write(" ")

sys.stdout.write(self.__Landscape[x][y].Warren.GetRabbitCou
nt())
      else:
        sys.stdout.write(" ")
      if not self.__Landscape[x][y].Fox is None:
        sys.stdout.write("F")
      else:
        sys.stdout.write(" ")
      sys.stdout.write(self.__Landscape[x][y].Terrain)
      sys.stdout.write("|")
    print
```

PYTHON 3

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

C#

```
private void DrawLandscape()
```

```
{
  Console.WriteLine();
  Console.WriteLine("TIME PERIOD: "+TimePeriod);
  Console.WriteLine();
  Console.Write("  ");
  for (int x = 0; x < LandscapeSize; x++)
  {
    Console.Write(" ");
    if (x < 10) { Console.Write(" "); }
    Console.Write(x + " |");
  }
  Console.WriteLine();
  for (int x = 0; x <= LandscapeSize * 5 + 3; x++)
  {
    Console.Write("-");
  }
  Console.WriteLine();
  for (int y = 0; y < LandscapeSize; y++)
  {
    if (y < 10) { Console.Write(" "); }
    Console.Write(" " + y + "|");
    for (int x = 0; x < LandscapeSize; x++)
    {
      if (Landscape[x, y].Warren != null)
      {
        if (Landscape[x, y].Warren.GetRabbitCount() < 10)
        {
          Console.Write(" ");
        }
        Console.Write(Landscape[x,
y].Warren.GetRabbitCount());
      }
      else { Console.Write(" "); }
      if (Landscape[x, y].Fox != null)
      {
        Console.Write("F");
      }
      else
      {
        Console.Write(" ");
      }
      Console.Write(Landscape[x, y].Terrain);
      Console.Write("|");
    }
    Console.WriteLine();
  }
}
```

PASCAL

```
procedure Simulation.DrawLandscape();
  var
    x : integer;
    y : integer;
  begin
    writeln;
    writeln('TIME PERIOD: ', TimePeriod);
    writeln;
    write('  ');
    for x := 0 to LandscapeSize - 1 do
      begin
        write(' ');
        if x < 10 then
          write(' ');
        write(x, ' |');
```

```
      end;
    writeln;
    for x:=0 to LandscapeSize * 5 + 3 do //CHANGE MADE HERE
      write('-');
    writeln;
    for y := 0 to LandscapeSize - 1 do
      begin
        if y < 10 then
          write(' ');
        write(' ', y, '|');
        for x:= 0 to LandscapeSize - 1 do
          begin
            if not(self.Landscape[x][y].Warren = nil) then
              begin
                if
self.Landscape[x][y].Warren.GetRabbitCount() < 10 then
                  write(' ');
                write(Landscape[x][y].Warren.GetRabbitCount
());
              end
            else
              write(' ');
            if not(self.Landscape[x][y].fox = nil) then
              write('F')
            else
              write(' ');
            write(Landscape[x][y].Terrain);
            write('|');
          end;
          writeln;
      end;
  end;
```

JAVA

```
private void DrawLandscape()
{
  Console.println();
  Console.println("TIME PERIOD: " + TimePeriod);
  Console.println();
  Console.print("   ");
  for(int x = 0; x < LandscapeSize; x++)
  {
    Console.print(" ");
    if (x < 10)
    {
      Console.print(" ");
    }
    Console.print(x + " |");
  }
  Console.println();
  for(int x = 0; x < LandscapeSize * 5 + 4; x++) //Change made
here
  {
    Console.print("-");
  }
  Console.println();
  for(int y = 0; y < LandscapeSize; y++)
  {
    if(y < 10 )
    {
      Console.print(" ");
    }
    Console.print(" " + y + "|");
    for(int x = 0; x < LandscapeSize; x++)
```

```
    {
      if ( Landscape[x][y].Warren != null )
      {
        if ( Landscape[x][y].Warren.GetRabbitCount() < 10)
        {
          Console.print(" ");
        }

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

(iv) **Marks are for AO3 (programming)**

1 mark: Warren/fox will not be placed in a river

1 mark: Warren will not be placed where there is a warren // fox will not be placed where there is a fox
R. if no sensible attempt at preventing warren/fox from being placed in a river

1 mark: Fully correct logic in second subroutine

3

VB.NET

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

Private Sub CreateNewFox()
  Dim x As Integer
  Dim y As Integer
  Do
    x = Rnd.Next(0, LandscapeSize)
```

```
    y = Rnd.Next(0, LandscapeSize)
  Loop While Not Landscape(x, y).Fox Is Nothing Or Landscape(x,
y).Terrain = "R"
  If ShowDetail Then
    Console.WriteLine(" New Fox at (" & x & "," & y & ")")
  End If
  Landscape(x, y).Fox = New Fox(Variability)
  FoxCount += 1
End Sub
```

PYTHON 2

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

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

PYTHON 3

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

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

C#

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

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

PASCAL

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

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

```
    inc(FoxCount);
  end;
```

JAVA

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

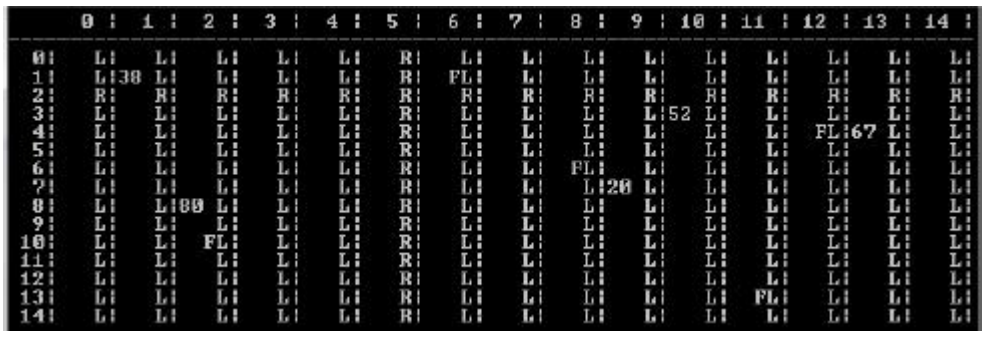

(v) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****

Must match code from part (c)(i) to (c)(iv). Code for these parts must be sensible

1 mark: Screen capture(s) indicating which locations are land and which are rivers

A. incorrect location of rivers if these match those set in parts (c)(ii)



1

(d) (i) **Marks are for AO3 (programming)**

Structure of subroutine:

1. **1 mark:** Subroutine created with correct name `CheckIfPathCrossesRiver` **I**. private/public/protected modifiers
2. **1 mark:** Subroutine has four parameters of appropriate data type, which are the coordinates of the two locations to check the path between **I**. `self` parameter in Python answers **I**. additional parameters
3. **1 mark:** Subroutine returns a Boolean value

Horizontal or vertical:

4. **1 mark:** Repetition structure created that has start and end points that correspond to one coordinate of the locations that need to be checked on the column/row **A**. if start and end points include the columns/rows that contain the fox and warren, even though this is not necessary
5. **1 mark:** Repetition structure will work regardless of whether or not the fox is to the left/right of or above/below the warren (depending on which direction is being checked) **A**. use of separate repetition structures to achieve this
6. **1 mark:** Within repetition structure a check is made of the type of terrain at the appropriate coordinate
7. **1 mark:** If a section of river is detected, subroutine will return true **R**. if subroutine would return true when the path does not cross a river

Other of vertical or horizontal:

8. **1 mark:** Correct cells are checked regardless of whether or not the fox is to the left/right of or above/below the warren **A**. if start and/or end points include the columns/rows that contain the fox and warren
9. **1 mark:** If a river is detected, subroutine will return true; **R**. if subroutine would return true when the path does not cross a river

MAX 7 if 2 and 5 are used instead of checking terrain type
MAX 5 if code does not use each of the relevant coordinates between fox and warren

9

VB.NET

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

```
    End While
  End If
  If FoxY - WarrenY > 0 Then
    yChange = 1
  Else
    yChange = -1
  End If
  If WarrenY <> FoxY Then
    y = WarrenY + yChange
    While y <> FoxY
      If Landscape(FoxX, y).Terrain = "R" Then
        Return True
      End If
      y += yChange
    End While
  End If
  Return False
End Function
```

PYTHON 2

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


PYTHON 3

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

```
    while y != FoxY:
      if self.__Landscape[FoxX][y].Terrain == "R":
        return True
      y += yChange
  return False
```

C#

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

PASCAL

```
function Simulation.CheckIfPathCrossesRiver(FoxX : integer;
Foxy : integer; WarrenX : integer; WarrenY : integer) : boolean;
  var
    xChange : integer;
    yChange : integer;
    x : integer;
    y : integer;
    Answer : boolean;
```

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

JAVA

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

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

(ii) **Marks are for AO3 (programming)**

1 mark: `CheckIfPathCrossesRiver` subroutine is called within the two repetition structures, with the coordinates of the warren and fox as parameters

1 mark: If the subroutine returns true, the fox will not eat any rabbits in the warren, otherwise it will eat rabbits if the warren is near enough

2

VB.NET

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

PYTHON 2

```
def __FoxesEatRabbitsInWarren(self, WarrenX, WarrenY):
  RabbitCountAtStartOfPeriod =
self.__Landscape[WarrenX][WarrenY].Warren.GetRabbitCount()
```

```
  for FoxX in range(0, self.__LandscapeSize):
    for FoxY in range (0, self.__LandscapeSize):
      if not self.__Landscape[FoxX][FoxY].Fox is None:
          if not self.CheckIfPathCrossesRiver(FoxX, FoxY,
WarrenX, WarrenY):  #INDENTATION CHANGED AFTER THIS LINE
      Dist = self.__DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY)
      if Dist <= 3.5:
        PercentToEat = 20
      elif Dist <= 7:
        PercentToEat = 10
      else:
        PercentToEat = 0
      RabbitsToEat = int(round(float(PercentToEat *
RabbitCountAtStartOfPeriod / 100)))
      FoodConsumed =
self.__Landscape[WarrenX][WarrenY].Warren.EatRabbits(Rabbit
sToEat)
      self.__Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsume
d)
      if self.__ShowDetail:
        sys.stdout.write(" " + str(FoodConsumed) + " rabbits
eaten by fox at (" + str(FoxX) + "," + str(FoxY) + ")." + "\n")
```

PYTHON 3

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

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

C#

```
private void FoxesEatRabbitsInWarren(int WarrenX, int
WarrenY)
{
  int FoodConsumed;
  int PercentToEat;
  double Dist;
  int RabbitsToEat;
  int RabbitCountAtStartOfPeriod = Landscape[WarrenX,
WarrenY].Warren.GetRabbitCount();
  for (int FoxX = 0; FoxX < LandscapeSize; FoxX++)
```

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

PASCAL

```
procedure Simulation.FoxesEatRabbitsInWarren(WarrenX :
integer; WarrenY : integer);
  var
    FoodConsumed : integer;
    PercentToEat : integer;
    Dist : double;
    RabbitsToEat : integer;
    RabbitCountAtStartOfPeriod : integer;
    FoxX : integer;
    FoxY : integer;
  begin
    RabbitCountAtStartOfPeriod :=
Landscape[WarrenX][WarrenY].Warren.GetRabbitCount();
    for FoxX := 0 to LandscapeSize - 1 do
      for FoxY := 0 to LandscapeSize - 1 do
        if not(Landscape[FoxX][FoxY].fox = nil) then
          if not(CheckIfPathCrossesRiver(FoxX, Foxy,
WarrenX, WarrenY)) then
            begin
              Dist := DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
              if Dist <= 3.5 then
                PercentToEat := 20
              else if Dist <= 7 then
```

```
              PercentToEat := 10
            else
              PercentToEat := 0;
            RabbitsToEat := round(PercentToEat *
RabbitCountAtStartOfPeriod / 100);
            FoodConsumed :=
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)
;
            Landscape[FoxX][FoxY].fox.GiveFood(FoodConsum
ed);
            if ShowDetail then
              writeln(' ', FoodConsumed, ' rabbits eaten by
fox at (', FoxX, ',', FoxY, ')');
          end;
  end;
```

JAVA

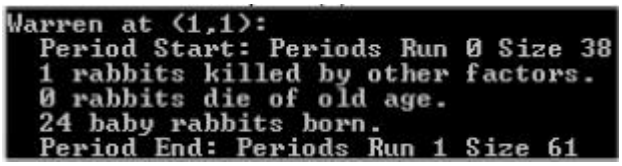
```
private void FoxesEatRabbitsInWarren(int WarrenX, int
WarrenY)
{
  int FoodConsumed;
  int PercentToEat;
  double Dist;
  int RabbitsToEat;
  int RabbitCountAtStartOfPeriod =
Landscape[WarrenX][WarrenY].Warren.GetRabbitCount();
  for(int FoxX = 0; FoxX < LandscapeSize; FoxX++)
  {
    for(int FoxY = 0; FoxY < LandscapeSize; FoxY++)
    {
      if (Landscape[FoxX][FoxY].Fox != null)
      {
        if (!CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY))
        {
          Dist = DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
          if ( Dist <= 3.5 )
          {
            PercentToEat = 20;
          }
          else if ( Dist <= 7 )
          {
            PercentToEat = 10;
          }
          else
          {
            PercentToEat = 0;
          }
          RabbitsToEat =
(int)(Math.round((double)(PercentToEat *
RabbitCountAtStartOfPeriod / 100)));
          FoodConsumed =
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)
;

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

```
            }
        }
    }
}
```

(iii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Must match code from part (d)(i) to (d)(ii). Code for these parts must be sensible

1 mark: Screen capture(s) show that no rabbits are eaten in the warren at (1, 1)



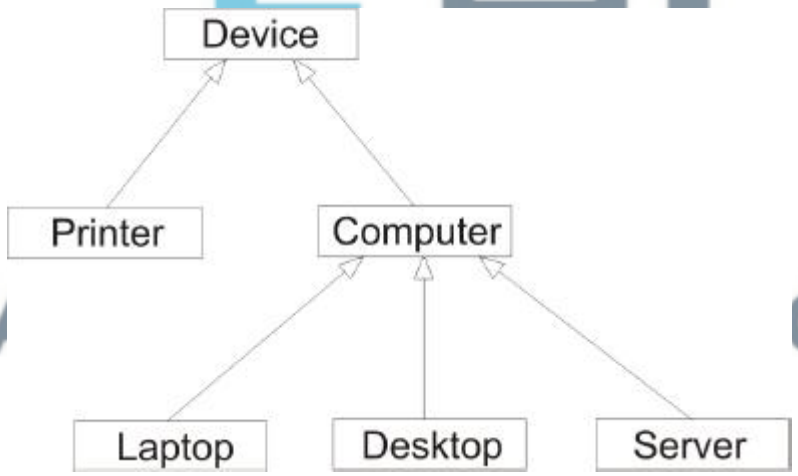
Note: Exact rabbit numbers killed/born do not need to match screenshot, but the start and end periods should be 0 and 1.

1

[35]

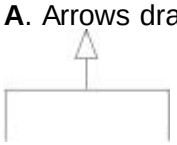
Q4.

(a)



1 mark for Device at top of diagram with Printer and Computer directly underneath it and linked to it and no other labels linked to it;
1 mark for Computer with Laptop, Desktop and Server directly underneath it and linked to it, and no other labels linked to it (except Device above);
1 mark for correctly styled diagram, i.e. lines drawn as arrows and boxes (any shape) around labels; – ***This mark is only available if candidate has already achieved at least one mark for correct contents of the diagram.***

A. Arrows drawn as:



A. Filled/empty arrowheads
A. Diagram rotated by 90 degrees

3

(b)

```
Computer = Class/Subclass/Extends(Device)        1
           (Public)
              Procedure AddDevice (Override)     1
              Function  GetProcessorName
              Function  GetRAMCapacity           1
              Function  GetHDDCapacity
            Private / Protected
              ProcessorName : String
              RAMCapacity : Integer              1
              HDDCapacity : Integer
           End
```

Accept answers that use different notations, so long as meaning is clear.
1 mark for correct header including name of class and parent class;
1 mark for redefining the AddDevice (constructor)procedure;
1 mark* for defining all 3 extra functions needed to read variable values, all identified as being public (keyword public is optional if functions are declared before variables);
1 mark# for defining all 3 extra variables, with appropriate data types and identified as being private;
A. Any sensible numeric types for RAMCapacity and HDDCapacity, do not have to be whole numbers
A. Answers that indicate separately that each variable is private or each method is public
R. Do not award mark for declaring new functions if any of the functions have the same name as the variables
I. Parameters to methods, minor changes to names that do not affect clarity

*. Do not award this mark if any extra functions/procedures have been declared, except for functions that would set values eg SetProcessorName or an incorrectly named procedure to add eg AddComputer
– Do not award this mark if any extra variables have been declared

4

(c)

```
Laptop = Class/Subclass (Computer)               1
         (Public)
            Procedure AddDevice (Override)       1
            Function  GetBluetoothInstalled
          Private / Protected                    1
            BluetoothInstalled : Boolean
         End
```

1 mark for correct header including name of class and parent class;

MAX 1 of the following two marks:
1 mark for redefining the AddDevice procedure;
1 mark* for:

- defining the GetBluetoothInstalled function needed to read this value, identified as being public (keyword public is optional if function is declared before variable)
- defining the BluetoothInstalled variable with an appropriate data type as being private.

A. Boolean or whole number types for BluetoothInstalled but reject string, character or real number types
A. Different sensible name for GetBluetoothInstalled function eg CheckBluetoothInstalled, IsBluetoothInstalled

A. Answers that indicate separately that each variable is private or each method is public
I. Parameters to methods, minor changes to names that do not affect clarity
I. Addition of any extra functions or variables

* Do not award this mark if any extra functions / procedures / variables declared, except for a SetBluetoothInstalled procedure.

2

[9]

Q5.

(a) It hides the detail of how the list will be stored/implemented from the programmer // a programmer working on the rest of the program does not need to know how the `LinkedList` class works // a programmer working on the rest of the program needs only concern themselves with the interface to the `LinkedList` class;
A. "user" for "programmer" as BOD mark

1

(b) The procedures/functions are public as programmer (writing the rest of the program) will need access to the operations defined in the procedures and functions from outside of the class / elsewhere in the program (so they must be public); **A.** just one of procedures or functions **A.** Procedures/functions will be accessible
The data items are private to prevent them being changed directly from outside of the class // to avoid the integrity of the data structure being damaged / changed accidentally (from outside the class); **A.** "elsewhere in program" for "outside of the class"
So that the implementation of `LinkedList` can be changed and programs written using only the public functions and procedures will still work;
MAX 2

2

(c) **OVERALL GUIDANCE:**

Solutions should be marked on this basis:

- Up to **5 marks** for correctly locating the position to delete the item from.
- Up to **3 marks** for deleting the item and updating pointers as required.

The addition of any unnecessary steps that do not stop the algorithm working should not result in a reduction in marks.

Responses should be accepted in pseudo-code or structured English but not in prose.

If you are unsure about the correctness of a solution please refer it to a team leader.

SPECIFIC MARKING POINTS:

Correctly locating deletion point (5 marks):

1. Initialising `Current` to `Start` before any loop;
2. Use of loop to attempt to move through list (regardless of correct terminating condition);
3. Advancing `Current` within loop;
4. Correctly maintaining the `Previous` pointer within loop;
5. Sensible condition to identify position to delete from (suitable terminating condition for loop);

Correctly deleting item (3 marks):

6. Update `Next` pointer of node before node to delete to point to node after it;
7. Test if item to delete was first item in list, and if so update `Start` pointer instead of `Next` pointer of node before the one to delete;
8. Release the memory used by the item being deleted back to the operating system;

Mark point 2 should be awarded if, within the loop, `Current` is being changed (even if not correctly changed).

Mark point 4 can be awarded if `Previous` is set to `Current` before `Current` is changed, even if `Current` is not being correctly updated.

Mark point 5 can be awarded if there is a sensible condition, even if `Current` is not correctly updated.

Mark point 6 can be awarded even if the value of `Previous` was not correctly maintained in the loop.

Mark points 6 and 7 can only be awarded if `Current` has not already been released (or attempted to be released).

Mark point 8 should only be awarded if this is done after and a loop to search for the item to delete, regardless of whether or not the correct item would be found or if it is done inside the loop but also within an if statement that correctly identifies the item to delete.

A. Deletion takes place inside of loop if the correct item to delete had been identified with an if statement and the loop will be exited at some point after deletion.
A.. Use of any type of condition controlled loop, as long as logic is correct.
A. Use of alternative variable names and instructions, so long as the meaning is clear.
A. Use of clear indentation to indicate start/end of iteration and selection structures.
A. Responses written in structured English, so long as variable names are used and the descriptions of what will be done are specific.
A. Use of Boolean variable to control loop as long as it is set under the correct conditions and has been initialised.
R. Responses written in prose.

R. Do not award mark points if incorrect variable names have been used, but allow minor misspellings of variable names.

EXAMPLE SOLUTIONS:

The examples below are complete solutions that would achieve full marks. **Refer recursive solutions to Team Leaders.**

Example 1

```
If Start.DataValue = DelItem Then
  Start ← Start.Next
  Release(Start)
Else
  Current ← Start
  Repeat
    Previous ← Current
    Current ← Current.Next
  Until Current.DataValue = DelItem
  Previous.Next ← Current.Next
  Release(Current)
EndIf
```

Example 2

```
Current ← Start
While Current.DataValue DelItem
  Previous ← Current
  Current ← Current.Next
EndWhile
If Current = Start Then
  Start ← Current.Next
Else
  Previous.Next ← Current.Next
EndIf
Release(Current)
```


Example 3

```
If Start.DataValue = DelItem Then
  Start ← Start.Next
  Release(Start)
Else
  Deleted ← False
  Current ← Start
  While Deleted = False
    If Current.DataValue = DelItem Then
      Previous.Next ← Current.Next
      Release(Current)
      Deleted ← True
    Else
      Previous ← Current
      Current ← Current.Next
```

```
    EndIf
  EndWhile
EndIf
```

8

[11]

Q6.

(a) **All marks AO2 (analyse)**

1 mark: The arrow should be pointing towards the base class;
1 mark: There is no class called `Monster` / / it should say `Enemy`, not `Monster;`

2

(b) **Mark is for AO2 (apply)**

VB.Net

```
Dim MyGame As New Game(False) / /
Dim MyGame As New Game(True) / /
Private Player As New Character / /
Private Cavern As New Grid(NSDistance, WEDistance) / /
Private Monster As New Enemy / /
Private Flask As New Item / /
Private Trap1 As New Trap / /
Private Trap2 As New Trap;
```

R If any additional code
R If spelt incorrectly
I Case

1

(c) **Mark is for AO2 (apply)**

VB.Net

```
CavernState;
```

R If any additional code
R If spelt incorrectly
I Case

1

(d) **Mark is for AO2 (apply)**

```
Trap / / Character / / Enemy;
```

A `SleepyEnemy`
R If any additional code
R If spelt incorrectly
I Case

1

(e) **Mark is for AO2 (apply)**

```
Choice / / NoOfCellsEast / / NoOfCellsSouth / / Count / / NSDistance
/ / WEDistance / / Count1 / / Count2;
```

R If any additional code
R If spelt incorrectly
I Case

1

(f) **Mark is for AO2 (apply)**

`Game;`

R If any additional code
R If spelt incorrectly
I Case

1

(g) **Mark is for AO2 (analyse)**

So that a position of (0,0) is rejected / / so that the item can't be in the player's starting position;

1

(h) **Marks are for AO1 (understanding)**

Makes the program code easier to understand;
Makes it easier to update the program;
Makes it easier to change the size of the cavern (in the game);
Max 2 points from the list above

2

(i) **Marks are for AO2 (analyse)**
1 mark: Create a new object (`Trap3`) of class `Trap;`
1 mark: Change the (3rd) `If` statement in the `PlayGame` subroutine by adding conditions to check if the player is in the same cell as `Trap3` and that `Trap3` has not been triggered already;

2

[12]

Q7.

(a) (i) **Marks are for AO3 (programming)**

1 mark: Selection structure with one correct condition;
1 mark: Both conditions correct and correct logical operator(s);
1 mark: Subroutine returns the correct `True / False` value under all conditions;

A New conditions added to existing selection structure

VB.Net

```
Public Function CheckValidMove(ByVal Direction As Char) As
Boolean
  Dim ValidMove As Boolean
  ValidMove = True
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M") Then
    ValidMove = False
  End If
  If Direction = "W" And
Player.GetPosition.NoOfCellsEast = 0 Then
    ValidMove = False
  End If
  Return ValidMove
End Function
```

3

(ii) **Marks are for AO3 (programming)**

1 mark: Selection structure with correct condition added in correct place

in the code;
1 mark: Correct error message displayed which will be displayed when move is invalid, and only when the move is invalid;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net

```
  ...
  ValidMove = CheckValidMove(MoveDirection)
  If Not ValidMove Then
    Console.WriteLine("That is not a valid move, please try
again")
  End If
Loop Until ValidMove
...
```

2

(iii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (a)(i) and (a)(ii), including prompts on screen capture matching those in code. Code for (a)(i) and (a)(ii) must be sensible.

Screen capture(s) showing the error message being displayed after the player tried to move to the west from a cell at the western end of the cavern;

A Alternative output messages if match code for (a)(ii)

1

(b) (i) **Marks are for AO3 (programming)**

1 mark: `SleepyEnemy` class created;
1 mark: Inheritance from `Enemy` class;
1 mark: `MovesTillSleep` property declared;
1 mark: Subroutine `MakeMove` that overrides the one in the base class;
1 mark: `MovesTillSleep` decremented in the `MakeMove` subroutine;
1 mark: Selection structure in `MakeMove` that calls `ChangeSleepStatus` if the value of `MovesTillSleep` is 0; **A** Changing `Awake` property instead of call to `ChangeSleepStatus`
1 mark: Subroutine `ChangeSleepStatus` that overrides the one in the base class;
1 mark: Value of `MovesTillSleep` set to `4` in the `ChangeSleepStatus` subroutine;

I Case of identifiers
A Minor typos in identifiers

VB.Net

```
Class SleepyEnemy
  Inherits Enemy
  Private MovesTillSleep As Integer

  Public Overrides Sub MakeMove(ByVal PlayerPosition As
CellReference)
    MyBase.MakeMove(PlayerPosition)
```

```
    MovesTillSleep = MovesTillSleep - 1
    If MovesTillSleep = 0 Then
      ChangeSleepStatus()
    End If
  End Sub

  Public Overrides Sub ChangeSleepStatus()
    MyBase.ChangeSleepStatus()
    MovesTillSleep = 4
  End Sub
End Class
```

8

(ii) **Marks are for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (b)(i), including prompts on screen capture matching those in code. Code for (b)(i) must be sensible.

1 mark: Screen capture(s) showing the player moving east and then east again at the start of the training game. The monster then wakes up and moves two cells nearer to the player. The player then moves south;

1 mark: The monster moves two cells nearer to the player and then disappears from the cavern display;

2

(c) (i) **Mark is for AO3 (programming)**

Appropriate option added to menu;

VB.Net

```
Public Sub DisplayMoveOptions()
  Console.WriteLine()
  Console.WriteLine("Enter N to move NORTH")
  Console.WriteLine("Enter S to move SOUTH")
  Console.WriteLine("Enter E to move EAST")
  Console.WriteLine("Enter W to move WEST")
  Console.WriteLine("Enter A to shoot an arrow")
  Console.WriteLine("Enter M to return to the Main Menu")
  Console.WriteLine()
End Sub
```


1

(ii) **Marks are for AO3 (programming)**

1 mark: Direction of A is allowed;
1 mark: Direction of A allowed only if player has got an arrow;

Maximum 1 mark: If any other invalid moves would be allowed or any valid moves not allowed

VB.Net

```
Public Function CheckValidMove(ByVal Direction As Char) As
Boolean
  Dim ValidMove As Boolean
  ValidMove = True
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M" Or Direction = "A") Then
    ValidMove = False
  End If
```

```
  If Direction = "A" And Not Player.GetHasArrow Then
    ValidMove = False
  End If
  Return ValidMove
End Function
```

2

(iii) **Marks are for AO3 (programming)**

1 mark: Property `HasArrow` created;
1 mark: `HasArrow` set to `True` when an object is instantiated;
1 mark: Subroutine `GetHasArrow` created;
1 mark: `GetHasArrow` returns the value of `HasArrow`;
1 mark: Subroutine `GetArrowDirection` created;
1 mark: `GetArrowDirection` has an appropriate output message and then gets a value entered by the user;
1 mark: In `GetArrowDirection`, value keeps being obtained from user until it is one of N, S, W or E;
1 mark: `HasArrow` is set to `False` in `GetArrowDirection`;

I Additional output messages
I Case of identifiers
A Minor typos in identifiers

VB.Net

```
Class Character
  Inherits Item
  Private HasArrow As Boolean
  Public Sub MakeMove(ByVal Direction As Char)
    Select Case Direction
    Case "N"
      NoOfCellsSouth = NoOfCellsSouth - 1
    Case "S"
      NoOfCellsSouth = NoOfCellsSouth + 1
    Case "W"
      NoOfCellsEast = NoOfCellsEast - 1
    Case "E"
      NoOfCellsEast = NoOfCellsEast + 1
    End Select
  End Sub

  Public Sub New()
    HasArrow = True
  End Sub

  Public Function GetHasArrow() As Boolean
    Return HasArrow
  End Function

  Public Function GetArrowDirection() As Char
    Dim Direction As Char
    Do
      Console.Write("What direction (E, W, S, N) would you like
to shoot in?")
      Direction = Console.ReadLine
    Loop Until Direction = "E" Or Direction = "W" Or Direction
= "S" Or Direction = "N"
    HasArrow = False
    Return Direction
  End Function
End Class
```

8

(iv) **Marks are for AO3 (programming)**

1 mark: Check for `A` having been entered – added in a sensible place in the code;
1 mark: If `A` was entered there is a call to `GetArrowDirection`;
1 mark: Selection structure that checks if the arrow direction is `N`;
1 mark: Detects if the monster is in any of the cells directly north of the player's current position;
1 mark: If the monster has been hit by an arrow then the correct output message is displayed and the value of `FlaskFound` is set to `True`;
1 mark: The code for moving the player and updating the cavern display is inside an *else* structure (or equivalent) so that this code is not executed if the player chooses to shoot an arrow;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net

```
If MoveDirection  "M" Then
  If MoveDirection = "A" Then
    MoveDirection = Player.GetArrowDirection
    Select MoveDirection
      Case "N"
        If Monster.GetPosition.NoOfCellsSouth
        Console.WriteLine("You have shot the monster and it
cannot stop you finding the flask")
        FlaskFound = True
      End If
    End Select
  Else
    Cavern.PlaceItem(Player.GetPosition, " ")
    Player.MakeMove(MoveDirection)
    Cavern.PlaceItem(Player.GetPosition, "*")
    Cavern.Display(Monster.GetAwake)
    FlaskFound = Player.CheckIfSameCell(Flask.GetPosition)
  End If
  If FlaskFound Then
  ...
```


6

(v) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing the user shooting an arrow northwards at the start of the training game and the message about the monster being shot is displayed;

A Alternative output messages if match code for (c)(iv)

1

(vi) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****

Info for examiner: Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing an arrow being shot, no message about the monster being hit is displayed and then the invalid move message is displayed when the player tries to shoot an arrow for a second time;

1

[35]

Q8.

(a) **Mark is for AO1 (understanding)**
64 / 2^6;

1

(b) **Mark is for AO2 (apply)**
100;

1

(c) **Mark is for AO2 (apply)**
110;
A The response given to question part (b) with 10 added on.

1

(d) **Mark is for AO2 (apply)**
220;
A The response given to question part (c) multiplied by 2.

1

(e) **All marks AO1 (understanding)**
So that source code cannot be accessed by users;
So that it is more convenient for users to run it / / users do not need to have an interpreter;
So that the program will execute more quickly;
Max 2

2

(f) **All marks AO1 (understanding)**
1 mark: Can't know what type of processor will be in user's computer / / Internet users have range of computers / devices with different processors; **A** References to just different types of computer / device rather than specifically processors
1 mark: A compiled program will only execute on a processor of specific type / family / with same instruction set / / A program run using an interpreter can execute on a computer with any type of processor;
R No compiler exists

2

[8]

Q9.

(a) An abstraction / leaving out non-essential details // A representation of reality;

1

(b) **1 mark for how stack works:**
Stack / It is a Last-in-First-Out / LIFO / First-in-Last-Out / FILO (data structure);

1 mark for correspondence with siding (MAX 1):
The last wagon to enter will be the first to leave;
Wagons enter and leave from same end of siding;
Wagons cannot leave siding before wagons that have entered after them;
Note: Responses must refer to both entering and leaving to gain this mark
NE References to "start", "end", "front", "back" of siding, without further clarification, as not clear which end of siding these terms refer to
NE A siding is LIFO − the student must refer to wagon in their answers, for example the last wagon to enter will be the first to leave.

2

(c)

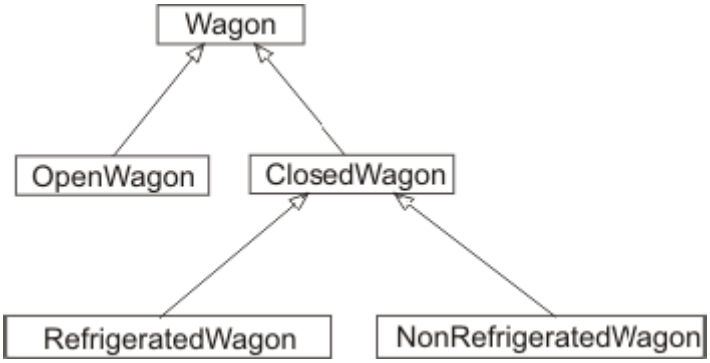
```
If TopOfStackPointer = 0
Then
  Stack Empty Error
Else
  CurrentWagon ← StackArray [TopOfStackPointer]
  Decrement TopOfStackPointer
EndIf
```

1 mark for appropriate `If` structure including condition (does not need both `Then` and `Else`) − Do not award this mark if value is popped off stack outside of `If`.
1 mark for reporting error in correct place
1 mark* for decrementing `TopOfStackPointer`
1 mark* for transferring value from correct position in array into `CurrentWagon` variable

* = if the `CurrentWagon` assignment is performed after the decrement instruction OR the `If` structure then award **MAX 1** of these two marks UNLESS the item is removed from position `TopOfStackPointer+1` so the code would work.
I unnecessary initialisation of any variables
A `Stack Is Empty` for `TopOfStackPointer = 0`
A Logic of `If` structure reversed i.e. `If stack is not empty / TopOfStackPointer>0 / 0 / !=0` and `Then, Else` swapped
A Any type of brackets or reasonable notation for the array index
A Award the mark for dealing with the error situation even if the condition in the `IF` statement is not correct, as long as the purpose of the condition is clearly correct
A Dealing with error in another sensible way eg by setting `CurrentWagon` to `Null`
A Additional lines of code that do not affect behaviour but **MAX 3** if these lines of code would stop the algorithm working correctly
DPT If candidate has used a different name for any variable then do not award first mark but award subsequent marks as if correct name used.

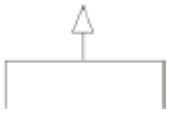
4

(d)



1 mark for Wagon at top of diagram with OpenWagon and ClosedWagon directly underneath it and linked to it and no other labels linked to it;
1 mark for ClosedWagon with RefrigeratedWagon and NonRefrigeratedWagon directly underneath it and linked to it, and no other labels linked to it (except Wagon above);
1 mark for correctly styled diagram, i.e. lines drawn as arrows and boxes (any shape) around labels; - ***This mark is only available if candidate has already achieved at least one mark for correct contents of the diagram.***

A Arrows drawn as:



A Filled / empty arrowheads
A Diagram rotated by 90 degrees

3

(e)

```
ClosedWagon = Class / Subclass / Extends Wagon      1
        (Public)
           Procedure CreateWagon (Override)         1
           Function GetHeight
           Function GetNumberOfDoors                1
           Function GetSuitableForFoodStuffs
         Private / Protected
           Height : Real
           NumberOfDoors : Integer                  1
           SuitableForFoodstuffs : Boolean
        End
```

Accept answers that use different notations, so long as meaning is clear.

1 mark for correct header including name of class and parent class;
1 mark for redefining the `CreateWagon` procedure;
1 mark* for defining all 3 extra functions needed to read variable values, all identified as being public (keyword `public` is optional if functions are declared before variables);
1 mark# for defining all 3 extra variables, with appropriate data types and identified as being private;

A Any sensible numeric types for `Height` and `NumberOfDoors`. `Height` must accept non-integer values and `NumberOfDoors` integer values only.
A Answers that indicate separately that each variable is private or each method is public
R. Do not award mark for declaring new functions if any of the functions have the same name as the variables

I Parameters to methods, minor changes to names that do not affect clarity
* - Do not award this mark if any extra functions / procedures have been declared, EXCEPT for functions that would set values individually e.g. `SetHeight` or an incorrectly named procedure to add e.g. `CreateClosedWagon` which are acceptable for this mark
# - Do not award this mark if any extra variables have been declared

4

[14]

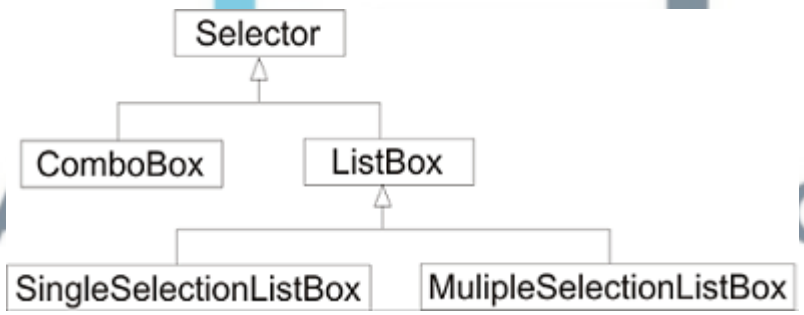
Q10.

(a) A class / subclass has / shares / can access properties and methods of the (parent) class it is derived from;
Building a hierarchy of classes with each child class inheriting access to its parent class' methods and properties;
Relationship between two object (types) in which one object (type) is a kind of the other;
MAX 1

A Just one of properties and methods, do not need both.
A Use of the word "inherits" in the response only if the relationship between parent and subclass is stated explicitly otherwise it is **NE**
A The following as alternatives to properties: fields, attributes, characteristics, data.
A The following as alternatives to methods: procedures, functions, code.
A The following as alternatives to parent: base, super.
A The following as alternative to child: descendent, subclass, derived.

1

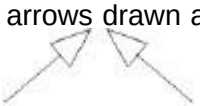
(b)



1 mark for Selector at top of diagram with ComboBox and ListBox directly underneath it and linked to it and no other labels linked to it;
1 mark for ListBox with SingleSelectionListBox and MultipleSelectionListBox directly underneath it and linked to it, and no other labels linked to it (except Selector above);
MAX 1 of the above **2 marks** if any additional links drawn in

1 mark for correctly styled diagram, i.e. lines drawn as arrows and boxes (any shape) around labels; - ***This mark is only available if candidate has already achieved at least 1 mark for correct contents of the diagram.***

A arrows drawn as:



A any type of arrowheads.
A diagram rotated through 90 / 180 / 270 degrees.
A arrows draw wrong way round (but cannot get mark for correctly styled

diagram).
A class diagrams.

3

(c)
```
ComboBox = Class (Selector)
  Public
    Procedure SelectItemFromList
    Procedure Display
    Procedure KeyPressed
    Function  GetTextTyped
    Function  GetSelectedItemNumber
    Procedure SetAllowNonListInputs
  Private
    TextTyped: String
    SelectedItemNumber: Integer
    AllowNonListInputs: Boolean
End
```

Accept answers that use different notations, so long as meaning is clear. Accept any sensible names for subroutines, except SelectItemFromList which must have this name as it overrides a procedure in the parent class.

1 mark for correct header including name of class (ComboBox) and parent class (Selector);
1 mark for overriding the SelectItemFromList procedure (it is not necessary to state that overriding is occurring but must be public);
2 marks for defining all 5 other extra functions / procedures needed, all identified as being public (keyword public is optional if they are declared before variables);; **OR 1 mark** if at least 2 of them defined;
1 mark for defining all 3 extra variables, with appropriate data types and identified as being private;

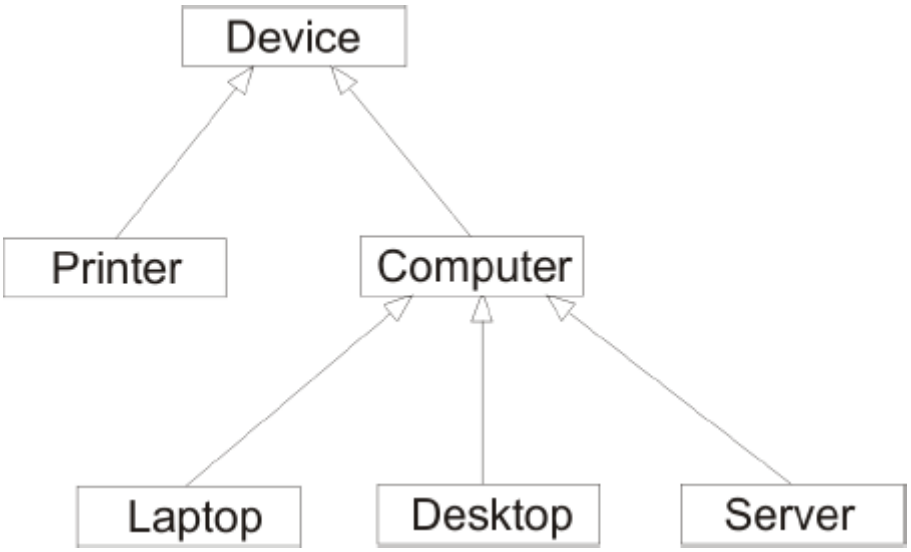
A Array of characters as alternative to string for TextTyped
A Any sensible numeric types for SelectedItemNumber (must be whole numbers)
A Answers that indicate separately that each variable is private or each method is public
A Two procedures instead of one for setting the value of AllowNonListInputs by result, eg, Procedure AllowTextInputs and Procedure OnlyAllowSelection
A Procedure instead of Function and vice-versa
I parameters to methods, minor changes to names that do not affect clarity
R do not award marks for functions / procedures with the same name as variables
DPT if any additional functions / procedures / variables declared do not award the first of the three marks for correctly defining new functions and variables, but award subsequent marks. However, do not penalise answers that include any of the following procedures / functions: `GetAllowNonListInputs`, `SetTextTyped`, `SetSelectedItemNumber`

5

[9]

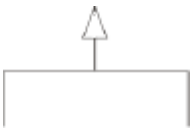
Q11.

(a)



1 mark for Device at top of diagram with Printer and Computer directly underneath it and linked to it and no other labels linked to it;
1 mark for Computer with Laptop, Desktop and Server directly underneath it and linked to it, and no other labels linked to it (except Device above);
1 mark for correctly styled diagram, i.e. lines drawn as arrows and boxes (any shape) around labels; – ***This mark is only available if candidate has already achieved at least one mark for correct contents of the diagram.***

A arrows drawn as:



A filled / empty arrowheads
A diagram rotated by 90 degrees

3

(b)

```
Computer = Class/Subclass/Extends(Device)                    1
           (Public)
               Procedure AddDevice (Override)                1
               Function GetProcessorName
               Function GetRAMCapacity                       1
               Function GetHDDCapacity
           Private / Protected
               ProcessorName : String
               RAMCapacity : Integer                         1
               HDDCapacity : Integer
           End
```

A answers that use different notations, so long as meaning is clear.

1 mark for correct header including name of class and parent class;
1 mark for redefining the AddDevice procedure;
1 mark for* defining all 3 extra functions needed to read variable values, all identified as being public (keyword public is optional if

functions are declared before variables);
1 mark# for defining all 3 extra variables, with appropriate data types and identified as being private;

A any sensible numeric types for RAMCapacity and HDDCapacity, do not have to be whole numbers
A answers that indicate separately that each variable is private or each method is public
R do not award mark for declaring new functions if any of the functions have the same name as the variables
I parameters to methods, minor changes to names that do not affect clarity

** – Do not award this mark if any extra functions / procedures have been declared, except for functions that would set values e.g. SetProcessorName or an incorrectly named procedure to add e.g. AddComputer*
 # – Do not award this mark if any extra variables have been declared

4

(c)

```
Laptop = Class/Subclass (Computer)                    1
         (Public)                                     1
             Procedure AddDevice (Override)
             Function GetBluetoothInstalled           1
         Private / Protected
             BluetoothInstalled : Boolean
         End
```

1 mark for correct header incl
1 mark for redefining the AddDevice procedure;*
1 mark for:*

- defining the GetBluetoothInstalled function needed to read this value, identified as being public (keyword public is optional if function is declared before variable)
- defining the BluetoothInstalled variable with an appropriate data type as being private.

A Boolean or whole number types for BluetoothInstalled but reject string, character or real number types
A Different sensible name for GetBluetoothInstalled function e.g. CheckBluetoothInstalled, IsBluetoothInstalled
A answers that indicate separately that each variable is private or each method is public
I parameters to methods, minor changes to names that do not affect clarity
I addition of any extra functions or variables

** Do not award this mark if any extra functions / procedures / variables declared, except for a SetBluetoothInstalled procedure.*

2

(d) **What** *(2 marks)*:
Wireless/RF (protocol/standard/technology);
For exchanging data over short distances // for creating Personal Area Network;
NE "uses waves" for "wireless"

Example *(1 mark):*
Any sensible example, related to the use of Bluetooth with the laptop e.g. synchronising contacts between phone/ laptop, sending photographs from phone to laptop, Bluetooth mouse, Bluetooth headset / headphones (used with laptop) etc;
NE connecting to wireless network
NE mouse

If the example makes clear that the technology is wireless, but this is not explicitly stated in the "What" part of the response then the "Wireless" mark should be awarded in the "What" part.

3

[12]

Q12.

Meaningful/appropriate/suitable identifiers //
A example;
Indentation // effective use of white space;
Subroutines / Procedures and functions/methods/modules; with interfaces // using parameters to pass values;
Subroutines / Procedures and functions/methods/modules should execute a single task;
Appropriate use of structured statements // use of (selection and repetition)/repetition;
Avoid use of goto statements;
Consistent use of case/style for identifier names;
Use of named constants;
Use of user-defined data types;
Use of libraries;
House-style naming conventions // following conventions;
A by explained example
A Use of local variables
R Commenting
R "easier to understand"
Max 3

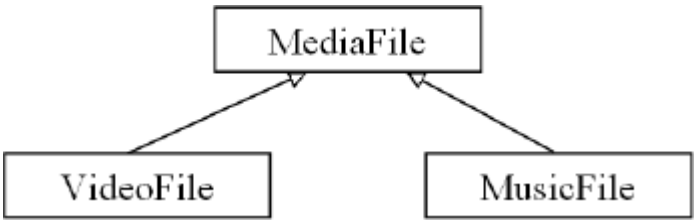
[3]

Q13.

(a) A class/subclass has/shares/inherits properties and methods with the (parent) class (it is derived from);
A another class
Building a hierarchy of classes with each child class inheriting access to its parent class's methods and properties;
Relationship between two object types/objects in which one object (type) is a kind of the other;
A Just one of properties and methods, do not need both.
A The following as alternatives to properties: fields, attributes, characteristics, data with data as BOD
A The following as alternatives to methods: procedures, functions, code.
A The following as alternatives to parent: base, super.
A The following as alternative to child: descendent, subclass, derived.

Max 1

(b)



1 mark for class names in boxes, with MediaFile drawn above the other two;
1 mark for correct arrows;

A arrows drawn as:
A filled/empty arrowheads
A rotated through 90 degrees

2

(c) Method can be defined with same name;
A method can be redefined, an inherited method (but not just inheritance) as implying same name
But have different implementation/code // perform different function;
The redefined method will be used instead of the parent's method;
A This is an example of polymorphism
A Procedure, function, subroutine for method.

2

(d)
```
MusicFile = Class/Subclass (MediaFile)                    1
    Public
        Procedure PlayFile (Override)        Function     1
    GetArtist
        Function GetSampleRate                            1
        Function GetBitDepth
    Private
        Artist : String                                   1
        SampleRate : Real 1
        BitDepth : Integer
    End
```


1 mark for correct header including name of class and parent class;
1 mark for redefining the PlayFile procedure;
1 mark for defining all 3 extra functions needed to read variable values;
1 mark for defining all 3 extra properties, with appropriate data types in private section;

A any numeric types for SampleRate and BitDepth
A answers that indicate separately that each variable is private
DPT if any extra functions/procedures/variables included but do not penalise answers that have extra procedures to set variable values.
DPT if any of the functions/procedures are private
I parameters to methods, minor changes to names that do not affect clarity, case

OR

```
(Public) class/subclass MusicFile extends/inherits
  MediaFile {                                             1
     public void PlayFile (Override)                      1
     public string GetArtist()
```

```
        public float GetSampleRate()                    1
        public int GetBitDepth()
        private string Artist
        private float SampleRate                        1
        private int BitDepth
}
```

1 mark for correct header including name of class and parent class;
1 mark for redefining the PlayFile procedure;
1 mark for defining all 3 extra <u>functions</u> needed to read variable values;
1 mark for defining all 3 extra properties, with appropriate data types as private;

A any numeric types for SampleRate and BitDepth
DPT if any extra functions/procedures/variables included but do not penalise answers that have extra procedures to set variable values.
DPT if any of the functions/procedures are private
I parameters to methods, minor changes to names that do not affect clarity, case
A mixes of two methods if meaning is clear

4

[9]

Q14.

(a)

1 mark for correct boxes
1 mark for correct lines
1 mark for correct line endings

3

(b)
```
Loan = class
                Public
                Procedure CreateLoan
                Procedure DeleteLoan
                Procedure GetLoanDetails;
                Private
                Person: Borrower
                BookLoaned: BookCopy;
                DateOfLoan: Time/Date          A string
                ReturnDate: Time/Date;         A string
    End;
```

1 mark for `Loan = Class + Public + Private + End`
1 mark for `CreateLoan + DeleteLoan + GetLoanDetails`
1 mark for `Person + BookLoaned`
1 mark for `DateOfLoan + ReturnDate`
A any reasonable names for operations and data items.

4

(c) Add a new data item ShortLoan; of type Boolean;
A loanlength; integer;
A loantype; string;

Modify the code for the operations;

Max 2

[9]

Q15.

(a) (i) (User defined) functions // program // object // class // data type // constant // record// label //control/component/ by example e.g. textbox ;

Max 2

(ii) Maximum number of characters ;
No <Space> or other punctuation characters ;
No use of reserved words ;
Must not start with a digit character ;
Case sensitive / permitted case only ;
Cannot define the same identifier name more than once ;
R any reference to filenames

Max 1

(b) Their use matches closely the (modular/structured) design ;
Code can be used 'repeatedly' within the same program ;
Code may originate from a program library/module ;
To make program debugging/testing/maintenance easier ;

Max 1

(c) (i) 10 ;

1

(ii) -1 ;

1

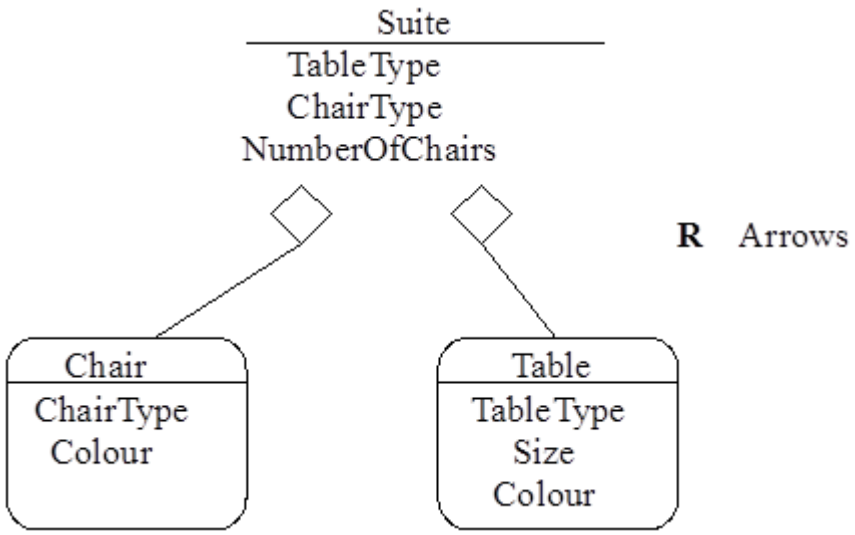
[6]

Q16.

(a) An object that contains other objects;
A a class containing other classes;

1

(b) (i)



R Arrows

1 mark for class entries
1 mark for connections
A circles or diamonds, filled or not

2

(ii)

```
Chair = class;
   Private;
      ChairType : string/text   A integer
      Colour : string/text    A integer/color  } ;

Table = class;
   Private
      TableType : string/text A integer      } ;
      Size : string/text A integer           }
      Colour : string/text A integer/color   }

Suite = class;
  (Public)
     (procedure) DisplayDetails;
   Private
      TableType : Table     }
      ChairType : Chair     } ;
      NumberOfChairs : integer;
```

A any sensible syntax
R implied inheritance

Max 8

[11]

Q17.

(a) *Any three from*
Procedures which have an interface / using parameters to pass values ;
Use of modules / use of libraries ;
Avoid global variables / use of local variables;
Meaningful identifier/variable/constant/ procedure / function / program / parameter names;
Consistent use of case for identifiers ;
Use of selection / loops / iteration ;
Avoid the use of GoTo structures ;
Effective use of white space / indentation;
R spacing/ space out the Code
Use of named constants ;
Use of user-defined data types ;
Use of pseudo-code / top down approach / Jackson methodology / process Decomposition ;
R the use of comments/documentation
R declaration of variables

3

(b) (i)

Surname	String / Text ; **A.** String[n]
NoOfYearsService	Integer /Byte / Int / Short;
PayRate	Single / Real / Float / Currency;

BasicRate	Single/Real/Float / Currency;
AdditionalRate	Single / Real / Float / Currency;

Sensible name + correct data type for single mark

***BUT Penalise once** occurrence of names containing space/other illegal character(s) which would have scored*

Max 3

(ii) 3.1 If NoOfYearsService > 5 ;

1

A >= in the statement **R** =>
A mathematical notation
NoOfYearsService := 5 ;

1

A = or := or ←

3.2 PayRate := 7.88 + NoOfYearsService * 0.65

1

A £ symbol
R use of undefined/unassigned variable(s) in the calculation

A in words 'greater than', 'equals'

3

[9]

Q18.

(a) Mouse click// mouse movement// keyboard operation// any interrupt;

1

(b) Event-driven programs service an event and wait for another; non event-driven programs run to completion/ are sequential;

2

(c) Contains its own data/fields/variables/properties;
Contains its own
Operations/methods/functions/procedures/behaviours/code;
Responds to messages;
A Based on a Class definition

Max 2

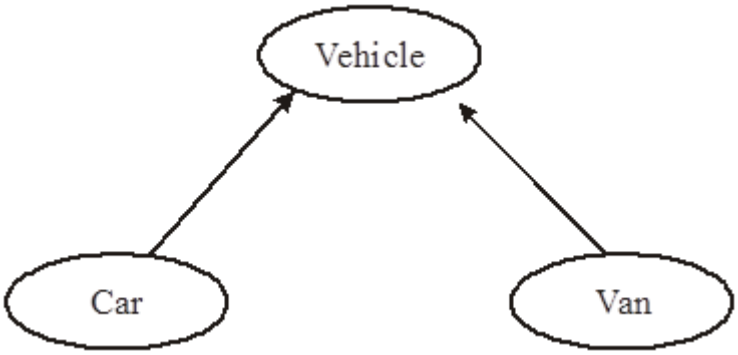
(d) Frame/form/window/button/check box/radio button/menu/text box;
A any sensible widget
R Plurals

1

[6]

Q19.

(a)



1 mark for all three classes in appropriate single enclosures
1 mark for correct independent arrows in correct directions

2

(b) (Insert) a SetColour Procedure;
A Function into the Public section;
R make Colour Public

2

(c) Van = Class/ subclass (Vehicle)ie. Clearly identify Van as a (sub) class of vehicle
1 mark

(Public)

Procedure SetVehicleDetails (Override) condone if not included
1 mark

Function GetCapacity
1 mark

Function GetTailLift
1 mark

(penalise extra functions/procedures once)
Private

Capacity : Integer/real/fixed/float
1 mark

TailLift: Boolean
1 mark

penalise once if not private and once if extra variables listed)

End

A Procedure SetCapacity and Procedure SetTailLift/
AProcedure AddNewVan instead of Procedure SetVehicleDetails

OR

Public class/subclass Van extends/inherits Vehicle
1 mark

{

public void SetVehicleDetails

1 mark

public int GetCapacity
1 mark

public boolean/int GetTailLift
1 mark

private int Capacity
1 mark

private boolean/int TailLift
1 mark

A public void SetCapacity and public void SetTailLift//public void AddNewVan instead of public void SetVehicleDetails
R any diagrams
I any parameters to methods

6

[10]

Q20.

(a) A class has properties/fields/attributes/characteristics and methods/procedures/functions of the parent class it is derived from // a subclass/derived class inherits all the properties/fields/attributes/characteristics and methods/procedures/functions from a super-class/base-class/parent class;

1

(b) StockItem (=) Class ***//*** Class (=) StockItem;

1 mark for keywords Class and StockItem
(***A*** Object instead of Class)

1 { Public (procedure DisplayDetails) (virtual)(virtual;abstract) *1 mark for keyword Public*
(procedure) SetLoan (virtual)(virtual;abstract) *and correct methods*

Private; **A** protected

Title: String
OnLoan: Boolean
DateAcquired: String/DateTime/Date
} 1 **A** text *instead of* string; *1 mark for correct data fields & data types*

End *don't allow the other fields*

Book = Class (StockItem) // Class Book extends/derives from StockItem
// Book Subclass: StockItem;
A without keyword Class

Private

Author: String

ISBN: String

} 1

Public

(Procedure) DisplayDetails (override)

} 1

End

If candidate declared 'getters' and 'setters' for the base class fields then don't have to have DisplayDetails as a base class method

CD = Class (StockItem) // Class CD extends/derives from StockItem // CD Subclass: StockItem;

Private

Artist: String

PlayingTime: Integer/Real/Time/DateTime

} 1

Public

(Procedure) DisplayDetails (override)

End

No marks for a diagrammatic answer.

I method parameters

Java version:

```
Public Class StockItem
{
Private String title;
Private boolean onLoan;
Private String dateAquired;
Public void displayDetails ();
Public void setLoan ();

}
Public Class Book extends StockItem
{
Private string author;
Private string isbn;
Public void displayDetails ();

}
Public Class CD extends StockItem
{
Private string artist;
Private integer playingTime;
Public void displayDetails();
}
```

Max 7

[8]

Q21.

(a) Produces re-useable code because of inheritance/encapsulation;
Produces re-useable objects;
Data is protected // only accessible in well-defined ways (because of encapsulation);
More efficient to write programs which use pre-defined / inherited objects / classes;
Storage structure of data and method code of a class may be altered without affecting programs that make use of the class;
Code produced contains fewer errors / more reliable;
Solutions are easier to understand (when expressed in terms of objects);
Easier to enforce design consistency; easier to debug;
Less maintenance effort required by developer since objects can be re-used;
New functions can be added to objects easily (because of inheritance);
R Easier to program
I references to GUIs

2

(b) *1 mark for correct base class and derived classes incl. containers;*
1 mark for 2 correctly directed arrows;
R E-R diagrams
I methods listed in containers

2

(c) Member = Class
(Public)
(procedure) AddNewMember(s); }
(procedure) AmendMember(s) } ; *no mark if methods are private*
(Procedure) ShowMember(s); }
A proc instead of procedure
R function instead of procedure

Private *(1 mark for all data fields marked as private)*
MembershipNo : Integer }**A** string/text as data type **R** number
FirstName: string/text };
Surname: String/text }
A ID
A FName
A SName
A Tel

TelephoneNumber: string/text :
R number/integer as data type
End (Class)

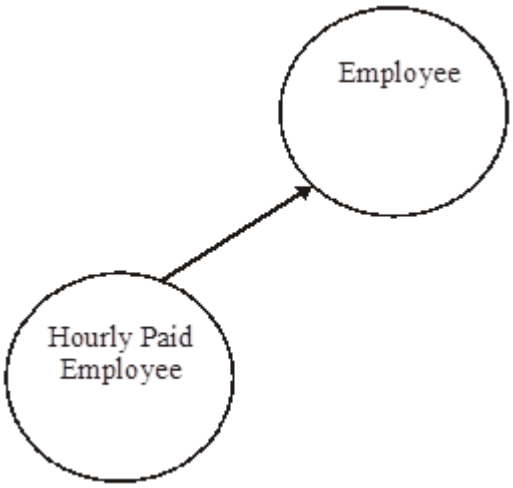
Public may come after Private. Each line may be preceded by Public or Private & in no particular order
R diagrammatic answer
I case
I white space

4

[8]

Q22.

(a)



1 mark if correct hierarchy (including rectangles or round/oval shapes) in an inheritance diagram;
A no shapes this year only
1 mark for arrow in correct direction

2

(b) THourlyPaidEmployee = Class (Employee)
(Public)
procedure CalculatePay (override)
procedure GetNumberOfHoursWorkedInMonth
Private
hourlyrate/hourlypay/HourlyPayRate: Currency
NumberOfHoursWorkedInMonth : Integar/Real/Float
End

OR

public class/subclas THourlyPaidEmployee extends/inherits TEmployee;{(1)
public void calculatePay;(1)
public void getNumberOfHoursWorkedInMonth;(1)
private; float hourlyPayRate;
private int numberOfHoursWorkedInMonth;(1)}

1 mark for private, 1 mark for var name
Accept "Object" instead of "Class"
Accept Public implied
Lose one mark if properties from parent class included
***R** any diagrams*

6

[8]

Q23.

TForm1 = Class(TForm)(1)

Button1:Tbutton;(1)

Button2:Tbutton; (1)

End

NB 1 mark for BOTH buttons

//

```
Class Tform1 extends Tform
        {Tbutton Button1;
        Tbutton Button 2;
        }
```

Must look like code.
*1 mark for connecting TForm1 to Tform **A** inherits, :*
*1 mark for defining both buttons as type Tbutton **A** As*
1 mark for {} or End

[3]

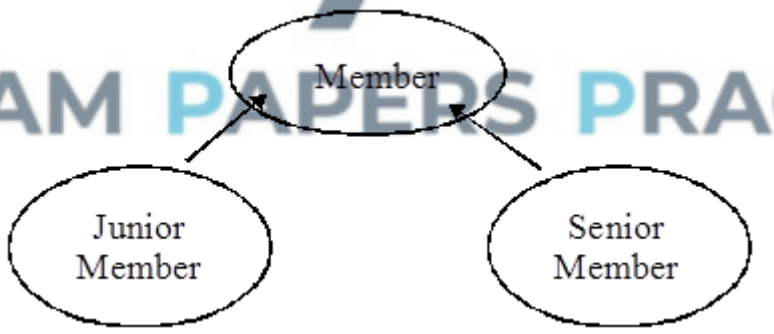
Q24.

(a) Produces re-usable code because of inheritance / encapsulation;
Data is only accessible in well defined ways (because encapsulated);
More efficient to write programs which uses pre-defined / inherited objects / classes;
Storage structure of data and the code in an object may be altered without affecting programs that make use of the object;
Code produced contains fewer errors / more reliable;
Solutions are easier to understand when expressed in terms of objects;
Easier to enforce design consistency – Windows GUI functionality;
Cheaper production costs / Less maintenance effort required by developer since reliable 'objects' can be re-used / bought in;
New functions can be added to objects easily (because encapsulated);

Any 2 advantages × 1 each – must state an advantage, not make a statement.
R Object is independent.

2

(b)



1 mark for correct base class and derived classes ;
1 for 2 correctly directed arrows. ;

2

(c)
```
Member = Class
        (Procedure) AddNewMembers;          }
        (Procedure) AmendMembers;           }      ;
        (Procedure)ShowMembers;             }
        Private                                    ;
            MembershipNo : Integer
            Name : String;
            Address : String;              ;;
        End;
```

Exact syntax not required, but must be in style of . 3 procedures (1)
Private (1)
All 3 field (property) names (1)
3 reasonable data types (1)

4

[8]

Q25.

(a) (i) **A class**

A grouping of data structures and behaviours / methods / procedures / functions; A set of objects / object type which share a common data structure and common behaviour / methods / procedures / functions;

Need both structure and behaviour
A variables, attributes

1

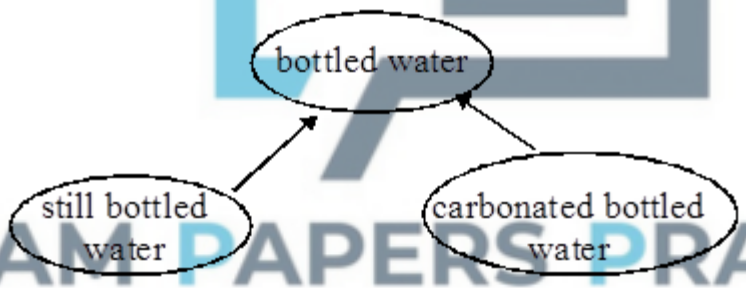
(ii) **Inheritance**

Relationship among classes wherein one class shares the data structure and behaviour/methods / procedures / functions / actions of another class

OR when a class has the same characteristics as its parent class
A attributes / features / properties

1

(b)



1 mark for correct base class
1 mark for two correct derived classes
*1 mark for **2** correctly directed arrows*

3

(c) **Advantages**

Produces re-usable components (you do not have to know how they are written);
A code
Data is protected – only accessible in well defined ways;
Easier to write programs which use pre-defined objects / classes;
Storage structures of data of an object may be altered without affecting programs that make use of the object;
Code of an object may be altered without affecting programs that make use of the object;
Solutions that use objects tend to contain fewer errors / more reliable;
Solutions are easier to understand when expressed in terms of objects;
Easier to enforce design consistency;
Cheaper production costs when software can be re=used;
Less maintenance effort required by developer since reliable 'objects' can be

bought in;
New functions / features can be added to objects / classes easily (inheritance);

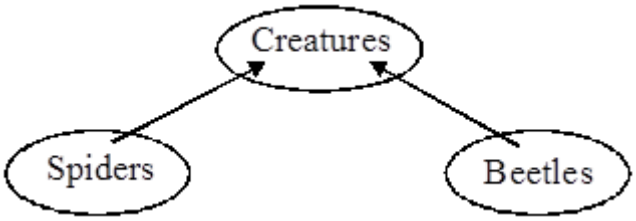
1 mark for each of 3 points

3

[8]

Q26.

(a) Diagram



2

(b) *1 for position, 1 for correct arrows.*
Circles not necessary
OK if completely upside down.
e.g. number of legs, colour, web type;

Property Method
Spin web/eat;
R instance of property, e.g. 8 legs

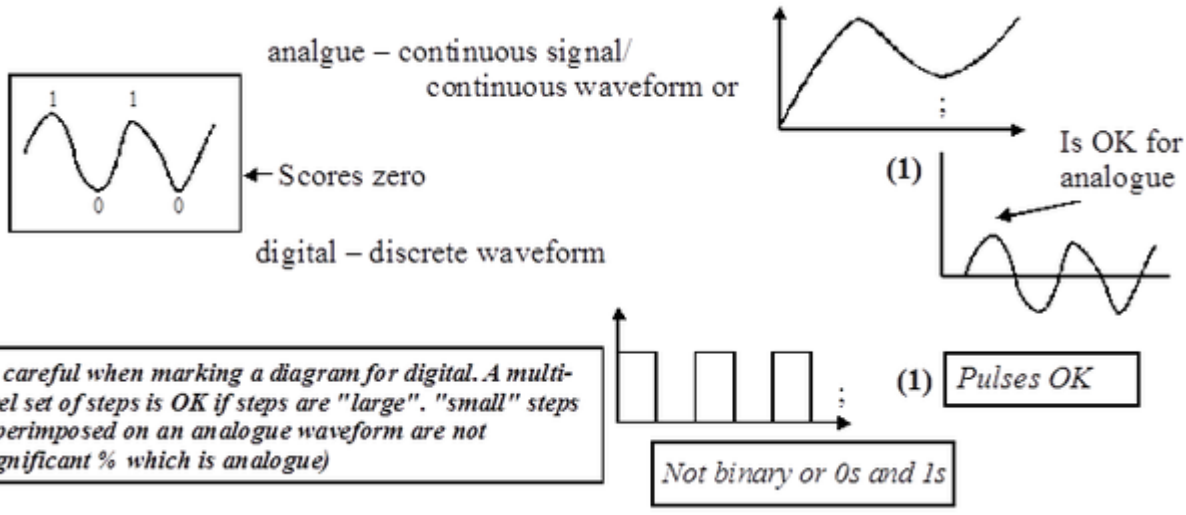
Does **NOT** have to be biologically correct but sensible!

2

[4]

Q27.

(a)



2

(b) (i) A class is a set of objects that share a common structure and a common behaviour;
A class is a set/collection of objects with same attributes/properties/characteristics/fields & methods (accept procedures

or functions for methods)
/behaviours/operations/code;

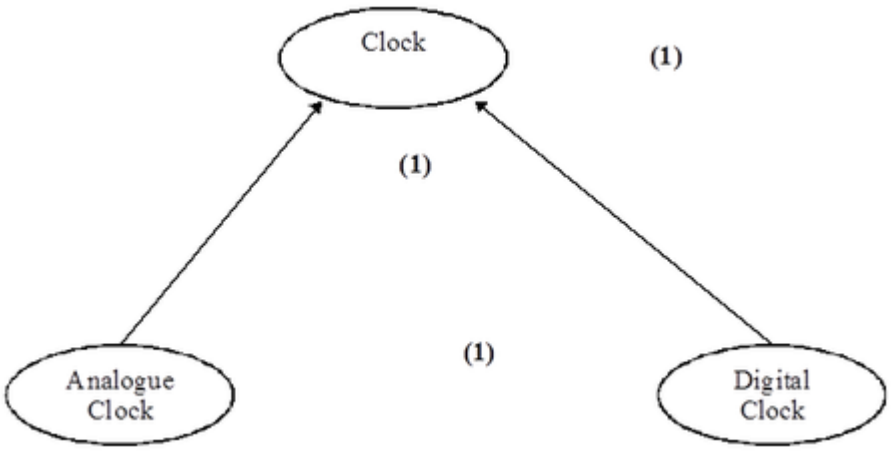
NOT set of objects with same data

1

(ii) Inheritance is a relationship/link among classes wherein one class shares the structure and behaviour of another class;
It is where one class is derived from another class.
It is where one class uses attributes/properties/etc/ from another class;
It is where one class uses methods/procedures/etc from another class;
It is where one class inherits from a parent class(hierarchy must be clear

1

(c)



1 mark for clock in root position. 1 mark for both Analogue and Digital clocks in leaf positions.
1 mark for correct arrow-headed lines.
Must be correctly vertically aligned for these two marks

3

[7]

Q28.

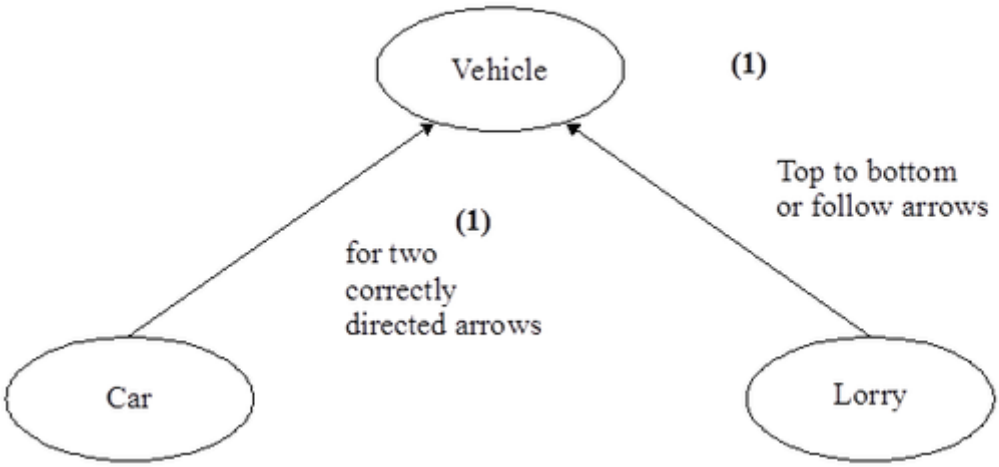
(a) (i) A class is a set of objects which share a common structure and a common behaviour / Object type that defines a data structure /fields /properties and the methods /procedures /functions that act on these fields

1

(ii) Inheritance is a relationship among classes wherein one class shares the structure/data structure /fields /properties and behaviour/methods/procedures/functions/actions of another class
Or
Inheritance is when a class has the same characteristics as its parent class

1

(b)



(1)

Top to bottom
or follow arrows

(1)
for two
correctly
directed arrows

(1)
for both

3

[5]

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.

Q2.

Almost all students obtained some marks on question 8 though very few got full marks. In

question 8.1 the most common error was to state that there was a protected method present in Figure 11. Most students got the mark for 8.2 with `Warren` being the most frequently seen incorrect answer.

For question 8.3 the concept of a private attribute was better understood than a protected attribute. Many students though that a protected attribute was an attribute that could not be changed. Students who did well on the exam paper overall normally had no issues answering 8.4 but students with less understanding of the code in the Skeleton Program often gave answers that explained why knowing the number of rabbits in a warren was useful instead of answering the question set.

Question 8.5 was not well answered with many students writing about the functionality of the program as a whole rather than the `CompressRabbitList` method. Answers often described rabbits being killed by other factors even though this was not done by this method.

Most students were able to get some marks for writing the class definition in question 8.6. The most common errors were to have `HDRabbit` inheriting from `Animal` instead of Rabbit, including the gender attribute in the `HDRabbit` definition and not overriding the `Inspect` method.

Q3.

(a) This was, for most students, the easiest of the programming questions on the paper with about half obtaining full marks. Less confident programmers often had the wrong logic in their conditions (either getting `AND/OR` mixed-up or `</>`). Some students did not write code to get the validation condition to continually repeat until a valid value was entered. A significant minority of students did not add the validation routine to the `InputCoordinate` routine and instead tried to add it the constructor for the `Simulation` class.

Some students used recursion instead of iteration and full marks could be obtained from using this method if it was done correctly however many of these students did not return the value from the recursive call to the calling routine in a way that it could then be used by the rest of the program.

(b) The majority of students were able to get at least half the marks on this question and were clearly familiar with how to create a method that overrides a method in a base class in the programming language they were using. A significant minority of students did not attempt this question and had clearly not prepared for answering questions using OOP within the Skeleton Program.

A number of students did not identify the correct variable to use and wrote code that tried to change the default probability instead of the protected attribute inherited from the `Animal` class storing the probability for that animal.

Some students did not call the overridden method in the base class even though the question specified this should be done. The equivalent functionality could be obtained by copying the code in the `CalculateNewAge` method in the `Animal` class into the new `CalculateNewAge` method in the `Rabbit` class but this is poor programming practice as the original code would now be in two places in the program rather than reusing the existing code.

(c) One fifth of students did not provide any evidence of their attempt to answer this question. All students should be encouraged to include any program code they have written as it may be worth some marks even if it doesn't work correctly.

The most common mistake in reasonable attempts at the tasks in this question was to have the incorrect logic (for example, getting muddled between `AND/OR`) when writing the code to prevent a warren/fox being placed in a river.

(d) Many students came up with creative answers to this question that showed a high-level of programming and problem-solving skill. However, a large number of students did not include any evidence of their attempt at writing the program code. Some students showed good exam technique by including a very limited answer which they knew was nowhere near correct but would allow them to get some marks (most frequently for creating a new subroutine with the name specified in the question).

The most challenging part of the question was to make sure that the solution worked irrespective of the relative position of the fox and the warren with a number of solutions working if the fox was to the left of and above the warren but not if it was to the right of and below the warren.

Q5.

This question was about abstraction, object-oriented programming and linked lists.

For part (a) candidates had to explain how the LinkedList class was a form of abstraction. Many gave a definition of abstraction but failed to apply this to the LinkedList class and so did not achieve a mark. Good responses made clear that the LinkedList class was an example of abstraction because it allowed a programmer to manipulate items in a linked list without having to be concerned about how the linked list was implemented.

For part (b) candidates had to explain why the functions and procedures in the class were public whilst the data items were not. Many candidates were able to obtain a mark for the former, but few did so for the latter. Good responses made clear that the functions and procedures were public as they would need to be called from outside of the class to implement the game, and the data items were private so that their values could only be modified in a controlled way from outside of the class, by calling the procedures of the class. It was not sufficient to state that the data items were private because they were only used by the class or because they should not be changed.

Candidates had to write an algorithm for deleting an item from a linked list for part (c). A question was asked in a previous year about inserting an item into a linked list and the standard of responses to this question was notably better than was the case in the previous year. The majority of candidates had at least a good attempt at writing the part of the algorithm that would find the correct item to delete and many were then able to change the pointers to delete the item. Common mistakes and omissions were to fail to keep track of the pointer to the previous item when searching, to release the item to delete back to the heap before changing the pointer around it or to increase the current pointer by the fixed value of 1 on each iteration of a search loop. Few candidates scored all eight marks. If a candidate achieved seven but not eight marks this was usually because the algorithm did not take account of the fact that the item to delete might be the first item in the list, in which case the start pointer would need to be changed.

Q9.

For question part (a) students had to explain what a model was. Good responses explained that, in the context of simulation, a model was an abstracted representation of reality. Common mistakes were to explain what a physical model was and to confuse a model with a prototype.

For part (b) students had to explain why a queue was an appropriate data structure to

represent a siding. Most students correctly explained that a stack was a first in last out structure, which was worth one mark. Fewer went on to successfully explain how this corresponded to the organisation of a siding. Students occasionally lost marks by using terms such as “in front of” in relation to the wagons, when it was not clear which end of a siding this related to.

For part (c) students had to write an algorithm for popping an item off a stack. A good range of responses was seen, with approximately half of students achieving at least two marks and a quarter achieving all four marks. The error that had to be dealt with was a potentially empty stack. Appropriate methods of dealing with this included displaying an error message or returning a rogue value. Some students made the mistake of using the pop operation within the algorithm that was supposed to define it.

This question part (d), drawing an inheritance diagram, was very well answered, with almost all students getting two marks and over half achieving all three. The most common mistake was to represent the relationships between the classes correctly but to fail to style the diagram appropriately.

For part (e) students had to define a class. This was well answered, with over half of students achieving at least three of the four marks. It is clear that students’ understanding of this topic has improved significantly over the last few years. The two most frequently made errors were to fail to express the relationship between the ClosedWagon and Wagon classes and to forget to override the CreateWagon procedure.

Q10.

(a) This part asked students to explain what inheritance was. Just under two thirds of students were able to do this. Those students who failed to achieve the mark usually did not make clear that the relationship was between a parent class and a sub class or, if they did this correctly, failed to explain the nature of the relationship, ie that the sub class could share some of the methods or properties of the parent.

(b) Students had to draw an inheritance diagram in this part. Almost 90% of candidates achieved at least two of the three available marks, but only half of the candidates achieved full marks. The failure to achieve the third mark was usually because a candidate failed to style the diagram correctly, either not drawing arrowheads, drawing them at the wrong end of the lines, or not enclosing the names of the classes in some type of box.

(c) A very good range of responses was made to this part. The most common mistakes that students made were to fail to identify that the new class was a sub class of the Selector class, to fail to redefine the SelectItemFromList procedure so that it was overridden, to add in extra unnecessary functions and procedures, and to redefine the data items from the parent class.

Q11.

Part (a): Students were required to draw an inheritance diagram. Most students scored two of the available three marks which were for identifying correctly the class hierarchy. The third mark was for drawing a correctly styled diagram and many students failed to do this. Students who did achieve the third mark correctly enclosed the class names and also drew arrows that pointed upwards to a class’ parent class.

Part (b): In this question part students had to write a class definition for the Computer class. Most students had a reasonable understanding of how to do this, with almost all achieving some marks, but less than a fifth scored full marks. To achieve all four marks students needed: to make clear that the class inherited from the Device class, to redefine

the AddDevice procedure, to declare private variables to store the additional properties, and to declare public functions to provide access to the values in these variables. The most commonly made mistakes were to fail to make the inheritance clear and to forget to redefine the AddDevice procedure. Some students lost marks by unnecessarily redeclaring the functions or variable from the parent class or by giving the functions the same names as the variables.

Part (c): The purpose of this question was to test if students understood that the Laptop class inherited from the Computer class, rather than the Device class. The vast majority of students who dealt with inheritance in this question part correctly identified this.

Part (d): Most students were able to identify that Bluetooth is a wireless protocol. Many, but not all of these, then went on to explain that it was designed for use over short distances. Many good examples, such as transferring photos from a mobile phone to a laptop or using a Bluetooth mouse were given. Some students lost marks by giving an example that was not in context.

Q12.

This was a straight-forward question. Most candidates got good marks on it although a surprising number of candidates gave incorrect answers.

Q13.

Part (a): Most candidates were able to explain what inheritance meant. Marks were sometimes lost because candidates wrote about a child class sharing 'things' with its parent, rather than methods or properties.

Part (b): The inheritance diagram was poorly drawn, surprisingly. Many candidates failed to put boxes around the class names or to put arrowheads on the lines and consequently lost marks.
Candidates need to remember to do both of these things.

Part (c): Answers to this question part were quite mixed. Overriding is when a method from a parent class is redefined with the same name by a child class to perform a different function.

Part (d): Most candidates got some marks for this question part but full mark responses were seen very infrequently. Common mistakes were to incorrectly redeclare the fields and methods from the parent class and to forget to override the PlayFile function.

Q14.

The class diagram was a challenge to most candidates. Most candidates assumed that it was an inheritance relationship. As a result they often lost marks later in the question as well. Class definitions were given in this question and this should have allowed candidates to score very highly in part (b). This was not the case as many candidates did not read the question carefully enough. Very few candidates realised that Person and BookLoaned needed to be of types Borrower and BookCopy respectively. While many candidates realised that the solution to part (c) was to add an additional variable, many candidates seemed to think that an extra procedure was required.

Q15.

(a) (i) Well answered with the most popular answers being constants and functions.

(ii) Many candidates then misunderstood what was wanted here and proceeded

to give answers which generally described how programs were constructed with loops, selection statements, etc.

Due to the range of differences with different languages, a wide range of answers were considered acceptable; the most popular being 'it must not contain any <Space> characters' and 'the use of reserved words is not permitted'. Some candidates confused what is allowed in a programming language with what is permitted by the operating system, proceeding to explain what was not allowed for filenames. Worse, was the suggested answer that 'names must be more than 6 characters long' which suggested that the rules about the choice of passwords were being described.

(b) No great detail was expected for the mark and most candidates were able to give an answer which mapped to those on the mark scheme. Use of language was an issue for some candidates who described 'chunks of program code'! There were also answers which clearly were answering 'last year's question' suggesting procedures may or may not return values, contrasting with functions which always return a value.

(c) This was similar to questions which have previously been set and was well answered.

Q16.

Very few candidates scored well on this question. Aggregation was a mystery to most candidates. Many answers described inheritance and then went on to give class definitions using inheritance. Hardly any candidates were able to draw a class diagram using the correct symbols. It was disappointing to see the number of answers that suggested that the candidates had not had experience of object oriented programming. The concepts of private and public were often misused and incorrect data types were often suggested.

Q17.

(a) Despite an extensive (and perhaps 'generous') mark scheme list it was rare for candidates to score more than 1 mark, and this was usually for a "selection/iteration" answer.

(b) (i) Candidates often failed to score three easy marks. The inclusion of <Space> or other illegal characters used in the identifier names was penalised once only. The other common error was the suggestion of incorrect data types, the most common being 'Number' and 'Decimal'. However, this was answered significantly better than on previous papers.

(ii) Despite a question of this type not having been set previously, it was clear from answers seen that candidates knew what was required. The most common error was simply not to make the connection between part (b)(i) and (b)(ii); for example, by introducing new identifiers to answer (ii) which gained no credit.

Q18.

It is hard to understand why so many candidates were unable to gain good marks for this question since all will be very familiar with GUIs. Many candidates used poor language and were unable to express themselves but there is a greater worry that candidates do not seem to understand how the underlying object-oriented system operates. It was common misconception that an event is something the user does. There were also many

candidates who did not seem to understand the event handling mechanism. Once again many marks were lost due to poor explanations in part (b). Part (c) showed that many candidates do not understand object-oriented programming and it appeared that they had not been exposed to this technique.

Q19.

The inheritance diagram for part (a) returned two very easy marks for a large majority of candidates. The question then got harder but, as a whole, discriminated very well, indeed. In part (b) the examiners expected that the candidates would state that a public procedure should be inserted to set the colour. Where a candidate wrote of a function instead of a procedure the answer was condoned.

It was hoped that candidates would apply the concept of inheritance in part (c) and introduce functions GetCapacity and GetTailLift with an override of the procedure SetVehicleDetails. The candidates were also expected to choose suitable data types for the two new data fields. Since only six marks were available for the class definition as a whole it was marked quite leniently with regard to the use of Get and Set in the function names. In future examinations, however, similar questions may well be assessed more strictly. Even with the generous mark scheme only a few candidates obtained all six marks in part (c).

Q20.

(a) Not many candidates could explain that inheritance is when a class gains the methods and properties of another class, while having some of its own methods and/or properties. Many candidates mixed up the terms object and class indiscriminately.

(b) Many candidates seemed to lack the knowledge and skill to structure a class definition, even though this topic has been examined many times before.

An example of an answer worthy of full marks is:

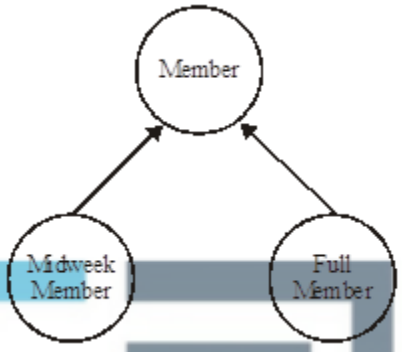
```
Type StockItem = Class
                 Public
                      Procedure DisplayDetails : virtual;
                      Procedure SetLoan;
                 Private
                      Title: String;
                      OnLoan: Boolean;
                      DateAcquired: Date;
                 End;
Type Book = Class (StockItem)
            Public
              Procedure DisplayDetails: override;
            Private
              Author: String;
              ISBN: String;
            End;
Type CD = Class (StockItem)
            Public
              Procedure DisplayDetails: override;
            Private
              Artist: String;
              PlayingTime: Integer;
            End;
```

Syntax of other languages such as Java was just as acceptable.

Q21.

In part (a) few candidates could give advantages of the object-oriented approach to programming. Many candidates are under the false impression that object-oriented programming means using environments such as Delphi or Visual Basic where beginners can get a working program with a nice user interface without much programming of their own. Few candidates also noticed that OOP should be compared to the structured approach. So re-useable code is not sufficient, since that is possible in the structured approach also. OOP produces re-useable objects, however, is a response worthy of a mark. Other acceptable advantages were: data is accessible in well-defined ways because of encapsulation; it is time-saving to write programs which use pre-defined objects; solutions are easier to understand when expressed in terms of objects.

For part (b), the drawing of inheritance diagrams have been required for several years now and candidates are expected to draw these in standard format:



It is important to convey the hierarchy (the super class above the sub-class) and draw the arrow from the sub-class to the super class. Classes should be shown in circles, rectangles or oval shapes.

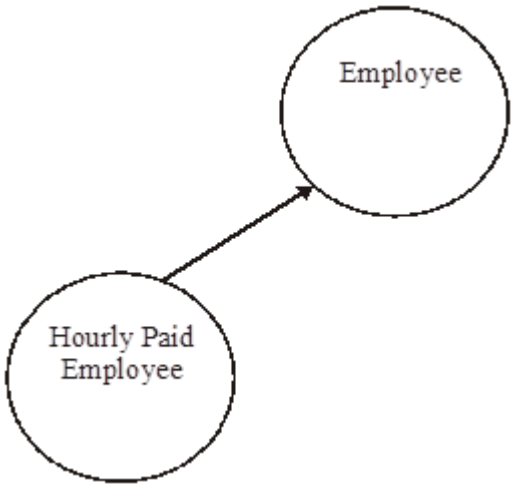
Part (c) expected the class definition of Member, not of any subclasses. The required methods and fields were clearly stated in the question. Candidates needed to ensure that methods were public and fields private and the right data type. Some candidates still seem to be under the false impression that a telephone number can be stored in an integer data type. Candidates should also be aware that the choice of identifiers is important since this is the programmer's interface when using an object, so meaningless abbreviations such as Tno instead of TelephoneNumber were not acceptable. A correct answer is:


```
Member = Class
Public
  Procedure AddNewMemberDetails
  Procedure AmendMemberDetails
  Procedure ShowMemberDetails
Private MembershipNumber: Integer
  FirstName : String
  Surname: String
  TelephoneNumber: String
End
```

Credit was also given to answers that followed Java-style syntax.
Diagrams were not accepted.

Q22.

(a) The drawing of inheritance diagrams has been required for several years now and candidates are expected to draw these in standard format:



It is important to convey the hierarchy (the super class above the sub-class) and draw the arrow from the sub-class to the super class. Classes should be shown in circles or oval shapes.

(b) This question showed the class definition of TEmployee. The sub-class ThourlyEmployee inherits all the fields and methods of TEmployee. The stem of the question stated that pay is calculated differently for an employee object of TEmployee than for THourlyPaidEmployee. This should have alerted candidates that this method needs to be redefined. The question also states that an additional operation to collect the number of hours worked in a month was required. From this, candidates should be able to see that extra fields are also required to store the number of hours worked a month and the hourly pay rate. Taking account of inheritance, the definition of the subclass should therefore be:

```
THourlyPaidEmployee =Class (TEmployee)
Public
       procedure CalculatePay override
       procedure GetNumberOfHoursWorkedInMonth
Private
       HourlyPayRate: Currency
       NumberOfHoursWorkedInMonth :Integer
End
```


Repeated declaration of fields from the parent class was not appropriate. Candidates also need to be aware that a sensible choice of identifiers is important as these are the interface for programmers using classes defined by others. For example the procedure identifier CalculateHoursWorked instead of GetHoursWorked did not gain credit.

Credit was also given to answers that followed a Java-style syntax. Diagrams were not accepted.

Q23.

Some candidates simply re-wrote the question. Others gave quite commendable inheritance diagrams, which, as they were not asked for, gained no marks. Examiners were lenient in their assessment here, giving marks for genuine attempts at coding.

Q24.

Again, vague answers lost marks in this question on object oriented programming, although it was felt that candidates were more familiar with this topic than previously. To say that two advantages of an object oriented approach were ‘can add new code more

efficiently' and 'easier to program' is not creditworthy.

The terms 'inheritance' and 'encapsulation' were liberally scattered over the page, but rarely in a context that gave two advantages of the object oriented approach. The term 'modules' was often used in place of 'objects' or 'classes', which made an otherwise intelligent answer incorrect. A good candidate wrote 'Data and instructions are encapsulated into the object, therefore objects can be reused', and, 'Attributes and activities can be inherited by subclasses meaning less new code is required.'

Most candidates drew a correct inheritance diagram for the classes described in part (b), but few gained marks for writing the class definition in part (c), although considerable latitude was given in assessing their efforts.

Q25.

This focused on object oriented programming. A class is defined as a grouping of data structures and behaviours (methods, procedures or functions). Both data structures and behaviours are needed. Inheritance is a relationship between classes where one class shares the data structures and behaviours with its parent class. The sharing is of all data structures and behaviours, not some and not similar ones. A definition in terms of itself is not sufficient. Thus a definition of inheritance which refers to one class inheriting characteristics from its parent class was not credited.

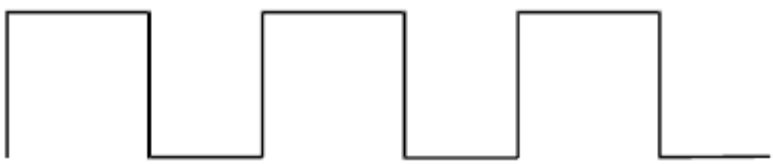
The inheritance diagram was mostly accurate, although the arrows were frequently omitted, but the benefits of an object oriented program over a structured approach, in part (c), rarely gained full marks. Many candidates stated vague facts about object oriented programming, some of which also applied to a structured approach, but failed to explain how these facts were an advantage.

Q26.

Too many candidates simply did not know what an inheritance diagram looked like. Those who gained full marks for this part of Question 1 placed the classes in the correct relative position and joined them by arrows pointing to the base class, **Creature**. The property of a class is comparable to a field, so 'number of legs' was acceptable, but '8 legs' was not. For the method, acceptable answers were commands such as 'eat' or 'spin web'.

Q27.

(a) Several candidates confused a digital signal with digital data and gave inappropriate answers. Digital data are discrete values. Examples are text and integers, e.g. sequences of binary 0s and 1s or denary numbers such as 1, 2, 3, 4, et cetera... In a communications system, data are propagated from one point to another by means of electric signals. An *analogue signal* is a continuously varying electromagnetic wave. A *digital signal* is a sequence of voltage pulses that may be transmitted over a wire medium; for example, a constant positive voltage level may represent binary 1 and a constant negative voltage may represent binary 0 as illustrated below.



Some candidates drew diagrams that depicted a digital signal with more than two discrete levels. This is fine. However, a few candidates drew diagrams of a digital

signal that essentially depicted an analogue signal modulated by a very small digital component. It was not sufficiently clear that the diagram represented a digital signal and hence no credit was given. Candidates are advised, for clarity's sake, to stick to a two state representation when drawing diagrams.

Some candidates drew a diagram of two computers linked by a telecommunications link with a modem at the interface of each with the link. This gained no credit.

(b) Many candidates were unable to define the terms class and inheritance precisely and so failed to gain credit. Many candidates defined a class as an object where in fact it is an object type describing a set of objects that share a common structure and common behaviour. Answers from several candidates focussed on data being stored and gave a definition that described a record not an object type, e.g. " a set of objects with the same data". The idea of encapsulation - combining a record with the functions and procedures that manipulate it - was not well expressed.

(c) The inheritance diagram was drawn vertically by most candidates but with many failing to show the correct direction for the arrows. The correct direction is illustrated below.

Q28.

Object-oriented programming concepts were not well understood by a significant number of candidates. Few candidates could define *class* or *inheritance* correctly. In part (b) an informed guess gave many candidates two out of three marks. The third mark was lost because candidates failed to use the accepted notation for indicating the relationship between a derived class and its parent class. This notation is as follows: