2.1 Data structures, abstract data types part 1 Mark Scheme

Q1.

Marks are for AO1 (understanding)

Static data structures have storage size determined at compile-time / before program is run / when program code is translated; dynamic data structures can grow/shrink during execution / at run-time;
//
Static data structures can waste storage space/memory if the number of data items stored is small relative to the size of the structure; whereas dynamic data structures only take up the amount of storage space required for the actual data;
//
Static data structures have fixed (maximum) size; whereas size of dynamic data structures can change;
//
Dynamic data structures (typically) require memory to store pointer(s) to the next item(s); which static data structures (typically) do not need; **NE**. Dynamic data structures use pointers
//
Static data structures (typically) store data in consecutive memory locations; which dynamic data structures (typically) do not;

[2]

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.

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;
```

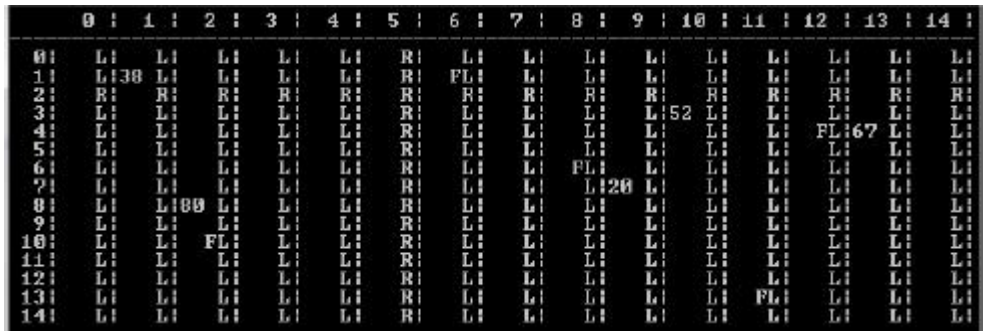
JAVA

```
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;
```

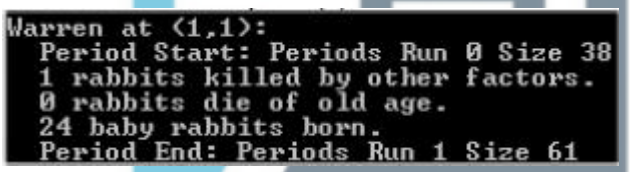
```
            }
            RabbitsToEat =
(int)(Math.round((double)(PercentToEat *
RabbitCountAtStartOfPeriod / 100)));
            FoodConsumed =
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)
;

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)

				List				
ListLength	New	p	q	[1]	[2]	[3]	[4]	[5]
4	*48*	*–*	*–*	*19*	*43*	*68*	*107*	
		1						
		2						
		3						
			4					107
			3				68	
						48		
5								

1 mark: 5 is the only value written in the `ListLength` column
1 mark: 1, 2, 3 in that order are the only values written in the `p` column
1 mark: 4, 3 in that order are the only values written in the `q` column
1 mark: final list is: 19, 43, 48, 68, 107

Ignore values being repeated unnecessarily in trace
MAX 3 if any incorrect values in table

4

(b) Inserts an item/variable `New` into list at correct position/preserving order//into sorted list (or equivalent);

1

(c) Heap is (pool of) free/unused/available memory;
Memory allocated/deallocated at run-time (to/from dynamic data structure(s));

Max 1

[6]

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) **Implementation One** would need to use a linear search // would need to look at every word in the array (before the one that is being searched for) // lookup time is proportional to number of words in list // lookup is O(N); **N.E.** "search" without further clarification that this would be linear
Implementation Two would use the hash function/hashing to directly calculate where the word would be stored // could jump directly to the correct position/location/index for the word in the array // lookup time is constant regardless of how many words in list // lookup is O(1); **A.** No need to go through words in list

2

(b) The (record for) each word/both words would be stored at the same position/index/location in the array;
A. The second word would be stored over/replace the first;
N.E. A collision has occurred

Store record/word in the next available position in the array // store a pointer (in each array position) that points to a list of records that have all collided at the position // rehash the word;
A. Idea that each array position could store more than one record e.g. five records per location, if explained.
A. Example of what "next available" might be.
R. The use of a different hashing function at all times ie not just rehashing.

2

(c) The hash function could compute the same value/location for more than one/two English word(s), so need to verify if the English word stored at the location is the one that is being looked up;
To avoid returning a French translation that is for a different English word, which is stored at the same location as the word that is being looked up // if a collision occurred (when storing the words) it will not be possible to tell if the translation is correct;

A. More than one word could be stored in each location
R. So that French to English translation can be done
MAX 1

1

[5]

Q7.

(a) `BoardDimension;`

R if any additional code
R if spelt incorrectly
I case

1

(b) `DisplayWhoseTurnItIs // DisplayWinner // DisplayBoard // WriteWithMsg` (VB6 only) `// WriteLineWithMsg` (VB6 only)`// WriteLine` (VB6 only)`// WriteNoLine` (VB6 only)`// ReadLine` (VB6 only);
R if any additional code
R if spelt incorrectly
I case

1

(c) `Board;`
R if any additional code
R if spelt incorrectly
I case

1

(d) Delete the three lines and add one copy of the line after the `If` statement(s);

1

(e) If (`PlayAgain` contains) a lowercase letter; it is converted into uppercase;

2

(f) The 123 in the 2nd condition should be 122;
A Change <= 123 to <123
The 3rd column should have condition values of `N` and `N` // the 1st column should have condition values of `N` and `N;`
There should only be an X in the last column; there should not be an X in any of the first three columns; //
there should be a Y (**A** other sensible indicator) in the last column; there should be Xs (**A** other sensible indicator) in the first three columns;

Note for examiners
Marks can be awarded for answers that show a corrected version of Table 4. An example of a possible correct Table 4:

Conditions	>= 97	N	N	Y	Y
	<= 122	N	Y	N	Y
Action	Change value of `PlayAgain`				X

MAX 3

[9]

Q8.

(a) (i) Repetition structure in the correct place in the code with correct termination condition;
Correct error message displayed;
Error message will be displayed every time an invalid name has been

entered and will only be displayed when an invalid name has been entered;
Getting name from user is inside the repetition structure;

A Minor typos in error message
I Capitalisation and spacing in error message

Pascal

```
...
Writeln;
Repeat
    Write('Please enter your name: ');
    Readln(PlayerName);
    If Length(PlayerName) = 0
      Then Writeln('You must enter a name');
Until Length(PlayerName) > 0;
Writeln;
...
```

Alternative answer:

```
...
Writeln;
Repeat
  Write('Please enter your name: ');
  Readln(PlayerName);
  If PlayerName = ''
    Then Writeln('You must enter a name');
Until PlayerName <> '';
Writeln;
...
```

Alternative answer:

```
...
Writeln;
PlayerName := '';
While PlayerName = ''
  Do
    Begin
      Write('Please enter your name: ');
      Readln(PlayerName);
      If Length(PlayerName) = 0
        Then Writeln('You must enter a name');
    End;
Writeln;
...
```

VB.Net

```
...
Console.WriteLine()
Do
  Console.Write("Please enter your name: ")
  PlayerName = Console.ReadLine
  If PlayerName.Length = 0 Then
    Console.WriteLine("You must enter a name")
  End If
Loop Until PlayerName.Length > 0
Console.WriteLine()
...
```

Alternative answer:

```
...
Console.WriteLine()
```

```
Do
  Console.Write("Please enter your name: ")
  PlayerName = Console.ReadLine
  If PlayerName = "" Then
    Console.WriteLine("You must enter a name")
  End If
Loop Until PlayerName <> ""
Console.WriteLine()
...
```

Alternative answer:

```
...
Console.WriteLine()
PlayerName = ""
While PlayerName = ""
  Console.Write("Please enter your name: ")
  PlayerName = Console.ReadLine
  If PlayerName = "" Then
    Console.WriteLine("You must enter a name")
  End If
End While
Console.WriteLine()
...
```

VB6

```
...
WriteLine("")
Do
PlayerName = ReadLine("Please enter your name: ")
  If Len(PlayerName) = 0 Then
    WriteLineWithMsg ("You must enter a name")
  End If
Loop Until Len(PlayerName) > 0
WriteLine("")
...
```

Alternative answer:

```
...
WriteLine("")
Do
  PlayerName = ReadLine("Please enter your name: ")
  If PlayerName = "" Then
    WriteLineWithMsg ("You must enter a name")
  End If
Loop Until PlayerName <>""
WriteLine("")
...
```

Alternative answer:

```
...
WriteLine("")
PlayerName = ""
While PlayerName = ""
PlayerName = ReadLine("Please enter your name: ")
  If PlayerName = "" Then
    WriteLineWithMsg ("You must enter a name")
  End If
Wend
WriteLine("")
...
```

Alternative answers could use some of the following instead of WriteLineWithMsg:

```
Console.Text = Console.Text & ...
WriteLine
WriteWithMsg
Msgbox
WriteNoLine
```

Python 2

```
...
print
Playername = ''
while len(PlayerName) == 0:
  print 'Please enter your name: '
  PlayerName = raw_input()
  if len(PlayerName) == 0:
    print 'You must enter a name'
print
```

Alternative answer:

```
...
print
PlayerName = ''
while PlayerName == '':
  print 'Please enter your name: ',
  PlayerName = raw_input()
  if PlayerName == '':
    print 'You must enter a name'
print
```

Python 3

```
...
print()
PlayerName = ''
while len(PlayerName) == 0:
  print('Please enter your name: '),
  PlayerName = input()
  if len(PlayerName) == 0:
    print('You must enter a name')
print()
```


Alternative answer:

```
...
print()
PlayerName = ''
while PlayerName == '':
  print('Please enter your name: '),
  PlayerName = input()
  if PlayerName == '':
    print('You must enter a name')
print()
...
```

Java

```
...
console.println();
do {
  playerName = console.readLine("Please enter your name: ");
  if (playerName.length() == 0) {
    console.println("You must enter a name");
  }
} while (playerName.length() == 0);
console.println();
...
```

Alternative answer:

```
...
console.println();
do {
  playerName = console.readLine("Please enter your name: ");
  if (playerName.equals("") {
    console.println("You must enter a name");
  }
} while (playerName.equal(""));
console.println();
...
```

4

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

Mark as follows:
No name entered and either error message displayed or asked to enter name; **R** If does not match code for part (a)(i)
Name of `Emily` entered and no error message displayed;

2

(b) (i) Selection structure that checks if the current and last card have the same rank; **A** equivalent logic
Selection structure that checks if the suit of the next card is higher than the suit of the last card; **A** equivalent logic
A one selection structure with two conditions
Note: if overall logic for the first 2 mark points is not correct only one of the 2 marks is to be given
`Higher` assigned value of `True` if the two cards have the same rank and the suit of the next card is higher; **R** If `Higher` always assigned the value of `True`
Value of `Higher` is returned to the calling routine; **R** if no evidence of code used to calculate value of `Higher` when the two cards have the same rank

MAX 3 if any existing functionality is incorrectly changed or if an incorrect value is returned under any circumstances

Pascal

```
Function IsNextCardHigher(LastCard, NextCard : TCard) :
Boolean;
  Var
    Higher : Boolean;
  Begin
    Higher := False;
    If NextCard.Rank > LastCard.Rank
      Then Higher := True;
    If NextCard.Rank = LastCard.Rank
      Then
        If NextCard.Suit > LastCard.Suit
          Then Higher := True;
    IsNextCardHigher := Higher;
  End;
```

Alternative answer:

```
Function IsNextCardHigher(LastCard, NextCard : TCard) :
Boolean;
  Var
```

```
    Higher : Boolean;
  Begin
    If NextCard.Rank > LastCard.Rank
      Then Higher := True
    Else
      If NextCard.Rank < LastCard.Rank
        Then Higher := False
        Else
          If NextCard.Suit > LastCard.Suit
            Then Higher := True
            Else Higher := False;
    IsNextCardHigher := Higher;
  End;
```

Alternative answer:

```
Function IsNextCardHigher(LastCard, NextCard : TCard) :
Boolean;
  Var
    Higher : Boolean;
  Begin
    If NextCard.Rank > LastCard.Rank
      Then Higher := True
      Else Higher := (NextCard.Rank = LastCard.Rank) AND
(NextCard.Suit > LastCard.Suit);
    IsNextCardHigher := Higher;
End;
```

VB.Net

```
Function IsNextCardHigher(ByVal LastCard As TCard, ByVal
NextCard As TCard) As Boolean
  Dim Higher As Boolean
  Higher = False
  If NextCard.Rank > LastCard.Rank Then
    Higher = True
  End If
  If NextCard.Rank = LastCard.Rank Then
    If NextCard.Suit > LastCard.Suit Then
      Higher = True
    End If
  End If
  Return Higher
End Function
```


Alternative answer:

```
Function IsNextCardHigher(ByVal LastCard As TCard, ByVal
NextCard As TCard) As Boolean
  Dim Higher As Boolean
  If NextCard.Rank > LastCard.Rank Then
    Higher = True
  ElseIf NextCard.Rank < LastCard.Rank Then
    Higher = False
  ElseIf NextCard.Suit > LastCard.Suit Then
    Higher = True
  Else
    Higher = False
  End If
  Return Higher
End Function
```

Alternative answer:

```
Function IsNextCardHigher(ByVal LastCard As TCard, ByVal
NextCard As TCard) As Boolean
  Dim Higher As Boolean
```

```
  Higher = False
  If NextCard.Rank > LastCard.Rank Then
    Higher = True
  ElseIf (NextCard.Rank = LastCard.Rank) And (NextCard.Suit >
LastCard.Suit) Then
    Higher = True
  End If
  Return Higher
End Function
```

VB6

```
Private Function IsNextCardHigher(ByRef LastCard As TCard,
ByRef NextCard As TCard) As Boolean
  Dim Higher As Boolean
  Higher = False
  If NextCard.Rank > LastCard.Rank Then
    Higher = True
  End If
  If NextCard.Rank = LastCard.Rank Then
    If NextCard.Suit > LastCard.Suit Then
      Higher = True
    End If
  End If
  IsNextCardHigher = Higher
End Function
```

Alternative answer:

```
Private Function IsNextCardHigher(ByRef LastCard As TCard,
ByRef NextCard As TCard) As Boolean
  Dim Higher As Boolean
  If NextCard.Rank > LastCard.Rank Then
    Higher = True
  ElseIf NextCard.Rank < LastCard.Rank Then
    Higher = False
  ElseIf NextCard.Suit > LastCard.Suit Then
    Higher = True
  Else
    Higher = False
  End If
  IsNextCardHigher = Higher
End Function
```


Alternative answer:

```
Private Function IsNextCardHigher(ByRef LastCard As TCard,
ByRef NextCard As TCard) As Boolean
  Dim Higher As Boolean
  Higher = False
  If NextCard.Rank > LastCard.Rank Then
    Higher = True
  ElseIf (NextCard.Rank = LastCard.Rank) And (NextCard.Suit >
LastCard.Suit) Then
    Higher = True
  End If
  IsNextCardHigher = Higher
End Function
```

Python 2

```
def IsNextCardHigher(LastCard, NextCard):
  Higher = False
  if NextCard.Rank > LastCard.Rank:
    Higher = True
  if NextCard.Rank == LastCard.Rank:
    if NextCard.Suit > LastCard.Suit:
```

```
      Higher = True
  return Higher
```

Alternative answer:

```
def IsNextCardHigher(LastCard, NextCard):
  Higher = False
  if NextCard.Rank > LastCard.Rank:
    Higher = True
  else:
    if NextCard.Rank < LastCard.Rank:
      Higher = False
    else:
      if NextCard.Suit > LastCard.Suit:
        Higher = True
      else:
        Higher = False
  return Higher
```

Alternative answer:

```
def IsNextCardHigher(LastCard, NextCard):
  Higher = False
  if NextCard.Rank > LastCard.Rank:
    Higher = True
  elif NextCard.Rank == LastCard.Rank and NextCard.Suit >
LastCard.Suit:
    Higher = True
  return Higher
```

Python 3

```
def IsNextCardHigher(LastCard, NextCard):
  Higher = False
  if NextCard.Rank > LastCard.Rank:
    Higher = True
  if NextCard.Rank == LastCard.Rank:
    if NextCard.Suit > LastCard.Suit:
        Higher = True
  return Higher
```


Alternative answer:

```
def IsNextCardHigher(LastCard, NextCard):
  Higher = False
  if NextCard.Rank > LastCard.Rank:
    Higher = True
  else:
    if NextCard.Rank < LastCard.Rank:
        Higher = False
    else:
      if NextCard.Suit > LastCard.Suit:
        Higher = True
      else:
        Higher = False
  return Higher
```

Alternative answer:

```
def IsNextCardHigher(LastCard, NextCard):
  Higher = False
  if NextCard.Rank > LastCard.Rank:
    Higher = True
  elif NextCard.Rank == LastCard.Rank and NextCard.Suit >
LastCard.Suit:
    Higher = True
  return Higher
```

Java

```
boolean isNextCardHigher(TCard lastCard, TCard nextCard){
  boolean higher;
  higher = false;
  console.println(lastCard.rank);
  console.println(nextCard.rank);
  if (nextCard.rank > lastCard.rank){
    higher = true;
  }
  if (nextCard.rank == lastCard.rank) {
    if (nextCard.suit > lastCard.suit) {
      higher = true;
    }
  }
  return higher;
}
```

Alternative answer:

```
boolean isNextCardHigher(TCard lastCard, TCard nextCard){
  boolean higher;
  higher = false;
  console.println(lastCard.rank);
  console.println(nextCard.rank);
  if (nextCard.rank > lastCard.rank){
    higher = true;
  } else if (lastCard.rank == nextCard.rank) {
    if (nextCard.suit > lastCard.suit) {
      higher = true;
    }
  }
  return higher;
}
```

Alternative answer:

```
boolean isNextCardHigher(TCard lastCard, TCard nextCard){
  boolean higher;
  higher = false;
  console.println(lastCard.rank);
  console.println(nextCard.rank);
  if (nextCard.rank > lastCard.rank){
    higher = true;
  }
  if (nextCard.rank == lastCard.rank) && (nextCard.suit >
lastCard.suit) {
    higher = true;
  }
  return higher;
}
```

4

(ii) ****SCREEN CAPTURE****

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

Mark as follows:

`y` entered by user results in message `Well done! You guessed correctly.`

Followed by `n` entered by user resulting in message `Well done! You guessed correctly.;`

I If first `y` entered and first message not shown on screen capture

A If code for part (b)(i) has been attempted and screen capture matches what would be produced by code for (b)(i)

Followed by `y` entered by user resulting in message `Well done! You guessed correctly.;`

R If test is for a random (shuffled deck) game
R If answer for 34 has no code for checking if the ranks of the two cards are equal / not equal

2

(c) (i) Modified message in sensible place in code `Do you think the next card will be higher than the last card (enter y or n or j to play a joker)?;`

A Any sensible message
A Two messages instead of a modified message
A No evidence in (c)(i) of this modified message being in the correct place in the code if there is supporting evidence from screen capture(s) for (c)(iii) that it is in the correct place

Pascal

```
...
Var
Choice : Char;
Begin
  Write('Do you think the next card will be higher than the last card (enter y or n or j to play a joker)? ');
  Readln(Choice);
...
```

VB.Net

```
...
Dim Choice As Char
Console.Write("Do you think the next card will be higher than the last card (enter y or n or j to play a joker)? ")
Choice = Console.ReadLine
...
```

VB6

```
...
  Dim Choice As String
  Choice = ReadLine("Do you think the next card will be higher than the last card (enter y or n or j to play a joker)? ")
  GetChoiceFromUser = Choice...
```

Python 2

```
...
print 'Do you think the next card will be higher than the last card (enter y or n or j to play a joker)? '
Choice = raw_input()
...
```

Python 3

```
...
  print('Do you think the next card will be higher than the last card (enter y or n or j to play a joker)? ')
  Choice = input()
...
```

Java

```
...
choice = console.readChar("Do you think the next card will be higher than the last card (enter y or n or j to play a joker)? ");
```

```
...
```

1

(ii) Appropriately named variable (eg `NoOfJokers`), of sensible data type, given initial value of 2;

Modify loop condition so that `y, n` and `j` are all allowed;
Additional condition so that `j` is only allowed if `NoOfJokers` is greater than 0; correct logic used;
A Player loses game if they try to play a 3rd joker as long as correct final score is displayed – note that using this method it is possible that a selection structure is being used instead of a modified loop;;

A Equivalent logic

Value of `NoOfJokers` decremented by 1 inside a selection structure; which has correct condition to check if `j` was option chosen by user;

Modify selection structure so that correct guess is called if either the user has guessed correctly or the player used a Joker; **R** If code will not allow the player to always use two jokers

Alternative answer:
Appropriately named variable (e.g.`NoOfJokers`) of sensible data type, given initial value of 0; **A** value of 0 not explicitly given if code would work without this

Modify loop condition so that `y, n` and `j` are all allowed;
Additional condition so that `j` is only allowed if `NoOfJokers` is less than 2; correct logic used;
A Player loses game if they try to play a 3rd joker as long as correct final score is displayed – note that using this method it is possible that a selection structure is being used instead of a modified loop;;

A Equivalent logic

Value of `NoOfJokers` incremented by 1 inside a selection structure; which has correct condition to check if `j` was option chosen by user;

Modify selection structure so that correct guess is called if either the user has guessed correctly or the player used a Joker; **R** If code will not allow the player to always use two jokers

MAX 5 if, when the game continues, there are unwanted side-effects (e.g. 3rd joker allowed, `Deck` changed when it shouldn't be, score goes up when `j` entered for a third time, etc...)

Pascal

```
...
Choice : Char;
NoOfJokers : Integer;
Begin
  NoOfJokers := 2;
  GameOver := False;
...
  While (NoOfCardsTurnedOver < 52) And Not GameOver
...
     Repeat
```

```
      Choice := GetChoiceFromUser;
    Until (Choice = 'y') Or (Choice = 'n') Or (Choice = 'j')
And (NoOfJokers > 0);
    If Choice = 'j'
      Then NoOfJokers := NoOfJokers - 1;
   DisplayCard(NextCard);
   NoOfCardsTurnedOver := CardsTurnedOver + 1;
   Higher := IsNextCardHigher(LastCard, NextCard);
   If Higher And(Choice='y') Or Not Higher And (Choice = 'n')
Or (Choice = 'j')
     Then
       Begin
       DisplayCorrectGuessMessage(NoOfCardsTurnedOver);
...
```

A equivalent logic for condition (eg `NoOfJokers >=1`)

Alternative answer:

```
...
Choice : Char;
NoOfJokers : Integer;
Begin
  NoOfJokers := 0;
  GameOver := False;
...
  While (NoOfCardsTurnedOver< 52) And Not GameOver
...
    Repeat
      Choice := GetChoiceFromUser;
    Until (Choice = 'y') Or (Choice = 'n') Or (Choice = 'j')
And (NoOfJokers <=1);
    If Choice = 'j'
      Then NoOfJokers := NoOfJokers + 1;
    DisplayCard(NextCard);
    NoOfCardsTurnedOver := CardsTurnedOver + 1;
    Higher := IsNextCardHigher(LastCard, NextCard);
    If Higher And(Choice='y') Or Not Higher And (Choice = 'n')
Or (Choice = 'j')
      Then
       Begin
       DisplayCorrectGuessMessage(NoOfCardsTurnedOver);
...
```


A equivalent logic for condition (eg `NoOfJokers < 2`)

VB.Net

```
...
Dim Choice As Char
Dim NoOfJokers As Integer
NoOfJokers = 2
GameOver = False
...
While NoOfCardsTurnedOver < 52 And Not GameOver
  ...
  Do
    Choice = GetChoiceFromUser()
  Loop Until Choice = "y" Or Choice = "n" Or Choice = "j" And
NoOfJokers > 0
  If Choice = "j" Then
    NoOfJokers = NoOfJokers - 1
  End If
  DisplayCard(NextCard)
  CardsTurnedOver = CardsTurnedOver + 1
```

```
  Higher = IsNextCardHigher(LastCard, NextCard)
  If Higher And Choice = "y" Or Not Higher And Choice = "n" Or
Choice = "j" Then
    DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
...
```

A equivalent logic for condition (eg `NoOfJokers >= 1`)

Alternative Answer:

```
...
Dim Choice As Char
Dim NoOfJokers As Integer
NoOfJokers = 0
GameOver = False
...
While NoOfCardsTurnedOver < 52 And Not GameOver
  ...
  Do
    Choice = GetChoiceFromUser()
  Loop Until Choice = "y" Or Choice = "n" Or Choice = "j" And
NoOfJokers <= 1
  If Choice = "j" Then
    NoOfJokers = NoOfJokers + 1
  End If
  DisplayCard(NextCard)
  CardsTurnedOver = CardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  If Higher And Choice = "y" Or Not Higher And Choice = "n" Or
Choice = "j" Then
    DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
...
```

A equivalent logic for condition (eg `NoOfJokers < 2`)

VB6

```
...
Dim Choice As String
Dim NoOfJokers As Integer
NoOfJokers = 2
GameOver = False
...
While NoOfCardsTurnedOver< 52 And Not GameOver
  ...
  Do
    Choice = GetChoiceFromUser()
  Loop Until Choice = "y" Or Choice = "n" Or Choice = "j" And
NoOfJokers > 0
  If Choice = "j" Then
    NoOfJokers = NoOfJokers - 1
  End If
  Call DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  If Higher And Choice = "y" Or Not Higher And Choice = "n" Or
Choice = "j" Then
    DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
...
```

A equivalent logic for condition (eg `NoOfJokers >= 1`)

Alternative answer:

```
Dim Choice As String
```

```
Dim NoOfJokers As Integer
NoOfJokers = 0
GameOver = False
...
While NoOfCardsTurnedOver < 52 And Not GameOver
  ...
  Do
    Choice = GetChoiceFromUser()
  Loop Until Choice = "y" Or Choice = "n" Or Choice = "j" And
NoOfJokers <=1
  If Choice = "j" Then
    NoOfJokers = NoOfJokers + 1
  End If
  Call DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  If Higher And Choice = "y" Or Not Higher And Choice = "n" Or
Choice = "j" Then
    DisplayCorrectGuessMessage(NoOfCardsTurnedOver)
...
```

A equivalent logic for condition (eg `NoOfJokers < 2`)

Python 2

```
...
NoOfJokers = 2
GameOver = False
...
while (NoOfCardsTurnedOver < 52) and (not GameOver):
...
  Choice = ''
  while (Choice != 'y') and (Choice != 'n') and ((Choice != 'j')
or (NoOfJokers == 0)):
    Choice = GetChoiceFromUser()
  if Choice == 'j':
    NoOfJokers = NoOfJokers - 1
  DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  if (Higher and Choice == 'y') or (not Higher and Choice ==
'n') or (Choice == 'j'):
...
```


A Equivalent logic for condition (eg `NoOfJokers < 1`)

Alternative answer:

```
...
NoOfJokers = 0
GameOver = False
...
while (NoOfCardsTurnedOver < 52) and (not GameOver):
...
  Choice = ''
  while (Choice != 'y') and (Choice != 'n') and ((Choice != 'j')
or (NoOfJokers == 2)):
    Choice = GetChoiceFromUser()
  if Choice == 'j':
    NoOfJokers = NoOfJokers + 1
  DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  if (Higher and Choice == 'y') or (not Higher and Choice ==
'n') or (Choice == 'j'):
```

```
...
```

A equivalent logic for condition (eg `NoOfJokers > 1`)

Python 3

```
...
NoOfJokers = 2
GameOver = False
...
while (NoOfCardsTurnedOver < 52) and (not GameOver):
...
  Choice = ''
  while (Choice != 'y') and (Choice != 'n') and ((Choice != 'j')
or (NoOfJokers == 0)):
    Choice = GetChoiceFromUser()
  if Choice == 'j':
    NoOfJokers = NoOfJokers - 1
  DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  if (Higher and Choice == 'y') or (not Higher and Choice ==
'n') or (Choice == 'j'):
...
```

A Equivalent logic for condition (eg `NoOfJokers < 1`)

Alternative answer:

```
...
NoOfJokers = 0
GameOver = False
...
while (NoOfCardsTurnedOver < 52) and (not GameOver):
...
  Choice = ''
  while (Choice != 'y') and (Choice != 'n') and ((Choice != 'j')
or (NoOfJokers == 2)):
    Choice = GetChoiceFromUser()
  if Choice == 'j':
    NoOfJokers = NoOfJokers + 1
  DisplayCard(NextCard)
  NoOfCardsTurnedOver = NoOfCardsTurnedOver + 1
  Higher = IsNextCardHigher(LastCard, NextCard)
  if (Higher and Choice == 'y') or (not Higher and Choice ==
'n') or (Choice == 'j'):
...
```


A equivalent logic for condition (eg `NoOfJokers > 1`)

Java

```
...
char choice;
int noOfJokers;
...
noOfJokers = 2;
gameOver = false;
...
while (noOfCardsTurnedOver < 52 && !gameOver) {
  getCard(nextCard, deck, noOfCardsTurnedOver);
  do {
    choice = getChoiceFromUser();
  } while (!(choice == 'y' || choice == 'n' || choice == 'j'
&& noOfJokers != 0));
```

```
if (choice == 'j') {
    noOfJokers = noOfJokers - 1;
  }
  displayCard(nextCard);
  noOfCardsTurnedOver = noOfCardsTurnedOver + 1;
  higher = isNextCardHigher(lastCard, nextCard);
  if (higher && choice =='y' || !higher && choice=='n' || choice
== 'j') {
...
```

A equivalent logic for condition (eg `NoOfJokers > 0`)

Alternative answer:

```
...
char choice;
int noOfJokers;
...
noOfJokers = 0;
gameOver = false;
...
while (noOfCardsTurnedOver < 52 && !gameOver) {
  getCard(nextCard, deck, noOfCardsTurnedOver);
  do {
    choice = getChoiceFromUser();
  } while (!(choice == 'y' || choice == 'n' || choice == 'j'
&& noOfJokers != 2));
  if (choice == 'j') {
    noOfJokers = noOfJokers + 1;
  }
  displayCard(nextCard);
  noOfCardsTurnedOver = noOfCardsTurnedOver + 1;
  higher = isNextCardHigher(lastCard, nextCard);
  if (higher && choice =='y' || !higher && choice=='n' || choice
== 'j') {
...
```

A equivalent logic for condition (eg `NoOfJokers < 2`)

7

(iii) *****SCREEN CAPTURE*****

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

Mark as follows:

`j` entered by user results in message `Well done! You guessed correctly.`; **R** if this aspect of test is for a random (shuffled deck) game

2nd `j` entered by user results in message `Well done! You guessed correctly.`; **R** if this aspect of test is for a random (shuffled deck) game

3rd `j` entered by user results in message `Do you think the next card will be higher than the last card (enter y or n)?`; **A** if test for 3rd joker being played is for a random (shuffled deck) game **A** message not being displayed and game ends (only if matches code for (c)(ii)) **I** additional error messages being displayed after `j` entered and before the message `Do you think the next card will be higher than the last card (enter y or n)?` as long as error messages match code for (c)(ii) **R** if player's score is increased when they play a 3rd joker

(d) (i) **A** Any sensibly named identifiers for variables / parameters instead of those used in this mark scheme

There are 5 marks available for setting up a new subroutine and the routine interface:
Created a new subroutine named `CalculateProbability;`
Correct routine interface with parameters of `LastCard` and `Deck` of correct data type;
All data needed by new subroutine is passed to the subroutine via the routine interface (ie no data values obtained from global variables);
Mechanism to return a numeric value to the calling routine set up; **R** use of global variable
Value calculated by subroutine is returned to calling routine;
I Additional parameters

There are then 6 marks available for calculating the probability:

Repetition structure set up to look at each card in `Deck` that has not yet been used in the game;
Selection structure, inside repetition structure, that checks if `LastCard` is higher than a card in `Deck;`

Inside the selection structure `NoOfCardsHigher` incremented if the condition in the selection structure is for a comparison of two cards; **R** If `NoOfCardsHigher` always incremented
Inside the selection structure `NoOfCardsLower` incremented **R** If `NoOfCardsLower` is always incremented;
//
Inside the selection structure `NoOfCardsHigher` incremented if the condition in the selection structure is for a comparison of two cards; **R** If `NoOfCardsHigher` always incremented
Correct calculation for `NoOfCardsInDeck` (does not matter if inside or outside repetition structure);
//
Inside the selection structure `NoOfCardsLower` incremented if the condition in the selection structure is for a comparison of two cards; **R** If `NoOfCardsLower` always incremented
Correct calculation for `NoOfCardsInDeck` (does not matter if inside or outside repetition structure);

Correctly calculates the number of cards, that have not been used in the game so far, that are higher / lower than `LastCard` in `Deck`;
Dividing `NoOfCardsHigher` by `NoOfCardsInDeck` / / Dividing `NoOfCardsHigher` by the sum of `NoOfCardsHigher` and `NoOfCardsLower;` **A** any equivalent calculation **A** correct expression using incorrectly calculated values for `NoOfCardsHigher` / `NoOfCardsLower`

Note: alternative methods that calculate the probability correctly should be referred to team leader.

Pascal

```
Function CalculateProbability(Deck : TDeck;
NoOfCardsTurnedOver : Integer; LastCard : TCard) : Real;
  Var
```

```
    Probability : Real;
    Count : Integer;
    NoOfCardsHigher : Integer;
    NoOfCardsLower : Integer;
  Begin
    NoOfCardsHigher := 0;
    NoOfCardsLower := 0;
    For Count := 1 To (52 - NoOfCardsTurnedOver)
      Do
        If IsNextCardHigher(LastCard, Deck[Count])
          Then NoOfCardsHigher := NoOfCardsHigher + 1
          Else NoOfCardsLower := NoOfCardsLower + 1;
    Probability := NoOfCardsHigher / (NoOfCardsHigher +
NoOfCardsLower);
    CalculateProbability := Probability;
  End;
```

Alternative answer:

```
...
For Count := 1 To (52 - NoOfCardsTurnedOver)
  Do
    If IsNextCardHigher(LastCard, Deck[Count])
      Then NoOfCardsHigher := NoOfCardsHigher + 1;
Probability := NoOfCardsHigher / (52 - NoOfCardsTurnedOver);
CalculateProbability := Probability;
...
```

Alternative answer:

```
Function CalculateProbability(Deck : TDeck; LastCard : TCard)
: Real;
Var
  Probability :Real;
  Count : Integer;
  NoOfCardsHigher : Integer;
  Begin
    NoOfCardsHigher := 0;
    Count := 1;
    While (Count < 52) And (Deck[Count].Suit <> 0)
      Do
        Begin
          If IsNextCardHigher(LastCard, Deck[Count])
            Then NoOfCardsHigher := NoOfCardsHigher + 1;
            Count := Count + 1;
        End;
      Probability := NoOfCardsHigher / (Count - 1);
      CalculateProbability:= Probability;
  End;
```

A `Deck[Count].Rank` instead of `Deck[Count].Suit`

VB.Net

```
Function CalculateProbability(ByVal Deck() As TCard, ByVal
NoOfCardsTurnedOver As Integer, ByVal LastCard As TCard) As
Single
  Dim Probability As Single
  Dim Count As Integer
  Dim NoOfCardsHigher As Integer = 0
  Dim NoOfCardsLower As Integer = 0
  For Count = 1 To(52 - NoOfCardsTurnedOver)
    If IsNextCardHigher(LastCard, Deck(Count)) Then
      NoOfCardsHigher = NoOfCardsHigher + 1
    Else
      NoOfCardsLower = NoOfCardsLower + 1
    End If
```

```
  Next
  Probability = NoOfCardsHigher / (NoOfCardsHigher +
NoOfCardsLower)
  Return Probability
End Function
```

Alternative answer:

```
...
For Count = 1 To (52 - NoOfCardsTurnedOver)
  If IsNextCardHigher(LastCard, Deck(Count)) Then
    NoOfCardsHigher = NoOfCardsHigher + 1
  End If
Next
Probability = NoOfCardsHigher / (52 -NoOfCardsTurnedOver)
CalculateProbability = Probability
...
```

Alternative answer:

```
Function CalculateProbability(ByVal Deck() As TCard, ByVal
LastCard As TCard) As Single
  Dim Probability As Single
  Dim Count As Integer
  Dim NoOfCardsHigher As Integer = 0
  Count = 1
  While Count < 52 And Deck(Count).Suit <> 0
    If IsNextCardHigher(LastCard, Deck(Count)) Then
      NoOfCardsHigher = NoOfCardsHigher + 1
    End If
    Count = Count + 1
  End While
  Probability = NoOfCardsHigher / (Count - 1)
  Return Probability
End Function
```

A `Deck(Count).Rank` instead of `Deck(Count).Suit`

Note: return mechanism does not need to be explicitly set up in routine interface

VB6

```
Private Function CalculateProbability(ByRef Deck() As TCard,
ByVal NoOfCardsTurnedOver As Integer, ByRef LastCard As TCard)
As Single
  Dim Probability As Single
  Dim Count As Integer
  Dim NoOfCardsHigher As Integer
  Dim NoOfCardsLower As Integer
  NoOfCardsHiger = 0
  NoOfCardsLower = 0
  For Count = 1 To (52 - NoOfCardsTurnedOver)
    If IsNextCardHigher(LastCard, Deck(Count)) Then
      NoOfCardsHigher = NoOfCardsHigher + 1
    Else
      NoOfCardsLower = NoOfCardsLower + 1
    End If
  Next
  Probability = NoOfCardsHigher / (NoOfCardsHigher +
NoOfCardsLower)
  CalculateProbability = Probability
End Function
```

Alternative answer:

```
...
For Count = 1 To (52 - NoOfCardsTurnedOver)
```

```
  If IsNextCardHigher(LastCard, Deck(Count)) Then
    NoOfCardsHigher = NoOfCardsHigher + 1
  End If
Next
Probability = NoOfCardsHigher / (52 - NoOfCardsTurnedOver)
CalculateProbability = Probability
...
```

Alternative answer:

```
Private Function CalculateProbability(ByRef Deck() As TCard,
ByRef LastCard As TCard) As Single
  Dim Probability As Single
  Dim Count As Integer
  Dim NoOfCardsHigher As Integer
  NoOfCardsHigher = 0
  Count = 1
  While Count < 52 And Deck(Count).Suit <> 0
    If IsNextCardHigher(LastCard, Deck(Count)) Then
      NoOfCardsHigher = NoOfCardsHigher + 1
    End If
    Count = Count + 1
  Wend
  Probability = NoOfCardsHigher / (Count - 1)
  CalculateProbability = Probability
End Function
```

A `Deck(Count).Rank` instead of `Deck(Count).Suit`

Python 2

```
def CalculateProbability(Deck, NoOfCardsTurnedOver,
LastCard):
  NoOfCardsHigher = 0
  NoOfCardsLower = 0
  for Count in range(1, 53 - NoOfCardsTurnedOver):
    if (IsNextCardHigher(LastCard, Deck[Count]):
      NoOfCardsHigher = NoOfCardsHigher + 1
    else:
      NoOfCardsLower = NoOfCardsLower + 1
  Probability = NoOfCardsHigher / (NoOfCardsHigher +
NoOfCardsLower)
  return Probability
```


Alternative answer:

```
...
for Count in range(1, 53 - NoOfCardsTurnedOver):
  if (IsNextCardHigher(LastCard, Deck[Count]):
    NoOfCardsHigher = NoOfCardsHigher + 1
Probability = NoOfCardsHigher / (52 - NoOfCardsTurnedOver)
return Probability
...
```

Alternative answer:

```
def CalculateProbability(Deck,LastCard):
  NoOfCardsHigher = 0
  Count = 1
  while (Count < 52) and (Deck[Count].Suit != 0):
    if (IsNextCardHigher(LastCard, Deck[Count]):
      NoOfCardsHigher = NoOfCardsHigher + 1
    Count = Count + 1
  Probability = NoOfCardsHigher / (Count - 1)
  return Probability
```

A `Deck[Count].Rank` **instead of** `Deck[Count].Suit`

Python 3

```
def CalculateProbability(Deck, NoOfCardsTurnedOver,
LastCard):
  NoOfCardsHigher = 0
  NoOfCardsLower = 0
  for Count in range(1, 53 - NoOfCardsTurnedOver):
    if (IsNextCardHigher(LastCard, Deck[Count]):
      NoOfCardsHigher = NoOfCardsHigher + 1
    else:
      NoOfCardsLower = NoOfCardsLower + 1
  Probability = NoOfCardsHigher / (NoOfCardsHigher +
NoOfCardsLower)
  return Probability
```

Alternative answer:

```
...
for Count in range(1, 53 - NoOfCardsTurnedOver):
  if (IsNextCardHigher(LastCard, Deck[Count]):
    NoOfCardsHigher = NoOfCardsHigher + 1
Probability = NoOfCardsHigher / (52 - NoOfCardsTurnedOver)
return Probability
...
```

Alternative answer:

```
def CalculateProbability(Deck, LastCard):
  NoOfCardsHigher = 0
  Count = 1
  while (Count < 52) and (Deck[Count].Suit != 0):
    if (IsNextCardHigher(LastCard, Deck[Count]):
      NoOfCardsHigher = NoOfCardsHigher + 1
    Count = Count + 1
  Probability = NoOfCardsHigher / (Count - 1)
  return Probability
```

A `Deck[Count].Rank` **instead of** `Deck[Count].Suit`

Java

```
float calculateProbability(TCard[] deck, int
noOfCardsTurnedOver, TCard lastCard) {
  int noOfCardsHigher;
  int noOfCardsLower;
  float probability;
  noOfCardsHigher = 0;
  noOfCardsLower = 0;
  for (int count = 1; count <= 52 - noOfCardsTurnedOver;
count++) {
    if (isNextCardHigher(lastCard, deck[count])) {
     noOfCardsHigher = noOfCardsHigher + 1;
    } else {
      noOfCardsLower = noOfCardsLower + 1;
    }
  }
  probability = (float) noOfCardsHigher / (noOfCardsHigher +
noOfCardsLower);
  return probability;
}
```

Alternative answer:

```
float calculateProbability(TCard[] deck, int
noOfCardsTurnedOver, TCard lastCard) {
  int noOfCardsHigher;
  float probability;
  noOfCardsHigher = 0;
  for (int count = 1; count <= 52 - noOfCardsTurnedOver;
```

```
count++) {
    if (isNextCardHigher(lastCard, deck[count])) {
      noOfCardsHigher = noOfCardsHigher + 1;
    }
  }
  probability = (float) noOfCardsHigher / (52 -
noOfCardsTurnedOver);
  return probability;
}
```

Alternative answer:

```
float calculateProbability(TCard[] deck, int
noOfCardsTurnedOver, TCard lastCard) {
  int noOfCardsHigher;
  float probability;
  int count;
  count = 1;
  noOfCardsHigher = 0;
  while (count < 52 && deck[count].suit != 0) {
    if (isNextCardHigher(lastCard, deck[count])) {
      noOfCardsHigher = noOfCardsHigher + 1;
    }
    count = count + 1;
  }
  probability = (float) noOfCardsHigher / (count - 1);
  return probability;
}
```

A `deck[count].rank` **instead of** `deck[count].suit`

11

(ii) Call to `CalculateProbability` subroutine in correct place;
R If parameter list does not match answer to (d)(i)

Displays "`The probability of the next card being higher is `" in correct place;
A Minor typos in prompt
I Capitalisation

Displays the calculated probability;
R If probability not returned by `CalculateProbability` subroutine
A Use of temporary variable to store the value returned by `CalculateProbability` with contents of temporary variable then displayed using output message
A Incorrect probability as long as value displayed is the value returned by `CalculateProbability` subroutine
I Case of identifiers and output messages
A Minor typos in output messages
I Spacing in output messages

Pascal

```
...
Writeln('The probability of the next card being higher is ',
CalculateProbability(Deck, NoOfCardsTurnedOver,
LastCard):3:2);
GetCard(NextCard, Deck, NoOfCardsTurnedOver);
Repeat
  Choice := GetChoiceFromUser;
...
```

VB.Net

```
...
  Console.WriteLine("The probability of the next card being
higher is " & CalculateProbability(Deck, NoOfCardsTurnedOver,
LastCard))
  GetCard(NextCard, Deck, NoOfCardsTurnedOver)
  Do
    Choice = GetChoiceFromUser()
    ...
```

VB6

```
...
WriteLine("The probability of the next card being higher is "
& CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard))
Call GetCard(NextCard, Deck, NoOfCardsTurnedOver)
Do
  Choice = GetChoiceFromUser()
  ...
```

Alternative answers could use some of the following instead of WriteLine:

```
Console.Text = Console.Text & ...
WriteLineWithMsg
WriteWithMsg
Msgbox
WriteNoLine
```

Python 2

```
...
print 'The probability of the next card being higher is %3.2f'
%CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard)
GetCard(NextCard, Deck, NoOfCardsTurnedOver)
Choice = ''
while (Choice != 'y') and (Choice != 'n'):
  Choice = GetChoiceFromUser()
...
```


Python 3

```
...
print('The probability of the next card being higher is %3.2f'
%CalculateProbability(Deck, NoOfCardsTurnedOver, LastCard))
GetCard(NextCard, Deck, NoOfCardsTurnedOver)
Choice = ''
while (Choice != 'y') and (Choice != 'n'):
  Choice = GetChoiceFromUser()
...
```

Java

```
...
console.print("The probability of the next card being higher
is ");
console.println(calculateProbability(deck,
noOfCardsTurnedOver, lastCard));
getCard(nextCard, deck, noOfCardsTurnedOver);
...
```

3

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

This is conditional on sensible code for (d)(i) and / or (d)(ii)

```
The probability of the next card being higher is 1;
```

User enters `y` followed by `The probability of the next card being higher is 0.9 ;`

A probabilities expressed as percentages (100, 90)
A Probabilities expressed as fractions (51 / 51, 45 / 50)
A Probabilities expressed in scientific form (1.00E+00, 0.90E+00)
A 0.9411765 and 0.88 instead of 1 and 0.9 - if (b) not completed / completed after (d)
A Other values for probabilities that are correct based on incorrect answer for (b) only if code for (d) is correct
R If test is for a random (shuffled deck) game
R Probability of 0

2

[39]

Q9.

(a)

Algorithm Name	**Requires Sorted List? (Tick one box)**
Binary search	✓
Linear search	

1 mark for having a tick in the "Binary search" row.
A alternative indicators for tick eg "Yes"
A a tick for "Binary search" and a cross for "Linear search"
R answers where two ticks have been used.

1

(b)

				List			
List Length	Outer Pointer	Current Value	Inner Pointer	[1] 9	[2] 8	[3] 5	[4] 6
4	2	8	1		9		
			0	8			
	3	5	2			9	
			1		8		
			0	5			
	4	6	3				9
			2			8	
			1		6		

Award **1 mark** for each of the highlighted rectangles which has the correct values written in it in the unshaded cells.
Accept responses in which correct values are unnecessarily written out again.
Do not award a mark for any rectangle which has an incorrect value written in it.

3

(c) The value being moved / CurrentValue / 6 does not need to be put at the start of the list / / should be inserted at position 2 not position 1;
Because the second condition (in the While statement) is not satisfied;
MAX 1

1

(d)

Order of Time Complexity	Tick one box
O(n)	
$O(n^2)$	✓
$O(2^n)$	

A alternative indicators instead of a tick eg a cross, Y, Yes
R responses in which more than one box is ticked

1

(e) Insertion sort;
A Insert sort

1

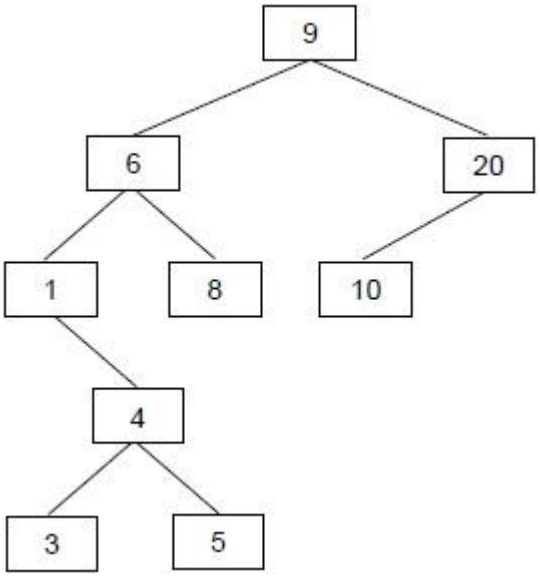
(f) (i) 9, 6, 8;
Must be in the order above. Can be separated by any character or a space

1

(ii) 9, 20,10;
Must be in the order above. Can be separated by any character or a space.

1

(g)



1 mark for inserting number 4 in the correct place
1 mark for inserting both numbers 3 and 5 in the correct place relative to 4
MAX 1 if any numbers added in the wrong place / any extra numbers added

2

[11]

Q10.

(a) Values / cards need to be taken out of the data structure from the opposite end that they are put in / / cards removed from top / front and added at end / bottom / rear;
Values / cards need to be removed in the same order that they are added;
A It is First in First Out / / It is FIFO;
A It is Last in Last Out / / It is LILO;
MAX 1

1

(b) (i)
```
FrontPointer = 11
RearPointer = 52
QueueSize = 42
```
1 mark for all three values correct

1

(ii)
```
FrontPointer = 11
RearPointer = 2
QueueSize = 44
```

1 mark for all three values correct
A incorrect value for `FrontPointer` if it matches the value given in part (i) and incorrect value for `QueueSize` if it is equal to the value given for `QueueSize` in part (i) incremented by two (follow through of errors previously made)

1

(iii)
```
If DeckQueue is empty then
  Report error
Else
  Output DeckQueue[FrontPointer]
  Decrement QueueSize
  Increment FrontPointer
  If FrontPointer>52 Then
     FrontPointer = 1
```

```
EndIf
```

1 mark for If statement to check if queue is empty – alternative for test is `QueueSize = 0`.
1 mark for reporting an error message if the queue is empty / / dealing with the error in another sensible way – this mark can still be awarded if there is an error in the logic of the If statement, as long as there is an If statement with a clear purpose.
1 mark for only completing the rest of the algorithm if the queue is not empty – this mark can still be awarded if there is an error in the logic of the If statement, as long as there is an If statement with a clear purpose.
1 mark for outputting the card at the correct position
1 mark for incrementing FrontPointer and decrementing QueueSize
1 mark for If statement testing if the end of the queue has been reached
1 mark for setting FrontPointer back to 1 if this is the case – this mark can still be awarded if minor error in logic of If statement, eg >= instead of =
A `FrontPointer = (FrontPointer` MOD 52) + 1 for **3 marks** or FrontPointer = (FrontPointer MOD 52) for **2 marks**, both as alternatives to incrementing and using and the second If statement - **deduct 1 mark** from either of the above if
QueueSize has not been decremented
A any type of brackets for array indexing
I Additional reasonable EndIf Statements
MAX 5 unless all of the steps listed above are carried out

6

(c) Flow of program / execution sequence determined by events / / program executes relevant code-handling block / procedure / sub-routine in response to events;
Example of event such as clicking a button;
Message sent to program when event occurs;
System / message loop executes until application closes; this receives and processes messages / / use of event-listener / handler;
If several events occur they are queued;
MAX 2

2

(d) The smartphone operating system:

- Will not have to support as wide a range of hardware devices / peripherals / / may not support external storage devices;
- Will not / less likely to need to support the addition of new hardware devices to the system;
- Will have minimising power consumption as a higher priority; **A** will have more sophisticated power management features
- Will run application software in a sandbox / / will (more tightly) restrict access to resources by application software;
- Must be capable of running on a device with less processing power / / less RAM / memory / / smaller memory footprint;
- Needs to work with specialised hardware devices eg GPS receiver, accelerometer (**Note:** A relevant example must be given);

A Points made in reverse, but do not give **2 marks** for one point and its reverse.
MAX 2

2

[13]

Q11.

(a) +;
4, 9, 6; (in any order)

2

(b) **A** Store the data / value (in the vertices / nodes);
A: holds the expression
B: Left pointer // points to the left child / left sub tree;
C: Right pointer // points to the right child / right sub tree;
A “indicates”, "index" or other synonym for “points” / “pointer”
R Stores left / right subtree

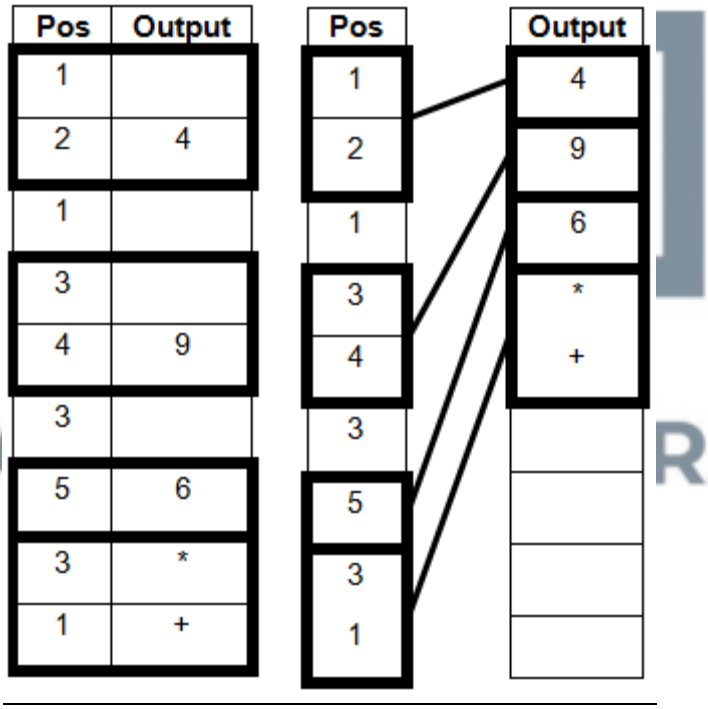
3

(c) The node has no left child / sub tree;
A there is nothing to the left
A this is a null pointer

1

(d) One mark for each area outlined with a dark rectangle. Lines that are not outlined can be missed out.

Alternative 1 Alternative 2



Mark against whichever alternative gives the highest mark.

Stop marking as soon as incorrect output is given.

4

(e) Post-order;
A Depth-first
A Depth-first search as BOD
TO Depth-first pre / in-order

1

(f) (4 + 9 * 6 in) Reverse Polish (Notation) // Postfix (Notation) // RPN;

1

[12]

Q12.

(a) Static structures have fixed (maximum) size whereas size of dynamic structures can change // Size of static structure fixed at compile-time whereas size of dynamic structure can change at run-time;
Static structures can waste storage space / memory if the number of data items stored is small relative to the size of the structure whereas dynamic structures only take up the amount of storage space required for the actual data;
Dynamic data structures (typically) require memory to store pointer(s) to the next item(s) which static structures (typically) do not need // Static structures (typically) store data in consecutive memory locations, which dynamic data structures (typically) do not;
Max 2
A just one side of points, other side is by implication
NE Dynamic data structures use pointers

2

(b) Not possible to simply insert item into middle of list;
Must move all items that should come after the new process down in the array;
NE move all data
Moving items is time consuming;
In a dynamic implementation, insertion achieved by adjusting pointers;
Max 2

2

(c) Priority (queue);

1

(d) (i) Memory allocated / deallocated at run-time / for new items (to dynamic data structure);
(Provides a) pool of free / unused / available memory;
NE to store new items
Max 1

1

(ii) (Memory) address // memory location // position in memory;
NE position or location without reference to memory
R index

1

(iii) **OVERALL GUIDANCE:**

Solutions should be marked on this basis:

- Up to **4 marks** for correctly locating the position to insert the new process at.

- Up to **4 marks** for creating a new node and storing the correct data into it and the associated pointers.
  Some marks can be awarded for this even if the locating process is incorrect / missing.

The full 7 marks should only be awarded for a complete fully working solution. If any steps are missed out, then award a Maximum of six marks (Max 6).

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. ***If you are unsure about the correctness of a solution please refer it to a team leader. Also, responses in prose should be referred to team leaders.***

SPECIFIC MARKING POINTS

Correctly locating insertion point (Max 4):

1 Initialising current node pointer to start pointer;
2 Use of loop to attempt to move through list (regardless of correct terminating condition);
3 Advancing current node pointer within loop;
4 Correctly maintaining a pointer to the node before the position that the new node should be inserted at;
5 Sensible condition to identify place to insert (suitable terminating condition for loop or condition in selection statement);

Correctly inserting new process (Max 4):

6 Create a new node / obtain new node from heap;
7 Store new process name and priority (in new node);
8 Update NextNodePointer of node before newly inserted one to point to new node;
9 Set NextNodePointer of new node to point to node after it;

Mark point 2 can only be awarded if, within the loop, current node pointer is being changed (even if not correctly changed).

Mark point 4 can only be awarded if mark point 3 had been awarded.

Mark point 5 can be awarded if there is a sensible condition, even if current node pointer is not correctly updated.

Mark points 8 and 9 can only be awarded if the correct insertion point has been found.

For any solution:

A use of either while or repeat loops, as long as logic is correct.
A storage of values into new node in any order, and regardless of whether the node has been created or not.
A use of ^ symbol to indicate the value stored at an address referenced by a pointer, for example CurrentNodePointer^. Priority indicates the value stored in the Priority field of the node pointed to by the pointer CurrentNodePointer.
A use of alternative variable names so long as the meaning is clear.

EXAMPLE SOLUTIONS AND MARKS:

These four examples show where marks should be awarded in some possible solutions (subject to a maximum mark of 7):

Example 1:

```
CurrentNodePointer = StartPointer;
Repeat
  PreviousNodePointer =
   CurrentNodePointer;
  CurrentNodePointer =
   NextNodePointer of current node;
Until priority of process in current node < priority
of process to add
//
 priority = "Low";;
Create new node;
Store new process name (and priority)
 in new node;
New node's NextNodePointer = Next
 NodePointer of item at position
 PreviousNodePointer;
NextNodePointer of item at position
 PreviousNodePointer = Address of new
 node;
```

Example 2:

This is an alternative way of expressing Example 1:

1 Load the Start Pointer value into the Current Node Pointer;
2 Copy value from Current Node Pointer into Previous Node Pointer;
3 Set Current Node Pointer to Next Node Pointer of current node;
4 If the priority of the data item at the current node is higher than or the same as the priority of the process to be added; then go back to step 2;
5 Create a new node;
6 Store the name of the new process (and its priority) in the new node;
7 Copy value from Next Node Pointer of list entry at position stored in Previous Node Pointer into the Next Node Pointer of the new node;
8 Set the Next Node Pointer of the list entry at position stored in the Previous Node Pointer to point to the new node;

Example 3:

```
CurrentNodePointer = StartPointer;
Inserted = False
While; Inserted = False Do
  If Current Node's priority < new item
    priority // = "Low"; Then
      Create new node;
      Store new process name (and priority)
        in new node;
      New node's NextNodePointer =
        CurrentNodePointer;
      NextNodePointer of item at position
        PreviousNodePointer = Address of new
        node;
      Inserted=True
```

```
  End If
  PreviousNodePointer = CurrentNodePointer;
  CurrentNodePointer =
    NextNodePointer of current node;
End While
```

Example 4:

```
CurrentNodePointer = StartPointer;
While; not at end of list // While
 CurrentNodePointer <> Nil // While
 priority of process at CurrentNodePointer
 >= priority of process to add Do
  If Current Node's priority is the required
   priority // = "Normal"; Then
    LastNodeOfCurrentPriorityPointer =
     CurrentNodePointer;
  End If
  CurrentNodePointer =
    NextNodePointer of current node;
End While
Create new node;
Store new process name (and priority)
  in new node;
New node 's NextNodePointer = Next
  NodePointer of item at position
  LastNodeOfCurrentPriorityPointer;
NextNodePointer of item at position
  LastNodeOfCurrentPriorityPointer = Address
  of new node;
```

7

[14]

Q13.

(a) Correct variable declarations for `Bit, Answer` and `Column;`
I additional variable declarations
`Column` initialised correctly before the start of the loop;
`Answer` initialised correctly before the start of the loop;
`While/Repeat` loop, with syntax allowed by the programming language used, after the variable initialisations; and correct condition for the termination of the loop;
R `For` loop
A any `While/Repeat` loop with logic corresponding to that in flowchart
(for a loop with a condition at the start accept >=1 or >0 but reject <>0)
Correct prompt `"Enter bit value:"` ;
followed by `Bit` assigned value entered by user;
followed by `Answer` given new value;
R if incorrect value would be calculated [followed by value of `Column` divided by 2;
A multiplying by 0.5
Correct prompt and the assignment statements altering `Bit, Answer` and `Column` are all within the loop;
After the loop – output message followed by value of `Answer`;

I Case of variable names, player names and output messages
A Minor typos in variable names and output messages
I spacing in prompts
A answers where formatting of decimal values is included e.g.
`Writeln('Decimal value is: ', Answer : 3)`
A initialisation of variables at declaration stage
A no brackets around column * bit

Pascal

```
Program Question;
    Var
         Answer : Integer;
         Column : Integer;
         Bit : Integer;
    Begin
          Answer := 0;
          Column := 8;
          Repeat
              Writeln('Enter bit value: ');
              Readln(Bit);
              Answer := Answer + (Column * Bit);
              Column := Column DIV 2;
         Until Column < 1;
         Writeln('Decimal value is: ', Answer);
         Readln;
    End.
```

VB.NET

```
Sub Main()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Console.Write("Enter bit value: ")
        Bit = Console.ReadLine
        Answer = Answer + (Column * Bit)
        Column = Column / 2
   Loop Until Column < 1
   Console.Write("Decimal value is: " & Answer)
   Console.ReadLine()
End Sub
```


Alternative Answer

```
Column = Column \ 2
```

VB6

```
Private Sub Form_Load()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Bit = InputBox("Enter bit value: ")
        Answer = Answer + (Column * Bit)
        Column = Column / 2
   Loop Until Column < 1
   MsgBox ("Decimal value is: " & Answer)
End Sub
```

Alternative Answer

```
Column = Column \ 2
```

Java

```
public class Question {
   AQAConsole console=new AQAConsole();
   public Question(){
         int column;
         int answer;
         int bit;
         answer=0;
         column=8;
         do{
              console.print("Enter bit value: ");
              bit=console.readInteger("");
              answer=answer+(column*bit);
              column=column/2;
         }while(column>=1);
         console.print("Decimal value is: ");
         console.println(answer);
      }
      public static void main(String[] arrays){
          new Question();
      }
}
```

Python 2.6

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print "Enter bit value: "
   # Accept: Bit = int(raw_input("Enter bit value: "))
   Bit = input()
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print "Decimal value is: ", Answer
# or + str(Answer)
```


Python 3.1

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print("Enter bit value: ")
   # Accept: Bit = int(input("Enter bit value: "))
   Bit = int(input())
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print("Decimal value is: " + str(Answer))
# or print("Decimal value is: {0}".format(Answer))
```

A. Answer and Bit not declared at start as long as they are spelt correctly and when they are given an initial value that value is of the correct data type

11

(b) ****SCREEN CAPTURE****

Must match code from 16, including prompts on screen capture matching those in code

Mark as follows:

"Enter bit value:" + first user input of 1

'Enter bit value: ' + second user input of 1
'Enter bit value: ' + third user input of 0
'Enter bit value: ' + fourth user input of 1
Value of 13 outputted;

3

(c) 15;

1

(d) 16 // twice as many // double;

1

(e) Design;
A Planning

1

(f) Implementation;

1

[18]

Q14.

(a) `ResetCavern;` (all languages)
// `GetNewRandomPosition` (Pascal only)
// `WriteWithMsg` (VB6 only)
// `WriteLineWithMsg` (VB6 only)
// `WriteLine` (VB6 only)
// `WriteNoLine` (VB6 only)
// `ReadLine` (VB6 only);
// `SetUpTrapPostions` (Python / Java only);
R if any additional code (including routine interface)
R if spelt incorrectly
I case

1

(b) `DisplayMenu // DisplayMoveOptions // DisplayWonGameMessage // DisplayTrapMessage // DisplayLostGameMessage // WriteWithMsg` (VB6 only) // `WriteLineWithMsg` (VB6 only) // `WriteLine` (VB6 only) // `WriteNoLine`(VB6 only);
A `DisplayCavern;`
R if any additional code (including routine interface)
R if spelt incorrectly
I case

1

(c) `Count1 // Count2 // Count;`
R if any additional code
R if spelt incorrectly
I case

1

(d) `Cavern // TrapPositions;`
R if any additional code (including routine interface)
R if spelt incorrectly
A `LoadedGameData.TrapPositions`
A `CurrentGameData.TrapPositions`
I case

1

(e) When the value of the cell in the `Cavern` array // When the value of the cell to

place the item in;
Indicated by the `Position` variable;
Contains a space // does not contain another item;
R is empty

Max 2 if no variable names used in description

3

(f) The number of times to repeat is known;
A fixed

1

(g) Makes the program code easier to understand;
Makes it easier to update the program;
Makes it easier to change the number of traps (in the game);

Max 1

(h) In text files all data is stored as strings / ASCII values / lines/characters // Text files use only byte values that display sensibly on a VDU // stores only values that can be opened and read in a text editor;

Binary files store data using different data types; **A** by example **A** Binary files can only be correctly interpreted by application that created them

2

(i) Easier reuse of routines in other programs;
Routine can be included in a library;
Helps to make the program code more understandable;
Ensures that the routine is self-contained // routine is independent of the rest of the program;
(Global variables use memory while a program is running) but local variables use memory for only part of the time a program is running;
reduces possibility of undesirable side effects;
Using global variables makes a program harder to debug;

Max 2

(j) (If it was not then) `MonsterAwake` is set to the Boolean value returned by the second call to `CheckIfSameCell;`
this would overwrite any True value returned by the first call to `CheckIfSameCell;`
//
Otherwise the monster would never wake up when the player triggers the first trap;;
//
Otherwise the monster would only wake up when the player triggers the second trap;;

2

[15]

Q15.

(a) **VB.Net/VB6**

```
Const MaxSize = 4
```

I capitalisation

Pascal

```
Const MaxSize = 4
```

I missing semicolon, capitalisation

NE MaxSize
A MaxSize = 4

Java
`final int MAX_SIZE = 4;`
I missing semicolon, capitalisation
NE `MAX_SIZE`

Python 2.6 and 3
`MAX_SIZE = 4`

1

(b) Improves readability of code // Easier to update the programming code if the value changes (**A** by implication) // reduce the likelihood of errors;

1

(c) `PlayerOneName // PlayerTwoName;`
R if any additional code
R if spelt incorrectly
I case & spaces
A `Max_SIZE (Python only)`
A `Currentfile` (**R** for VB6/VB.Net)

1

(d) `LowestCurrentTopScore ;`
A `PositionOfLowestCurrentTopScore;`
R if any additional code
R if spelt incorrectly
I case & spaces

1

(e) `b;`

1

(f) `True;`

1

(g) `False;`

1

(h) `UpdateTopScores;`
R if spelt incorrectly
I case & spaces

1

(i) `VirtualDiceGame;`
R if spelt incorrectly
I case & spaces

1

(j) `AppealDieResult;`
`RollAppealDie;`
R if spelt incorrectly
R RollAppealDie (Python only)
I case & spaces

1

(k) Until PlayerOut // Until PlayerOut = True // until player is out;
A any unambiguous description of the loop termination condition

1

(l) Because the scope; of the two variables is different; //
Because they are both local variables; in different subroutines;
A Because where they are accessible is different;

2

(m) 3;

1

(n) It compares the score of the current record/position (in the `TopScores` array);
with the lowest score found so far // with LowestCurrentTopScore;
if it is less than it then it changes the lowest score found so far; **R** swaps
and makes the position of the lowest top score equal to count / equal to the
current position in the array;

4

[18]

Q16.

(a)

Reverse Polish Notation	Equivalent Infix Expression
45 6 +	*45 + 6* ***R*** *6 + 45*
12 19 + 8 *	*(12 + 19) * 8* ***R*** *12+19*8, (19+12)*8* ***A*** *x for **

2

(b) Simpler for a machine / computer to evaluate // simpler to code algorithm
A easier **R** to understand
Do not need brackets (to show correct order of evaluation/calculation);
Operators appear in the order required for computation;
No need for order of precedence of operators;
No need to backtrack when evaluating;
A RPN expressions cannot be ambiguous as **BOD**

1

(c)

String Pos	Token	Integer Val	Op1	Op2	Result	Stack
0	-	-	-	-	-	
1	6	6				6
2	4	4				4 6
3	+		**6**	**4**	10	10
4	3	3				3 10
5	2	2				2 3 10
6	+		**3**	**2**	5	5 10
7	*		**10**	**5**	50	50

Output : **50**

1 mark for each of rows 1–3
1 mark for rows 4 and 5 together
1 mark for rows 6 and 7 together
1 mark for correct final output
Values of Op1 and Op2 MUST be assigned in rows 3, 6 and 7 to award the marks for these rows. They cannot be inferred from incorrectly entered previous values.

***I** values in empty cells, even if they are incorrect.*

6

(d)
```
If StackArray is full
  Then Stack Full Error
  Else
    Increment TopOfStackPointer
    StackArray [TopOfStackPointer]  ←
        ANumber
EndIf
```

1 mark for appropriate `If` structure including condition (does not need both `Then` and `Else`) – Do not award this mark if `ANumber` is put into `StackArray` outside the `If`.
1 mark for reporting error in correct place
1 mark for incrementing `TopOfStackPointer`*
1 mark for storing value in `ANumber` into correct position in array*
** = if the store instruction is given before the increment instruction OR*

*the `If` structure then award **Max 1** of these two marks UNLESS the item is inserted at position `TopOfStackPointer+1` so the code would work.*

I initialisation of `TopOfStackPointer` to 0
A `TopOfStackPointer=20/>=20` for Stack is full
A Logic of `if` structure reversed i.e. If stack is not full / `TopOfStackPointer<20 / <>20/!=20` and `Then, Else` swapped
A Any type of brackets or reasonable notation for the array index
DPT If candidate has used a different name any variable then do not award first mark but award subsequent marks as if correct name used.
Refer answers where candidate has used a loop to find position to insert item into stack to team leaders.

4

[13]

Q17.

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

1

(b) *1 mark for naming or describing **two** pointers from this list:*

- Front/start/head pointer
- Next node pointer
- Previous node pointer
- Rear/end/tail pointer
  R Next free space pointer

*1 mark for stating the purpose of **one** of the pointers that have been named:*

- (Front/start/head pointer) to indicate where to remove items from // who should be served next // who is currently being served;
  NE to points to start of list
- (Next node pointer) to link items in list together // to show order of list // so items can be inserted into middle of list // to traverse list;
- (Previous node pointer) to link items in list together // to show order of list // so items can be inserted into middle of list // to traverse list backwards;
- (Read / end / tail pointer) to indicate where to add new items to // so new people can be added to queue
  NE to point to end of list
  A Contextualised answers which refer to queue instead of list or adding people to a queue.
  R Answers which clearly relate to the use of a fixed-size array.

2

(ii) Priority (queue);

1

(c) Allow any reasonable example that would require randomness e.g. time next person joins queue, inter-person arrival time, time to be served, choice of meal, type (student / teacher) of next person to arrive;
R number of students / teachers / people in queue

1

[5]

Q18.

(a) (i) **VB.NET / VB6**

```
   If YCoordinate < 1 Or YCoordinate > 3 Then ValidMove = False
    If ValidMove = True then
      If Board(XCoordinate, YCoordinate) <> " " Then ValidMove
= False
    End If
```

A `If Board(XCoordinate, YCoordinate) = "X" Or Board(XCoordinate, YCoordinate) = "O" Then`

A `If Not(Board(XCoordinate, YCoordinate) = " ") Then`

A `If ValidMove = True` **AndAlso** `Board(XCoordinate, YCoordinate) <> " " Then ValidMove = False` **(VB.NET only)**

Pascal

```
If (YCoordinate < 1) Or (YCoordinate > 3) Then
ValidMove:=False;
If ValidMove = True Then
    If Board[XCoordinate, YCoordinate] <> ' ' Then
ValidMove:=False;
```

Java

```
boolean checkValidMove(int xCoordinate, int yCoordinate,
char[][] board) {
   boolean validMove = true;
   //check the x Coordinate is valid
   if (xCoordinate < 1 || xCoordinate > 3) validMove = false;
   //check the y Coordinate is valid
   if (yCoordinate < 1 || yCoordinate > 3) validMove = false;
   //check the cell is empty
   if (validMove) {
    if (board[xCoordinate][yCoordinate] != ' ')
validMove = false;
   } // end if
  return validMove;
} // end method checkValidMove
```

Python

```
def CheckValidMove(XCoordinate, YCoordinate,Board):
    ValidMove = True
    # Check x coordinate is valid
    if (XCoordinate <1) or (XCoordinate > 3):
        ValidMove = False
   if (YCoordinate <1) or (YCoordinate > 3):
        ValidMove = False
   if (ValidMove == True):
      if (Board[XCoordinate][YCoordinate] != ' '):
          ValidMove = False
    return ValidMove
```

Mark as follows:

IF statement with condition YCoordinate<1, correct logic and second condition of YCoordinate>3;

Return a value of false if y coordinate is an illegal value; **R** if value would not actually be returned;

IF statement checking that move is valid so far;

IF statement comparing value of Board(XCoordinate, YCoordinate) with " ";

returning a value of false if cell is not empty; **R** if value would not actually be returned;

A Equivalent logic

A Alternative answers where Return statements are used after each validation check instead of assigning a Boolean value to ValidMove

Alternative Answer (Java, Python, VB.NET)
Using only one IF statement **and** short-circuit evaluation operators, one mark
for each correct condition plus one mark for correct Boolean operators - as
long as the check that the Board cell is empty is the last condition (if Board
cell is not the last condition marks can only be awarded for any correct conditions that appear before it). Operators for short-circuit evaluation: VB.NET AndAlso/OrElse instead of And/Or; Python and/or instead of &/|; Java &&/|| instead of &/|

Alternative Answer (Pascal)
Using only one IF statement with all conditions connected by OR operators
and the check for non-empty cell being the last condition. If non-empty cell
test is not the last condition maximum of 4 marks.

Alternative Answer
VB.NET / VB6

```
If XCoordinate < 1 Or XCoordinate >3 then
     ValidMove = False
  Else
     If YCoordinate < 1 Or YCoordinate > 3
          Then ValidMove = False
     Else
          If Board(XCoordinate, YCoordinate) <> " " Then
ValidMove = False
     End If
  End If
```

Pascal

```
If (XCoordinate < 1) Or (XCoordinate > 3)
    Then
        Begin
           ValidMove := False;
        End
    Else
        Begin
          If (YCoordinate < 1) Or (YCoordinate > 3)
             Then
                Begin
                   ValidMove := False;
                End
             Else
                Begin
                   If Board[XCoordinate, YCoordinate] <> ' '
Then ValidMove := False;
                End
         End;
```

Mark as follows:
IF statement with condition YCoordinate<1, correct logic and second condition of YCoordinate>3;
Return a value of false if y coordinate is an illegal value; **R** if value would not actually be returned;

Correct use of nested ifs so that checking cell is empty on board only occurs if xcoordinate and ycoordinate are in the allowed range;
IF statement comparing value of Board(XCoordinate, YCoordinate) with " ";
returning a value of false if cell is not empty; **R** if value would not actually be returned
A Equivalent logic
A Alternative answers where Return statements are used after each validation check instead of assigning a value to ValidMove

5

(ii) ****SCREEN CAPTURE(S)****
This is conditional on sensible code for (a)(i)

Mark as follows:
Test showing coordinate (2,-3) and error message;
Test showing coordinate (2, 7) and error message;

R other coordinates
A In VB6 a test showing only Y value of the coordinate i.e. -3, 7 and error message.

2

(iii) ****SCREEN CAPTURE****
This is conditional on sensible code for (a)(i). Mark should not be awarded if code would not work.
E.g. if Boolean values are assigned to ValidMove and there is no Return statement after the validation check.
E.g. trying to reference a position in the array that is out of bounds and would result in an error

Mark as follows:
Screen capture showing board position, coordinates of illegal move **and** error message;

1

(b) (i) **VB.NET/VB6**

```
If Board(2, 2) = Board(3, 3) And Board(2, 2) =
Board(1, 1) And Board(2, 2) <> " " Then xOrOHasWon = True
     If Board(2, 2) = Board(3, 1) And Board(2, 2) =
Board(1, 3) And Board(2, 2) <> " " Then xOrOHasWon = True
```

Alternative answer

```
((Board(2,2)= "X") OR (Board(2,2) ="O"))
instead of <> " "
```

Alternative answer

```
   If Board(2, 2) = Board(3, 3) Then
        If Board(2, 2) = Board(1, 1) Then
             If Board(2, 2) <> " " Then
               xOrOHasWon = True
             End If
        End If
   End If
   If Board(2, 2) = Board(3, 1) Then
        If Board(2, 2) = Board(1, 3) Then
            If Board(2, 2) <> " " Then
               xOrOHasWon = True
            End If
       End If
```

```
   End If
```

Pascal

```
If (Board[2, 2] = Board[3, 3]) And (Board[2, 2] =
Board[1, 1]) And (Board[2, 2] <> ' ') Then xOrOHasWon :=
True;
If (Board[2, 2] = Board[3, 1]) And (Board[2, 2] =
Board[1, 3]) And (Board[2, 2] <> ' ') Then xOrOHasWon :=
True;
```

Alternative answer

```
((Board[2,2]= 'X') OR (Board[2,2] ='O'))
instead of <> ' '
```

Alternative answer

```
If (Board[2, 2] = Board[3, 3]) Then
   If (Board[2, 2] = Board[1, 1]) Then
        If (Board[2, 2] <> ' ') Then
            xOrOHasWon := True;
If (Board[2, 2] = Board[3, 1]) Then
   If (Board[2, 2] = Board[1, 3]) Then
        If (Board[2, 2] <> ' ') Then
            xOrOHasWon := True;
```

Java

```
if (board[1][1] == board[2][2] &&
    board[2][2] == board[3][3] &&
    board[1][1] != ' ') {
  xOrOHasWon = true;
} // end if diagonal
if (board[3][1] == board[2][2] &&
    board[2][2] == board[1][3] &&
    board[3][1] != ' ') {
  xOrOHasWon = true;
} // end if other diagonal
return xOrOHasWon;
```

Python

```
# check diagonals
if (Board[2][2] == Board[3][3]) and (Board[2][2] ==
Board[1][1]) and (Board[2][2] != ' '):
         xOrOHasWon = True # accept return True
if (Board[2][2] == Board[3][1]) and (Board[2][2] ==
Board[1][3]) and (Board[2][2] != ' '):
         xOorOHasWon = True # accept return True
```

Mark as follows:

Comparison of two cells on one diagonal;
Comparison of other cell on the diagonal with one of the two cells just checked;
Check that the line is of Xs or Os (not blanks);
Return True if line of three symbols found on the 1st diagonal;
R if value would not actually be returned
All correct conditions for 2nd diagonal;
Return True if line of three symbols found on the 2nd diagonal;
R if value would not actually be returned
I. additional comparisons of cells – as long as they do not result in check for three symbols in a line not working
Max 4 if diagonal check is inside a loop.

6

(ii) ****SCREEN CAPTURE****
This is conditional on sensible code for (b)(i)

Mark as follows:
Screen capture showing winning message and three symbols in a line in positions [1,1], [2,2], [3,3] // Screen capture showing winning message **and** three symbols in a line in positions [1,3], [2,2], [3,1];

1

(iii) ***SCREEN CAPTURE***
This is conditional on sensible code for (b)(i)

Mark as follows:
Screen capture showing winning message **and** three symbols in a line in positions [1,1], [2,2], [3,3] // Screen capture showing winning message **and** three symbols in a line in positions [1,3], [2,2], [3,1];
R Same diagonal line as shown in part (i)

1

(c) (i) **VB.NET**

```
Else
    Console.WriteLine("A draw this time! ")
    PlayerOneScore = PlayerOneScore + 0.5
    PlayerTwoScore = PlayerTwoScore + 0.5
Endif
```

VB6

```
Else
    MsgBox ("A draw this time!")
    PlayerOneScore = PlayerOneScore + 0.5
    PlayerTwoScore = PlayerTwoScore + 0.5
End If
```

Pascal

```
Else
    Begin
        Writeln('A draw this time!');
        PlayerOneScore := PlayerOneScore + 0.5;
        PlayerTwoScore := PlayerTwoScore + 0.5;
    End;
```


Java

```
} else {
    console.println("A draw this time!");
    playerOneScore = playerOneScore + 0.5f;
    playerTwoScore = playerTwoScore + 0.5f;
} // end if/else
```

Python 2

```
          else:
               print "A draw this time!"
               PlayerOneScore += 0.5 # accept
PlayerOneScore = PlayerOneScore + 0.5
              PlayerTwoScore += 0.5
```

Python 3

```
         else:
              print("A draw this time!")
              PlayerOneScore += 0.5 # accept
PlayerOneScore = PlayerOneScore + 0.5
             PlayerTwoScore += 0.5
```

Mark as follows:
At least one player's score changed within the existing IF statement;
A if in THEN part of NoOfMoves=9 statement
Both scores increased by correct amount;

2

(ii) ****SCREEN CAPTURE****

This is conditional on sensible answer for (c)(i).

Drawn board position with 9 symbols (as defined in preliminary material);
Messages saying players have score of 0.5; **R** other scores

2

(d) (i) **VB.NET**

```
Dim Board(4, 4) As Char
```

VB6

```
Dim Board(1 to 4, 1 to 4) As String
```

Pascal

```
TBoard = Array[1..4,1..4] Of Char;
```

Java

```
char board[][] = new char[5][5];
```

Python

```
Board = [[0,0,0,0,0],
         [0,0,0,0,0],
         [0,0,0,0,0],
         [0,0,0,0,0],
         [0,0,0,0,0],
         ]
```

Mark as follows:
Existing declaration of Board modified correctly;
A No change made as position 0 of array will be used (not Pascal / VB6) – only accept if explanation is given.
A 0..3 instead of 1..4 (Pascal)
A 0 to 3 instead of 1 to 4 (VB6)

1

(ii) **VB.NET / VB6 / Pascal**

```
If NoOfMoves = 16
```

Java

```
if (noOfMoves == 16) {
      gameHasBeenDrawn = true;
}
```

Python

```
if NoOfMoves == 16:
```

Mark as follows: Value of 9 changed to 16;

1

(iii) **VB.NET / VB6**

```
For Row = 1 To 4
    For Column = 1 To 4
```

Pascal

```
For Row := 1 To 4
    Do
        Begin
          For Column := 1 To 4
```

Java

```
for (row = 1; row <= 4; row++) {
    for (column = 1; column <= 4; column++) {
```

Python

```
def ClearBoard(Board):
    for Row in range(1,5):
        for Column in range(1,5):
            Board[Column][Row] = ' '
```

A range(4) if candidate has used 0 for array position instead of 4.

Mark as follows:
Outer FOR loop changed to iterate 4 times **and**
Inner FOR loop changed to iterate 4 times;

A 0 to 3 instead of 1 to 4 – only if indicated 0th position would be used in answer to (d)(i).

1

(iv) **VB.NET**

```
Console.WriteLine(" | 1 2 3 4 ")
Console.WriteLine("--+-------- ")
For Row = 1 To 4
    Console.Write(Row & " | ")
    For Column = 1 To 4
```

VB6

```
BoardAsString = " | 1 2 3 4 "
    BoardAsString = BoardAsString & vbCrLf & "--+--------" &
vbCrLf
    For Row = 1 To 4
        BoardAsString = BoardAsString & Row & " | "
        For Column = 1 To 4
```


Pascal

```
Writeln(' | 1 2 3 4 ');
Writeln('--+---------');
For Row := 1 To 4
   Do
     Begin
        Write(Row, ' | ');
        For Column := 1 To 4
          Do
             Begin
```

Java

```
console.println(" | 1 2 3 4 ");
console.println("--+---------");
for (row = 1; row <= 4; row++) {
    console.write(" | ");
    for (column = 1; column <= 4; column++) {
```

Python 2

```
def DisplayBoard(Board):
    print ' | 1 2 3 4 '
    print '--+---------'
```

```
    for Row in range(1,5):
        print str(Row) + '| ',
        for Column in range(1,5):
           print Board[Column][Row]
        print
   print '\n'
```

Python 3

```
def DisplayBoard(Board):
    print(' | 1 2 3 4 ')
    print('--+---------')
    for Row in range(1,5):
        print(Row, '|', end=' ')
        for Column in range(1,5):
            print(Board[Column][Row],end=" ")
        print()
   print('\n')
```

A range(4) if candidate has used 0 for array position instead of 4.

Mark as follows:
Change message so that 4th column heading is shown;
Outer FOR loop changed to iterate 4 times **and**
Inner FOR loop changed to iterate 4 times;

A `0 to 3` instead of `1 to 4` – only if indicated 0th position would be used in answer to (d)(i).

2

(v) ****SCREEN CAPTURE****
This is conditional on sensible answers for (d)(i) and (iv)

displays 4 rows;
displays 4 columns;

2

(vi) **VB.NET / VB6**

```
If XCoordinate < 1 Or XCoordinate > 4 Then ValidMove =
False
If YCoordinate < 1 Or YCoordinate > 4 Then ValidMove =
False
```


Pascal

```
If (XCoordinate < 1) Or (XCoordinate > 4) Then ValidMove
:= False;
If (YCoordinate < 1) Or (YCoordinate > 4) Then ValidMove
:= False;
```

Java

```
if (xCoordinate < 1 || xCoordinate > 4) validMove = false;
//check the y Coordinate is valid
if (yCoordinate < 1 || yCoordinate > 4) validMove = false;
//check the cell is empty
```

Python

```
def CheckValidMove(XCoordinate, YCoordinate, Board):
    ValidMove = True
    if (XCoordinate <1) or (XCoordinate > 4):
        ValidMove = False
   if (YCoordinate <1) or (YCoordinate > 4):
        ValidMove = False
   if (ValidMove == True) and
```

```
(Board[XCoordinate][YCoordinate] != ' '):
        ValidMove = False
   return ValidMove
```

Mark as follows:
Change upper boundary to 4 for both X **and** Y coordinates;

A Change lower boundary to 0 for both X and Y coordinates instead of upper boundary change – only if indicated 0th position would be used in answer to (d)(i);

1

(vii) **VB.NET / VB6**

```
For Row = 1 To 4
    If Board(2, Row) = Board(3, Row) And (Board(2, Row) =
Board(1, Row) Or Board(2, Row) = Board(4, Row)) and Board(2,
Row) <> " " Then xOrOHasWon = True
Next
```

Pascal

```
For Row := 1 To 4
    Do
        If (Board[2, Row] = Board[3, Row]) And ((Board[2,
Row] = Board[1, Row]) Or (Board[2, Row] = Board[4, Row]))
And (Board[2, Row] <> ' ')
        Then xOrOHasWon := True;
```

Java

```
for (row = 1; row <= 4; row++) {
    if (board[1][row] == board[2][row] &&
          board[2][row] == board[3][row] &&
          board[2][row] != ' ') {
          xOrOHasWon = true;
    } // end if
     if (board[2][row] == board[3][row] &&
          board[3][row] == board[4][row] &&
          board[row][2] != ' ') {
          xOrOHasWon = true;
    } // end if
} // end column
```


Python

```
if (Board[2][Row] == Board[3][Row]) and (Board[2][Row]
= = Board[1][Row]) or (Board[2][Row] = = Board[4][Row])
and (Board[2][Row] != ' '):
          xOrOHasWon = True
```

Mark as follows:
Change FOR loop so it iterates 4 times;
Board(4, Row); compared with Board(3, Row)/Board(2, Row);
Solution works for all 8 legal winning positions on the rows;

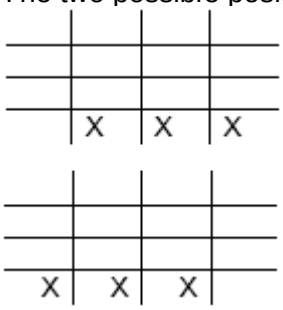
A Two loops (both go from 1 to 4) – both loops need to be included in the
code shown by the candidate to get full marks
A Additional IF statements, as long as logic is correct
Max 3 4 IF statements instead of a FOR loop – one IF statement for each
row in the grid
Max 2 if only works for four symbols in a row
Max 2 if solution detects a winning solution when it shouldn't

A Answers coordinates using 0 instead of 4 – only if indicated 0th position would be used in answer to (d)(i).

4

(viii) ****SCREEN CAPTURE****
This is conditional on sensible answers for (d)(i), (iv) and (vii).

Symbol shown in (2,4);
Winning message shown and three symbols in a horizontal line including a symbol in position (2,4); **R** if solution for 45 is for four symbols in a line, not three
The two possible positions for full marks (could be O instead of X):



A If candidate has used array position 0 instead of 4, accept a winning position on either the bottom or top line of the board.

2

(ix) Declare Board as a 3-dimensional array; Board(4,4,4) / /Board (6,4,4);
OR
Declare 6 (one for each surface); 4x4 arrays;
OR
Declare 4; 4x4 arrays;

NE. 3D
A. Answer that imply creating a new data type / using array structure that will be used with the Board variable; that allows 64/96 cells to be represented;

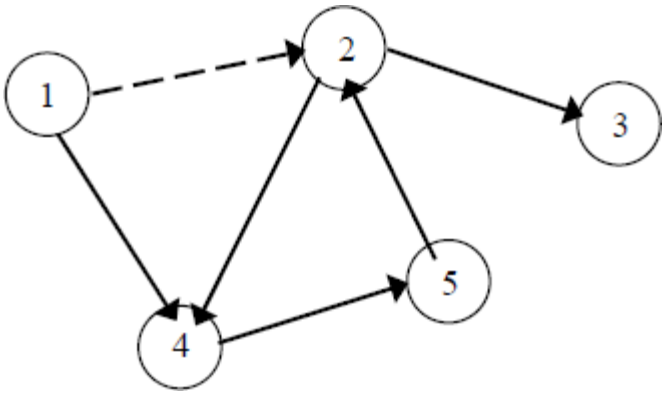
Description of further list nesting (similar to 3d array) **(Python only)**

2

36

Q19.

(a)



1 mark for all 5 lines correctly drawn

1 mark for all 5 arrowheads pointing in correct directions
Max 1 if more than 5 lines drawn by candidate (note that dotted arrow is given in question)

A arrowheads at any position on line

2

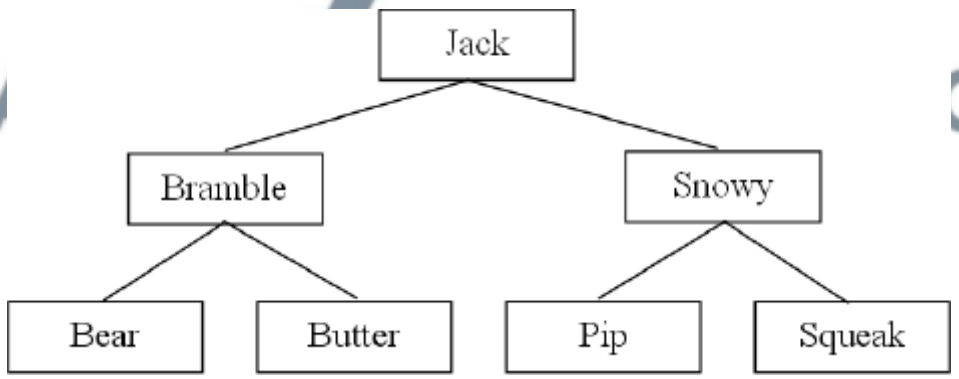
(b) Adjacency matrix appropriate when there are many edges between vertices // when edges may be frequently changed // when presence/absence of specific edges needs to be tested (frequently)
Adjacency list appropriate when there are few edges between vertices // when graph is sparse // when edges rarely changed //when presence/absence of specific edges does not need to be tested (frequently)
A alternative words which describe edge e.g. connection, line

2

(c) Connected // There is a path between each pair of vertices;
Undirected // No direction is associated with each edge;
Has no cycles // No (simple) circuits // No closed chains // No closed paths in which all the edges are different and all the intermediate vertices are different // No route from a vertex back to itself that doesn't use an edge more than once or visit an intermediate vertex more than once;
Alternative definitions:
Graph with no cycles, and a simple cycle is formed if any edge is added to it;;
Graph which is connected, and it is not connected anymore if any edge is removed from it;;
Graph in which any two vertices can be connected by a unique simple path;;
(Note: not just adjacent vertices)
Graph which is connected and has n - 1 edges where n is no of vertices;;
Graph which has no simple cycles and has n - 1 edges where n is no of vertices;;

Max 2

(d)



1 mark for Jack as root
1 mark for Bramble and Snowy as children of Jack
1 mark for four correct children of Bramble and Snowy

DPT if arrows drawn instead of lines
DPT if any node has more than 2 child nodes
A "mirror image" answers which are consistent.

3

(e) **For solution with 3 arrays:**
One array stores data items;
One array for left child pointers;

One array for right child pointers;
Pointers stored at same location in arrays as corresponding data item;
For solution with 1 array of records:
Record created to store data item and pointers;
One pointer to left child;
One pointer to right child;
For either of the above solutions:
Rogue value (allow example) used to indicate no child;
Variable indicates position in array(s) of root node // Root node stored at first location/start of array(s);
If answered as diagram:
Column for data with at least three correct data items in it;
Use of rogue value for a node that does not have child;
Correct value for a start pointer variable indicating position of root node in the array (not drawn as an arrow, array indices must be labelled);
Column for left child pointers*;
Column for right child pointers*;
* = To get these marks, there must be a sufficient number of pointers to demonstrate that the data structure is a representation of a binary tree, but it is not necessary for every item to be shown. Also the array indices must be shown.

Max 3

[12]

Q20.

(a)

				List				
ListLength	New	P	q	[1]	[2]	[3]	[4]	[5]
4	*38*	-	-	*9*	*21*	*49*	*107*	
		1						
		2						
		3						
			4					107
			3				49	
						38		
5								

4,5 in sequence for ListLength;
1,2,3 in sequence for p;
4,3 in sequence for q;
Final list in array is 9, 21, 38, 49, 107;

Do not award a mark if additional values indicated e.g. 4 for p

4

(b) Inserts an item/variable New into list at correct position/preserving order//into sorted list (or equivalent);

1

(c) (i) Static structures have fixed (maximum) size whereas size of dynamic structures can change // Size of static structure fixed at compile-time whereas size of dynamic structure can change at run–time;
Static structures can waste storage space/memory if the number of data items stored is small relative to the size of the structure whereas dynamic structures only take up the amount of storage space required for the actual data;
Dynamic data structures (typically) require memory to store pointer(s) to the next item(s) which static structures do not need;
A just one side of points, other side is by implication

Max 1

(ii) Heap is pool of free/unused/available memory;
Memory allocated/deallocated at run-time (to dynamic data structure);

Max 1

[7]

Q22.

(a) (i) Picture element // smallest resolvable/rectangular area/unit (**A** smallest dot) which can be drawn on screen // smallest addressable part/unit of a picture;
smallest unit which is mapped to memory;

1

(ii) Pixels are stored as numbers/bit patterns (**A** values) which represent different colours;
A or by example;

1

(b) 1;

1

(c) (picture / image) width;
(picture / image) height;
A (picture / image) dimensions
R size
image resolution / colour depth / No. of bits per pixel;
colour palette / No. of colours in image;
offset to the start of image data;
compression type;

Max 2

(d) (i) loop counter / (loop) control variable // array subscript/index;
array of Byte;
A array of Integer

2

(ii) 1101;
I. any additional leading 0's

1

(e) (i) ThisWidth;
X;

2

(ii) 2-dimensional array (of Byte);

1

(f)

ThisWidth	ThisHeight	Counter	X	Y	ThisByte		Final
8	5	0	1	1	255	[0]	25
				2	255	[1]	96
				3	255	[2]	96
				4	255	[3]	24
				5	255	[4]	24
				6	255	[5]	113
				7	255	[6]	
				8	255	[7]	
			2	1	255	[8]	
				2	25	[9]	
		1		3	25	[10]	
		2		4	96	[11]	
		3		5	96	[12]	
		4		6	24	[13]	
		5		7	24	[14]	
		6		8	113	[15]	

Mark as follows:
Counter has incremented from 0 to 6 (only);
X variable has incremented 1 and 2 (only);
Y variable has incremented 1 – 8 (only) at least once;
ThisByte contains first ten correct values;
Final[0] contains 25;
Final[1] to Final[5] are correct and with no other array subscripts used;

A correct six values (only) in Final array (in consecutive but wrong positions)

Max 6

(g) (i) program / constant / module / unit / user defined type / label /object / component / control / class;

A 'control' by example e.g. text box, drop down list
A any elements which are SQL specific

1

(ii) Maximum number of characters;
No punctuation characters;
No use of reserved words;
Must not start with a digit character;
case critical e.g. must start with lower case character;

A any answer which describes 'general' programming language restrictions.
identifier names must be unique;free-format not allowed for certain constructs, e.g. statement must not spread over two lines;
restrictions on identifiers used for labels;
loop control variable must be ordinal/integer;
array index range is restricted;
all variables must be pre-declared;

Max 2

[20]

Q23.

(a) (i) Empty entries waste memory // Maximum size// fixed size;

1

(ii) Memory used by pointers//takes more time to add / delete nodes//
indirect access takes more time;
R programming difficulties

1

(b) Place next item in first location/ location 0/ location 1//
Implement a circular array/queue // allow wraparound;

1

(c) IsFull/IsQueueFull;

1

[4]

Q24.

(a)
```
CarFailed : = False
Input NextCar
For Position ← 1 to 4
     Do
     NextCategory ← SingleCharacter( A. NextCar; , B. Position; )
     If C. NextCategory = '0'/ NextCategory <> '1';
     Then CarFailed ← True
```

Etc…
Part C – **I.** omission of quotes **A** double quotes

3

(b)

Variable	Data Type	Comment
Position	D **Integer**;	E **loop counter/loop control;** **Takes the range of value 1, 2, 3 and 4;** **Indicates the current test/category or**

		suitable description; **Provides an index for the <u>string</u> // indicates the position in the <u>string</u>;**
`NextCar`	String	
`NextCategory`	F **Char;**	
`CarFailed`	Boolean	

3

[6]

Q25.

(a) <u>2–D</u> array;

1

(b) Shows that <u>sales person 2</u>; <u>did meet their target;</u> for <u>Quarter 3</u> /July – September;

Max 1

(c)

Person	Quarter	Target [Person, Quarter]	NewArray [1]	[2]	[3]	[4]
1	1,2,34		0	0	0	0;
1	1	Y				
	2	N		1;		
	3	Y				
	4;	N				1
2	1	N	1			
	2	N		2		
	3	Y				
	4	Y				
3;	1	N	2;			
	2	N		3		
	3	N			1	
	4	N				2;

NewArray initial values all 0;

1

Person loop counter 1 to 3;

1

Person 1 – is followed by quarters 1 to 4 in sequence;

1

NewArray[2] = 1 for person = 1 and Quarter = 2;

1

Final NewArray[1] = 2;

1

Final NewArray[2 and 3 and 4] values are correct;

1

(d) Stores the (total) number of sales staff who did not meet their target // the(total) number of sales targets not met;
for <u>each</u> quarter;

2

[10]

Q26.

(a) Last (item) in, is the first (item) out / first (item) in is the last (item) out ;
R LIFO / FILO

1

(b) (i)

600	'A'
601	'V'
602	'E'
603	'R'
604	'Y' ;
605	

All items in the correct locations

1

(ii)

599	
600	'A'
601	'V'
602	'E' ;
603	
604	
605	

Correct three items // ft from an incorrect (i) including 605 as the first

location used ;
A 'R' and 'Y' entries indicated in some way as 'deleted'

1

(iii)

600	'A'
601	'V'
602	'E'
603	'S'
604	'P' ;
605	

Correct list of five items // ft from an incorrect (i) + a correct ft (ii) including 605 as the first location used ;

1

(c) (i) Queue ;
A First In – First Out FIFO / LILO

1

(ii) Items are removed/popped from the stack (one at a time) (and items are then added to the queue);

1

(iii) Items leave the queue on a 'first in-first out' basis ; **A** from the front of the queue

1

(iv) 'Y', 'R', 'E', 'V', 'A' on the queue ;
Y', 'R', 'E', 'V', 'A' on the final stack ;
A using 701 for the first queue location

2

[9]

Q27.

(a) A procedure that is defined in terms of itself;
A A procedure that calls itself
R re-entrant

1

(b) Store return addresses;
Store parameters;
Store local variables/ return values;

Max 1

(c)

Number	Entry	Output
11	1	

11	2;	
11	3;	
11	4;	4;

4

(d) A linear search//
To find/output the position/index of Number in Items;

1

(e) Number is not an entry in Items// Stack overflows;

1

(f) Test for reaching the end of Items;

1

(g) Binary Search;
An iterative solution;

Max 1

[10]

Q28.

(a) Next item to be added is at position/location/address (Tail + 1);
Position/location/address Tail is the last item in the queue ;
R 'points to the end of the queue'

Max 1

(b) Cat // item at position Head ;

1

(c)

	6	
Tail →	5	'Shark'
	4	'Eel'
	3	'Snake'
Head	2	'Frog'
	1	'Dog'
	0	'Cat'

Snake + Eel + Shark at positions 3,4,5 ;(1)
Tail points to 5 ;(1)
Head points to 2 ;(1)
I. Dog and Cat crossed through

3

(d) Tail will eventually reach position 99 (**A** 100) ;

Head will eventually reach 99 (**A** 100);
Memory/queue will become full ;
Space is not re-useable ;

Max 2

[7]

Q29.

(a) Temp ← Front;
Front ← Temp.Next//Front ← Temp^.Next;
Dispose (Temp); **A** Free(Temp)

Alternative

Temp ← Front.Next// Temp ← Front^.Next;
Dispose (Front); A Free(Temp)
Front ← Temp;

3

(b) AddItem//Add;

1

(c) (i) Full/FullQueue;

1

(ii) No memory used for pointers;
I Faster
R Easier to program

1

(iii) Size is limited by array size;
memory wasted when not full;

2

[8]

Q30.

(a)

Number	Lower	Upper	Current
12	1	9	
		5	5
	3		3
	4	4	4

Value returned	4

1 mark for 1st row (12, 1, 9)
2 marks for second row (1 mark for each 5)
2 marks for 3rd row (3 and 3)
2 marks for 4th row (1 mark for Lower = 4, 1mark for upper = 4)
1 mark for correct return value

8

(b) Find the position of 12/ a number in the array// search for 12/ a number in the array;

1

[9]

Q31.

(a) Salesperson 7;
April /month 4;
The number of storecards 'taken out';

Max 2

(b) StoreCards + sensible subscripts [1..10, 1..6] / (1 to 10, 1 to 6) / [0..10, 0..6] / (0 to 10, 0 to 6) / (10,6) / [10] 6];
StoreCards + Integer / Byte;

2

(c) StoreCards (8, 1);
= 13 / := 13 / ← 13;
Must be an assignment statement

2

(d) Key in / Input the employee number; the program calculates the total number of store cards for a single person // print/outputs/displays the total for a single person; over six months;

Max 2

(e) (i) Single / real / float;
R Floating point / Double

1

(i) Boolean /Yes-No / True-False; **R** Y/N / T/F

1

(iii) Integer/ byte;

1

[11]

Q32.

(a) (i) 101 0110;

1

(ii) 1101 0110 / or follow-through from 'their 7 bit code' from (a);
A Parity bit positioned in bit position 0

1

(b) (i) 'D';
R Lower case

1

(ii) 'J';
R Lower case

1

(c) (i) FirstName // Surname;

1

(ii) Surname // FirstName; (i) and (ii) must be different

1

(iii) FullName;
I. any incorrect case

1

(d)

Position	NextNumber	NextChar	FinalString
			‘’
1	(65)	‘A’	‘A’
2	78	‘N’	‘AN’
3	32	<Space>	‘AN ‘
4	69	‘E’	‘AN E’
5	82	‘R’	‘AN ER’
6	82	‘R’	‘AN ERR’
7	79	‘O’	‘AN ERRO’
8	82	‘R’	‘AN ERROR’

Position values incrementing to at least 4;to maximum 8;
NextNumber[2] has value 78;
Remaining NextNumber values are all correct and in correct positions;
NextChar[3] has <Space> character + NextNumber[3] is 32;
FinalString correct / f/t from their NextChar column;

6

[13]

Q33.

(a) (i) 271;

1

(ii) The required item might be the 271st one/last one/ not be present// Every item accessed;

1

(b) (i) 9;

1

(ii) Each comparison halves the number of items to be accessed//271 lies between 2^8 and 2^9;

1

(c)

Count1	Count2	Temp	A				
			[1]	[2]	[3]	[4]	[5]
-	-	-	23	45	16	12	31
1	1						
	2	45		16	45		
	3	45			12	45	
	4	45				31	45
2	1	23	16	23			
	2	23		12	23		
	3						
	4						
3	1	16	12	16			
	2						
	3						
	4						
4	1						
	2						
	3						
	4						

1 mark

1 mark

1 mark for Count1
1 mark for Count2
1 mark for Temp

5

(ii) (bubble) sort the items into ascending order;

1

(iii) Reduce the number of tests each pass// stop when no swaps occur during a pass//Add a flag No Swaps to indicate when no swaps occur// change loop control to Repeat until no swaps// sort variable sized array;

1

[11]

Q34.

(a) (i)
- poorly structured code;
- uses GoTo statements;
- the flow of control jumps out of a loop;
- nothing reported to the user when no matching name found;
- abbreviated variable for 'position' variable;
- ReadLn is better than Read;
- Program only iterates once / considers only the first array element;
- (if duplicates) only the first matching surname is found;
- (loop terminates at 20) does not allow for additional array /name entries;

A poor layout - excessive indentation used;
I. variable declaration // reference to the syntax

Max 2

(ii) All statements must have correct identifier name correct data type (String / Text // Integer / Byte / Word / Int / Shortint / Short as appropriate)

In addition, either array must have brackets to indicate an 'array' 19/20 to indicate a range;

Max 2

(b) Intialisation of counter or Boolean variable
P := 1 / P := 0 / For P := 1 to 20 // IsFound := False;

Looping
LOOP UNTIL // DO WHILE // WHILE DO // REPEAT UNTIL and used at the beginning/end of a code block as appropriate;

Some loop condition is met
(P = 20/21) OR IsFound = TRUE / P = 20/21 // IsFound = TRUE / IsFound;

IF with use of the array
IF NoOfClaims [P];

Selection condition
>4 / >=5;

Loop counter incremented
P = P+1

Final output
Correct logic followed with OUTPUT 'Yes'
A multiple times

Final output
Correct logic followed with OUTPUT 'No'
R Multiple times
R 'Prose' scores 0

5

[9]

Q35.

(a) (i) Empty entries waste space // Maximum/fixed/static size
A stack may overflow

1

(ii) Space used by pointers // more complex to program;

1

(b) (i) The size of the stack /amount of data is known/limited/predictable
Memory saved since no pointers (if not given in a (ii))
R easier to program

1

(ii) The size of the stack is unknown//
The stack is volatile/ number of items fluctuates widely;

1

[4]

Q1.

The difference(s) between static and dynamic data structures was generally well understood. A common answer that was not awarded marks was to simply state that static data structures cannot be changed, this does not make it clear that the size cannot change.

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.

Q6.

This question was about the use of hashing.

In part (a) candidates had to compare the efficiency of searching a hash table with searching an unordered list. There were many good responses to this which explained that a slow linear search would be required for an unordered list but a fast calculation of a hash value is all that would be needed for the hash table implementation, and using this the location of the translation could be directly found.

For part (b) candidates had to explain what a collision was and how it could be dealt with. The majority of candidates appeared to understand both of these but some failed to achieve marks by not stating points explicitly. For example, too many candidates failed to explain the basic point that if two items hashed to the same value then they would be stored at the same location, and the second value would overwrite the first. Various sensible methods of dealing with a collision were well described.

Part (c) required candidates to explain why the English word had to be stored in addition to the French word. Some correctly identified that when performing English to French translation, if two English words had hashed to the same value, it would not be possible to tell which the correct translation was unless the English word was stored. A small number of candidates incorrectly believed that the translation was being done in reverse (French to English) and explained that the hash function would be one-way, which whilst true was not a correct answer to the question that had been asked.

Q7.

Answers to Section C were often of poor quality and very few students achieved good marks on this question. A number of students are still including additional code when asked for the name of an identifier (parts (a) – (c), though there were fewer students this year who were doing this. This means that they are not getting the marks for these questions as they have not made it clear which entity is the identifier (sometimes there is more than one identifier in the lines of code that they have copied from the Skeleton Program).

Parts (d) – (f) were not well answered. Many students could find one error in the decision table for part (f) but few could find more than one. Answers to parts (d) and (e) were often vague with many students providing answers that were about different parts of the Skeleton Program from the ones asked for in these questions.

Q8.

(a) This was a fairly straightforward programming question with most students getting good marks. Some students did not read the question carefully and created a selection structure instead of a loop that would repeatedly get a value from the user until a valid value was entered. A number of answers were seen where a recursive solution was attempted but the name entered was not actually returned to the calling routine.

A significant number of students did not complete the test specified in the question, often entering their own name as test data.

(b) Most students got reasonable marks on this question. Less able students sometimes got confused between the < and > operators and a number of students only compared the suits of the two cards – forgetting to compare for rank equality.

(c) This was a more challenging question and was a good discriminator between students. It was pleasing to see some interesting answers to this question where able students had clearly thought through the problem and come up with their own method for solving it under exam conditions.

Most students were able to adapt the code so that it would allow a joker to be played, though a number did not attempt to write code that would limit the number of jokers that could be played.

(d) It was disappointing that a large number of students did not include any attempt at answering the question. There was a mark available just for creating a correctly-named subroutine (even if the subroutine did not do anything or use any parameters) and a mark for displaying a message (even if the message did not include the calculated probability). Students should be encouraged to include partial solutions to questions they have not been able to answer wholly successfully.

As was the case for the last few years, less able students often struggled to create a new subroutine even though there are numerous examples of subroutines in the Skeleton Program. Again, a number of students developed a solution that would correctly calculate the probability but just included code inside the subroutine that displayed this value rather than setting up a mechanism to return the calculated number to the calling routine.

Q9.

(a) The overwhelming majority of students were able to correctly identify that it was the binary search algorithm that required the list to be sorted for this part.

(b) The trace for this part was also well completed with about three quarters of students getting some marks and well over half getting full marks.

(c) For this part, around half of the candidates were correctly able to explain that the value of InnerPointer did not decrease to zero because either the second while loop condition was not

(d) For this part, about two thirds of students correctly identified that the algorithm that they had traced was of time complexity $O(n^2)$.

(e) This part was poorly answered, with only about one third of students correctly identifying that the algorithm they had traced was an insertion sort. Bubble sort was a far more common but incorrect response.

(f) Parts (i), (ii) were all well answered. The most common error in both parts of was to perform a traversal of the tree instead of using it as a binary search tree.

(g) This part was all well answered. The most common error in both parts of was to perform a traversal of the tree instead of using it as a binary search tree.

Q10.

(a) This part was well tackled with over two thirds of candidates recognising that a queue was an appropriate data structure to represent the deck of cards because it was a first-in-first out structure.

(b) (i) Just over half of the candidates were able to correctly update the pointers to the queue in this part

(ii) And just under half were able to do likewise in this part. In the former, the most common mistakes were to change FrontPointer to 10 instead of 11 and to leave QueueSize at 52. In the latter, the most common mistake was to change RearPointer to 54, failing to recognise that this was a circular queue and RearPointer should change to 2.

(iii) Students achieved a broad range of marks on this part which required them to write an algorithm for dealing a card. Over two thirds got some marks, but just under 10% scored full marks. The most commonly made mistakes were to try to deal more than one card, to fail to wrap FrontPointer back to the start of the array when it went past the end of it, and to deal a card when the deck was empty.

(c) This part was well answered, with most students recognising that in an event-driven program, subroutines would be associated with events, such as a button being clicked or a timer reaching 0, and that the appropriate subroutine would be called when the event occurred. Some responses were not considered creditworthy as they could have applied to any style of programming, for example “the program waits for user input before executing instructions”.

(d) Approximately two thirds of students achieved some marks for comparing a mobile phone operating system and a desktop operating system in this part of the question. A common mistake was to compare the devices rather than the operating systems, for example stating that the desktop would have more memory that the mobile phone. Many students explained that the mobile phone OS would have lower hardware requirements but it would have been nice to see some more concrete examples of these reduced requirements, and phrases such as “smaller” and “more lightweight” were used too often. Some students gave responses to a question from an earlier paper that were not appropriate for this scenario.

Q11.

For part (a) almost all candidates were able to identify the content of the root node correctly. Leaf nodes were more problematic, with many candidates believing that all of the nodes that were not the root were leaves, and thus mistakenly included “+" in the list of contents of leaf nodes. Nevertheless just over half of the candidates achieved full marks for this question part.

Part (b) was poorly tackled, with only a third of candidates getting more than one mark. The most common error was to state that arrays B and C were used to store the left / right hand subtrees. Candidates needed to make clear that the arrays were storing pointers to these subtrees to be awarded the marks.

For (c), responses were disappointing with many candidates showing no understanding of the purpose of the entry 0 in the array. Others stated that it indicated that a node was a leaf node, or had no children, when in fact the 0 in array B only related to the left hand subtree so nothing could be told from it about whether or not a node had children without also consulting array C.

Part (d): candidates usually find recursive traces to be quite difficult. On this occasion, about half of them got at least one mark for the trace, and a quarter achieved all four marks.

For part (e), about a third of candidates made the expected response, but a

disappointingly large number wrote answers that were not even types of tree traversal.

For part (f), almost all of the candidates who had correctly completed the trace table and obtained the output recognised that this was the Reverse Polish Notation equivalent of the infix expression in the expression tree.

Q12.

For (a), about three quarters of the candidates achieved one or two marks for this question part. The most common creditworthy response was that the size of a dynamic data structure could change at run time whereas a static data structure's size was fixed at compile time. Some candidates missed out on this mark by making the vaguer statement that a dynamic structure could change at run time whereas a static structure could not, without any reference to size. Some candidates lost a mark by just giving opposite sides of the same point.

Question (b) was well answered. Some candidates missed out on one mark scheme point by writing that all elements in the array would have to be moved down, without making clear that it was only the elements below the insertion point that needed to move. Many of this group of candidates nevertheless achieved full marks by making other valid points. A small but surprising number of students wrote about deleting an item when the question was about inserting one.

For part (c), just over half of the candidates correctly identified that this was a priority queue.

For part (d)(i), only about a quarter of the candidates offered a reasonable explanation of what the heap would be used for. Many stated that it would be used to store the list or to store pointers. Some asserted that it would be used to store new items, which was a better response but was not creditworthy as the suggestion was that the new items would be stored in the heap. Good responses recognised that when a new item was added, memory would be claimed from the heap to store the new item.

Part (d)(ii) was poorly answered with only about a third of the candidates recognising that the pointer value was a memory address. Answers referring to a value being an index or location were not enough for a mark without further explanation that made clear that this was to a memory location.

In part (d)(iii), this algorithm was the most complex one that candidates have been asked to write on a COMP3 paper. Answers given in pseudo-code or structured English were acceptable and all reasonable styles of syntax were marked as we are aware that candidates will have used different programming languages. Responses written in prose were acceptable if they included clear steps and showed how the variables would be updated. There were very few prose answers, but they often scored poorly due to being too general.

There were some very good responses and marks were awarded later on in the algorithms even if earlier parts were not working, although this was not always possible as sometimes whether or not later parts of the given algorithm worked depended fundamentally on earlier parts.

The most common misconception was to treat the linked list as if it were a list stored at sequential locations in an array and to adjust the CurrentNodePointer by adding one onto it inside a loop rather than by setting its value to the NextNodePointer of the current node. It remained possible to pick up some marks for this type of implementation by, for example having a suitable loop.

Very few candidates, even those achieving almost full marks, included the step of actually claiming the memory from the heap to store the new item in.

A small number of candidates produced recursive solutions which were also accepted.

Q13.

For the first time a flowchart was used to represent an algorithm in a COMP1 exam. There was no increase in difficulty resulting from this and the standard of answers was the same as seen in the previous year.

Some students did not follow the algorithm given and instead developed their own program to convert binary to denary. This resulted in them not getting many marks as they had not answered the question.

Students using VB6 tended to get lower marks on this question than those using the other languages available for COMP1. This was partly due to not providing the correct evidence for the testing (screen captures needed to show the data entered for the test as well as the result of the test), although many students using VB6 also seemed to have weaker programming skills.

Students need to be aware that an algorithm is not the same as a program and that simply copying the algorithm into their development environment will not result in a working program in any of the COMP1 programming languages – the pseudo–code/flowchart needs to be adapted to match the syntax of the programming language they are using. As in previous years, a number of students simply copied parts of the algorithm into their program code eg trying to use a keyword of OUTPUT. These appeared to be less able students who generally struggled on the Section D programming as well. The vast majority of students were able to convert the algorithm successfully into working program code and the marks obtained on this question were virtually identical to those achieved on Section B on the 2011 COMP1 exam.

Q14.

Answers to this section were often of poor quality and very few students achieved good marks on this question.

A number of students are still including additional code when asked for the name of an identifier. This means that they are not getting the marks for these questions as they have not made it clear which entity is the identifier (sometimes there is more than one identifier in lines of code that they have copied from the Skeleton Program). To reduce the chance of errors, when asked to give the name of an identifier students should be encouraged to copy and paste the identifier from the Skeleton Program, rather than typing the identifier into the EAD.

Very few students showed any understanding of binary files, even though these were used in the Skeleton Program. Part (a) was answered better than most other parts of Section C with most students able to give at least one reason why the use of global variables should be avoided. The majority of students were also able to state an advantage of using a named constant.

Q15.

Most candidates were not well prepared for this section and did not do as well on these questions about the Skeleton Program as they did on the questions where they were asked to modify the Skeleton Program. In particular, little understanding of structure charts or decision tables was shown by a significant number of candidates.

It was pleasing to note that most candidates only gave the name of an identifier when asked to do so – those who copied and pasted sections of code from the Skeleton Program did not get the marks for these questions as they had not demonstrated that they understood what an identifier is (some candidates gave answers that contained multiple identifiers). Some candidates did not get the mark for giving an example of a constant declaration as they provided only the name of the constant. Candidates should ensure that when asked for the name of an identifier they provide only the identifier in their answer and when asked for an example of a type of program statement that the entire program statement is given in their answer.

For part (n) many candidates described the repetition structure rather than the selection structure inside the repetition structure.

Q16.

Part (a): This question part was very well answered with the majority of candidates getting both marks. The only common mistake was to miss out the brackets in the expression that should be (12+19)*8.

Part (b): As with part (a), this question part was also well answered. The most common correct response was that brackets are not required. It would have been nice to see some more detailed explanations of this point, rather than just a brief statement of it. A common incorrect answer was that RPN was easier for a computer to understand. The word "understand" is not appropriate in this context.

Part (c): Responses to this question part were excellent, with relatively few errors made. The majority of candidates got full marks which is unusual for a question involving a trace table. The only two recurring mistakes were to pop the numbers off the stack in the wrong order, resulting in the transposition of the values in Op1 and Op2 and forgetting to push 50 back onto the stack at the very end.

Part (d): This question was well answered, with most candidates getting some marks and a significant number more than half marks. The most common mistake was to increment the `TopOfStackPointer` in the wrong place – either before the If construct which tested for the stack full condition or after the value in `ANumber` was stored into the `StackArray`. Some candidates implemented solutions that used a loop to find the first empty position in the array to insert the number into. These were awarded credit if they would have worked, but many failed to test properly for the stack being full. It is important that candidates use the correct variable names when they are given on the question paper.

Q17.

Part (a): This question part was poorly answered with many candidates giving vague responses or explaining what a simulation is rather than a model. In this context, a model is an abstraction of the real-world problem that leaves out unnecessary details. Some candidates confused a model with a prototype.

Part (b)(i): Again, this question part was poorly answered. A significant number of candidates appeared to have no understanding of what was being asked, although more than half got at least one mark. Candidates who made a reasonable attempt at an answer often named two pointers, but then offered inadequate explanations of their purpose. For example, the purpose of the pointer to the end of the list is to enable new items to be added to the list, not simply to know where the end is.

Part (b)(ii): Some candidates correctly identified that a priority queue was required, but many invented new types of queues.

Part (c): This question part was well answered with many candidates giving well thought out answers such as determining whether the next person entering the cafeteria was a student or teacher or generating a time taken to serve the person at the front of the queue. The most common incorrect answer was the number of people / students / teachers in a queue. In each case, the number in a queue would be a consequence of other randomly determined occurrences rather than determined randomly itself.

Q18.

(a) The checks for a valid `YCoordinate` were done correctly by most candidates. Some candidates dropped marks by having code that would not return the correct value from the function (by adding the validation checks after the value was assigned to the function) or by combining the `XCoordinate` and `YCoordinate` checks in one statement with an AND operator (this would not work unless brackets were added in the correct places).

The check for overwriting moves was harder and was not done as well as the `YCoordinate` check. Code that would not compile was often seen. Many candidates did not ensure that the overwriting of moves was only checked for if the coordinates were valid – this would result in checking an out-of-bounds position on an array which could cause the program to crash when run (e.g. VB.Net) or to return spurious results by checking a different memory location (e.g. Pascal). A few candidates (mostly in Java and C#) used exception handling to deal with this problem. While this was not on the mark scheme it was deemed to be worthy of the mark available, though it would be better practice to write code where exception handling was not needed.

Some candidates had either code that would not compile for the overwriting check or code that would crash when tested with an out-of-bounds coordinate but they had included screen captures for part (ii). Marks were not awarded for part (ii) in these cases as the marks were dependent on the code from part (i) – these candidates had run a different version of their code for their testing from that they had included for part (i).

(b) Most candidates did very well on this question and had obviously anticipated that this would be asked and prepared for it accordingly.

Some answers clearly demonstrated that checking for a win on a row/column being in a loop had not been understood, as they put the check for a line in a diagonal in a loop that repeated three times unnecessarily e.g.

```
For Diagonal = 1 To 3
    Do
        If Board(1,1)= Board(2,2) And Board(2,2) = Board(3,3)
            And Board(2,2)  " " Then XorOHasWon := True
```

(c) Most candidates answered this question well. A few dropped marks for part (ii) by showing a drawn position for a second or third game in a match. Part (i) asked for the code for the selection structure used in the Skeleton Program – if this was not included (i.e. candidate only included the code for adding to the scores) then only one mark could be awarded. Some candidates added a new selection structure rather than amending the existing structure as asked for in the question – again only one mark was awarded in this case.

(d) Answers to this question were generally good with many candidates getting full marks for parts (i) to (vi). The most common incorrect answer for part (ii) was to change the maximum number of moves to 12, not 16. Part (vii) was more

challenging and many candidates dropped marks here. Many incorrectly gave (correct) code for 4-in-a-row rather than 3-in-a-row. Another common error was to add a second loop for the rows that went from 2 to 4 instead of 1 to 4. Some candidates did not read the question carefully and gave an answer that checked for a win in a column not a row. Part (viii) was done well by those who had done part (vii); some candidates did not read the question carefully and did not test for a winning row in the position asked for. There were a lot of correct answers for part (ix) although some dropped a mark by stating the change and not describing it as well. It is important that candidates recognise key words used in questions, like describe and explain, and understand how these should be answered. The most common correct answer was actually the one not on the specification about using a 3D array. A significant number of candidates did not describe how the data structure could be represented and instead wrote about how the displaying of the board would have to be modified.

Q19.

Part (a): The use of the adjacency matrix was clearly well understood with all but a few candidates achieving full marks.

Part (b): There were some good responses to this question part, but also quite a lot of confused answers. An adjacency matrix is more appropriate when there are many edges in a graph, or if these edges need to be checked or updated frequently. An adjacency list is appropriate for graphs with few edges (sparse graphs) or where the edges are not checked / updated frequently.
Neither the number of vertices in a graph nor the available memory would influence the choice.
There was confusion over the use of terminology with some candidates apparently using the term vertex to mean edge.

Part (c): This question part was poorly answered, with only a third of candidates scoring any marks. A tree is a graph that is connected, undirected and has no cycles. Some candidates gave responses that referred only to specific types of tree, either rooted trees or binary trees.
These responses did not gain credit.

Part (d): The vast majority of candidates knew how to construct a binary search tree. The most common cause of error appeared to be candidates forgetting the order of the letters in the alphabet rather than forgetting the principles that should be used to construct the tree. A small number of candidates mistakenly drew arrows instead of lines between nodes.

Part (e): There were some good responses to this question part but many were disappointing and a surprising number of candidates did not write a response at all. Many candidates who did answer chose to use a diagram to illustrate their response which was quite acceptable, so long as the diagram included enough detail to make clear that it was a representation of a binary tree.

Q20.

Part (a): This question part was very well answered with the many candidates getting full marks.

Part (b): The algorithm inserted an item into an ordered list at the correct position. Some candidates lost marks by failing to refer to the ‘correct position’. Others stated that a sort was performed which was not true, as the order of the items in the existing list was not changed.

Part (c): For part (i), around half of the candidates were able to explain that a static data structure has a fixed size which could not change at compile time, whereas the size of a dynamic data structure can change at runtime.

There were few correct answers to part (ii). The heap is a pool of available memory locationsand as the linked list grew, memory would be allocated to it from the heap and deallocated memory would be returned to it. Some candidates erroneously believed that the heap would be used as a temporary store or that the pointers in the linked list would be stored in the heap.

Q22.

(a) See earlier comment in the General section of this Report.

(b) All that was required in this question was the association between a number value and a colour, and hence that different numbers are used to represent different colours. The suspicion was that the candidates were not clear of the meaning of the word 'encoding' in the question stem. Some candidates described the idea that the picture was formed by putting together many pixels. A common misconception was that the pixel value stored its location.

(c) A common wrong answer – as seen in a previous examination – described 'data which is stored in the file directory' (not the file header).

(d) (i) Very poorly answered, despite a very similar question on a recent January series question paper.

(e) (i) Well answered.

(ii) Often candidates latched on to the term 'data structure' and then chose from the stack, queue options, failing to appreciate that an array is referred to as a data structure.

(f) On the one previous question paper on which the algorithm trace used a nested loop, the quality of answers seen was encouraging and the Report commented on this. Alas, the impetus was not maintained, and the number of candidates who were able to score 5 or 6 marks was small. The common error on the better scripts was not to make the final increment of the Counter variable value 6.

(g) (i) Most candidates came up with a valid answer from the large range deemed acceptable.

(ii) Many candidates were able to come up with two restrictions on the choice of identifier names. Some scored 1 mark only by quoting two near identical reasons e.g. cannot contain a 'comma' character followed by 'cannot contain a question mark' character. Some candidates answered their own question e.g. 'cannot store a text character in a integer data type variable'. Other candidates read the question as 'general restrictions' of the programming language and so gave answers such as 'variables must be declared before they can be used,' Answers of this nature were given credit.

Q23.

This question produced many disappointing responses. The question was specifically about a queue but many candidates failed to grasp this. There were a large number of responses that showed that the candidate was concerned about the difficulties of inserting or removing items from the middle of the queue. There was a mixed response to part (a).

Many candidates were well prepared and were able to give satisfactory responses. However, a large number of candidates were unable to express themselves clearly and there was a sizeable minority who seemed to be unprepared for this question.

In part (b) it was pleasing to see the large number of candidates who understood that a circular implementation was required. There were a number of misconceptions, in particular a number of candidates suggested looking for empty spaces in the middle of the array where items had been removed.

Part (c) was badly answered with many wild guesses showing a lack of comprehension. Few candidates showed that they understood that the array was being used to implement a queue and that a queue operation was required. Even fewer recognised that the array is finite and that this will affect the implementation of the queue.

Q24.

(a) The algorithm given contained a single loop, one selection statement and a function. However, the majority of candidates were unable to correctly complete the given algorithm, which begs the question - how many candidates would be able to write a similar algorithm from scratch?

(b) The candidates were given for the first time a specific list of data types from which to choose. This approach is likely to be used again. If other data types are to be used in future questions e.g. Date, then they would clearly be included in a given table.

Candidates did better for D and F, although securing a mark for the purpose of the Position identifier for E was more elusive.

Q25.

It has been commented on in previous papers how the standard of candidates' answers for this question has improved and this continued with this paper.

In part (c) a nested loop (given for the first time) was the key control structure and the majority of candidates identified this. Some confused the nesting, although were still able to secure 5 of the available 6 marks. The most commonly lost mark was at the start of the algorithm for the failure to initialise the array.

Generally part (d) was well answered. Where a mark was lost the suspicion was that it was through poor expression, not lack of understanding of the algorithm. Incorrect answers often stated the numbers for individual sales staff or started to include the concept of target setting, etc.

Q26.

(a) The majority of candidates were able to describe a stack structure as a 'first in last out' or 'last in first out' operation.

(b) The weaker answers seen here moved values to a different memory location once additions and deletions occurred, or used location 605 as the first available and so qualified for a maximum of two (only) 'follow through' marks.

(c) Many candidates were clear about the basic operation which was taking place but then their communication skills let them down in the descriptions required for (ii) and (iii). For (ii) the answer looked for was the idea that items leave the stack one after the other. For (iii) a description was required for the principle of operation of a

queue.

Q27.

Candidates generally scored well on this question. Recursively-defined was well understood although many candidates were unable to describe the use of the stack well enough. It was pleasing to see the majority of candidates obtaining most of the marks on part (c). Candidates often failed to obtain the mark for part (d) due to inadequate descriptions. Although many candidates provided a situation where the algorithm will fail, fewer were able to suggest a suitable modification. Once again this was often due to an inability to express themselves well. A wide range of answers were supplied for part (g) but a substantial number of correct responses were given.

Q28.

This was the first time this topic was assessed with a question like this, but it was well understood by most candidates.

(a) This mark proved elusive for the majority, as an answer which suggested the pointer indicated a location or address value was required. The majority of answers seen simply suggested that “it indicates the end of the queue” and was considered insufficient.

(c) The diagram generally scored well. A common error was to leave the head pointer at address 0.

(d) This was another question where candidates were often let down by their poor expression; the examiner often suspected that the candidate knew that continual addition would result in the queue running out of space, but the candidate was unable to express this. Answers which vaguely stated that the queue ‘runs of out addresses’ were considered insufficient.

Weaker candidates simply said that continually changing the pointer values would prove difficult to keep track of, with the implication that this was a human task and not performed by the computer. Some candidates suggested wrongly that head and tail could reach the same value at which point the queue would become unusable.

Q29.

This question discovered that most candidates did not really understand the concept of a queue as a data structure. Throughout the question many candidates referred to insert/delete operations thus showing a lack of understanding of the operation of a queue. Part (a) was quite beyond the ability of most candidates and this suggested that they had never attempted to program a queue using a linked data structure. Very few candidates recognised that the value of Front had to be stored in a temporary variable in order to free the memory used. The answers to part (c) suggested that they had not programmed a queue using an array either. Most candidates, however, managed to obtain some marks on this question, most frequently parts (c)(i) and (c)(ii). Many marks were lost in part (c)(iii) due to inadequate descriptions.

Q30.

Very few candidates did not get some marks for the trace and many returned full marks for this part of the question. Some candidates who did achieve full marks on part (a) could not say what the algorithm does. Many candidates seemed to make a wild guess.

Q31.

This was the first question paper on which two-dimensional arrays had been set and the answers seen were encouraging.

(a) Most candidates correctly described that this was the issues figure for salesperson 7 in month 4. Some candidates described the figure as the highest sales figure for April which gained no credit.

(b) Only better candidates wrote an acceptable declaration statement which required the correct identifier StoreCards with the correct subscripts in the correct order.

(c) Few acceptable statements were seen.

(d) Encouragingly, this was well answered, with most candidates able to describe the purpose of the algorithm. Answers which did little more that re-write statement(s) from the given algorithm into a narrative form - e.g. “person total set to zero” - which was little different, did not gain credit. The common error was stating that the algorithm calculated a total for ‘each’ salesperson.

(e) Somewhat surprisingly – despite similar questions on previous papers - candidates were often unable to state a correct data type, which would suggest the fundamental concept in programming that “identifiers will have a stated or implied data type” is not understood.

For (ii) almost all gave Boolean, with every possible phonetic spelling, and some gave integer for (iii). Real/Float or other acceptable alternatives for (i) were rare.

Q32.

(a) Most candidates were able to write the correct binary pattern and successfully add a correct parity bit. Candidates need reminding that it is normal practice to use bit position 7 (the left-most bit) for the parity bit.

(b) (i) Most candidates wrote character ‘D’.

(ii) Some candidates left the answer as 74 failing to use the Chr( ) function as the final step.

(c) All candidates scored well for the structure chart appreciating that such a diagram can be used to represent the interface of a function. The mark scheme was generous in allowing a mixture of case used in the identifier labels in the diagram. This might not gain credit in future papers.

(d) This question was well answered by the majority of candidates - possibly as there had been a ‘build up’ to the algorithm in the earlier parts of the question.

This is an encouraging trend as the algorithm contained a loop, the use of functions, the use of arrays (which had been poorly tackled on previous papers), yet candidates scored highly.

Q33.

It was pleasing to see many good answers to parts (a) and (b) although a number of candidates failed to obtain full marks through inadequate explanations. Part (c) was disappointing with few candidates completing the trace table correctly. Nevertheless it was pleasing to see a greater number of candidates able to partially complete a dry run. A surprising number of candidates were able to state that this was a bubble sort even

though they failed to complete the trace table. Fewer were able to give a suitable improvement. The most common incorrect suggestion was to “make the algorithm recursive”.

Q34.

(a) (i) The use of GoTo statements has not previously been examined on this paper and most candidates struggled to suggest a single reason why this was poorly designed code, despite a large number of acceptable answers. The most common correct answers were that the use of GoTo statements gives rise to code which is difficult to follow and trace; there is no output produced when the SearchName value is not found; when there is more than one occurrence of SearchName in the PolicyHolder array, the program will output the number of claims value for the first occurrence of the name only.

(ii) Few marks were obtained here with most candidates failing to give the bounds of the array for PolicyHolder or NoOfClaims, or omitting a data type for the identifier.

(b) Candidates should be able to write small amounts of program code in a unit that has the word ‘programming’ in its title. Knowledge of loops other than a For loop was rare. It was hoped that candidates would have constructed a Repeat – Until or While loop which terminated when a NoOfClaims value of 5 or more was found. Candidates who used a For loop were, however, still able to score the maximum 5 marks.

Examiners were not looking for the correct use of exact syntax for the language as stated by the candidate.

The use of IF statements was better understood, but this often did not extend to using an array index for the NoOfClaims as part of the IF statement. Very many candidates used the maths operator incorrectly, e.g. ≥ or more usually =>. Quite a few candidates reversed the logic testing for <5 and gave appropriate output for which they gained marks. Most popular languages seen were Pascal and Visual Basic but the candidates that used C on the whole answered the question very well indeed.

Q35.

Although a short question, it proved difficult for most candidates. Many missed the point that both part (a) and part (b) were about the *implementation* of a stack, and in part (b) gave answers that were about applications that were suitable for a linked list or an array. However, we can note one particularly lucid answer to part (a)(i): “This is a static data structure with a finite pre-declared capacity.”