1.1 Programming part 2 Mark schemes.

Q1.

(a) `TCard` //
`TRecentScore` //
`TDeck` (Pascal only) //
`TRecentScores` (Pascal only);
R If any additional code
R If spelt incorrectly
I Case

1

(b) `CInt` (VB.Net / VB6 only) //
`Val` (Pascal only) //
`StrToInt` (Delphi only) //
`parseInt` (Java only) //
`Integer.parseInt` (Java only) //
`int` (Python only);

R If any additional code
R If spelt incorrectly
I Case

1

(c) `Deck//RecentScores;`
R If any additional code
R If spelt incorrectly
I Case

1

(d) Temporary;

1

(e) Most recent holder;

1

(f) Stepper;

1

(g) When the name in the variable `PlayerX` is not in the array `RecentScores;`
A Answer that does not use identifiers but clearly suggests that the name is not in the array

1

(h) `WHILE Found = False AND Position`
A Alternative loop conditions that would provide correct functionality
eg

```
Position  10
        Console.Write(“Not a valid choice, please enter another
number: ”)
        NumberToGuess = Console.ReadLine()
    End While
    Guess = 0
    NumberOfGuesses = 0
    While Guess <> NumberToGuess And NumberOfGuesses < 5
        Console.Write(“Player Two have a guess: ”)
        Guess = Console.ReadLine()
        NumberOfGuesses = NumberOfGuesses + 1
    End While
    If Guess = NumberToGuess Then
        Console.Write(“Player Two wins”)
    Else
        Console.Write(“Player One wins”)
    End If
    Console.ReadLine()
  End Sub
End Module
```

VB6

```
Private Sub Form_Load()
    Dim NumberToGuess As Integer
    Dim NumberOfGuesses As Integer
    Dim Guess As Integer
    NumberToGuess = ReadLine(“Player One enter your
chosen number: ”)
    While NumberToGuess < 1 Or NumberToGuess > 10
      NumberToGuess = ReadLine(“Not a valid choice,
please enter another number: ”)
    Wend
    Guess = 0
    NumberOfGuesses = 0
    While Guess < > NumberToGuess And NumberOfGuesses <
5
      Guess = ReadLine(“Player Two have a guess: ”)
      NumberOfGuesses = NumberOfGuesses + 1
    Wend
    If Guess = NumberToGuess Then
      WriteLineWithMsg (“Player Two wins”)
    Else
      WriteLineWithMsg (“Player One wins”)
    End If
End Sub
```

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

```
Text1.Text = Text1.Text & ...
WriteLine
WriteWithMsg
Msgbox
InputBox
WriteNoLine
```

Python 3

```
print(‘Player One enter your chosen number: ’)
NumberToGuess = int(input())
while (NumberToGuess < 1) or (NumberToGuess > 10) :
  print(‘Not a valid choice, please enter another
number: ’)
  NumberToGuess = int(input())
Guess = 0
NumberOfGuesses = 0
while (Guess != NumberToGuess) and (NumberOfGuesses <
5) :
  print(‘Player Two have a guess: ’)
  Guess = int(input())
  NumberOfGuesses = NumberOfGuesses + 1
```

```
if Guess == NumberToGuess :
  print(‘Player Two wins’)
else :
  print(‘Player One wins’)
```

Alternative print / input combinations:

```
NumberToGuess = int(input('Player One enter your
chosen number: ’))

Guess = int(input(‘Player Two have a guess: '))
```

Python 2

```
print ‘Player One enter your chosen number: '
NumberToGuess = int(raw_input())
while (NumberToGuess  10) :
  print ‘Not a valid choice, please enter another
number: ’
  NumberToGuess = int(raw_input())
Guess = 0
NumberOfGuesses = 0
while (Guess != NumberToGuess) and (NumberOfGuesses <
5) :
  print ‘Player Two have a guess: ’
  Guess = int(raw_input())
  NumberOfGuesses = NumberOfGuesses + 1
if Guess == NumberToGuess :
  print ‘Player Two wins’
else :
  print ‘Player One wins’
```

Alternative print / input combinations:

```
NumberToGuess = int(raw_input(‘Player One enter your
chosen number: ’))

Guess = int(raw_input(‘Player Two have a guess: ’))
```

JAVA

```
int numberToGuess;
int numberOfGuesses;
int guess;
numberToGuess = console.readInteger(“Player One enter
your
chosen number: ”);
while(numberToGuess < 1 || numberToGuess > 10){
  numberToGuess = console.readInteger(“Not a valid
choice,
please enter another number: ”);
}
guess = 0;
numberOfGuesses = 0;
while (guess != numberToGuess && numberOfGuesses < 5){
  guess = console.readInteger(“Player Two have a guess:
```

```
”);
  numberOfGuesses++;
}
if(guess == numberToGuess){
  console.println(“Player Two wins”);
}else{
  console.println(“Player One wins”);
}
```

13

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

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

Mark as follows:
'Player One enter your chosen number: ' + user input of 0
'Not a valid choice, please enter another number: ' Message shown;
user input of 11
'Not a valid choice, please enter another number: ' Message shown;
user input of 5
'Player Two have a guess: ' + user input of 5;
'Player Two wins' message shown; **R** If no evidence of user input
A alternative output messages if match code for (a)

4

(c) ****SCREEN CAPTURE****

Must match code from (a), including prompts on screen capture matching those in code. Code for 19 must be sensible.

Mark as follows:
'Player One enter your chosen number: ' + user input of `6;`
'Player Two have a guess: ' + user input of `1`
'Player Two have a guess: ' + user input of `3`
'Player Two have a guess: ' + user input of `5`
'Player Two have a guess: ' + user input of `7`
'Player Two have a guess: ' + user input of `10;`
'Player One wins' message shown; **R** If no evidence of user input
A alternative output messages if match code for (a)

3

(d) If a `FOR` loop was used then Player Two will always have 5 guesses // a `WHILE` loop will mean that the loop will terminate when Player Twoguesses correctly // the number of times to iterate is not known before the loop starts;

1

[21]

Q4.

(a) `AmountToShift // StartPosition // EndPosition // SizeOfRailFence // N // Count // Key // ASCIICode // NewASCIICode // Count2 // Count1 // NoOfColumns // NoOfRows // NoOfCiphertextCharacters // NoOfCiphertextCharactersProcessed // i // j // PositionOfNextCharacter // LastFullRowNo //`

```
AmountToReduceNoOfColumnsTimesjBy //
BeginningofNextRowIndex // CurrentPosition;
```

R if any additional code
R if spelt incorrectly
I case & spaces

1

(b)

```
EveryNthCharacterSteganography;
```

R if any additional code (including routine interface)
R if spelt incorrectly
I case & spaces

1

(c) **Pascal**

```
Ord // Length;
```

VB.Net

```
Asc // Length;
```

VB6

```
Asc // Len;
```

Python

```
ord // len // int;
```

Java

```
int // length;
```

R if any additional code
R if spelt incorrectly
I case & spaces

1

(d) **Pascal**

Ciphertext := '' //
Plaintext := '' //
ChangedText := '' //
TextFromFile := '' //
HiddenMessage := '';
I semicolons

VB.Net / VB6

```
Ciphertext = "" //
Plaintext = "" //
ChangedText = "" //
TextFromFile = "" //
HiddenMessage = "";
```

Python

```
Ciphertext = '' //
Plaintext = '' //
ChangedText = '' //
TextFromFile = '' //
HiddenMessage = ''
```

Java

```
ciphertext = "" //
```

`plaintext = "" //`
`changedText = "" //`
`textFromFile = "" //`
`hiddenMessage = ""`
I semicolons

R if any additional code
R if spelt incorrectly
I case & spaces

1

(e) Because if decrypt has been selected; then the plaintext alphabet needs to be shifted in the opposite direction;

2

(f) **Mark as follows:**
Identify the problem that will occur;
Explanation of how `MOD 26` solves the problem;
Max 1 if no example used in explanation

Example answer
Without `MOD 26` then the shift will only be applied correctly to letters early in the alphabet e.g. if the `AmountToShift` is 1 then the letter Z will be given a `NewASCIICode` of 91 (ASCII code for Z is 90) and this does not represent a letter; Using `MOD 26` ensures that the ciphertext alphabet wraps round to the beginning of the alphabet (in this example `NewASCIICode` would become 65 the ASCII Code for A);

2

(g) `ApplyShiftToASCIICodeForCharacter;`
R if spelt incorrectly
I case & spaces

1

(h) `NewASCIICode;`
A `ApplyShiftToASCIICodeForCharacter` (Pascal / VB.Net / VB6 only);
R if spelt incorrectly
I case & spaces

1

(i) `GetTypeOfCharacter //`
`Ord` (Pascal / Python only) //
`Asc` (VB only) //
`int` (Java only);
R if spelt incorrectly
I case & spaces

1

(j) **Pascal / VB6**
`For 1 To Length(OriginalText);`

VB.Net
`For 0 To (OriginalText.Length - 1);`

Python 2 / 3
`for in range (0, len(OriginalText)):;`

Java

```
for (count = 0; count
count++);
```

A Alternative correct logic

A Any clear description that conveys correct logic

1

[12]

Q5.

(a) One mark per correct response.

Construct	Example	Valid?
identifier	`Player2name`	No;
parameter	`x, y:bool`	Yes;
procedure-def	`procedure square(s:real)`	No;
procedure-def	`procedure rect(w:int,h:int)`	No;

A alternative clear indicators of Yes / No such as Y / N, True / False and Tick / Cross.

4

(b) (i) The <`type`> rule has an extra type `char`;

The <`procedure-def`> rule does not allow a procedure without parameters // cannot be just an identifier;

A answers comparing the figures the other way around, i.e.

- The type rule does not allow a char
- The procedure does not have to have parameters / can be just an identifier

2

(ii) Required as there can be a list of parameters // required as there can be more than one parameter;

BNF does not support iteration // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise // recursion allows for more than one parameter;

Max 1

A Input for parameter

NE Rule needs to loop

1

[7]

Q6.

		Answer	Carry	
0	0	0	0	
0	1	1	0	;
1	0	1	0	;
1	1	0	1	;

A 10 instead of 0 in the Answer column for the final row of the table

[3]

Q7.

(a) 011 0010;
R If not 7 bits

1

(b) 1011 0000

Mark as follows:
Correct data bits;
Correct parity bit for the candidate's data bits;
R If not 8 bits

2

(c) Error correction (not just error detection) (for single errors);
Can detect when two errors have occurred in data transmission;
Reduces the need for the retransmission of data;
Decreases the likelihood of an undetected error // improved error detection;
Can locate an error (not just detect that an error has occurred);

Max 1

[4]

Q8.

(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]

Q9.

(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]

Q10.

(a) (i) Appropriate option added;

Pascal

```
Procedure DisplayMoveOptions;
   Begin
        Writeln;
        Writeln('Enter N to move NORTH');
        Writeln('Enter E to move EAST');
        Writeln('Enter S to move SOUTH');
        Writeln('Enter W to move WEST');
        Writeln('Enter D to move SOUTHEAST');
        Writeln('Enter M to return to the Main Menu');
        Writeln;
End;
```

VB.NET

```
Sub DisplayMoveOptions()
    Console.WriteLine()
    Console.WriteLine("Enter N to move NORTH")
    Console.WriteLine("Enter E to move EAST")
    Console.WriteLine("Enter S to move SOUTH")
    Console.WriteLine("Enter W to move WEST")
    Console.WriteLine("Enter D to move SOUTHEAST")
   Console.WriteLine("Enter M to return to the Main Menu")
   Console.WriteLine()
End Sub
```

VB6

```
Private Sub DisplayMoveOptions()
    WriteLine ("")
    WriteLine ("Enter N to move NORTH")
    WriteLine ("Enter E to move EAST")
    WriteLine ("Enter S to move SOUTH")
    WriteLine ("Enter W to move WEST")
    WriteLine ("Enter D to move SOUTHEAST")
    WriteLine ("Enter M to return to the Main Menu")
    WriteLine ("")
End Sub
```

A Text1.Text = Text1.Text & "Enter D to move SOUTHEAST "

Java

```
void displayMoveOptions() {
    console.println();
    console.println("Enter N to move NORTH");
    console.println("Enter E to move EAST");
    console.println("Enter S to move SOUTH");
    console.println("Enter W to move WEST");
    console.println("Enter D to move SOUTHEAST");
   console.println("Enter M to return to the Main Menu");
   console.println();
}
```

Python 2

```
def DisplayMoveOptions():
    print ''
    print 'Enter N to move NORTH'
    print 'Enter E to move EAST'
    print 'Enter S to move SOUTH'
    print 'Enter W to move WEST'
    print 'Enter D to move SOUTHEAST'
    print 'Enter M to return to the Main Menu'
    print ''
```

Python 3

```
def DisplayMoveOptions():
    print ()
    print ('Enter N to move NORTH')
    print ('Enter E to move EAST')
    print ('Enter S to move SOUTH')
    print ('Enter W to move WEST')
    print ('Enter D to move SOUTHEAST')
    print ('Enter M to return to the Main Menu')
    print ()
```

A Any sensible prompt
A Prompt added anywhere in subroutine
R If prompt asks for character other than D

1

(ii) Additional case statement for option D added correctly and all of the rest of the code added inside the correct option of the case statement;
A any character instead of D (except N, S, W, E) – only if matches prompt from (a)(i)
`NoOfCellsSouth incremented within the correct option of the case statement;`
`NoOfCellsEast incremented within the correct option of the case statement;`

Pascal

```
Case Direction Of
    'N' : PlayerPosition.NoOfCellsSouth :=
PlayerPosition.NoOfCellsSouth - 1;
    'S' : PlayerPosition.NoOfCellsSouth :=
PlayerPosition.NoOfCellsSouth + 1;
    'W' : PlayerPosition.NoOfCellsEast :=
PlayerPosition.NoOfCellsEast - 1;
    'E' : PlayerPosition.NoOfCellsEast :=
PlayerPosition.NoOfCellsEast + 1;
    'D' : Begin
            PlayerPosition.NoOfCellsSouth :=
PlayerPosition.NoOfCellsSouth + 1;
            PlayerPosition.NoOfCellsEast :=
PlayerPosition.NoOfCellsEast + 1;
          End;
End;
```

VB.NET

```
Case "E"
    PlayerPosition.NoOfCellsEast =
PlayerPosition.NoOfCellsEast + 1
Case "D"
    PlayerPosition.NoOfCellsSouth =
PlayerPosition.NoOfCellsSouth + 1
    PlayerPosition.NoOfCellsEast =
PlayerPosition.NoOfCellsEast + 1
```

VB6

```
Case "E"
    PlayerPosition.NoOfCellsEast =
PlayerPosition.NoOfCellsEast + 1
Case "D"
    PlayerPosition.NoOfCellsSouth =
PlayerPosition.NoOfCellsSouth + 1
    PlayerPosition.NoOfCellsEast =
PlayerPosition.NoOfCellsEast + 1
```

Java

```
switch (direction) {
case 'N':
   playerPosition.noOfCellsSouth--;
   break;
case 'S':
    playerPosition.noOfCellsSouth++;
    break;
case 'W':
   playerPosition.noOfCellsEast--;
```

```
        break;
    case 'E':
        playerPosition.noOfCellsEast++;
        break;
    case 'D':
        playerPosition.noOfCellsSouth++;
        playerPosition.noOfCellsEast++;
        break;
    }
```

Python

```
elif Direction == 'E':
    PlayerPosition.NoOfCellsEast += 1
elif Direction == 'D':
    PlayerPosition.NoOfCellsSouth += 1
    PlayerPosition.NoOfCellsEast += 1
```

3

(iii) Additional condition added to `IF statement` ;

A answers using an additional `IF statement`

R if value of ‘D’ will result in `False` being returned by function

R if function will always return True

Pascal

```
ValidMove := True;
If Not (Direction In ['N','S','W','E','D','M'])
    Then ValidMove := False;
CheckValidMove := ValidMove;
```

VB.NET

```
ValidMove = True
If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M" Or Direction = "D") Then
    ValidMove = False
End If
CheckValidMove = ValidMove
```

VB6

```
ValidMove = True
If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M" Or Direction = "D") Then
   ValidMove = False
End If
CheckValidMove = ValidMove
```

Java

```
validMove = true;
if (!(direction = = 'N' || direction = = 'S' || direction = =
'W'|| direction = = 'E' || direction = = 'D' || direction = =
'M')) {
    validMove = false;
}
return validMove;
```

Python

```
def CheckValidMove(PlayerPosition,Direction):
  ValidMove = True
  if not (Direction in ['N','S','W','E','D','M']):
    ValidMove = False
  return ValidMove
```

1

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

This is conditional on sensible code for (i), (ii) and (iii)

Screen capture(s) showing modified menu shown to user and option 'D' selected;

Screen capture(s) showing both original position of player in the cavern and the new position of the player in the cavern;

2

(b) (i) Selection structure with correct condition;

Inside the selection structure there is code that will display the correct message on the screen;

I Capitalisation and minor typos in message

R different message

Selection structure is in the correct place in the code;

Pascal

```
Repeat
    DisplayMoveOptions;
    MoveDirection := GetMove;
   ValidMove := CheckValidMove(PlayerPosition,
MoveDirection);
   If Not ValidMove
     Then Writeln('That is not a valid move, please try
again');
Until ValidMove;
```

Alternative answer

```
If ValidMove = False...
```

VB.NET

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

VB6

```
Do
    Call DisplayMoveOptions()
    MoveDirection = GetMove()
   ValidMove = CheckValidMove(PlayerPosition, MoveDirection)
   If Not ValidMove Then
     WriteLine("That is not a valid move, please try again")
   End If
Loop Until ValidMove
```

A `Text1.Text = Text1.Text & "That is not a valid move, please try again "`

A `WriteLineWithMsg`

Java

```
do {
    displayMoveOptions();
    moveDirection = getMove();
   validMove = checkValidMove(playerPosition,
moveDirection);
```

```
   if (!validMove) {
     console.println("That is not a valid move, please try
again");
   }
} while (!validMove);
```

Alternative answer

```
if (validMove == false)
```

Python

```
while not ValidMove:
    DisplayMoveOptions()
    MoveDirection = GetMove()
   ValidMove = CheckValidMove(PlayerPosition, MoveDirection)
   if not ValidMove:
        # Python 2:
        print 'That is not a valid move, please try again'
        # Python 3:
        print('That is not a valid move, please try again')
```

Alternative answer

```
if ValidMove = False...
```

3

(ii) If statement with a correct condition;
Correct logic and 2nd condition for `If` statement;
Value of `False` returned correctly by the function if illegal north move is made;

R if a value of `False` will always be returned by the function
R if all north moves will return false
R if all moves when `PlayerPosition.NoOfCellsSouth` is in row 1 will return false

Value of `True` returned correctly by the function if legal north move is made;

A Answers which combine all the checks for a valid move into one `If` statement
I missing option 'D' in code

Pascal

```
ValidMove := True;
If Not (Direction In ['N','S','W','E','D','M'])
   Then ValidMove := False;
If (PlayerPosition.NoOfCellsSouth = 1) And (Direction = 'N')
   Then ValidMove := False;
CheckValidMove := ValidMove;
```

Alternative answer

```
If ValidMove And (Direction = 'N')
   Then ValidMove := ValidMove And
   (PlayerPosition <> 1);
```

VB.NET

```
If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "D" Or Direction = "M") Then
   ValidMove = False
End If
If PlayerPosition.NoOfCellsSouth = 1 And Direction = "N" Then
   ValidMove = False
```

```
End If
CheckValidMove = ValidMove
```

Alternative answer

```
If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M") Then
   ValidMove = False
End If
If ValidMove And (Direction = "N") Then
   ValidMove = (ValidMove And (PlayerPosition.NoOfCellsSouth
<> 1))
End If
```

VB6

```
If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "D" Or Direction = "M") Then
   ValidMove = False
End If
If PlayerPosition.NoOfCellsSouth = 1 And Direction = "N" Then
   ValidMove = False
End If
CheckValidMove = ValidMove
```

Alternative answer

```
If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M") Then
   ValidMove = False
End If
If ValidMove And (Direction = "N") Then
   ValidMove = (ValidMove And (PlayerPosition.NoOfCellsSouth
<> 1))
End If
```

Java

```
validMove = true;
if (!(direction = = 'N' || direction = = 'S' || direction = =
'W'|| direction = = 'E' || direction = = 'D' || direction = =
'M')) {
   validMove = false;
}
if (validMove && direction = = 'N') {
   validMove = validMove &&
   (playerPosition.noOfCellsSouth != 1);
    }
return validMove;
```

Alternative answer

```
if (playerPosition.noOfCellsSouth = = 1 && direction = = 'N')
{
validMove = false;
}
```

Python

```
def CheckValidMove(PlayerPosition,Direction):
   ValidMove = True
   if not (Direction in ['N','S','W','E','D','M']):
      ValidMove = False
   if (PlayerPosition.NoOfCellsSouth = = 1) and (Direction =
= 'N'):
      ValidMove = False
   return ValidMove
```

Alternative answer

```
if not (Direction in ['N','S','W','E','D','M']):
     ValidMove = False
if ValidMove and (Direction = = 'N'):
   ValidMove = (ValidMove and (PlayerPosition. NoOfCellsSouth
!= 1))
```

4

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

This is conditional on sensible code for (b)(i) and correct code for (b)(ii).

Screen capture(s) showing correct cavern state with a player at the northern end of the cavern (top line), 'N' being entered at prompt, followed by correct error message being displayed;

1

(c) (i) `NoOfMoves` is assigned the value 0 – before the first repetition structure in `PlayGame`;

I. Case of variable names

A. Minor typos in variable name

A assignment statement(s) in other subroutine(s) if correct functionality would be obtained

`NoOfMoves` incremented in any sensible place in the code inside the first selection structure in `PlayGame` subroutine;

One correct message displayed with `NoOfMoves`;

Second correct message displayed with `NoOfMoves`;

Correct logic – both of the messages will be displayed only under the correct circumstances;

A. minor typos in messages **I**. capitalisation & spacing in messages

A. message displayed on more than one line

A. more than one line of code used to display a message

A. `NoOfMoves` declared as global

I. `NoOfMoves` declaration not shown in PROGRAM SOURCE CODE

Pascal

```
Eaten:= False;
FlaskFound := False;
DisplayCavern(Cavern, MonsterAwake);
NoOfMoves := 0;
Repeat
   ...
   If MoveDirection <> 'M'
      Then
          Begin
            MakeMove(Cavern, MoveDirection, PlayerPosition);
            NoOfMoves := NoOfMoves + 1;
            DisplayCavern(Cavern, MonsterAwake);
            ...
            If FlaskFound
              Then
                 Begin
                   DisplayWonGameMessage;
                   Writeln('The number of moves you took to
find the flask was ',NoOfMoves);
                 End;
                 ...
                 If Eaten
                   Then
```

```
                     Begin
                        DisplayLostGameMessage;
                        Writeln('The number of moves you
survived in the cavern for was ', NoOfMoves);
                     End;
```

Alternative answer

```
Until Eaten Or FlaskFound Or (MoveDirection = 'M');
If Eaten
   Then Writeln('The number of moves that you survived in the
cavern for was ', NoOfMoves);
If FlaskFound
   Then Writeln('The number of moves you took to find the flask
was ', NoOfMoves);
```

Alternative answer

```
If FlaskFound
   Then DisplayWonGameMessage(NoOfMoves);
...
If Eaten
   Then DisplayLostGameMessage(NoOfMoves);
```

together with modified `DisplayWonGameMessage` to include additional output message (I missing parameter if `NoOfMoves` declared as global)

```
Procedure DisplayWonGameMessage(NoOfMoves : Integer);
  Begin
     Writeln('Well done! You have found the flask containing
the Styxian potion.');
     Writeln('You have won the game of MONSTER!');
     Writeln('The number of moves you took to find the flask
was ',NoOfMoves);
     Writeln;
   End
```

and modified `DisplayLostGameMessage` to include additional output message (I missing parameter if `NoOfMoves` declared as global)

```
Procedure DisplayLostGameMessage(NoOfMoves : Integer);
   Begin
     Writeln('ARGHHHHHH! The monster has eaten you. GAME
OVER.');
     Writeln('Maybe you will have better luck next time you
play MONSTER!');
     Writeln('The number of moves you survived in the cavern
for was ', NoOfMoves);
     Writeln;
   End;
```

VB.NET

```
Dim ValidMove As Boolean
Eaten = False
FlaskFound = False
DisplayCavern(Cavern, MonsterAwake)
NoOfMoves = 0
Do
...
If MoveDirection <> "M" Then
    MakeMove(Cavern, MoveDirection, PlayerPosition)
    NoOfMoves = NoOfMoves + 1
    DisplayCavern(Cavern, MonsterAwake)
...
If FlaskFound Then
   DisplayWonGameMessage()
   Console.WriteLine("The number of moves you took to find the
flask was " & NoOfMoves)
End If
...
If Eaten Then
   DisplayLostGameMessage()
   Console.WriteLine("The number of moves that you survived in
the cavern for was " & NoOfMoves)
End If
...
```

Alternative answer

```
Loop Until Eaten Or FlaskFound Or MoveDirection = "M"
If Eaten Then
   Console.WriteLine("The number of moves that you survived in
the cavern for was " & NoOfMoves)
End If
If FlaskFound Then
   Console.WriteLine("The number of moves you took to find the
flask was " & NoOfMoves)
End If
```

Alternative answer

```
If FlaskFound Then
   DisplayWonGameMessage(NoOfMoves)
End If
...
If Eaten Then
   DisplayLostGameMessage(NoOfMoves)
End If
```

together with modified `DisplayWonGameMessage` to include additional output message (**I** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayWonGameMessage(ByVal NoOfMoves As Integer)
   Console.WriteLine("Well done! You have found the flask
containing the Styxian potion.")
   Console.WriteLine("You have won the game of MONSTER!")
   Console.Writeline("The number of moves you took to find the
flask was " & NoOfMoves)
   Console.WriteLine()
End Sub
```

and modified `DisplayLostGameMessage` to include additional output message (**I** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayLostGameMessage(ByVal NoOfMoves As Integer)
   Console.WriteLine("ARGHHHHHH! The monster has
eaten you. GAME OVER.")
   Console.WriteLine("Maybe you will have better luck next
time you play MONSTER!")
   Console.WriteLine("The number of moves you survived in the
cavern for was " & NoOfMoves);
   Console.WriteLine()
End Sub
```

VB6

```
Dim ValidMove As Boolean
Eaten = False
FlaskFound = False
Call DisplayCavern(Cavern, MonsterAwake)
NoOfMoves = 0
Do
...
```

```
If MoveDirection <> "M" Then
   Call MakeMove(Cavern, MoveDirection, PlayerPosition)
   NoOfMoves = NoOfMoves + 1
   Call DisplayCavern(Cavern, MonsterAwake)
...
If FlaskFound Then
   Call DisplayWonGameMessage()
   WriteLine("The number of moves you took to find the flask was " & NoOfMoves)
End If
...
If Eaten Then
   Call DisplayLostGameMessage()
   WriteLine("The number of moves that you survived in the cavern for was " & NoOfMoves)
End If
...
```

Alternative answer

```
Loop Until Eaten Or FlaskFound Or MoveDirection = "M"
If Eaten Then
   WriteLine("The number of moves that you survived in the cavern for was " & NoOfMoves)
End If
If FlaskFound Then
   WriteLine("The number of moves you took to find the flask was " & NoOfMoves)
End If
```

Alternative answer

```
If FlaskFound Then
   DisplayWonGameMessage(NoOfMoves)
End If
...
If Eaten Then
   DisplayLostGameMessage(NoOfMoves)
End If
```

together with modified `DisplayWonGameMessage` to include additional output message (**I** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayWonGameMessage(ByVal NoOfMoves As Integer)
   WriteLine("Well done! You have found the flask containing the Styxian potion.")
   WriteLine("You have won the game of MONSTER!")
   Writeline("The number of moves you took to find the flask was " & NoOfMoves);
   WriteLine("")
End Sub
```

and modified `DisplayLostGameMessage` to include additional output message (**I** missing parameter if `NoOfMoves` declared as global)

```
Sub DisplayLostGameMessage(ByVal NoOfMoves As Integer)
   WriteLine("ARGHHHHHH! The monster has eaten you. GAME OVER.")
   WriteLine("Maybe you will have better luck next time you play MONSTER!")
   WriteLine("The number of moves you survived in the cavern for was " & NoOfMoves);
   WriteLine("")
End Sub
```

A. `Text1.Text = Text1.Text & "The number of moves that you survived in the cavern for was "`

A. `Text1.Text = Text1.Text & "The number of moves you took to find the flask was "`

A. **`WriteLineWithMsg`**

Java

```
eaten = false;
flaskFound = false;
displayCavern(cavern, monsterAwake);
noOfMoves = 0;
do {
   ...
   if (moveDirection != 'M') {
      makeMove(cavern, moveDirection, playerPosition);
      noOfMoves++;
      displayCavern(cavern, monsterAwake);
      flaskFound = checkIfSameCell(playerPosition,
flaskPosition);
      if (flaskFound) {
          displayWonGameMessage();
          console.println("The number of moves you took to
find the flask was " + noOfMoves);
      }
      ...
      if (eaten) {
          displayLostGameMessage();
          console.println("The number of moves you survived in
the " + "cavern for was " + noOfMoves);
     }
```

Alternative answer

```
} while (!(eaten || flaskFound || moveDirection == 'M'));
if (flaskFound) {
   console.println("The number of moves you took to find the
flask was " + noOfMoves);
}
if (eaten) {
   console.println("The number of moves you survived in the "
+ "cavern for was " + noOfMoves);
}
```

Alternative answer

```
eaten = false;
flaskFound = false;
displayCavern(cavern, monsterAwake);
noOfMoves = 0;
do {
   ...
   if (moveDirection != 'M') {
      makeMove(cavern, moveDirection, playerPosition);
      noOfMoves++;
      displayCavern(cavern, monsterAwake);
      ...
```

together with modified `displayLostGameMessage` and `displayWonGameMessage` to include additional output message (I missing parameter if `NoOfMoves` declared as global)

```
void displayWonGameMessage(int noOfMoves){
console.println("ARGHHHHHHH! The monster has eaten you. GAME
OVER.");
   console.println("Maybe you will have better luck next time
you play MONSTER!");
   console.println("The number of moves you survived in the
cavern was " + noOfMoves);
   console.println();
}
void displayWonGameMessage(int noOfMoves){
   console.println("Well done! You have found the flask
containing the Styxian potion.");
   console.println("You have won the game of MONSTER!");
   console.println("The number of moves you took to find the
flask was " + noOfMoves);
}
```

Python

```
Eaten = False
FlaskFound = False
MoveDirection = ''
DisplayCavern(Cavern, MonsterAwake)
NoOfMoves = 0
while not (Eaten or FlaskFound or (MoveDirection == 'M')):
    ValidMove = False
    while not ValidMove:
       DisplayMoveOptions()
       MoveDirection = GetMove()
      ValidMove = CheckValidMove(PlayerPosition,
MoveDirection)
      if not ValidMove:
          print 'That is not a valid move, please try again'
   if MoveDirection != 'M':
      MakeMove(Cavern, MoveDirection, PlayerPosition)
      NoOfMoves += 1
      DisplayCavern(Cavern, MonsterAwake)
...
   if FlaskFound:
      DisplayWonGameMessage()
       # Python 2:
      print 'The number of moves you took to find the flask
was', NoOfMoves
       # Alternative answer:
      print 'The number of moves you took to find the flask was
' + str(NoOfMoves)
       # Python 3:
      print('The number of moves you took to find the flask was
' + str(NoOfMoves)
       # Alternative answer:
      print('The number of moves you took to find the flask was
{0}'.format(NoOfMoves)) #Py3
...
if Eaten:
   DisplayLostGameMessage()
    # Python 2:
   print 'The number of moves that you survived in the cavern
for was', NoOfMoves
  # Alternative answer:
   print 'The number of moves that you survived in the cavern
for was ' + str(NoOfMoves)
    # Python 3:
   print('The number of moves that you survived in the cavern
for was ' + str(NoOfMoves))
  # Alternative answer:
   print('The number of moves that you survived in the cavern
for was {0}'.format(NoOfMoves))
```

Alternative Answer

```
# Python 2
if Eaten:
   print 'The number of moves that you survived in the cavern
for was', NoOfMoves
else:
   print 'The number of moves you took to find the flask was',
NoOfMoves
```

```
# Python 3
if Eaten:
   print('The number of moves that you survived in the cavern
for was' + str(NoOfMoves))
else:
   print('The number of moves you took to find the flask was'
+ str(NoOfMoves))
```

A `.format(NoOfMoves)`

Alternative answer

```
if FlaskFound:
   DisplayWonGameMessage(NoOfMoves)
...
if Eaten:
   DisplayLostGameMessage(NoOfMoves)
```

together with modified `displayLostGameMessage` and `displayWonGameMessage` to include additional output message (**I** missing parameter if `NoOfMoves` declared as global)

```
# Python 2
def DisplayWonGameMessage(NoOfMoves):
   print 'Well Done! You have found the flask containing the
Styxian potion.'
   print 'You have won the game of MONSTER!'
   print 'The number of moves you took to find the flask was
', NoOfMoves
def DisplayLostGameMessage(NoOfMoves):
   print 'ARGHHHHHH! The monster has eaten you. GAME OVER.'
   print 'Maybe you will have better luck the next time you play
MONSTER!'
   print 'The number of moves that you survived in the cavern
for was', NoOfMoves
```

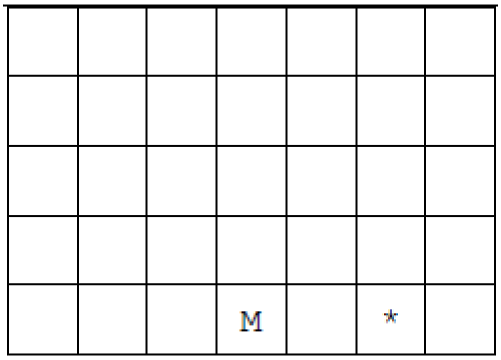
```
# Python 3
def DisplayWonGameMessage(NoOfMoves):
   print('Well Done! You have found the flask containing the
Styxian potion.')
   print('You have won the game of MONSTER!')
   print('The number of moves you took to find the flask was'
+ str(NoOfMoves))
def DisplayLostGameMessage(NoOfMoves):
   print('ARGHHHHHH! The monster has eaten you. GAME OVER.')
   print('Maybe you will have better luck the next time you play
MONSTER!')
print('The number of moves that you survived in the cavern for
was'+ str(NoOfMoves))
```

5

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

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

Screen capture(s) showing correct cavern state:



			M		*	

followed by message "`The number of moves you took to find the flask was 3`";

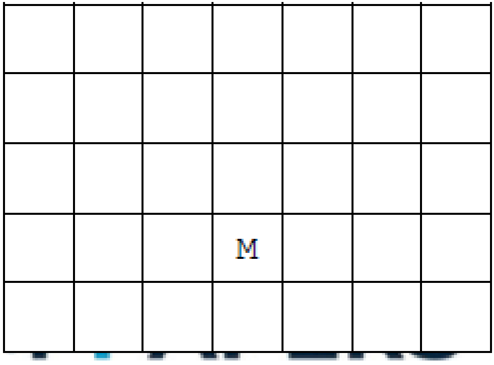
A Different message – if it matches code in (c)(i) and displays final value of `NoOfMoves` correctly
R If message "`The number of moves that you survived …`" is also shown

1

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

This is conditional on sensible code for (c)(i)

Screen capture(s) showing correct cavern state:



			M			

followed by message "`The number of moves that you survived in the cavern for was 2`";

A Different message – if it matches code in (c)(i) and displays final value of `NoOfMoves` correctly
R If message "`The number of moves you took…`" is also shown

1

(d) (i) `CalculateDistance` subroutine created – with begin and end of subroutine;
`PlayerPosition` and `MonsterPosition` passed as parameters to the `CalculateDistance` subroutine;
I additional unnecessary parameters
R global variables
A four integer values instead of two `CellReference` values
R passing by value for parameters of type `CellReference` (VB6 only)

Integer value returned by subroutine either as parameter passed by reference or by function return value; **R** global variable **A** real value

Calculates difference between the `NoOfCellsEast` for the monster and the player; **R** if the result can be a negative distance

Calculates difference between the `NoOfCellsSouth` for the monster and the player; **R** if the result can be a negative distance

Calculates the total distance between the monster and the player; **A** Incorrect values for differences in `NoOfCellsEast` and `NoOfCellsSouth` being added together

Distance calculated is actually returned by the subroutine; **A** use of global variable

I Case of identifiers
A Minor typos in identifiers
I Order of parameters in routine interface

Pascal

```
Function CalculateDistance(PlayerPosition, MonsterPosition :
TCellReference) : Integer;
   Var
      Distance : Integer;
   Begin
      If PlayerPosition.NoOfCellsEast >
MonsterPosition.NoOfCellsEast
         Then Distance := PlayerPosition.NoOfCellsEast –
MonsterPosition.NoOfCellsEast
         Else Distance := MonsterPosition.NoOfCellsEast –
PlayerPosition.NoOfCellsEast;
      If PlayerPosition.NoOfCellsSouth >
MonsterPosition.NoOfCellsSouth
         Then Distance := Distance +
PlayerPosition.NoOfCellsSouth –
MonsterPosition.NoOfCellsSouth
         Else Distance := Distance +
MonsterPosition.NoOfCellsSouth –
PlayerPosition.NoOfCellsSouth;
   CalculateDistance := Distance;
End;
```

Alternative answer

```
Distance := Abs(PlayerPosition.NoOfCellsEast –
MonsterPosition.NoOfCellsEast) +
Abs(PlayerPosition.NoOfCellsSouth –
MonsterPosition.NoOfCellsSouth));
```

Alternative answer

```
Distance := Trunc(Sqrt(Sqr(PlayerPosition.NoOfCellsEast –
MonsterPosition.NoOfCellsEast)) +
Sqrt(Sqr(PlayerPosition.NoOfCellsSouth –
MonsterPosition.NoOfCellsSouth)));
```

Alternative answer

```
Distance := Round(Sqrt(Sqr(PlayerPosition.NoOfCellsEast –
MonsterPosition.NoOfCellsEast)) +
Sqrt(Sqr(PlayerPosition.NoOfCellsSouth –
MonsterPosition.NoOfCellsSouth)));
```

Alternative answer

```
Distance2 : Integer;
...
Distance := PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast;
If Distance < 0
   Then
      Distance := Distance * -1;
Distance2 := PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth;
If Distance2 < 0
   Then
      Distance2 := Distance2 * -1;
Distance := Distance + Distance2;
```

VB.NET

```
Function CalculateDistance(ByVal PlayerPosition As
CellReference, ByVal MonsterPosition As CellReference) As
Integer
   Dim Distance As Integer
   If PlayerPosition.NoOfCellsEast >
MonsterPosition.NoOfCellsEast Then
      Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
   Else
      Distance = MonsterPosition.NoOfCellsEast -
PlayerPosition.NoOfCellsEast
   End If
   If PlayerPosition.NoOfCellsSouth >
MonsterPosition.NoOfCellsSouth Then
      Distance = Distance + PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
   Else
      Distance = Distance + MonsterPosition.NoOfCellsSouth -
PlayerPosition.NoOfCellsSouth
   End If
   CalculateDistance = Distance
End Function
```

Alternative answer

```
Distance = System.Math.Abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
System.Math.Abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

A this alternative answer if `System.Math` included

A give benefit of doubt for this answer if no evidence of `System.Math` included

Alternative answer

```
Distance = (((PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) ^ 2) ^ 0.5) +
(((PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth) ^ 2) ^ 0.5)
```

Alternative answer

```
Dim Distance2 As Integer
...
Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
If Distance < 0 Then
   Distance = Distance * -1
End If
```

```
Distance2 = PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
If Distance2 < 0 Then
   Distance2 = Distance2 * -1
End If
Distance = Distance + Distance2
```

VB6

```
Private Function CalculateDistance(ByRef PlayerPosition As
CellReference, ByRef MonsterPosition As CellReference) As
Integer
   Dim Distance As Integer
   If PlayerPosition.NoOfCellsEast >
MonsterPosition.NoOfCellsEast Then
      Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
   Else
      Distance = MonsterPosition.NoOfCellsEast -
PlayerPosition.NoOfCellsEast
   End If
   If PlayerPosition.NoOfCellsSouth >
MonsterPosition.NoOfCellsSouth Then
      Distance = Distance + PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
   Else
      Distance = Distance + MonsterPosition.NoOfCellsSouth -
PlayerPosition.NoOfCellsSouth
   End If
   CalculateDistance = Distance
End Function
```

Alternative answer

```
Distance = (((PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) ^ 2) ^ 0.5) +
(((PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth) ^ 2) ^ 0.5)
```

Alternative answer

```
Distance = Abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
Abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

Alternative answer

```
Dim Distance2 As Integer
...
Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
If Distance < 0 Then
   Distance = Distance * -1
End If
Distance2 = PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
If Distance2 < 0 Then
   Distance2 = Distance2 * -1
End If
Distance = Distance + Distance2
```

Java

```
int calculateDistance(CellReference playerPosition,
CellReference monsterPosition) {
   int distance;
if(playerPosition.noOfCellsEast>monsterPosition.noOfCellsEa
st){
   distance=playerPosition.noOfCellsEast-monsterPosition.no
OfCellsEast;
   } else{
   distance=monsterPosition.noOfCellsEast-playerPosition.no
OfCellsEast;
   }
if(playerPosition.noOfCellsSouth>monsterPosition.noOfCellsS
outh){
distance=distance+playerPosition.noOfCellsSouth-monsterPosi
tion.noOfCellsSouth;
   }else{
distance=distance+monsterPosition.noOfCellsSouth-playerPosi
tion.noOfCellsSouth;
   }
   return distance;
}
```

Alternative Answer

```
int calculateDistance(CellReference playerPosition,
CellReference monsterPosition) {
   int distance;
   distance = Math.abs(playerPosition.noOfCellsSouth -
monsterPosition.noOfCellsSouth);
   distance += Math.abs(playerPosition.noOfCellsEast -
monsterPosition.noOfCellsEast);
   return distance;
}
```

Alternative Answer

```
distance=(int)Math.sqrt(Math.pow((double)(playerPosition.no
OfCellsSouth - monsterPosition.noOfCellsSouth), 2))
+(int)Math.sqrt(Math.pow((double)(playerPosition.noOfCellsE
ast - monsterPosition.noOfCellsEast), 2));
```

Alternative Answer

```
distance=(int)Math.round(Math.sqrt(Math.pow((double)(player
Position.noOfCellsSouth - monsterPosition.noOfCellsSouth),
2))
+Math.sqrt(Math.pow((double)(playerPosition.noOfCellsEast -
monsterPosition.noOfCellsEast), 2)));
```

Alternative answer

```
int distance2;
...
distance = playerPosition.noOfCellsEast -
monsterPosition.noOfCellsEast;
if (distance < 0) {
   distance = distance * -1;
}
distance2 = playerPosition.noOfCellsSouth -
monsterPosition.noOfCellsSouth;
if (distance2 < 0) {
   distance2 = distance2 * -1;
}
distance = distance + distance2;
```

Python

```
def CalculateDistance(PlayerPosition, MonsterPosition):
   if PlayerPosition.NoOfCellsEast >
MonsterPosition.NoOfCellsEast:
      Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
   else:
      Distance = MonsterPositionNoOfCellsEast -
```

```
PlayerPosition.NoOfCellsEast
   if PlayerPosition.NoOfCellsSouth >
MonsterPosition.NoOfCellsSouth:
      Distance = Distance + PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
   else:
      Distance = Distance + MonsterPositionNoOfCellsSouth -
PlayerPosition.NoOfCellsSouth
   return Distance
```

Alternative Answer

```
Distance = abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

Alternative Answer

```
return abs(PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast) +
abs(PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)
```

Alternative Answer

```
import math
Distance =
math.trunc(math.sqrt(pow((PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast),2)) +
math.sqrt(pow((PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth),2)))
```

Alternative Answer

```
import math
Distance = round(math.sqrt((PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast)**2) +
math.sqrt((PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth)**2))
```

Alternative Answer

```
Distance = PlayerPosition.NoOfCellsEast -
MonsterPosition.NoOfCellsEast
if Distance < 0:
   Distance = Distance * -1
Distance2 = PlayerPosition.NoOfCellsSouth -
MonsterPosition.NoOfCellsSouth
if Distance2 < 0:
   Distance2 = Distance2 * -1
Distance = Distance + Distance2
```

7

(ii) Call to `CalculateDistance` subroutine;
R if parameter list does not match answer to (d)(i)
Displays "`Distance between monster and player: `" in correct place;
A. any place in code after call to `DisplayMoveOptions` and before call to `MakeMove`
A. minor typos in prompt
I capitalisation

Displays the calculated distance;
R. if no evidence of any calculation for the distance
R. if distance is displayed before call to `CalculateDistance` subroutine

R. if distance returned by `CalculateDistance` stored in a global variable
R. if distance calculated in part (d)(i) would not actually be displayed e.g. program would not compile/run
A. use of temporary variable to store the value returned by `CalculateDistance` with contents of temporary variable then displayed using output message

I Case of identifiers and output messages
A. Minor typos in output messages
I spacing in output messages

Pascal

```
DisplayMoveOptions;
Writeln('Distance between monster and player: ',
CalculateDistance(PlayerPosition, MonsterPosition));
```

VB.NET

```
DisplayMoveOptions()
Console.WriteLine("Distance between monster and player: " &
CalculateDistance(PlayerPosition, MonsterPosition))
```

VB6

```
DisplayMoveOptions()
WriteLine("Distance between monster and player: " &
CalculateDistance(PlayerPosition, MonsterPosition))
```

A `Text1.Text = Text1.Text & "Distance between monster and player: " & CalculateDistance(PlayerPosition, MonsterPosition)`

A `WriteLineWithMsg`

Java

```
   displayMoveOptions();
   console.println("Distance between monster and player: " +
calculateDistance(playerPosition, monsterPosition));
```

Python 2

```
DisplayMoveOptions()
print 'Distance to monster:',
CalculateDistance(PlayerPosition, MonsterPosition)
```

Alternative answer

```
DisplayMoveOptions()
print 'Distance to monster:' +
str(CalculateDistance(PlayerPosition, MonsterPosition))
```

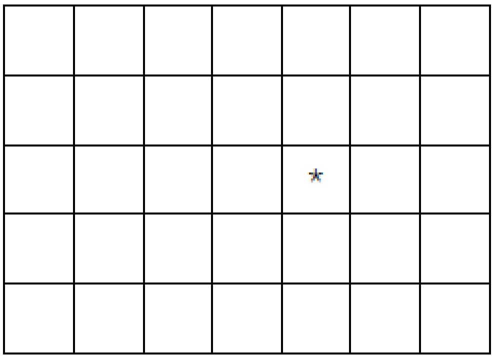
Python 3

```
DisplayMoveOptions()
print('Distance to monster:' +
str(CalculateDistance(PlayerPosition, MonsterPosition))
```

3

(iii) ****SCREEN CAPTURE(S)****
This is conditional on sensible code for (d)(i) and/or (d)(ii)

Player shown in the cell 3 south and 5 east of the northwest corner
AND
`"Distance between monster and player: 3"`
shown;

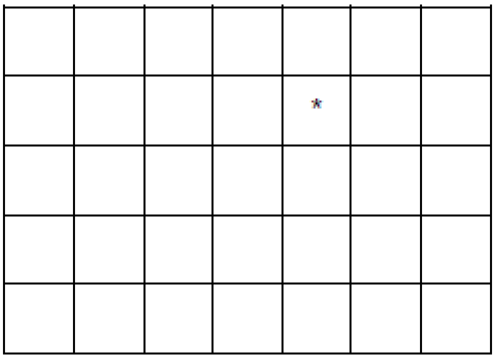


I monster symbol (M) displayed in the cavern

1

(iv) ****SCREEN CAPTURE(S)****
This is conditional on sensible code for (d)(i) and/or (d)(ii)

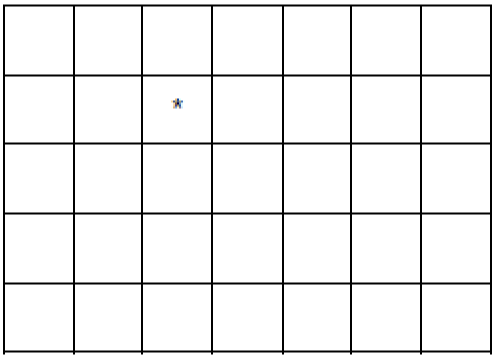
Player shown in the cell 2 south and 5 east of the northwest corner
AND
`"Distance between monster and player: 2"`
shown;



I monster symbol (M) displayed in the cavern

1

(v) ****SCREEN CAPTURE(S)****
This is conditional on sensible code for (d)(i) and/or (d)(ii)
Player shown in the cell 2 south and 3 east of the northwest corner
AND
`"Distance between monster and player: 2"`
shown;



I monster symbol (M) displayed in the cavern

1

35

Q11.

(a) **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;
A no loops
Alternative definitions:
A simple cycle is formed if any edge is added to graph;
Any two vertices can be connected by a unique simple path;

Max 1

(b) No route from entrance to exit / through maze;
Maze contains a loop/circuit ;
A more than one route through maze;
Part of the maze is inaccessible / enclosed;
R Responses that clearly relate to a graph rather than the maze

Max 1

(c)

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1
6	0	0	0	0	1	0	0
7	0	0	0	0	1	0	0

(allow some symbol in the central diagonal to indicate unused)

or

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	■	0	1	1	0	0	0
3	■	■	0	0	0	0	0
4	■	■	■	0	1	0	0
5	■	■	■	■	0	1	1
6	■	■	■	■	■	0	0
7	■	■	■	■	■	■	0

(with the shaded portion in either half – some indication must be made that half of the matrix is not being used. This could just be leaving it blank, unless the candidate has also represented absence of an edge by leaving cells blank)

1 mark for drawing a 7x7 matrix, labelled with indices on both axis and filled only with 0s and 1s, or some other symbol to indicate presence/absence of edge. e.g. T/F. Absence can be represented by an empty cell.
1 mark for correct values entered into matrix, as shown above;

2

(d) (i) Routine defined in terms of itself // Routine that calls itself;
A alternative names for routine e.g. procedure, algorithm
NE repeats itself

1

(ii) Stores return addresses;
Stores parameters;
Stores local variables; NE temporary variables
Stores contents of registers;
A To keep track of calls to subroutines/methods etc.

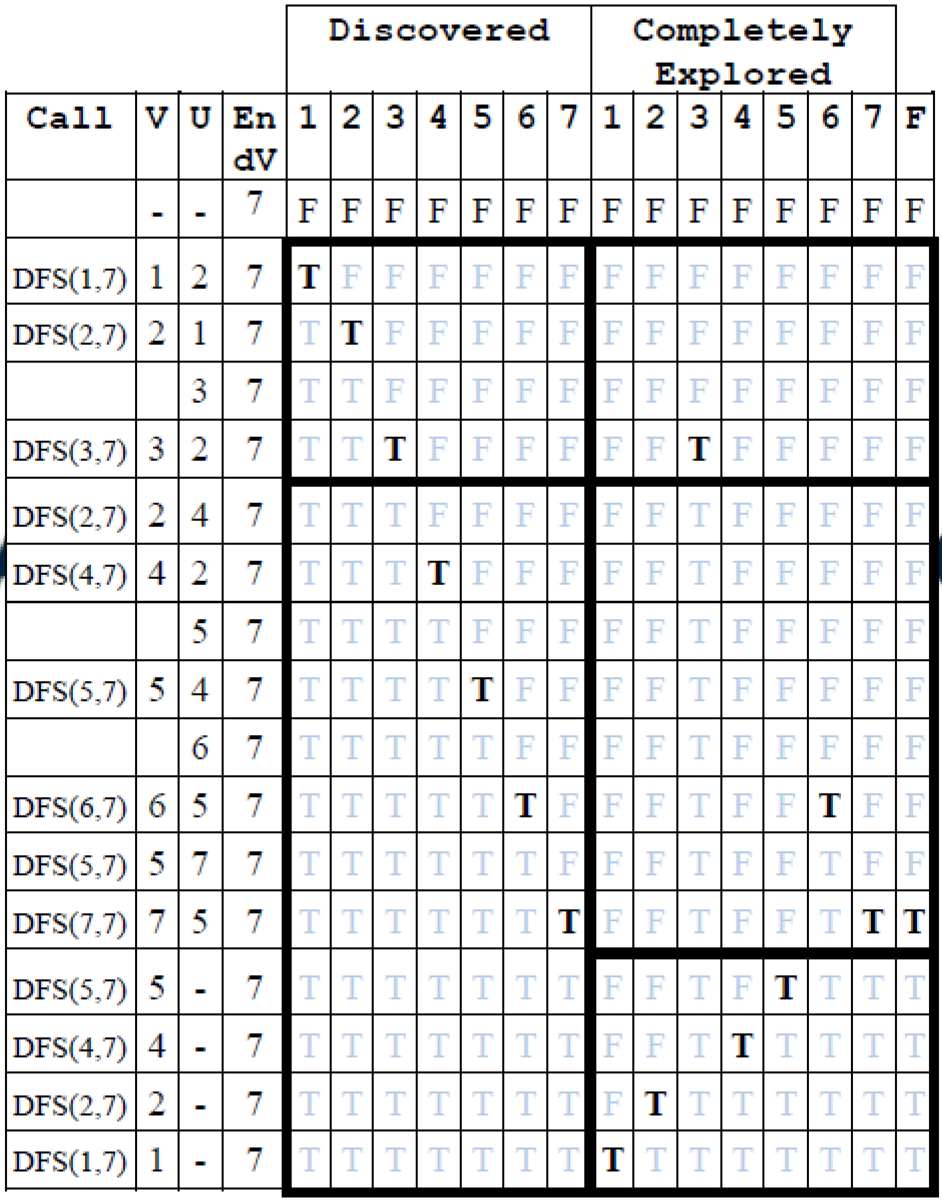
Max 1

Procedures / invocations / calls must be returned to in reverse order (of being called);
As it is a LIFO structure;
A FILO
As more than one / many return addresses / <u>sets of</u> values may need to be stored (at same time) // As the routine calls itself and for each call/invocation a new return address / new values must be stored;

Max 1

2

(e)



				Discovered							Completely Explored							
Call	V	U	EndV	1	2	3	4	5	6	7	1	2	3	4	5	6	7	F
	-	-	7	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(1,7)	1	2	7	**T**	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(2,7)	2	1	7	T	**T**	F	F	F	F	F	F	F	F	F	F	F	F	F
		3	7	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(3,7)	3	2	7	T	T	**T**	F	F	F	F	F	F	**T**	F	F	F	F	F
DFS(2,7)	2	4	7	T	T	T	F	F	F	F	F	F	T	F	F	F	F	F
DFS(4,7)	4	2	7	T	T	T	**T**	F	F	F	F	F	T	F	F	F	F	F
		5	7	T	T	T	T	F	F	F	F	F	T	F	F	F	F	F
DFS(5,7)	5	4	7	T	T	T	T	**T**	F	F	F	F	T	F	F	F	F	F
		6	7	T	T	T	T	T	F	F	F	F	T	F	F	F	F	F
DFS(6,7)	6	5	7	T	T	T	T	T	**T**	F	F	F	T	F	F	**T**	F	F
DFS(5,7)	5	7	7	T	T	T	T	T	T	F	F	F	T	F	F	T	F	F
DFS(7,7)	7	5	7	T	T	T	T	T	T	**T**	F	F	T	F	F	T	**T**	**T**
DFS(5,7)	5	-	7	T	T	T	T	T	T	T	F	F	T	F	**T**	T	T	T
DFS(4,7)	4	-	7	T	T	T	T	T	T	T	F	F	T	**T**	T	T	T	T
DFS(2,7)	2	-	7	T	T	T	T	T	T	T	F	**T**	T	T	T	T	T	T
DFS(1,7)	1	-	7	T	T	T	T	T	T	T	**T**	T	T	T	T	T	T	T

1 mark for having the correct values changes in each region highlighted by a *rectangle and no incorrect changes in the region. Ignore the contents of any cells that are not changed.*

A alternative indicators that clearly mean True and False.
A it is not necessary to repeat values that are already set (shown lighter in table)

5

[12]

Q12.

(a) **VB.Net**

```
Sub Main()
    Dim Names(4) As String
    Dim Current As Integer
    Dim Max As Integer
    Dim Found As Boolean
    Dim PlayerName As String

    Names(1) = "Ben"
    Names(2) = "Thor"
    Names(3) = "Zoe"
    Names(4) = "Kate"
    ;Max = 4
    Current = 1
    Found = False
    Console.WriteLine("What player are you looking for?")
    PlayerName = Console.ReadLine
    While Found = False And Current <= Max
        If Names(Current) = PlayerName Then
            Found = True
        Else
            Current = Current + 1
        End If
    End While
    If Found = True Then
        Console.WriteLine("Yes, they have a top score")
    Else
        Console.WriteLine("No, they do not have a top score")
    End If
    Console.ReadLine()
End Sub
```

VB6

```
Private Sub Form_Load()
    Dim Names(4) As String     A. Names(1 To 4)
    Dim Current As Integer
    Dim Max As Integer
    Dim Found As Boolean
    Dim PlayerName As String

    Names(1) = "Ben"
    Names(2) = "Thor"
    Names(3) = "Zoe"
    Names(4) = "Kate"
    Max = 4
    Current = 1
    Found = False
    PlayerName = InputBox("What player are you looking for?")
    While Found = False And Current <= Max
        If Names(Current) = PlayerName Then
            Found = True
        Else
            Current = Current + 1
        End If
    End While
    If Found = True Then
        MsgBox("Yes, they have a top score")
    Else
        MsgBox("No, they do not have a top score")
    End If
    End
End Sub
```

Pascal

```
Program Question;
Var
   Names : Array[1..4] Of String;
   Current : Integer;
   Max : Integer;
   Found : Boolean;
   PlayerName : String;
Begin
   Names[1] := 'Ben';
   Names[2] := 'Thor';
   Names[3] := 'Zoe';
   Names[4] := 'Kate';
   Max := 4;
   Current := 1;
   Found := False;
   Writeln('What player are you looking for?');
   Readln(PlayerName);
   While (Found = False) And (Current <= Max)
      Do
         Begin
            If Names[Current] = PlayerName
               Then Found := True
               Else Current := Current + 1;
         End;
   If Found = True
      Then Writeln('Yes, they have a top score')
      Else Writeln('No, they do not have a top score');
   Readln;
End.
```

Java

```
public class Question {

    AQAConsole console = new AQAConsole();

    public Question() {
         String[] names = new String[5];
         int max;
         int current;
         boolean found;
         String playerName;

         names[1] = "Ben";
         names[2] = "Thor";
         names[3] = "Zoe";
         names[4] = "Kate";
```

```
//possible alternative, which declares and
//instantiates in one.
//String[] names={"","Ben","Thor","Zoe","Kate"};
```

```
         current = 1;
         max = 4;
         found = false;

         playerName = console.readLine("What player are you
     looking for? ");
         while ((found == false) && (current <= max)) {
            if (names[current].equals(playerName)){
              found = true;
            } else {
              current++;
            } // end if/else
         } // end while

         if (found == true) {
             console.println("Yes, they have a top score");
         } else {
                    console.println("No, they do not have a top score");
         } // end if/else
     }// end CONSTRUCTOR
    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
    new Question();
    }
}
```

Python 2.6

```
Names = ["", "", "", "", ""]
Names[1] = "Ben"
Names[2] = "Thor"
Names[3] = "Zoe"
Names[4] = "Kate"
# Or:
# Names["", "Ben","Thor", "Zoe","Kate"]

# Or:
# Names = [""]
# Names.append("Ben")
# Names.append("Thor")
# Names.append("Zoe")
# Names.append("Kate")

Max = 4
Current = 1
Found = False
PlayerName = raw_input("What player are you looking
for?")
while (Found == False) and (Current <= Max):
     if Names[Current] == PlayerName:
         Found = True
     else:
         Current += 1
if Found == True: # accept if Found:
     print "Yes, they do have a top score"
else:
     print "No, they do not have a top score"
```

A Answers where `Max` is set to 5 and loop condition of `Current < Max`

A Answers where `Max` is set to 4 and loop condition of `Current < Max + 1`

Python 3

```
Names = ["", "", "", "", ""]
Names[1] = "Ben"
Names[2] = "Thor"
Names[3] = "Zoe"
Names[4] = "Kate"
# Or:
# Names["", "Ben","Thor", "Zoe","Kate"]

# Or:
# Names = [""]
# Names.append("Ben")
# Names.append("Thor")
# Names.append("Zoe")
# Names.append("Kate")

Max = 4
Current = 1
Found = False
PlayerName = input("What player are you looking
for?")
while (Found == False) and (Current <= Max):
     if Names[Current] == PlayerName:
         Found = True
     else:
         Current += 1
if Found == True: # accept if Found:
     print("Yes, they do have a top score")
else:
     print("No, they do not have a top score")
```

A Answers where `Max` is set to 5 and loop condition of `Current < Max`

A Answers where `Max` is set to 4 and loop condition of `Current <`

Mark as follows:

Correct variable declarations for `Max, Current, Found, PlayerName` and correct declaration for the `Names` array;
Four correct values assigned to the correct positions in the Names array;
`Max, Current, Found` initialised correctly;
Correct prompt followed by `PlayerName` assigned value entered by user;
`WHILE` loop formed correctly and correct conditions for the termination of the loop;
First `IF` followed by correct condition and `IF` statement is inside the loop;
`THEN` followed by correct assignment statement within a correctly formed `IF` statement;
`ELSE` followed by correct assignment statement within a correctly formed `IF` statement;
Second `IF` followed by correct condition and `IF` is after the loop;
`THEN` followed by correct output within a correctly formed `IF` statement;
`ELSE` followed by correct output within a correctly formed `IF` statement;

I Case of variable names, player names and output messages
A Minor typos in variable names and output messages
A Max declared as a constant instead of a variable
A Alternative conditions with equivalent logic for the loop
A Array positions 0–3 used instead of 1–4 if consistent usage throughout program

11

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

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

Mark as follows:

'What player are you looking for' + user input of ‘ Thor’ ;
'Yes, they have a top scor' message shown;
I spacing
R If code for (a) would not produce this test run

2

(c) * * * * SCREEN CAPTURE* * * *

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

Mark as follows:

'What player are you looking for?' + user input of ‘Imran’ ;
'No, they do not have a top score' message shown;
I spacing
R If code for (a) would not produce this test run

2

[15]

Q13.

(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 <u>found so far</u> // 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]

Q14.

(a) (i) **VB.Net**

```
If VirtualDiceGame Then
    AppealDieResult = Int(Rnd() * 5) + 1
Else
    Console.WriteLine("Please roll the appeal die and then
enter your result.")
    Console.WriteLine()
    Console.WriteLine("Enter 1 if the result is NOT OUT")
    Console.WriteLine("Enter 2 if the result is CAUGHT")
    Console.WriteLine("Enter 3 if the result is LBW")
    Console.WriteLine("Enter 4 if the result is BOWLED")
    Console.WriteLine("Enter 5 if the result is RUN OUT")
    Console.WriteLine()
    Console.Write("Result: ")
    AppealDieResult = Console.ReadLine
    Console.WriteLine()
End If
```

VB6

```
If VirtualDiceGame Then
    AppealDieResult = Int(Rnd() * 5) + 1
Else
    WriteLine ("Please roll the appeal die and then enter your
result.")
    WriteLine ("")
    WriteLine ("Enter 1 if the result is NOT OUT")
    WriteLine ("Enter 2 if the result is CAUGHT")
    WriteLine ("Enter 3 if the result is LBW")
    WriteLine ("Enter 4 if the result is BOWLED")
    WriteLine ("Enter 5 if the result is RUN OUT")
    WriteLine ("")
    AppealDieResult = ReadLine("Result:")
    WriteLine ("")
End If
```

A Text1.Text = Text1.Text & "Enter 5 if the result is RUN OUT"
A WriteLineWithMsg

Pascal

```
If VirtualDiceGame
    Then AppealDieResult := Random(5) + 1
   Else
      Begin
         Writeln('Please roll the appeal die and then enter
your result.');
         Writeln;
         Writeln('Enter 1 if the result is NOT OUT');
         Writeln('Enter 2 if the result is CAUGHT');
         Writeln('Enter 3 if the result is LBW');
         Writeln('Enter 4 if the result is BOWLED');
         Writeln('Enter 5 if the result is RUN OUT');
         Writeln;
         Write('Result: ');
         Readln(AppealDieResult);
         Writeln;
      End;
```

Java

```
if (virtualDiceGame) {
    appealDieResult = objRandom.nextInt(5) + 1;
} else {
    console.println("Please roll the appeal die and
then enter your result.");
    console.println();
    console.println("Enter 1 if the result is NOT
OUT");
    console.println("Enter 2 if the result is
CAUGHT");
    console.println("Enter 3 if the result is LBW");
    console.println("Enter 4 if the result is
BOWLED");
    console.println("Enter 5 if the result is RUN
OUT");
    console.println();
    appealDieResult = console.readInteger("Result:
```

```
");
    console.println();
}
```

Python 2.6

```
def RollAppealDie(VirtualDiceGame):
     if VirtualDiceGame:
         AppealDieResult = random.randint(1,5)
     else:
         print "Please roll the appeal die and then enter your 
result."
         print ""
         print "Enter 1 if the result is NOT OUT"
         print "Enter 2 if the result is CAUGHT"
         print "Enter 3 if the result is LBW"
         print "Enter 4 if the result is BOWLED"
         print "Enter 5 if the result is RUN OUT"
         print ""
         AppealDieResult = input("Result: ")
         print ""
     return AppealDieResult
```

Python 3

```
def RollAppealDie(VirtualDiceGame):
     if VirtualDiceGame:
         AppealDieResult = random.randint(1,5)
     else:
         print("Please roll the appeal die and then enter your 
result.")
         print()
         print("Enter 1 if the result is NOT OUT")
         print("Enter 2 if the result is CAUGHT")
         print("Enter 3 if the result is LBW")
         print("Enter 4 if the result is BOWLED")
         print("Enter 5 if the result is RUN OUT")
         print()
         AppealDieResult = int(input("Result: "))
         print()
     return AppealDieResult
```

Mark as follows:

Generates random number between 1 and 5;
Appropriate prompt added if real dice being used;
I minor typos and capitalisation in prompt
A alternative sensible prompt

2

(ii) **VB.Net**

```
Select Case AppealDieResult
    Case 1
        Console.WriteLine("Not out!")
    Case 2
        Console.WriteLine("Caught!")
    Case 3
        Console.WriteLine("LBW!")
    Case 4
        Console.WriteLine("Bowled!")
    Case 5
        Console.WriteLine("Run Out!")
End Select
```

VB6

```
Select Case AppealDieResult
    Case 1
        WriteLineWithMsg ("Not out!")
    Case 2
        WriteLineWithMsg ("Caught!")
    Case 3
        WriteLineWithMsg ("LBW!")
    Case 4
        WriteLineWithMsg ("Bowled!")
    Case 5
        WriteLineWithMsg ("Run out!")
End Select
```

A WriteLine / WriteWithMsg / Msgbox instead of WriteLineWithMsg
A Text1.Text = Text1.Text & “Run out!”

Pascal

```
Case AppealDieResult Of
        1 : Writeln('Not out!');
        2 : Writeln('Caught!');
        3 : Writeln('LBW!');
        4 : Writeln('Bowled!');
        5 : Writeln('Run out!');
    End;
```

Java

```
switch (appealDieResult) {
     case 1:
        console.println("Not out!");
        break;
     case 2:
        console.println("Caught!");
        break;
     case 3:
        console.println("LBW!");
        break;
     case 4:
        console.println("Bowled!");
        break;
     case 5:
        console.println("Run out!");
        break; /////////////optional
}
```

Python 2.6

```
def DisplayAppealDieResult(AppealDieResult):
    if AppealDieResult == 1:
       print "Not out!"
    elif AppealDieResult == 2:
       print "Caught!"
    elif AppealDieResult == 3:
       print "LBW!"
    elif AppealDieResult == 4:
       print "Bowled!"
    elif AppealDieResult == 5:
       print "Run out!"
```

Python 3

```
def DisplayAppealDieResult(AppealDieResult):
     if AppealDieResult == 1:
         print("Not out!")
     elif AppealDieResult == 2:
         print("Caught!")
```

```
        elif AppealDieResult == 3:
            print("LBW!")
        elif AppealDieResult == 4:
            print("Bowled!")
        elif AppealDieResult == 5:
            print("Run out!")
```

Mark as follows:
5th case option added;
Appropriate output message in 5th case option;
I minor typos and capitalisation in output message 2

2

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

Screen capture showing run out (option 5) message shown to user;
User enters “5” and correct output message showing ‘RUN OUT!’;
A Alternative output message if matches code for (a)(i) / (a)(ii)

2

(b) (i) **VB.Net**

```
If PlayerOneScore > PlayerTwoScore Then
Console.WriteLine(PlayerOneName & " wins!")
If PlayerTwoScore > PlayerOneScore Then
Console.WriteLine(PlayerTwoName & " wins!")
If PlayerOneScore = PlayerTwoScore Then
Console.WriteLine("A draw!")
```

VB6

```
If PlayerOneScore > PlayerTwoScore Then
WriteLineWithMsg (PlayerOneName & " wins!")
If PlayerTwoScore > PlayerOneScore Then
WriteLineWithMsg (PlayerTwoName & " wins!")
If PlayerOneScore = PlayerTwoScore Then
WriteLineWithMsg ("A draw!")
```

A Using MsgBox/WriteLine/WriteWithMsg for output instead of WriteLineWithMsg
A Text.Text1 = Text.Text1 & "A draw!"

Pascal

```
If (PlayerOneScore > PlayerTwoScore)
    Then Writeln(PlayerOneName, ' wins!');
If (PlayerTwoScore > PlayerOneScore)
    Then Writeln(PlayerTwoName, ' wins!');
If (PlayerOneScore = PlayerTwoScore)
    Then Writeln('A draw!');
```

Java

```
if (playerOneScore > playerTwoScore) {
    console.println(playerOneName + " wins!");
} // end if
if (playerTwoScore > playerOneScore) {
    console.println(playerTwoName + " wins!");
} // end if
if (playerTwoScore == playerOneScore) {
    console.println("A draw!");
}
```

Python 2.6

```
if PlayerOneScore > PlayerTwoScore:
     print PlayerOneName, " wins!"
if PlayerTwoScore > PlayerOneScore:
     print PlayerTwoName, " wins!"
if PlayerOneScore = = PlayerTwoScore:
     print "A draw!"
```

Python 3

```
if PlayerOneScore > PlayerTwoScore:
     print PlayerOneName, "wins!"
if PlayerTwoScore > PlayerOneScore:
     print PlayerTwoName, "wins!"
if PlayerOneScore = = PlayerTwoScore:
     print "A draw!"
```

Mark as follows:

IF statement;
with correct condition;
suitable output message shown under, and only under, correct circumstances;

3

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

Mark as follows:

Test showing both player scores are 0;
Correct message shown; *This is conditional on sensible code for (b)(ii)*

2

(c) (i) **VB.Net**

```
Console.Write("Result: ")
BowlDieResult = Console.ReadLine()
Console.WriteLine()
While BowlDieResult < 1 Or BowlDieResult > 6
    Console.WriteLine("Please enter a value between 1
and 6 only")
    BowlDieResult = Console.ReadLine
End While
```

Alternative Answer – VB.Net

```
Do
    Console.Write("Result: ")
    BowlDieResult = Console.ReadLine
    If BowlDieResult < 1 Or BowlDieResult > 6 Then
       Console.WriteLine("Please enter a number between 1 and
6 only")
    End If
Loop Until BowlDieResult >= 1 And BowlDieResult <=6
```

VB6

```
BowlDieResult = ReadLine("Result:")
While BowlDieResult < 1 Or BowlDieResult > 6
    BowlDieResult = ReadLine("Please enter a value
between 1 and 6 only")
End While
```

A `InputBox` instead of `ReadLine`

Alternative Answer – VB6

```
Do
    BowlDieResult = ReadLine("Result:")
    If BowlDieResult < 1 Or BowlDieResult > 6 Then
       BowlDieResult = WriteLine("Please enter a value between
1 and 6 only")
    End If
Loop Until BowlDieResult >= 1 And BowlDieResult <=6
```

Pascal

```
Repeat
  Write('Result: ');
  Readln(BowlDieResult);
  If (BowlDieResult < 1) Or (BowlDieResult > 6)
      Then Writeln('Please enter a value between 1 and 6
only');
  Until (BowlDieResult >= 1) And (BowlDieResult <=6);
```

Alternative Answer - Pascal

```
Write('Result: ');
Readln(BowlDieResult);
Writeln;
While (BowlDieResult < 1) Or (BowlDieResult > 6)
   Do
      Begin
          Writeln('Please enter a value between 1 and 6 only');
          Readln(BowlDieResult);
      End;
```

Java

```
do {
    bowlDieResult = console.readInteger("Result: ");
    if ((bowlDieResult < 1 || bowlDieResult > 6))
    {
     console.println("Please enter a value between 1 and 6
only");
     }
} while (bowlDieResult < 1 || bowlDieResult > 6);
```

Python 2.6

```
while BowlDieResult not in [1,2,3,4,5,6]:
while BowlDieResult not in range(1,7):
while BowlDieResult < 1 or BowlDieResult >6:
while not (1 <= BowlDieResult <= 6):
         BowlDieResult = input("Please enter a value
between 1 and 6 only: ")
```

Python 3

```
while BowlDieResult not in [1,2,3,4,5,6]:
while BowlDieResult not in range(1,7):
while BowlDieResult < 1 or BowlDieResult >6:
while not (1 <= BowlDieResult <= 6):
         BowlDieResult = int(input("Please enter a value
between 1 and 6 only: "))
```

Mark as follows:

Suitable iteration structure used in appropriate place in the Skeleton Program with one correct condition;

Use of OR logical operator and have second condition correct for iterative structure;

A Alternative logic using AND and NOT logical operators

Correct error message and get choice from user – both inside the loop;

Error message is displayed if, and only if, invalid data entered by user;

I. minor typos and capitalisation in output message

4

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

This is conditional on sensible code for (c)(i)

Mark as follows:

Test showing a value of 0 entered and the correct output message;
Test showing a value of 2 entered and the correct output message;
Test showing a value of 7 entered and the correct output message;

I Order of tests

A Alternative error message if matches code for (c)(i)

3

(d) (i) **VB.Net**

```
Console.WriteLine("4. Display top scores")
Console.WriteLine("5. Save top scores")
Console.WriteLine("9. Quit")
```

VB6

```
WriteLine ("4. Display top scores")
WriteLine ("5. Save top scores")
WriteLine ("9. Quit")
```

Pascal

```
Writeln('4. Display top scores');
Writeln('5. Save top scores');
Writeln('9. Quit');
```

Java

```
console.println("4. Display top scores");
console.println("5. Save top scores");
console.println("9. Quit");
```

Python 2.6

```
def DisplayMenu():
    print "Dice Cricket"
    print ""
    print "1. Play game version with virtual dice"
    print "2. Play game version with real dice"
    print "3. Load top scores"
    print "4. Display top scores"
    print "5. Save top scores"
    print "9. Quit"
```

Python 3

```
print("4. Display top scores")
print("5. Save top scores")
print("9. Quit")
```

A minor typos in output message

1

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

```
If OptionChosen < 1 Or (OptionChosen > 5 And
OptionChosen <> 9) Then
```

Pascal

```
If (OptionChosen < 1) Or ((OptionChosen > 5) And
(OptionChosen <> 9))
    Then
```

Java

```
if ((optionChosen < 1) || ((optionChosen > 5) &&
(optionChosen != 9))) {
```

Python 2.6

```
def GetMenuChoice():
     OptionChosen = input("Please enter your choice:")
     if (OptionChosen < 1 or (OptionChosen > 5 and
OptionChosen != 9)):
          Print ""
          print "That was not one of the allowed options.
Please try again: "
     return OptionChosen
```

Python 3

```
def GetMenuChoice():
     OptionChosen = int(input("Please enter your
choice: "))
     if (OptionChosen < 1 or (OptionChosen > 5 and
OptionChosen != 9)):
        print()
        print("That was not one of the allowed options. Please
try again: ")
      return OptionChosen
```

Mark as follows:
OptionChosen > 5 // OptionChosen >= 6; 1

(iii) **VB.Net**

```
Sub SaveTopScores(ByVal TopScores() As TTopScore)
   Dim Count As Integer
   Dim LineToAddToFile As String
   FileOpen(1, "HiScores.txt", OpenMode.Output)
   For Count = 1 To MaxSize
     LineToAddToFile = TopScores(Count).Name & "," &
TopScores(Count).Score
     PrintLine(1, LineToAddToFile)
   Next
   FileClose(1)
End Sub
```

VB6

```
Private Sub SaveTopScores(ByRef TopScores() As
TTopScore)
Dim Count As Integer
Open "HiScores.txt" For Output As #1
For Count = 1 To MaxSize
    Print #1, TopScores(Count).Name & "," &
Str(TopScores(Count).Score)
Next
Close #1
End Sub
```

Pascal

```
Procedure SaveTopScores(TopScores : TTopScores);
Var
   Count : Integer;
   LineToAddToFile : String;
   CurrentFile : TextFile;
Begin
   Assign(CurrentFile, 'HiScores.txt');
   ReWrite(CurrentFile);
   For Count := 1 To MaxSize
     Do
```

```
          Begin
            LineToAddToFile :=
IntToStr(TopScores[Count].Score)
            LineToAddToFile := TopScores[Count].Name + ',' +
LineToAddToFile;
            Writeln(CurrentFile, LineToAddToFile);
        End;
    Close(CurrentFile);
End;
```

A `Str(TopScores[Count].Score, LineToAddToFile);`
instead of
`LineToAddToFile := IntToStr(TopScores[Count].Score)`

Java

```
void saveTopScores(TopScore[] topScores) {
     AQAWriteTextFile currentFile = new
AQAWriteTextFile();
     currentFile.openFile("hitest.txt");
     int count;
     for (count = 1; count <= MAX_SIZE; count++) {
        String lineToAddToFile = topScores[count].name + ", ";
        lineToAddToFile = lineToAddToFile +
String.valueOf(topScores[count].score);
        currentFile.writeToTextFile(lineToAddToFile);
    } // end for count
    currentFile.closeFile();
}
```

Python 2.6

```
def SaveTopScores(TopScores):
    OutFile = open("HiScores.txt","w")
    Count = 1
    for Count in range(1, MAX_SIZE+1):
        LineToAddToFile = TopScores[Count].Name + "," +
str(TopScores[Count].Score) + "\n":
        OutFile.write(LineToAddToFile)
    OutFile.close()
# or more likely
def SaveTopScores(TopScores):
    Outfile = open(“HiScores.txt”,”w”)
    For score in (TopScores[1], TopScores[2],
TopScores[3], TopScores[4]):
            Line = score.Name + “,”+
str(score.Score) + “\n”
Outfile.write(line)
            Outfile.close()
```

Python 3

```
def SaveTopScores(TopScores):
      CurrentFile = open("HiScores.txt","w")
      Count = 1
      for Count in range(1, MAX_SIZE+1):
           LineToAddToFile = TopScores[Count].Name + "," +
str(TopScores[Count].Score) + "\n"
           CurrentFile.write(LineToAddToFile)
      CurrentFile.close()
```

Mark as follows:
Correctly named subroutine declared; **I** capitalisation **R** other mistakes in identifier
File opened correctly (for output);
First line to add into file consists of the 1st name; a comma and the

1st score;
First line written to file correctly;
2nd, 3rd and 4th lines would be written to the file correctly;
File closed correctly;

Additional marks for good programming practice=
(Max 3)
TopScores array passed as a parameter;
Use of iterative structure and counter used within iterative structure - going from 1 to MaxSize (**R 4**);
Sensible identifier names used for all variables/parameters;
Evidence of sensible commenting of source code;

10

(iv) **VB.Net**

```
Loop Until (OptionSelected >= 1 And OptionSelected
<= 5) Or OptionSelected = 9
Console.WriteLine()
If OptionSelected >= 1 And OptionSelected <= 5 Then
     Select Case OptionSelected
         Case 1 : PlayDiceGame(PlayerOneName,
PlayerTwoName, True, TopScores)
         Case 2 : PlayDiceGame(PlayerOneName,
PlayerTwoName, False, TopScores)
         Case 3 : LoadTopScores(TopScores)
         Case 4 : DisplayTopScores(TopScores)
         Case 5 : SaveTopScores(TopScores)
     End Select
```

VB6

```
Loop Until (OptionSelected >= 1 And OptionSelected
<= 5) Or OptionSelected = 9
If OptionSelected >= 1 And OptionSelected <= 5 Then
     Select Case OptionSelected
        Case 1: Call PlayDiceGame(PlayerOneName,
PlayerTwoName, True, TopScores)
        Case 2: Call PlayDiceGame(PlayerOneName,
PlayerTwoName, False, TopScores)
        Case 3: LoadTopScores(TopScores)
        Case 4: Call DisplayTopScores(TopScores)
        Case 5: Call SaveTopScores(TopScores)
```

Pascal

```
Until OptionSelected In [1..5, 9];
Writeln;
If OptionSelected In [1..5]
     Then
        Case OptionSelected Of
           1 : PlayDiceGame(PlayerOneName,
PlayerTwoName, True, TopScores);
           2 : PlayDiceGame(PlayerOneName,
PlayerTwoName, False, TopScores);
           3 : LoadTopScores(TopScores);
           4 : DisplayTopScores(TopScores);
           5 : SaveTopScores(TopScores);
End;
```

Java

```
do {
   displayMenu();
   optionSelected = getMenuChoice();
} while (!((optionSelected >= 1 && optionSelected
<= 5) || optionSelected == 9));
if (optionSelected >= 1 && optionSelected <= 5) {
   switch (optionSelected) {
      case 1:
          playDiceGame(playerOneName, playerTwoName, true,
topScores);
          break;
      case 2:
          playDiceGame(playerOneName, playerTwoName, false,
topScores);
          break;
      case 3:
          loadTopScores(topScores);
          break;
      case 4:
          displayTopScores(topScores);
          break;
      case 5:
          saveTopScores(topScores);
          break; //optional
   } // end case
} // end if
```

Python 2.6

```
while OptionSelected != 9:
    DisplayMenu()
    OptionSelected = GetMenuChoice()
    while OptionSelected not in [1,2,3,4,5,9]:
        DisplayMenu()
        OptionSelected = GetMenuChoice()
    print ""
    if OptionSelected in [1,2,3,4,5]:
        if OptionSelected == 1:
            PlayDiceGame(PlayerOneName,
PlayerTwoName,  True, TopScores)
        elif OptionSelected == 2:
PlayDiceGame(PlayerOneName,
            PlayerTwoName, False, TopScores)
        elif OptionSelected == 3:
            LoadTopScores(TopScores)
        elif OptionSelected == 4:
            DisplayTopScores(TopScores)
        elif OptionSelected == 5:
            SaveTopScores(TopScores)
```

Python 3

```
while OptionSelected != 9:
    DisplayMenu()
     OptionSelected = GetMenuChoice()
     while OptionSelected not in [1,2,3,4,5,9]:
          DisplayMenu()
          OptionSelected = GetMenuChoice()
     print()
     if OptionSelected in [1,2,3,4,5]:
        if OptionSelected == 1:
            PlayDiceGame(PlayerOneName, PlayerTwoName, True,
TopScores)
        elif OptionSelected == 2:
            PlayDiceGame(PlayerOneName,
            PlayerTwoName, False, TopScores)
        elif OptionSelected == 3:
            LoadTopScores(TopScores)
        elif OptionSelected == 4:
            DisplayTopScores(TopScores)
        elif OptionSelected == 5:
            SaveTopScores(TopScores)
```

Mark as follows:
Additional case statement for `OptionSelected` being 5;
Procedure call;
Passing `TopScores` as a parameter;
Loop terminating condition **and** selection condition range both changed from 1-4 to 1-5;

4

(iv) ****SCREEN CAPTURE****

Adapted menu is displayed; *This is conditional on sensible answer for question (d)(i)*

option 5 is selected, and accepted as valid input; *This is conditional on sensible answer for questions (d)(ii) and (d)(iv)*

2

(v) ****SCREEN CAPTURE****
This is conditional on sensible answer for (d)(ii), (ii) and (iv)

Contents of file are **exactly** as follows:

Ricky,12
Sachin,45
Brian,2
Janet,4

A Screen capture showing contents of text file
I Minor typos & capitalisation in Janet's name
R If Janet's name in the text file does not match the name used in (d)(iv)

(e) (i) Generate wider range of random numbers; add extra case statements for low score values / give low score values a bigger range in case statements than high score values;
//
Create a list/array containing a list of possible bowl die results where there are more 1s and 5s than 3s and 4s; generate a random number between 1 and the list size and use the bowl die result in that position in the list/array;

Mark as follows:
Generate a wider range of random numbers; Explain how the extra random numbers could be used to have a higher chance of getting a score of 1 or 0 than a score of 4 or 6;
A Replace case statement with if statements to allow different score values to have ranges of values associated with them (Pascal Only)
A Other sensible suggestions for modifications to the Skeleton Program that would result in the desired behaviour change.
MAX 1 if suggested changes would adversely effect other aspects of the game represented in the Skeleton Program e.g. does result in more lower scores than higher scores but would prevent a player from getting a result of out.

2

Q15.

(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]

Q16.

Meaningful/appropriate/suitable identifiers //
A example;
Indentation // effective use of white space;
Subroutines / Procedures and functions/methods/modules; with interfaces // using parameters to pass values;
Subroutines / Procedures and functions/methods/modules should execute a single task;
Appropriate use of structured statements // use of (selection and repetition)/repetition;
Avoid use of goto statements;
Consistent use of case/style for identifier names;
Use of <u>named</u> constants;
Use of user-defined data types;
Use of libraries;

House-style naming conventions // following conventions;
A by explained example
A Use of local variables
R Commenting
R "easier to understand"

Max 3

[3]

Q17.

(a) (i) Board // PlayerOneName // PlayerTwoName // PlayerOneScore // PlayerTwoScore // XCoord // YCoord // ValidMove // NoOfMoves // GameHasBeenWon // GameHasBeenDrawn // CurrentSymbol // StartSymbol // PlayerOneSymbol // PlayerTwoSymbol // Answer

Java only: console;

1

(ii) Row // Column // RandomNo // ValidMove // XOrOHasWon // WhoStarts;
VB6 only: BoardAsString;
Java and Python: X // Y;
Java and C#: ObjRandom;

1

(iii) A global variable is accessible/useable from anywhere in the program;
A local variable is only accessible / useable in the program block / procedure / function / subroutine / method in which it is declared;
//
Local variables only exist/use memory whilst the procedure / function / subroutine / method is executing; global variables exist / use memory the whole time the program is executing;

2

(iv) When the user enters ‘X' ; **or** ‘O’; // When PlayerOneSymbol contains ‘X’; **or** ‘O’;

2

(v) Because players could be making moves referring to non-empty cells; as no check is made for this (in the CheckValidMove subroutine); // Because some illegal moves are allowed;;

Mark as follows:
a move that is not legal being attempted (**A** by example); and is allowed (**A** by implication);

2

(vi) NoOfMoves // Row // Column;

1

(vii) PlayerOneName // PlayerTwoName // WhoStarts // PlayerTwoSymbol // RandomNo;
Python only: X // Y;[

1

(viii) CheckValidMove;

1

(ix) **VB.NET**
RandomNo = Rnd()*100 // WhoStarts = "X" // WhoStarts = “O”// GetWhoStarts = WhoStarts;

VB6

```
RandomNo = Rnd() *100 + 1 // WhoStarts = "X" // WhoStarts = "O"
// GetWhoStarts = WhoStarts;
```

Pascal

```
RandomNo := Random(100) // WhoStarts := 'O ' // [WhoStarts :=
'X' // GetWhoStarts := WhoStarts;
```

Java

```
Random objRandom = new Random() //
randomNo = objRandom.nextInt(100) // whoStarts = 'X' //
whoStarts = 'O'
```

Python

```
RandomNo = random.randint(0, 100) //
WhoStarts = 'X' // WhoStarts = 'O';
```

R if extra code included

1

(x) It looks at the remainder obtained by dividing `RandomNo` by 2;
A any explanation that clearly explains both sides of comparison
A if the random number /`RandomNo` is even;

If the value is 0/even it sets `WhoStarts` to 'X';
if the value is not 0/odd it sets `WhoStarts` to 'O';

Award only 1 mark of the 2 available marks labelled with asterisks(*) if candidate has identified conditions but described outcomes in terms of who will start game instead of assignment of value into WhoStarts. Candidate must cover both the `Then and Else` parts to get this 1 mark if specific variable name not used.

3

(b) (i) Boundary values are those that are just inside, on and just outside the range of allowed values;

1

(ii) 2; 3; 4;

R. non-integer values
Max 1 if additional values given

3

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

Screen capture showing boundary test resulting in correct behaviour;
Must match one of the boundary values given in(b)(ii).

R. If screen capture does not show a correct boundary value given as an answer to question (b)(ii)

1

[20]

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):

	X	X	X

X	X	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) (i) **** SCREEN CAPTURE ****
"The new word?" + setter input 'EAGLE' ;
input of correct guess 'EAGLE' ; (**A** 'eagle' if code in (b) has evidence for use of function Ucase, .ToUpper, etc.)
correct logic demonstrated with "CORRECT" ;
NB VB6 – all three stages must be evidenced

3

(ii) **** SCREEN CAPTURE ****
setter input 'BEAR'
"Your guess ?" + any incorrect guess ;
correct logic demonstrated with "INCORRECT" ;
NB VB6 – all three stages must be evidenced

3

(b) **Visual Basic**

```
Dim NewWord As String
Dim UserWordGuess As String
Console.Write("The new word?")
NewWord = Console.ReadLine

Console.Write("Your guess?")
```

```
UserWordGuess = Console.ReadLine

If UserWordGuess = NewWord

    Then Console.WriteLine("CORRECT")
    Else Console.WriteLine("INCORRECT")
End If
```

Pascal

```
Var
    NewWord : String;
    UserWordGuess : String;

Begin
    Write(‘The new word?');
    Readln(NewWord);
    Write(‘Your guess?');
    Readln(UserWordGuess);
    If UserWordGuess = NewWord
      Then Writeln(‘CORRECT')
      Else Writeln(‘INCORRECT');

    Readln;
End.
```

Mark as follows:
evidence of two variables declared ;
data types appropriate to the language for both variables ;
correct two identifier names used – NewWord □ UserWordGuess ;
(**A** case variations)

correct user prompt "The new word?" (**A** 'imprecise') ;

correctly formed IF followed by condition;
THEN clause followed by the logically correct output (**A** 'imprecise') ;
ELSE clause ;   followed by the logically correct output (**A** 'imprecise') ;

NB. Two separate IF statements scores Maximum 2

JAVA

```
class Question4 {

    Console console = new Console();
    String newWord = "";
    String userWordGuess;

    public Question4(){
        newWord=console.readLine("The new word?");
        userWordGuess=console.readLine("Your guess?");
        if(userWordGuess.equals(newWord)) {
            console.println("CORRECT");
        } else {
            console.println("INCORRECT");
        } // end if / else
    } // end construct or

    public static void main(String[] args) {
        new Question4();
        System.exit(0);
    } // end Main
} // end Question4
```

Max 7

Python

```
NewWord = raw_input("The new word?")
UserWordGuess = raw_input("Your Guess?")
if UserWordGuess == NewWord:
    print "CORRECT"
else:
    print "INCORRECT"
raw_input() # keep window on screen
```

Max 7

[27]

Q21.

(a) section of code can be referred to by name ;
aids readability ;
aids testing ;
code is easier to maintain / debug ;
the same block of code can be used repeatedly within the program ;
reusable within other programs ;
they encourage the use of local variables ;
reduces the complexity / results in less code in the main body of the program ;
they are ‘building blocks’ for structured programming ;

Max 3

(b) (i) **General:** Do not give credit for variables which are stated as part of an assignment statement **A** Variable shown in a declaration statement
`PhraseOK ; ThisNewPhrase` Java only `Phrase) ; Position ;`
`GuessedLetter ; MissingLetter ;`
Python only – `Choice`

Max 1

(ii) `NewPhrase ; PhraseHasBeenSet ; PhraseGuessed ; ;`
`GuessStatusArray ; LettersGuessedArray ;`
`NextGuessedLetter ; Index ; Choice` (not Python)

VB and VB6 only – `IndividualLettersArray`
Java only – `Console`

Max 1

(iii) `Len / Length/ StrLen;`
PHP – `Trim, , IntVal`
C# – `int.Parse`
Python – `Range`
`Java - ReadLine - ReadChar - CharAt`

1

(iv) `GuessStatusArray ; LettersGuessedArray ;`
VB.Net and VB6 only: `IndividualLettersArray ;`

Max 1

(v) `Position ; Index ; (`**A** `PhraseOK / Missingletter / Choice)`

Java only – Found – i

Max 1

(vi) DisplayMenu ; DisplayCurrentStatus ;

Max 1

(c) (i) DisplayCurrentStatus ; AllLettersGuessedCorrectly ; SetUpGuessStatusArray ;

Java only - GetNewPhrase ;
Java / Python only – HasLetterBeenUsed ;
C, C#, java – main

1

(ii) Check carefully with (c) (i)

AllLettersGuessedCorrectly (Not Python)	NewPhrase GuessStatusArray IndividualLettersArray(VB6 only)
SetUpGuessStatusArray	NewPhrase (+GuessStatusArray Java only) GuessStatusArray (not PHP / C#) IndividualLettersArray(VB.Net andVB6 only)
DisplayCurrentStatus	GuessStatusArray Phraselength
GetNewPhrase (Java only)	minimumLength
main (C, C#, java only)	args
hasLetterBeenUsed (Java and Python only)	LettersGuessedArray, myGuess

(Python Letter only)

1

(d) takes the original word / phrase (**A** by implication);
checks its length using <u>characters;</u>
“a length of less than 10 is not permitted” / equivalent statement with the <u>exact logic;</u>

3

(e) (i) PhraseOK = True / PhraseOK = False / PhraseOK / or explained ;

1

(ii) program will continually prompt the setter for a new phrase ;
there is a continuous loop ;

Max 1

(f) (i) a section of code needs to be repeated // **A** by implication e.g. “done for each character in the string” ;

1

(ii) the number of iterations is known // the loop is to iterate a (**R** fixed) known no. of times ;

1

(iii) The number of characters (**R** Letters) / length of the phrase ;

1

(g) Key positions are: 2; 5; 6; 10;

Index	1	2	3	4	5	6	7	8	9	10	11
		+			+	+				+	

Each correct index position ; (Max 4)
Some ‘indicator’ value e.g. True or equivalent used for all correct positions ;
A could be the actual letters stored (all in correct positions)

5

(h) No (change) // an attempt will be made to overwrite the existing ‘F' entry at position 6 in the array ;

1

(i) Key positions are: 1- 2-3-4 ;

Index	1	2	3	4	5	6	7	8	9	10	11
	‘C'	'G‘	'B‘	'H'							

First four cells used ;
and contain the correct letters ;

2

(j) No change // **A** changes;

1

(ii) No change followed by “the same letter is never stored more than once” / “the letter has already been entered” ;

A *different possible interpretation …*
Changes followed by “Second ‘B’ character is stored at position 5” ;

1

[29]

Q22.

(a) **Visual Basic**

```
Sub DisplayMenu()
Console.Writeline("______________________________")
Console.WriteLine("1. SETTER - Makes new word / phrase")
Console.WriteLine("")
Console.WriteLine("2. USER - Next letter guess")
Console.WriteLine("")
Console.WriteLine("3. USER – Make a complete word / phrase guess")
Console.WriteLine("")
Console.WriteLine("5. End")
End Sub
```

Pascal

```
Procedure DisplayMenu;
Begin
  Writeln('_____________________________');
  Writeln;
  Writeln('1. SETTER - Makes new word / phrase');
  Writeln;
  Writeln('2. USER - Next letter guess');
  Writeln('');
  Writeln('3. USER - Make a complete word / phrase
guess');
  Writeln;
  Writeln('5. End');
  Writeln;
End;
```

Java

```
    private void displayMenu() {

console.println("_____________________________________");
         console.println();
         console.println("1. SETTER - Makes new
word/phrase");
         console.println();
         console.println("2. USER - Next letter guess");
         console.println();
         console.println("3. USER - Make a complete
word/phrase guess");
         console.println();
         console.println("5. End");
         console.println();
    } // end method displayMenu
```

Python

```
def DisplayMenu():
    print “__________________________________”
    print “”
    print “1. SETTER - Makes new word/phrase”
    print “”
    print “2. USER - Next letter guess”
    print “”
    print “3. USER - Make a complete word/phrase guess”
    print “”
    print “5. End”
    print “”
```

Mark as follows:
additional choice for option 3 shown (**A** minor typos) ;
inside procedure DisplayMenu ;
VB6 – code added to listbox control lstMenu ; inside Form_Load event ;

2

(b) **Visual Basic**

```
Sub InputUsersCompletePhraseGuess()
    Console.WriteLine("Procedure
InputUsersCompletePhraseGuess has
    been called")
    Console.ReadLine()
End Sub
```

Pascal

```
Procedure InputUsersCompletePhraseGuess;
    begin
         Writeln('Procedure InputUsersCompletePhraseGuess
has been called
');
     end;
```

Java

```
   private void inputUsersCompletePhraseGuess() {
     console.println("Procedure
inputUsersCompletePhraseGuess has been called");
   } // end inputUsersCompletePhraseGuess
```

Python

```
def InputUsersCompletePhraseGuess():
    print "Procedure InputUsersCompletePhraseGuess has been
called"
    raw_input()
allow missing raw_input() – only keeps window open.
```

(NB no explicit "end" statement as in Pascal / VB) –
Award mark for correct indentation of print statement.

Mark as follows:
New procedure InputUsersCompletePhraseGuess() defined ;
Contains the required output (A. minor typos);

VB6 = `MsgBox " Appropriate text …"`

End of the procedure/function is clear ;

Python only : Award 3rd mark for correct indentation of print statement.

3

(c) **Visual Basic**

```
If Choice = 3 Then Call InputUsersCompletePhraseGuess()
```

Pascal

```
If Choice = 3
  Then
    Begin
      InputUsersCompletePhraseGuess
    End;
```

Java

```
if (choice == 3) {
      inputUsersCompletePhraseGuess();
   } // end if
```

Python

```
  elif Response == '3':
       InputUsersCompletePhraseGuess()
```

Inverted commas needed to indicate string value as returned by `raw_input()` function

Mark as follows:
Call to procedure InputUsersCompletePhraseGuess ;
IF statement for choice 3 ;

2

(d) **** SCREEN CAPTURE *****
Menu choice 3 selected ;
'Correct' output message displayed - Must match text in code for (b) ;

2

(e) **Visual Basic**

```
 Sub CountPhrasesFromFile()
      ' uses global variable NumberOfPhrasesInFile
      Dim TempPhrase As String

      FileOpen(1, "MyPhrases.txt", OpenMode.Input)
      NumberOfPhrasesInFile = 0
      Do
           TempPhrase = LineInput(1)
           NumberOfPhrasesInFile = NumberOfPhrasesInFile
+ 1
         Loop Until EOF(1)
         FileClose(1)
     End Sub
```

OR equivalent using the `FileStream` object and `StreamReader` method.

Pascal
answer with WHILE loop:

```
Procedure CountPhrasesFromFile;
{ uses global variable NumberOfPhrasesInFile }

Var
  TempPhrase:string;

Begin
  Reset(MyPhrasesPipe);
  NumberOfPhrasesInFile:=0;
  While Not Eof(MyPhrasesPipe)
   Do
    Begin
      ReadLn(MyPhrasesPipe, TempPhrase);
      NumberOfPhrasesInFile:=NumberOfPhrasesInFile+1;
      End;
  Close(MyPhrasesPipe);
End;
```

Alternative implementations:

```
Procedure CountPhrasesInFile(Var NumberOfPhrasesInFile :
Integer);
Function CountPhrasesInFile(Var NumberOfPhrasesInFile :
Integer) :
Integer;
```

Java

```
private void countPhrasesFromFile() {
    String fileNameIn = "MyPhrases.txt";
    String newLine;
    numberOfPhrasesInFile = 0;
    try {
      BufferedReader phrasesFile = new
BufferedReader(new FileReader(fileNameIn));

      while ((newLine = phrasesFile.readLine()) != null) {
           numberOfPhrasesInFile = numberOfPhrasesInFile +
1;
      } // end while
      phrasesFile.close();
    } catch (IOException e) {
      System.out.println(e.toString());
      System.exit(0);
    } // end try/catch
    console.println("Number of phrases: " +
numberOfPhrasesInFile);

  } // end countPhrasesFromFile
```

Python

```
def CountPhrasesFromFile1():
    global NumberOfPhrasesInFile
    f = open('MyPhrases.txt','r')
    AllPhrases = f.readlines()
         NumberOfPhrasesInFile = len(AllPhrases)
    f.close()
```

or

```
def CountPhrasesFromFile2():
    global NumberOfPhrasesInFile
    f = open('MyPhrases.txt','r')
    NumberOfPhrasesInFile = 0
    for phrase in f.readlines():
         NumberOfPhrasesInFile = NumberOfPhrasesInFile + 1
    f.close()
```

`Accept NumberOfPhrasesInFile += 1`

Mark as follows:

open file correctly formed ;
correctly formed loop (post or pre condition);
terminates with ‘EOF’ ;
each phrase read from file ;
temporary variable used to store the next line of text ;
file closed ;
“`NumberOfPhrasesInFile`” initialized ;
“`NumberOfPhrasesInFile`” incremented ;
return of the phrase count / assigned to global variable ;

Alternative solutions which include all or some of the following:

– **declaring a dynamic array**; **A** by implication if supported in language
opening file / specifying the file;

read entire text file into string;
split string into array;
closing file;
read size of array;
return of the phrase count / assigned to global variable;
N.B. More than one mark may be awarded if command combines multiple functions e.g. `ReadAllLines` which opens (1) and closes (1) file, reads entire text file (1) and splits into an array (1) is worth 4 marks

– Solutions which (do not require the loop structure and) **compute thenumber of phrases from object methods.**
The table below is an indicative (but not exhaustive) list so you need to checkany other feasible answers you see, particularly if the screen shot appears to work.

Max 7

Table 1 shows some of the methods for the supported languages which will be used for an alternativesolution.

Table 1

List of commands / methods

Language	**Function to read entire text file into a string or array***	**Function to split string into an array**	**Function to return array length**
Visual Basic 6	ReadAll [1 – read all phrases into string]	`Split [1]`	`UBound [1]`
.NET languages: VB C# Delphi Java	`ReadToEnd` [1 - read all phrases into string] `ReadAllText` [3 - 1 open, 1 close, 1 read all phrases into string] `ReadAllLines` [4 - 1 open, 1 close, 1 read all phrases, 1 split into array]	`Split [1]` except if this markalready given for `ReadAllLines`	`UBound` [1] `GetUpperBound` [1]
PHP	`File` [4 - 1 open, 1 close, 1 read all phrases, 1 split into array] `File Get Contents` [3 - 1 open, 1 close, 1 read all phrases into string]	`Explode`, `Split` (with some close variations e.g. `Split` `Split`[1] except if this mark already given for File)	`Count`[1]
Java	`Scanner` with delimiter '\\z □ [1 – read all phrases into string]	`Split` [1]	`Length` [1]
Python	`Read` [1 – read all phrases into string] `ReadLines` [2 – read all	`Split` [1]	`Shape/Len` [1]

	phrases and split into list]		
** Note that some of the commands in the second column are worth more than one mark as theyperform multiple tasks e.g.* `File_Get_Contents` *in PHP opens and closes the file and reads all the phrases into a string so is worth 3 marks, as shown in [ ]. To answer (e) the candidate would then need to use* `Split / Explode` *to break this string up into an array then* `Count`*to see how many elements there are in the array – i.e. how many phrases were loaded.*			

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

This is conditional on some code for (a) (i)

reports the number of phrases in the file - 24 (**A** 25) ;

1

(f) (i) **Visual Basic**

```
Sub GenerateRandomPhraseNumber()
    ' uses global variables NumberOfPhrasesInFile
               and PhraseNumber
      Randomize()
      ThisPhraseNumber = Int(Rnd() *
NumberOfPhrasesInFile) + 1
    End Sub
```

Pascal

```
Procedure GenerateRandomPhraseNumber;
{ uses global variables NumberOfPhrasesInFile and
PhraseNumber }
Begin
   Randomize;
   PhraseNumber:=Trunc(Random(NumberOfPhrasesInFile))
+1;
End;
```

Alternative Implementations

NB Several alternative implementations possible for both Pascal and Visual Basic

e.g. Pascal

```
Procedure GenerateRandomPhraseNumber
                        (Var
NumberOfPhrasesInFile:Integer);
Function GenerateRandomPhraseNumber : Integer;
Function GenerateRandomPhraseNumber
                   (Var
NumberOfPhrasesInFile:Integer):
Integer;
```

Java

```
   private void generateRandomPhraseNumber() {
      // .nextInt(n) produces nos [0..n[
      phraseNumber =
generator.nextInt(numberOfPhrasesInFile) + 1;
   } // end generateRandomPhraseNumber
```

Alternative implementation:

```
   private int generateRandomPhraseNumber() {
      return
generator.nextInt(numberOfPhrasesInFile) + 1;
   }
```

Python

Needs “`import random`” declared at start of program

```
def GenerateRandomPhraseNumber():
    global PhraseNumber, NumberOfPhrasesInFile
    PhraseNumber =
random.randrange(NumberOfPhrasesInFile)
```

Mark as follows:

Correct use of the RANDOM / RND function / class with “NoOfPhrasesInFile”) ;
correct range generated (from 1 to “NoOfPhrasesInFile”);
final answer is integer // implied by variable declaration / return value from a function ;

Note: Commentary in the 'C specific' MS

3

(ii) ***** SCREEN CAPTURE 1 *****
displays phrase number ;

***** SCREEN CAPTURE 2 *****
displays phrase number ;

2

(g) (i) **Visual Basic**

```
Sub SelectPhraseFromFile()
' uses global variable PhraseNumber
    Dim Counter As Integer
    Dim Found As Boolean
    Dim ThisPhraseFromFile As String
    Counter = 1
    Found = False
    FileOpen(1, "MyPhrases.txt", OpenMode.Input)
    Do
      ThisPhraseFromFile = LineInput(1)
      If Counter = PhraseNumber Then
         Found = True
      Else
         Counter = Counter + 1
      End If
    Loop Until Found = True Or EOF(1)

    FileClose(1)
End Sub
```

OR equivalent using the `FileStream` object and `StreamReader` method.

Pascal

```
Procedure SelectPhraseFromFile;
```

```
{ uses global variable PhraseNumber }
Var
  Counter:Integer;
  MyPhrasesPipe : TextFile;
  ThisPhraseFromFile : String;

Begin
  Assign(MyPhrasesPipe, 'MyPhrases.txt');
  Reset(MyPhrasesPipe);
  Counter:=0;
  While (Not Eof(MyPhrasesPipe)) And
(Counter<>PhraseNumber)
     Do
       Begin
          Readln(MyPhrasesPipe, ThisPhraseFromFile);
          Counter:=Counter+1;
       End;

     Close(MyPhrasePipe);
End;
```

Mark as follows:
File opened ;
Loop (post or pre-condition) / FOR-ENDFOR ;
Counter initialized ;
Read next phrase from file ;
Stored in a temporary variable ;
File closed ;
Return of the phrase / assigned to global variable ;

For loop only …
For 1 TO X ;

Conditional loop only …
Counter incremented ;
Boolean variable for trigger / Counter compared with PhraseNumber for trigger ;
Boolean variable set to True when located // terminated correctly ;

Alternative solution if entire text file read at once:
- declaring a dynamic array; A by implication if supported in language
opening file / specifying the file;
read entire text file into string;
split string into array;
closing file;
access correct cell in array;
return of the phrase / assigned to global variable;
N.B. More than one mark may be awarded if command combines multiplefunctions e.g. `ReadAllLines` which opens (1) and closes (1) file,
reads entire text file (1) and splits into an array (1) is worth 4 marks

- solutions which use object methods
As for Question (e)(ii), look for solutions which compute the phrase in this way. Refer to **Table 1** shown with (e)(i).

Java

```
    private void selectPhraseFromFile() {
```

```
    String fileNameIn = "MyPhrases.txt";

    int counter = 1;
    try {
        BufferedReader phrasesFile = new
BufferedReader(new FileReader(fileNameIn));
        while ((counter !=
phraseNumber)&((thisPhraseFromFile =
phrasesFile.readLine()) != null) ) {
             counter = counter + 1;
        } // end while
        console.println("Phrase/phrase selected
is: " + thisPhraseFromFile);
        phrasesFile.close();
    } catch (IOException e) {
        System.out.println(e.toString());
        System.exit(0);
    } // end try/catch
  } // end selectPhraseFromFile
```

Mark as follows:
File opened;
Loop (FOR, post or pre-condition) used to search for the phrase;
Counter initialised;
Counter used to control position in the file;
Counter incremented;

Test for ‘EOF’;
Boolean variable for trigger / counterphraseNumber for trigger;
Boolean variable set to true when located;
File closed;

Python

```
def SelectPhraseFromFile():
    global PhraseNumber, ThisPhraseFromFile
    f = open('MyPhrases.txt','r')
    Phrases = f.readlines()
    ThisPhraseFromFile = Phrases[PhraseNumber]
    print "The Phrase selected is ... %s" %
ThisPhraseFromFile
```

or

```
  print "The Phrase selected is ... ",
ThisPhraseFromFile
    f.close()
```

Max 7

(ii) **** SCREEN CAPTURE 1 ****

**** SCREEN CAPTURE 2 ****

Evidence for two different words selected ;

1(0)	MANCHESTER UNITED
2(1)	YELLOW SUBMARINE

3(2)	HIP HOP MUSIC
4(3)	DETERMINATION
5(4)	PABLO PICASSO
6(5)	THE GRAND CANYON
7(6)	BRICK LANE
8(7)	WIGAN ATHLETIC
9(8)	WORLD MUSIC
10(9)	THE COLISEUM
11(10)	WAR AND PEACE
12(11)	VIVIENNE WESTWOOD
13(12)	EAST ENDERS
14(13)	GRIZZLY BEAR
15(14)	NEW ZEALAND
16(15)	KATE WINSLET
17(16)	THE SUNDAY TIMES
18(17)	THE GUARDIAN
19 (18)	HOCKEY STICKS
20(19)	CORONATION STREET
21(20)	GLASTONBURY FESTIVAL
22(21)	SERENDIPITOUS
23(22)	FORTUITOUS
24(22)	FASHION STATEMENT

2

(h) **Visual Basic**

```
Dim NumberOfPhrasesInFile As Integer
Dim PhraseNumber As Integer
Dim ThisPhraseFromFile As String
```

Pascal

```
 Var
    NumberOfPhrasesInFile : Integer;
    PhraseNumber : Integer;
    ThisPhraseFromFile : String;
```

Java

```
int numberOfPhrasesInFile;
int phraseNumber;
String thisPhraseFromFile;
```

Python

Declare `NumberOfPhrasesInFile / PhraseNumber` and initialiseat start of program to assign data type.

```
NumberOfPhrasesInFile = 0
PhraseNumber = 0
ThisPhraseFromFile = ‘’
```

Mark as follows:

declare `NumberOfPhrasesInFile / PhraseNumber / ThisPhraseFromFile` or any plausible variable (Max 1) ;
correct matching plausible data type (Max 1) ;
Python only: Data type is implied by assignment
e.g. PhraseNumber = 0

A if complete code listing given and additional variable is identified

Max 2

[33]

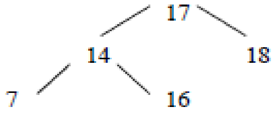
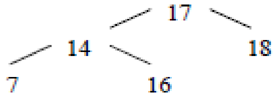
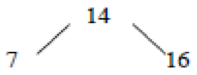
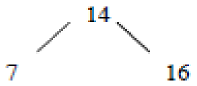
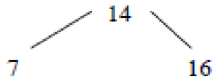
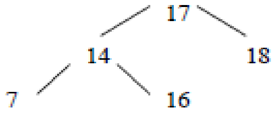
Q23.

(a) A procedure/routine that calls itself/ is defined in terms of itself;

A Function instead of procedure
R re-entrant
R program iteration (TO)

1

(b) (i)

Procedure Call	T	
P_1		
P_2	18	;
P_1		
P_3		;
P_4	16	;
P_3		
P_5	7	;
P_3		
P_1		

Output	18;	17;	16	14	7;

7

(ii) Reversed Inorder; Tree <u>traversal</u>;
I Sort/ Re-arrange

2

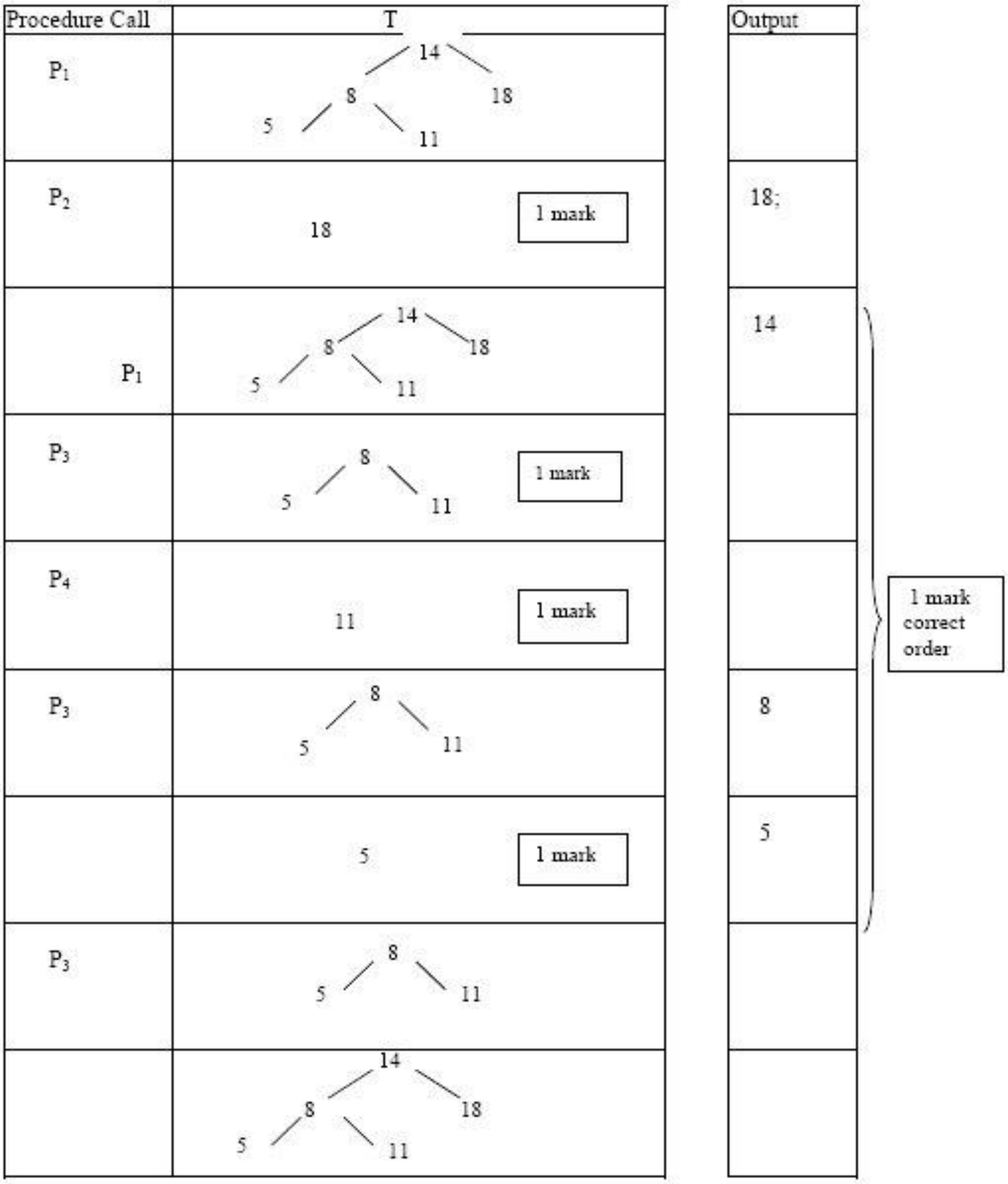
[10]

Q24.

(a) A procedure/routine that calls itself/ is defined in terms of itself;
A Function instead of procedure
R re-entrant **R** program **R** iteration

1

(b) (i)



Procedure Call	T	Output

P1	14
	8 18
	5 11	

P2	18 1 mark	18;

P1	14
	8 18
	5 11
		14

P3	8 5 11 1 mark	
P4	11 1 mark	
P3	8 5 11	8
	5 1 mark	5
P3	8 5 11	

	14
	8 18
	5 11	

1 mark correct order

6

(ii) Reverse Inorder// Reverse order; (tree) traversal;

2

[9]

Q25.

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

Max 2

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

Max 1

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

Max 1

(c) (i) 10 ;

1

(ii) -1 ;

1

[6]

Q26.

(a) (i) String / Text / Char ;
R alpha / alpha-numeric / character

1

(ii) Integer / Date (and Time) ;
A String

1

(iii) Boolean ;
R Yes/No

1

(b) (i) Book ;

1

(ii) False / F / No // f/t from the (a) (iii) answer e.g. stated as integer - value 0/1

1

(iii) True / T / Yes // f/t from the (a) (iii) answer e.g. stated as integer - value 1/0
(Max 1 for (ii) and (iii) if no indication of meaning when integer used)

1

(c) (i) T76542 ; 1 ;

2

(ii) T ;
I. the quote marks (i) and (ii)

1

(iii)

NextAvailableCode	Book	LocationLetter
1	1	'T'
2	2	'T'
3	3	(gap not required)
4	4	'M'

(in sequence – possible repeat of 3 and/or 4	5	Penalty -1 if the first 'M' is followed by either 'T' or 'X'
	6	

Figure 2

	Location
[1]	'Torrington'
[2]	'Torrington'
[3]	
[4]	'Morristown'
[5]	

Figure 3

	NewCode
[1]	1
[2]	2
[3]	
[4]	3
[5]	

Figure 4

6

[15]

Q27.

(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]

Q28.

(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]

Q29.

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

3

(b) (i)

Surname	String / Text ; **A.** String[n]

NoOfYearsService	Integer /Byte / Int / Short;
PayRate	Single / Real / Float / Currency;
BasicRate	Single/Real/Float / Currency;
AdditionalRate	Single / Real / Float / Currency;

Sensible name + correct data type for single mark

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

Max 3

(ii) 3.1 If NoOfYearsService > 5 ;

1

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

1

A = or := or ←

3.2 PayRate := 7.88 + NoOfYearsService * 0.65

1

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

A in words ‘greater than’, ‘equals’

3

[9]

Q30.

(a) Calculates the total rejects for the week / calculates the total of array DailyRejects ;
Outputs the total rejects for the week ;
A Output the total only (if already mentions that calculates total rejects for the week)

2

(b) (i) RejectTotal := RejectTotal + DailyRejects[DayNo] ;
A ; may be omitted
A minor spelling errors
A omission of the subscript

1

(ii) RejectTotal: Integer //
DayNo : Integer //
DailyRejects : Array[1 ..7] of integer;
I. Dim …

Max 1

(iii) Loop counter / control the loop / Loop control variable / inference of a loop counter ;
Index/subscript for the array DailyRejects / reference the array elements ;
R days of the week

Max 1

(iv) Array of integers // array

1

(c) If RejectTotal > 7 ;
Then WriteLn (‘Investigate’)
Else WriteLn (‘Inside weekly tolerance’) ;
A reversed logic for both parts

2

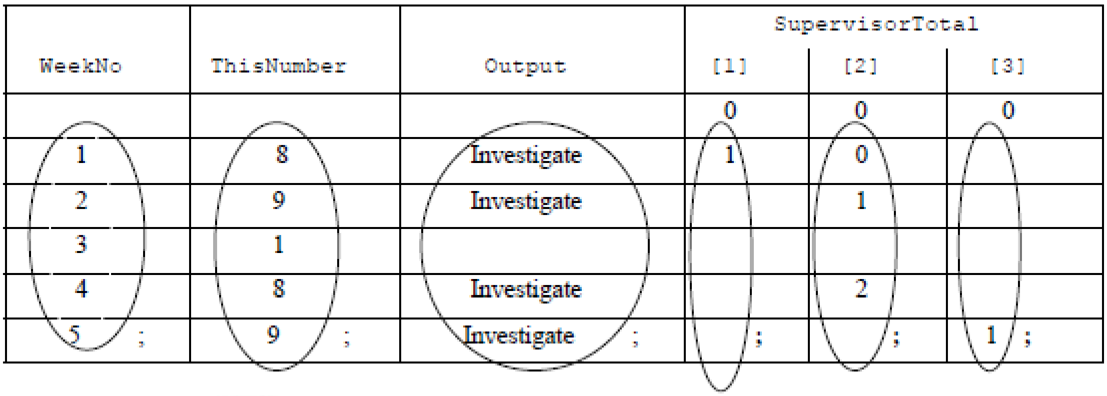
(d) *Library program …*
Tried and tested routines should reduce the debugging time;
Evelopment time may be reduced ; **A** less code to write
Code can be dynamically loaded only when needed ;
Library files can be shared between different applications ;
A previously written/saved program code can be reused/
A program routines were previously saved/compiled ;
A program code is available and used from third party providers ;

Max 2

(e) (i) 3 / [3] / SupervisorTotal[3] := etc …..;

1

(ii)



WeekNo	ThisNumber	Output	SupervisorTotal [1]	 [2]	 [3]
			0	0	0
1	8	Investigate	1	0	
2	9	Investigate		1	
3	1				
4	8	Investigate		2	
5 ;	9 ;	Investigate ;	;	;	1 ;

6

[17]

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) Functions always <u>return</u> some value when called;
Procedures <u>may</u> return a value;
Functions appear in expressions;
Procedures do not appear in expressions;
Procedures name alone makes up the statement / call <name>

Max 2

(ii) Anything named which is plausible;
Examples could include: computation / formatting / string handling;
R software features / button events / DLL
A Dynamic Linked Library

2

(b) (i) True/Yes/ 1;

1

(ii) False/No/0;

1

(iii) Error;

1

(c) Program / constant / function / procedure / module / unit / user defined type / record / label / object /class;

Max 2

(d) **Advantage of an Interpreter:**

- Should allow faster/easier program development // faster/easier testing / debugging / finding errors;
- Correcting mistakes is less time consuming;

Max 1

Advantage of a compiler:

- The executable code/object code/program will run faster;
- Once the executable file has been produced no further action;
- Software distribution requires no further software to be available to the user;
- Prevents tampering of the code by users other than the developer;

Max 1

2

[11]

Q33.

(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]

Q34.

(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

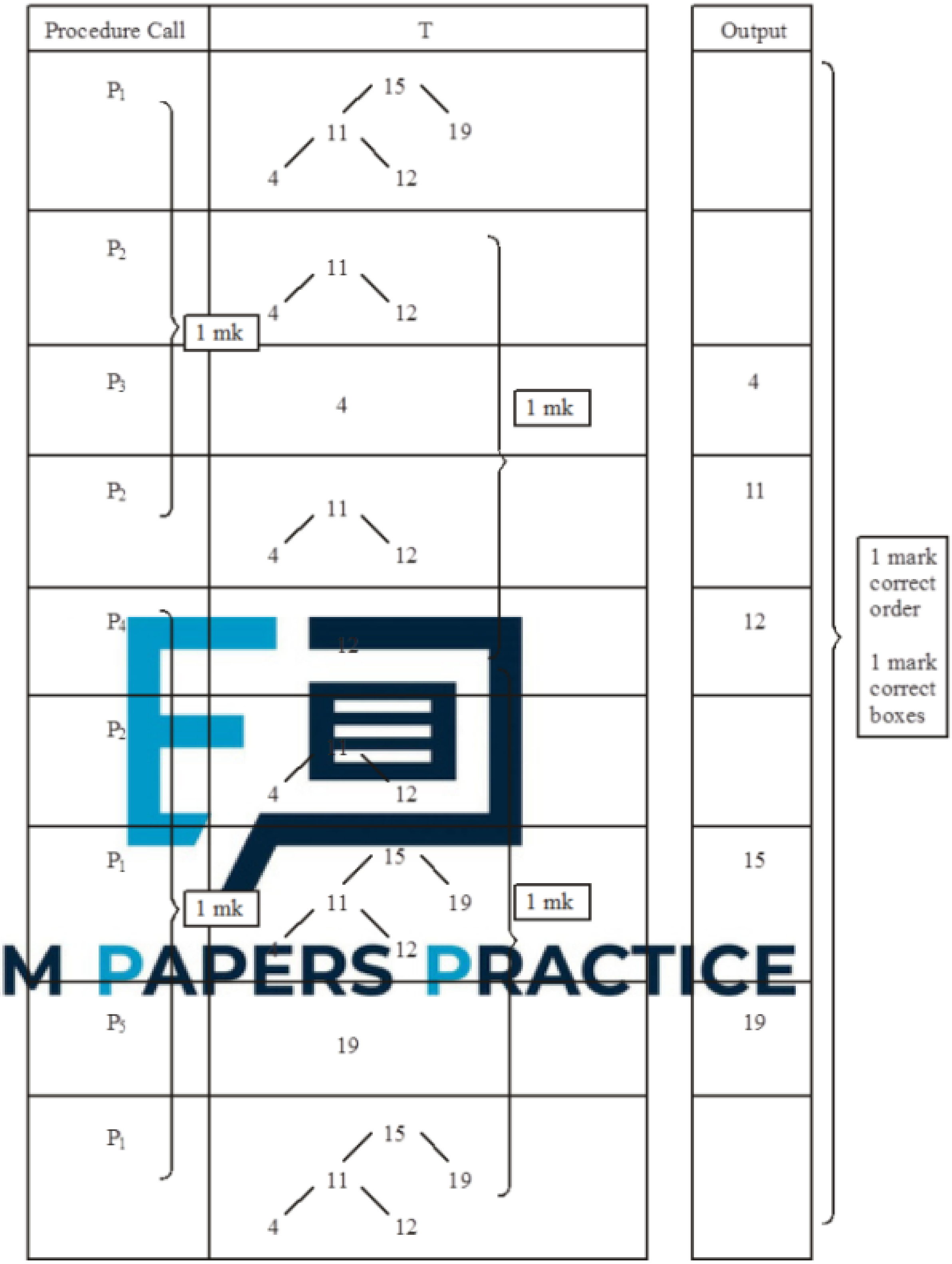
[4]

Q35.

(a) A procedure/routine that calls itself/ is defined in terms of itself;
A Function instead of procedure
R re-entrant
R program
R iteration

1

(b) (i)



Procedure Call	T	Output

P_1	15
	11 19
	4 12	

P_2	11 4 12	
P_3	4	4
P_2	11 4 12	11
P_4	12	12
P_2	11 4 12	

P_1	15
	11 19
	4 12
		15

P_5	19	19

P_1	15
	11 19
	4 12	

1 mk
1 mk
1 mk
1 mk
1 mark correct order
1 mark correct boxes

6

(ii) In order; (tree) traversal

2

[9]

Q1.

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)). 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).

Most students were able to identify that NoOfCardsTurnedOver was a stepper role variable but fewer were able to correctly identify the roles of Choice and SwapSpace. Many answers made it clear that the problem with the algorithm had been identified for part (g) but fewer were able to describe the changes that needed to be made to correct the problem. For part (i), search was the most frequently seen answer which was not worth a mark.

Q2.

(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.

Q3.

Most students did well on this question, with well over half getting 20 or 21 marks out of 21.

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, for example, trying to use a keyword of OUTPUT or students using VB.Net adding the word DO to their WHILE loops. 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. Minor differences between the messages / prompts in the given algorithm from those used in the student's program were not penalised but a number of students dropped marks by using substantially different messages / prompts in their program.

Q4.

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)). 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.

Part (d) was well-answered with most students giving a correct example. Parts (e) and (f) asked for students to explain parts of the Skeleton program code with very few getting good marks on these questions. Answers were often given that were too vague or about completely different parts of the Skeleton program. Some students described what Mod 26 does instead of explaining why it was needed. Students often seemed to be unfamiliar with structure charts getting few, if any, marks for parts (g) - (j).

Q5.

Candidates demonstrated a pleasing understanding of the use of syntax diagrams and Backus Naur Form to specify language syntax.

For (a), the overwhelming majority of candidates scored at least three of the four available marks. Candidates had most trouble identifying that the third example `procedure square (s:real)` was not valid, perhaps because they just assumed that real was a valid type rather than checking it against the diagrams.

For (b)(i), the majority of candidates recognised that the BNF definitions incorrectly included a new “char“ data type and almost half also identified that the BNF definitions did not allow for a procedure to have no parameters.

Part (b)(ii) was well answered with most candidates achieving a mark for recognising that there could be any number of parameters. Pleasingly, some also went on to explain that recursion had to be used because BNF does not support iteration. The most commonly seen incorrect response was to simply define what recursion was instead of addressing the specific question.

Q6.

The majority of students got full marks for this question.

Q7.

This question was generally well-answered. For part (a), some students did not use the number of bits specified in the question and some used even parity instead of odd parity. Part (b) was the first COMP1 question about Hamming code. Many students were able to give an advantage of Hamming code although occasionally answers were too vague, eg, "It can detect errors" and there were some students who clearly had no understanding of the topic and were just guessing eg, "It uses less memory."

Q8.

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.

Q9.

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.

Q10.

(a) This was a fairly straightforward programming question with most students getting close to full marks. Some students did not check their code carefully and subtracted one from `NoOfCellsSouth` or `NoOfCellsEast` (instead of adding one).

Care needs to be taken with screen captures of testing as for part (d) a number of students showed the after state of the cavern and the selection of option (iv), but did not show the original state of the cavern and thus the screen capture(s) provided did not include sufficient evidence for the mark to be awarded.

A common mistake made by weaker students in all Pascal, VB and Java was to try to combine into one instruction (using a AND Boolean operator) an instruction to increment the NoOfCellsSouth and an instruction to increment the NoOfCellsEast – suggesting that they did not know how to write a case statement that contains more than one instruction.

(b) A number of students had clearly anticipated that this question would be asked and prepared thoroughly for it. Weaker students struggled to write the correct conditions for the selection structures and often wrote code that would either prevent all moves in the northernmost row of the cavern or all moves northwards.
A number of answers included code to prevent the player moving out of bounds in each of the four possible directions (and some also prevented illegal moves in a southeast direction as well). This was not necessary as it was not what the question asked. Some weaker students ended up with more errors in their answers by trying to add (incorrect) code to prevent the other possible illegal moves.

(c) Most students obtained marks on this question. A number of students did not follow the question specification and changed the messages to be displayed to the user or added one to the NoOfMoves variable in the wrong place (often this was done inside the repetition structure used to ensure that a valid move had been entered – this would mean that the NoOfMoves variable would be incremented even when a valid move had not been entered). Students should be aware that if a question specifies a particular message to display then this is the message that their program must display – minor typos were ignored, but when a message was different by a whole word or more the mark was not awarded.

(d) This was the most challenging of the programming questions 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. One unusual correct answer seen from a few students was to pass a copy of the Cavern array to the CalculateDistance subroutine and use a loop inside the routine to count how many calls were made to the MakeMonsterMove subroutine until the monster and player were in the same cell.

The most commonly used method to calculate the distance was to subtract the monster's east value from the player's east value followed by a selection structure to deal with the scenario of a negative difference, then to do the same for the difference between the two south values and finally to add the two differences together. A number of students lost marks by dealing with negative values after adding the east difference and south difference together – this would only calculate the correct distance between the monster and player under some circumstances.

It was disappointing that a significant 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). Students should be encouraged to include partial solutions to questions they have not been able to answer wholly successfully.

Less able students often struggled to create a new subroutine even though there are numerous examples of subroutines in the Skeleton Program. A number of students, particularly those using VB, developed a solution that would correctly calculate the distance between the monster and the player but did not set up a mechanism to return the distance to the calling routine. This was often because they had used a procedure, rather than a function (although a few students did use passing by reference correctly as a return mechanism).

Q11.

Part (a): Two thirds of students were able to identify one property that a graph must have to be a tree. A small number confused a tree with a rooted tree and made assertions such as that a tree must have a root, which is incorrect.

Part (b): This question part tested students' understanding of the method being used to represent a maze as a graph. The majority of students correctly identified a feature of the maze that would stop its graph being a tree. The most commonly seen correct response identified that there could be a loop in the maze. Other possibilities included that part of the maze could be inaccessible or that part of the maze might only be traversable in one direction. Some students failed to achieve the mark because they re-answered part (a), discussing a feature of a graph that would stop it being a tree, rather than a feature of a maze.

Part (c): Students were asked to represent the graph of the maze as an adjacency matrix. Three quarters of students scored both marks for this question part. Responses where symbols other than 0s and 1s were used in the matrix were accepted, as long as they could be viewed as an accurate representation of the graph.

Part (d)(i): The vast majority of students were able to identify that a recursive routine would call itself. A small number asserted that a recursive routine would repeat itself, which was not considered to be enough for a mark as this could equally have been a description of iteration.

Part (d)(ii): Most students scored some marks for this question part, but less than a fifth achieved both. The most widely understood point was that the data would need to be removed from the stack in the reverse of the order that it was put onto it so that the recursion could be unwound. Less well understood was the types of data that would be stored, such as return addresses and local variables.

Part (e): Most students achieved some marks on this question part and around a quarter achieved all five for a fully complete trace. The most commonly made mistake was to update, incorrectly, the Completely Explored array as the recursive calls were made, as opposed to when the recursion unwound.

Q12.

This task was a more challenging question than those on the 2009 and 2010 COMP1 question papers. However, it was based on a standard algorithm (linear search) that is on the specification. Despite the Preliminary Material clearly stating that candidates should be familiar with declaring and using arrays (and there being examples of arrays in the Skeleton Program), a significant number of candidates were unable to write a syntactically correct array declaration in their programming language. A number of candidates provided screen captures that had not been produced by the programming code they had given in their answer for part (b); this meant that they did not get any marks for their screen captures. Candidates should understand that they could get marks for test runs which show only part of their program working correctly, but they will not get any marks for “correct” test evidence that was not produced by their programming code.

Most candidates were still able to score good marks on this question despite the increased difficulty of this task.

Q13.

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.

Q14.

(a) Most candidates attempted this question and were able to get the majority of the marks. Despite the question asking that the new option of ‘RUN OUT’ be available for both real and virtual dice versions of the game, a number of candidates did not alter the Skeleton Program to generate a random number between 1 and 5.

(b) For question (b) candidates were asked to adapt the DisplayResult subroutine so that an appropriate message would be displayed if the result of a game was a draw. Many candidates got good marks on this question. The most common mistake was to add an else clause to one of the existing IF statements rather than adding an additional IF statement – this would result in the message about a drawn game being displayed if one of the player’s had won the game as well as when a game was drawn. Some candidates adapted the Skeleton Program correctly, but then did not provide evidence for the test asked for in the question – a test showing both players getting a score of 0 was needed. Some candidates provided test evidence when the players have obtained a score of 1 or more.

(c) While there were a lot of good answers to this question, candidates generally found question (c) more difficult than questions (a) and (b). Candidates often used the incorrect logic. Common mistakes included using the wrong logical connective for the two conditions (i.e. AND instead of OR / OR instead of AND) and using the wrong logical operator with a numeric value e.g. ">=6" instead of ">6" or ">=7". It was clear that a significant proportion of candidates following the AS Computing course struggle to understand the logic of selection/repetition structures which have multiple conditions. A number of candidates did not read the question sufficiently carefully and did not include a repetition structure inside the RollBowlDie routine – only using a selection statement.

(d) Many candidates had clearly anticipated that they would be asked to write a routine to save the top scores to a file and did very well on this question with able candidates often obtaining full marks. Some candidates seemed to have tried to memorise the code for this task and then were unable to reproduce it under exam conditions (or simply copied and pasted the SaveTopScores subroutine and then tried to modify it) as they did not sufficiently understand the task they had been practising. For part (iv), a number of candidates did not modify the main program block to allow the 5th option to be selected.

(e) A wide range of responses were seen to this question. A large number of candidates were unable to express their ideas clearly and their description of how their suggested changes could be made was too vague to get full marks. Some answers would have achieved the desired result of getting the low scores more than the high scores, but also resulted in adverse, undesired changes to the Skeleton Program (e.g. a player could no longer get 2 runs and could never get a result of "out").

Q15.

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.

Q16.

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

Q17.

(a) In general, candidates were better prepared for Section C this year and candidates demonstrated a good understanding of the Skeleton Program.

When asked for the name of an identifier a one word answer is expected. A significant number of candidates included an entire line of code that included the name of a relevant identifier in it. Answers for parts (i), (ii), (vi), (vii), (viii) that gave a correct answer as part of a declaration were accepted this year; answers that included the identifier as part of some other statement (e.g. within an assignment statement) were rejected. In future examinations, any answer that includes anything other than the name of the identifier will **not** be deemed creditworthy.

Part (iii) was generally well-answered though some candidates gave an answer that global variables are declared at the start of a program. This is often true, but it is possible to declare global variables in other places in a program and this was not sufficient (on its own) for a mark.

Most candidates were able to answer part (iv). The most common error was stating that the instructions would stop being repeated when an ‘X’ or ‘Y’ is entered (instead of ‘X’ or ‘O’). Some candidates just copied and pasted code from the Skeleton Program rather than describe the stopping condition.
Most candidates seemed to be aware of the role of variables. More were able to identify stepper role variables than fixed-value role variables. The most common incorrect answers for the fixed-role variables were PlayerOneSymbol (this is given a value inside a loop and so its value can change several times) and StartSymbol (which changes value after each game).

Part (ix) was answered well, but some candidates gave a declaration rather than an assignment statement and others copied in several lines of code rather than just the assignment statement. A few candidates copied in the code for the entire subroutine which showed that they did not understand what an assignment statement was.
Good answers for part (x) referred to how the value ‘X’/‘O’ would be assigned to the variable WhoStarts. Most answers obtained some marks, but often referred to how a value of ‘X’/‘O’ would be returned – this was not a description of the selection statement, but the subroutine as a whole.

(b) The definition of boundary data was often unclear and most answers for part (i) did not get the mark available. Very few candidates stated that boundary data is that which is at the limit of what is allowed, just before the limit and just after the limit. Some candidates gave answers in which they wrote about boundary data being data which is only just allowed and then gave a (correct) example of boundary data as being 4 (which would not be allowed).
Some candidates did not get the mark for the screen capture of the test as their test did not show both the data entered and the behaviour that resulted from their test.

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.

The format of this paper – where candidates were required at an early stage to program a task from scratch for a relatively straight forward specification – seemed to work well and a large number of candidates scored the maximum seven marks for the program source code. The question assessed the candidate’s ability to implement the given problem description using the basic constructs of a high level language. However, candidates need to be made aware that the algorithm given had to be seen as a formal specification where the wording in any output or user prompts in their program code had to match exactly that given in the algorithm. The mark scheme reflected this and, as a result, candidates frequently lost marks for their screen shots because of their lack of attention to detail.

Q21.

Questions (a) to (c) required candidates to identify certain features of the Skeleton Program and this was generally well answered. Many candidates did not associate the term ‘*pre-defined function*’ to mean a built-in function and hence did not score the mark for question (b)(ii).

For question (e)(i) candidates were able to describe the condition which controlled the loop ‘PhraseOK=True’ and to describe for question (e)(ii) that the consequence would be a continuous loop. However, the explanation of why the programmer had used a ‘For’ loop was often poor with candidates unable to give a convincing explanation for this choice (and not a ‘repeat-until’ structure). Also candidates were unable to use precise language to describe a ‘known’ number of iterations.

This question was well answered with many candidates scoring the maximum 10 marks. Better answers for question (g) scored the final mark by describing a Boolean flag or an integer value of 1 indicating that a particular letter had been guessed. If the candidate described the letter itself stored as the indicator, then this was deemed creditworthy.

There was possible ambiguity between the wording of the stem for question (j)(i) and the statement in the Preliminary Material that 'An entered letter is never stored more than once.' As a result an answer of either yes or no for question (j)(i) scored the 1 mark and this followed through into the marking of question (j)(ii).

Q22.

(a) By this stage of the examination, weaker candidates were either starting to find the paper challenging or were struggling to complete the paper in the two hours. Attempts at this question ranged from not attempted (which were relatively few) to a completely correct solution. The question – similar to question (c)(ii) – required that the candidate followed precisely the specification given to gain full marks. It was suspected that many candidates’ practice for the examination had included the coding of a guess of the complete phrase and so included this code even though it had not been asked for in the question. Candidates should be reminded of the need to answer the question set; not one that they wish had been set! Candidates seemed to understand fully what was meant by a ‘procedure / function stub’ and followed the instructions to produce all the evidence required.

(b) The majority of candidates had clearly read the suggestions in the Preliminary Material and were well prepared for this task. As a general principle, no credit was given for any screen shot evidence – e.g. question (e)(ii) – which was not supported by relevant and plausible code. The able candidates had no difficulty answering this question and often gained very close to the maximum mark. Common shortcomings were solutions which read the phrases into an array which had been set to a particular size (24 or 25) and so assumed prior knowledge of the number of phrases in the file.

For question (f)(i) a common shortcoming was code which generated a random number between 1 and 24, not 1 and ‘the computed number of phrases in the file’.

Many candidates for question (h) included a complete listing of their final program code (possibly because this was a requirement on the COMP1 Specimen Paper). This was not in the rubric of the operational examination question.

Q23.

It was pleasing to see the number of candidates that scored highly on this question. Most candidates were able to obtain the mark for part (a) and a large number did very well on part (b). It must be emphasised that candidates were asked to dry run the algorithm and complete the trace table. A small number of candidates were able to produce the correct output but did not produce a satisfactory trace. Marks were given for the trace and so it is essential that candidates fill this in correctly. Although most candidates obtained one mark for part (b)(ii), few obtained two. Candidates must realise that correct technical terminology should be used.

Q24.

Most candidates obtained the mark for part (a). It was also very pleasing to see the number of candidates who were able to correctly trace the algorithm. Many candidates obtained good marks on this question. Although many candidates did go wrong with the trace, very few candidates failed to attempt it.

Q25.

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

(ii) Many candidates then misunderstood what was wanted here and proceeded to give answers which generally described how programs were constructed with loops, selection statements, etc.

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

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

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

Q26.

In general the dry run was poorly answered and left completely blank on too many scripts.

(a) Many candidates scored the maximum three marks for identifying the data types. Some candidates lost a mark for suggesting that 'yes/no' or a 'check box' was an acceptable data type. This comes from their practical experience with database design software and a visual programming language, but candidates should appreciate they are not acceptable names for programming language data types.

(b) This was a different style of question from that previously seen. Candidates seemed to cope well with being asked to 'fill in the blanks' in the algorithm.

(c) (ii) Answers were often incorrect, but then inexplicably candidates were able to use the same function correctly in part (iii).

Q27.

(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.

Q28.

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.

Q29.

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

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

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

Q30.

A general observation was that candidates scored significantly better with tracing the algorithm than with the first part of the question where they were asked to recognise various components of the given program.

(a) Almost all candidates got the idea that the program was calculating a weekly total.

Very few stated for the second mark that it output the result.

(b) (i) A common error was to copy the first assignment statement which appeared, ignoring the rubric that it should ‘perform a calculation’.

(ii) A common error was the statements that RejectTotal:=0 was a declaration statement.

(iii) Very few answers scored here. The most common (wrong) answer was that it represented the day of the week.

(c) This should have been an easy two marks. Common errors were for candidates to introduce their own output messages, or to use incorrect logic; typically where the equality condition produced both messages.

A wide variety of answers were considered acceptable including the use of two separate IF statements.

(d) This is only the second paper on which an explanation of the use of library programs was required and it is clearly still not well understood. The most common correct answers were that library programs are pre-written code which has the potential for reuse or code which is purchased from 3rd party suppliers. Such answers were however rare and there were far too many vague answers with statements such as “their use will make life easier for the programmer”.

(e) An encouraging sign on this paper, continuing on from June 2006, is much improved answers seen for the trace table question, especially as this question contained a procedure which had not appeared in previous questions.

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) Many candidates were able to explain that functions always return a value but few candidates were able to distinguish this from the way a procedure behaves.

For candidates who had covered this theory in a practical context this was an easy two marks. Candidates should have been exposed to a subset of the functions available in their programming language. The final part of the question stem "…or when using a generic software package" was intended to help the weaker candidates in triggering some of the functions they would have used; unfortunately, candidates often gave answers describing features of a generic software package.

(b) This question was generally well answered, although it was noticeable that the standard of answers varied between centres. Candidates who found the question easy were undoubtedly those who had practical experience of using functions which required none, one or two parameters when used.

(c) The most popular answer was to use identifier names for constants, followed by procedures and functions.

(d) This was well answered with most candidates able to score marks. The key word in the question stem was "advantage" and so answers required more than just a description of a compiler and an interpreter.

Q33.

(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.

Q34.

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.”

Q35.

This was another question which most candidates found difficult, if not impossible. However, some good candidates produced very good answers.

Most candidates were able to answer part (a).

The examiners only rarely awarded full marks for the trace table. A lot of candidates abandoned the trace once they realised that the numbers were being output in ascending order. This limited their reward to two or three marks at best since half of the marks depended on the trace being completed. Many candidates had difficulty logging the procedure calls even when they made a good attempt at showing the tree in the T column.

Some candidates got the two marks for part (b)(ii) without attempting the trace while others who showed the right output in (i) called the procedure a search or a bubble sort.