Version 1.0

General Certificate of Education

June 2010

Computing COMP1

Unit 1: Problem Solving, Programming, Data Representation and Practical Exercise

Final

Mark Scheme

Mark schemes are prepared by the Principal Examiner and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation meeting attended by all examiners and is the scheme which was used by them in this examination. The standardisation meeting ensures that the mark scheme covers the candidates' responses to questions and that every examiner understands and applies it in the same correct way. As preparation for the standardisation meeting each examiner analyses a number of candidates' scripts: alternative answers not already covered by the mark scheme are discussed at the meeting and legislated for. If, after this meeting, examiners encounter unusual answers which have not been discussed at the meeting they are required to refer these to the Principal Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of candidates' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

Further copies of this Mark Scheme are available to download from the AQA Website: www.aqa.org.uk

Set and published by the Assessment and Qualifications Alliance.

The Assessment and Qualifications Alliance (AQA) is a company limited by guarantee registered in England and Wales (company number 3644723) and a registered charity (registered charity number 1073334).
Registered address: AQA, Devas Street, Manchester M15 6EX

The following annotation is used in the mark scheme:

;	- means a single mark
//	- means alternative response
/	- means an alternative word or sub-phrase
A	- means acceptable creditworthy answer
R	- means reject answer as not creditworthy
I	- means ignore.

Qu	Part	Marking Guidance	Marks
1	**01**	167;	**1**
	02	10.4375 // $10\frac{7}{16}$;; 1 mark for correct integer part 1 mark for correct fractional part	**2**
	03	-;89; 1 mark for correct sign 1 mark for correct integer value	**2**
	04	A7;	**1**
2	**05**	128 // 2^7;	**1**
	06	1000010; **R.** more than 7 bits used	**1**
	07	01000001 **Mark as follows:** Correct parity bit for the candidate's data bits; Correct data bits; **R.** if not 8 bits	**2**

	08	Sender counts/checks the number of 1s in the bit pattern/value/data;
		(If even number of 1s then 0 parity bit is added; if odd 1 is added;) // Extra bit added to ensure even number of 1s;
		Receiver counts/checks the number of 1s in the bit pattern/value/data received; If odd it identifies that an error has occurred; and requests for data to be resent; **A.** If even it accepts the data received **A.** if even data is assumed to be correct; **A.** an even number of errors will be detected; **R** if even, data is correct
		// receiver regenerates parity bit from data received; compares generated parity bit with received parity bit; if different requests for data to be resent
		R. Implication that sender or receiver are people.
			Max 4

3	**09**	6/100; //600;;	**2**
	10	8 (bits); **A.** 1 byte;	**1**
	11	Sample at a frequency (at least) twice the rate; of the highest frequency (that can be present in the original signal);	**2**

4	**12**	Meaningful/appropriate/suitable identifiers // **A**. example; Indentation // effective use of white space; Subroutines / Procedures and functions/methods/modules; with interfaces // using parameters to pass values; Subroutines / Procedures and functions/methods/modules should execute a single task; Appropriate use of structured statements // use of (selection and repetition)/repetition; Avoid use of goto statements; Consistent use of case/style for identifier names; Use of named constants; Use of user-defined data types; Use of libraries; House-style naming conventions // following conventions; **A.** by explained example **A.** Use of local variables **R**. Commenting **R.** "easier to understand"	**Max 3**

<table>
<tr><td><b>5</b></td><td><b>13</b></td><td><i>Must have the concept of problem/task for the first mark</i>

A (step-by-step) description of how to complete a task / a description of a process that achieves some task / a sequence of steps that solve a problem / A sequence of unambiguous instructions for solving a problem;
<b>R.</b> Set of instructions

Independent of any programming language;
That can be completed in finite time;</td><td><b>Max 2</b></td></tr>
<tr><td></td><td><b>14</b></td><td>
<table>
<tr><th>Answer</th><th>Count</th><th>Remainder</th></tr>
<tr><td>True</td><td>-</td><td>-</td></tr>
<tr><td></td><td>2</td><td>1</td></tr>
<tr><td></td><td>3</td><td>1</td></tr>
<tr><td></td><td>4</td><td>3</td></tr>
<tr><td></td><td>5</td><td>2</td></tr>
<tr><td></td><td>6</td><td>1</td></tr>
</table>
<b>Mark as follows:</b>
answer column; <b>A.</b> True instead of blank cells <b>R.</b> if no evidence that dry run has been attempted
count column;</td><td></td></tr>
</table>

		1 mark per correct value in `remainder` column;;;;	**6**
	15	Works out if x is a prime number // Checks if x is divisible (with no remainder);	**1**

6	**16**	**VB.NET**	

```
Sub Main()
   Dim PlayerOneScore, PlayerTwoScore, NoOfGamesPlayed,
NoOfGamesInMatch As Integer
   Dim PlayerOneWinsGame As Char

   PlayerOneScore = 0
   PlayerTwoScore = 0
   Console.Write("How many games?")
   NoOfGamesInMatch = Console.ReadLine()
   For NoOfGamesPlayed = 1 To NoOfGamesInMatch
      Console.Write("Did Player One win the game (enter Y
or N)?")
      PlayerOneWinsGame = Console.ReadLine
      If PlayerOneWinsGame = "Y" Then
         PlayerOneScore = PlayerOneScore + 1
      Else
         PlayerTwoScore = PlayerTwoScore + 1
      End If
   Next
   Console.WriteLine(PlayerOneScore)
   Console.WriteLine(PlayerTwoScore)
   Console.ReadLine()
End Sub
```

VB6

```
Private Sub Form_Load()
   Dim PlayerOneScore As Integer
   Dim PlayerTwoScore As Integer
   Dim NoOfGamesPlayed As Integer
   Dim NoOfGamesInMatch As Integer
   Dim PlayerOneWinsGame As String

   PlayerOneScore = 0
   PlayerTwoScore = 0
   NoOfGamesInMatch = InputBox("How many games?")
   For NoOfGamesPlayed = 1 To NoOfGamesInMatch
      PlayerOneWinsGame = InputBox("Did Player One win
the game (enter Y or N)?")
      If PlayerOneWinsGame = "Y" Then
         PlayerOneScore = PlayerOneScore + 1
      Else
         PlayerTwoScore = PlayerTwoScore + 1
      End If
   Next
   MsgBox (PlayerOneScore)
   MsgBox (PlayerTwoScore)
End Sub
```

<table>
<tr>
<td></td>
<td></td>
<td>

Alternative answer – one msgbox instead of two

```
MsgBox (PlayerOneScore & vbCrLf & PlayerTwoScore)
```

Pascal

```
Program Question6;
Var PlayerOneScore, PlayerTwoScore, NoOfGamesPlayed,
NoOfGamesInMatch:Integer;
Var PlayerOneWinsGame:Char;
Begin
  PlayerOneScore := 0;
  PlayerTwoScore := 0;
  Writeln('How many games?');
  Readln(NoOfGamesInMatch);
  For NoOfGamesPlayed := 1 To NoOfGamesInMatch Do
    Begin
      Write('Did Player One win the game (enter Y or
N)?');
      Readln(PlayerOneWinsGame);
      If PlayerOneWinsGame = 'Y'
        Then PlayerOneScore := PlayerOneScore + 1
        Else PlayerTwoScore := PlayerTwoScore + 1;
    End;
  Writeln(PlayerOneScore);
  Writeln(PlayerTwoScore);
  Readln();
End.
```

Mark as follows:

All variables declared correctly; **I.** Case **A.** Minor typos **R.** If additional variables
`PlayerOneScore, PlayerTwoScore` initialised correctly;
Correct prompt (**I.** Case **A.** minor typos) followed by `NoOfGamesInMatch` assigned value entered by user;
FOR loop formed correctly including end of loop in correct place;
Correct prompt (**I.** Case **A.** minor typos) followed by `PlayerOneWinsGame` assigned value entered by user;
IF followed by correct condition; **R.** if does not cater for capital letter 'Y'
THEN followed by correct assignment statement;
ELSE followed by correct assignment statement;
output of both player's scores after loop; **A.** Message displayed with score

</td>
<td>

9

</td>
</tr>
<tr>
<td></td>
<td>

17

</td>
<td>

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

Mark as follows:

'How many games?' + user input of '3';
'Did Player One win the game (enter Y or N)? ' + user input of 'Y';
'N' entered by user for second/third game;
Correct scores shown for each player (**A.** follow through);
I. spacing

</td>
<td>

4

</td>
</tr>
</table>

7	**18**	`Board // PlayerOneName // PlayerTwoName // PlayerOneScore // PlayerTwoScore // XCoord // YCoord // ValidMove // NoOfMoves // GameHasBeenWon // GameHasBeenDrawn // CurrentSymbol // StartSymbol // PlayerOneSymbol // PlayerTwoSymbol // Answer` PHP: see PHP mark scheme Java only: `console;`	**1**
	19	`Row // Column // RandomNo // ValidMove // XOrOHasWon // WhoStarts;` VB6 only: `BoardAsString;` Java and Python： `X // Y;` Java and C#: `ObjRandom;` PHP: see PHP mark scheme	**1**
	20	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**
	21	When the user enters 'X' ; **or** 'O'; // When `PlayerOneSymbol` contains 'X'; **or** 'O';	**2**
	22	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**
	23	`NoOfMoves // Row // Column`; PHP: see PHP mark scheme	**1**
	24	`PlayerOneName // PlayerTwoName // WhoStarts // PlayerTwoSymbol // RandomNo;` Python only: `X // Y;` PHP: see PHP mark scheme	**1**
	25	`CheckValidMove;`	**1**

	26	**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;
		R. if extra code included
			1

	27	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**

8	**28**	Boundary values are those that are just inside, on and just outside the range of allowed values;	**1**
	29	2; 3; 4; **R.** non-integer values **Max** 1 if additional values given	**3**
	30	****SCREEN CAPTURE(S)**** Screen capture showing boundary test resulting in correct behaviour; Must match one of the boundary values given in 29. **R.** If screen capture does not show a correct boundary value given as an answer to question 29	**1**

9	**31**	**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;
```

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 (C, C#, PHP)

Using only one IF statement, 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)

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

32

****SCREEN CAPTURE(S)****
This is conditional on sensible code for 31

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

33

****SCREEN CAPTURE****
This is conditional on sensible code for 31. 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

10 **34**

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;
```

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**
	35	****SCREEN CAPTURE**** *This is conditional on sensible code for 34* **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**
	36	***SCREEN CAPTURE*** *This is conditional on sensible code for 34* **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**

11	**37**	**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;`
		
		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**
	38	****SCREEN CAPTURE**** *This is conditional on sensible answer for 37* Drawn board position with 9 symbols (as defined in preliminary material); Messages saying players have score of 0.5; **R.** other scores	**2**

12	**39**	**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;`
		
		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

	40	**VB.NET / VB6 / Pascal**
		`If NoOfMoves = 16`
		
		Mark as follows: Value of 9 changed to 16;
			1

	41	**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`
		
		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 39.
			1

42

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
```

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 39.

2

43

****SCREEN CAPTURE****
This is conditional on sensible answers for 39 and 42

displays 4 rows;
displays 4 columns;

2

44

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;
```

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 39;

1

45

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;
```

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 39.

4

46

****SCREEN CAPTURE****
This is conditional on sensible answers for 45, 42 and 39.

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**
	47	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;*	**2**

C Mark Scheme

<table>
<tr><th>Qu</th><th>Part</th><th>Marking Guidance</th><th>Mar</th></tr>
<tr><td>6</td><td>16</td><td><pre>
#include <stdio.h>
#include <conio.h>

int NoOfGamesInMatch;
int NoOfGamesPlayed;
int PlayerOneScore;
int PlayerTwoScore;
char PlayerOneWinsGame;

void main(void){
 PlayerOneScore = 0;
 PlayerTwoScore = 0;
 printf("How many games?\n");
 scanf("%i",&NoOfGamesInMatch);

for(NoOfGamesPlayed=1;NoOfGamesPlayed<=NoOfGamesInMatch;NoOfGamesPlayed++){
   printf("Did Player One win the game (enter Y or N)?\n");
   flushall();
   scanf("%c",&PlayerOneWinsGame);
   if(PlayerOneWinsGame == 'Y'){
      PlayerOneScore = PlayerOneScore + 1;
   }
   else {
      PlayerTwoScore = PlayerTwoScore + 1;
   }
 }
 printf("%i\n",PlayerOneScore);
 printf("%i\n",PlayerTwoScore);
 getch();
}
</pre></td><td>9</td></tr>
</table>

<table>
<tr><td>7</td><td>26</td><td><pre>RandomNo = rand() // whoStarts ='X' // whoStarts ='O';</pre></td><td>1</td></tr>
</table>

<table>
<tr><td>9</td><td>31</td><td><pre>
int CheckValidMove(int XCoordinate, int YCoordinate,
char Board[4][4]){
   int validMove;
   validMove = 1;
   // check x coordinate is valid
   if ((XCoordinate<1) || (XCoordinate>3)){
      validMove = 0;
   }
   if ((YCoordinate<1) || (YCoordinate>3)){
      validMove = 0;
   }
   if validMove == 1{
        if (Board[XCoordinate][YCoordinate] != ' '){
             validMove = 0;
        }
   }
   return validMove;
}
</pre></td><td>5</td></tr>
</table>

10	**34**	// check diagonals
		if((Board[1][1]==Board[2][2]) && (Board[1][1]==Board[3][3]) && (Board[2][2] !=' ')){
		xOrOHasWon = 1;
		}
		if((Board[1][3]==Board[2][2]) && (Board[3][1]==Board[2][2]) && (Board[2][2] !=' ')){
		xOrOHasWon = 1;
		}
		return xOrOHasWon;
			6

11	**37**	else {
		printf("A draw this time\n");
		PlayerOneScore = PlayerOneScore + 0.5;
		PlayerTwoScore = PlayerTwoScore + 0.5;
		}
			2

12	**39**	char Board**[5][5]**;	**1**

	40	if(NoOfMoves == **16**){
		GameHasBeenDrawn = 1;
		}
			1

	41	for(Row=1;Row<=**4**;Row++){
		for(Column=1;Column<=**4**;Column++){
			1

	42	printf(" | 1 2 3 **4**\n");
		printf("--+--------\n");
		for(Row=1;Row<=**4**;Row++){
		printf("%i |",Row);
		for(Column=1;Column<=**4**;Column++){
			2

	44	if ((XCoordinate<1) || (XCoordinate>**4**)){
		validMove = 0;
		}
		if ((YCoordinate<1) || (YCoordinate>**4**)){
		validMove = 0;
		}
			1

	45	for(Row=1;Row<=4;Row++){
		if((((Board[2][Row] == Board[3][Row]) && (Board[2][Row] == Board[1][Row])) || ((Board[2][Row] == Board[4][Row]) && (Board[2][Row] == Board[3][Row]))) && Board[2][Row] !=' '){
		xOrOHasWon = 1;
		}
			4

C# Mark Scheme

<table>
<tr><th>Qu</th><th>Part</th><th>Marking Guidance</th><th>Marks</th></tr>
<tr><td>6</td><td>16</td><td><pre>
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OsAndXsMatch
{
    class Program
    {
        static void Main(string[] args)
        {
            int PlayerOneScore = 0;
            int PlayerTwoScore = 0;
            int NoOfGamesPlayed;
            int NoOfGamesInMatch;
            char PlayerOneWinsGame;
            Console.Write("How many games?");
            NoOfGamesInMatch =
int.Parse(Console.ReadLine());
            for (NoOfGamesPlayed = 1; NoOfGamesPlayed <=
NoOfGamesInMatch; NoOfGamesPlayed++)
            {
                Console.Write("Did Player One win the
game (enter Y or N)?");
                PlayerOneWinsGame =
char.Parse(Console.ReadLine());
                if (PlayerOneWinsGame == 'Y')
                    PlayerOneScore++;
                else
                    PlayerTwoScore++;
            }
            Console.WriteLine(PlayerOneScore);
            Console.WriteLine(PlayerTwoScore);
            Console.ReadLine();
        }
    }
}
</pre></td><td>9</td></tr>
</table>

<table>
<tr><td>7</td><td>26</td><td><pre>
Random objRandom = new Random() // int RandomNo =
objRandom.Next(100) // WhoStarts = 'X' // WhoStarts =
'O';
</pre></td><td>1</td></tr>
</table>

<table>
<tr><td>9</td><td>31</td><td><pre>
public static bool CheckValidMove(int XCoordinate, int
YCoordinate, char[,] Board)
{
    bool ValidMove = true;

    if (XCoordinate < 1 || XCoordinate > 3)
         ValidMove = false;
</pre></td><td></td></tr>
</table>

```
    if (YCoordinate < 1 || YCoordinate > 3)
          ValidMove = false;
    if (ValidMove)
      if (Board[XCoordinate, YCoordinate] != ' ')
          ValidMove = false;
    return ValidMove;
}  // end CheckValidMove
```

Alternative not using local Boolean variable:

```
public static bool CheckValidMove(int XCoordinate, int
YCoordinate, char[,] Board)
{
    if (XCoordinate < 1 || XCoordinate > 3 ||
              YCoordinate < 1 || YCoordinate > 3 ||
               Board[XCoordinate, YCoordinate] != ' ')
        return false;
    else
       return true;
}  // end CheckValidMove
```

5

10 34

```
// check diagonals
if ((Board[1, 1] == Board[2, 2]) && (Board[2, 2] ==
Board[3, 3])
               && (Board[1, 1] != ' '))
   xOrOHasWon = true;
if ((Board[3, 1] == Board[2, 2]) && (Board[2, 2] ==
Board[1, 3])
               && (Board[3, 1] != ' '))
   xOrOHasWon = true;
```

6

11 37

```
else
{
    Console.WriteLine("A draw this time!");
    PlayerOneScore = PlayerOneScore + 0.5;
    PlayerTwoScore = PlayerTwoScore + 0.5;
}
```

2

12 39

```
public static char[,] Board = new char[5, 5];
```

1

40

```
if (NoOfMoves == 16)
```

1

41

```
for (Row = 0; Column <= 4; Row++)
{
    for (Column = 0; Column <= 4; Column++)
    {
       Board[Column, Row] = ' ';
    }
```

<table>
<tr><td></td><td></td><td><pre>}</pre></td><td><b>1</b></td></tr>
<tr><td></td><td><b>42</b></td><td><pre>Console.WriteLine(" | 1 2 3 4");
Console.WriteLine("--+--------");
for (Row = 0; Row <= 4; Row++)
    {
        Console.Write((Row + 1).ToString() + " | ");
        for (Column = 0; Column <= 4; Column++)
        {
           . . . .</pre></td><td><b>2</b></td></tr>
<tr><td></td><td><b>44</b></td><td><pre>bool ValidMove = true;

if (XCoordinate < 1 || XCoordinate > 4)
    ValidMove = false;
if (YCoordinate < 1 || YCoordinate > 4)
    ValidMove = false;
if (ValidMove)
      if (Board[XCoordinate, YCoordinate] != ' ')
           ValidMove = false;
return ValidMove;</pre>
Alternative not using local boolean variable:
<pre>if (XCoordinate < 1 || XCoordinate > 4 ||
      YCoordinate < 1 || YCoordinate > 4 ||
           Board[XCoordinate, YCoordinate] != ' ')
       return false;
else
       return true;</pre></td><td><b>1</b></td></tr>
<tr><td></td><td><b>45</b></td><td><pre>// check rows
for (Row = 1; Row <= 4; Row++)
{
      if (Board[1, Row] == Board[2, Row] &&
           Board[2, Row] == Board[3, Row] &&
                Board[2, Row] != ' ')
          XorOHasWon = true;
      if (Board[2, Row] == Board[3, Row] &&
              Board[3, Row] == Board[4, Row] &&
                Board[2, Row] != ' ')
         XorOHasWon = true;
}</pre></td><td><b>4</b></td></tr>
</table>

Java Mark Scheme

<table>
<tr><th>Qu</th><th>Part</th><th>Marking Guidance</th><th>Marks</th></tr>
<tr><td>6</td><td>16</td><td>

```
package comp1_java_2010_v4a; //this is optional
public class Question6 {
    public static void main(String[] args) {
        int noOfGamesInMatch;
        int noOfGamesPlayed;
        int playerOneScore;
        int playerTwoScore;
        char playerOneWinsGame;
        Console console = new Console();

        playerOneScore = 0;
        playerTwoScore = 0;
        noOfGamesInMatch = console.readInteger("How many
games? ");
        for (noOfGamesPlayed = 1; noOfGamesPlayed <=
noOfGamesInMatch; noOfGamesPlayed++) {
            playerOneWinsGame = console.readChar("Did
player One win the game (enter Y or N)? ");
            if (playerOneWinsGame == 'Y') {
                playerOneScore++;
            } else {
                playerTwoScore++;
            } // end if/else
        } // end for noOfGamesPlayed
        console.writeLine(playerOneScore);
        console.writeLine(playerTwoScore);
    } // end Question6
}
```

</td><td>9</td></tr>
<tr><td>7</td><td>26</td><td>

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

</td><td>1</td></tr>
<tr><td>9</td><td>31</td><td>

```
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
```

</td><td>5</td></tr>
</table>

<table>
<tr><td>10</td><td>34</td><td>   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;</td><td>6</td></tr>
</table>

<table>
<tr><td>11</td><td>37</td><td>} else {
    console.println("A draw this time!");
    playerOneScore = playerOneScore + 0.5f;
    playerTwoScore = playerTwoScore + 0.5f;
} // end if/else</td><td>2</td></tr>
</table>

<table>
<tr><td>12</td><td>39</td><td>char board[][] = new char[5][5];</td><td>1</td></tr>
<tr><td></td><td>40</td><td>if (noOfMoves == **16**) {
      gameHasBeenDrawn = true;
}</td><td>1</td></tr>
<tr><td></td><td>41</td><td>for (row = 1; row <= **4**; row++) {
    for (column = 1; column <= **4**; column++) {</td><td>1</td></tr>
<tr><td></td><td>42</td><td>console.println(" | 1 2 3 **4** ");
console.println("--+---------");
for (row = 1; row <= **4**; row++) {
    console.write(" | ");
    for (column = 1; column <= **4**; column++) {</td><td>2</td></tr>
<tr><td></td><td>44</td><td>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</td><td>1</td></tr>
<tr><td></td><td>45</td><td>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</td><td>4</td></tr>
</table>

PHP Mark Scheme

<table>
<tr><th>Qu</th><th>Part</th><th>Marking Guidance</th><th>Marks</th></tr>
<tr><td>6</td><td>16</td><td>&lt;?php
/* Question 6
*/
$PlayerOneScore = 0;
$PlayerTwoScore = 0;
$NoOfGamesInMatch = 0;
fwrite(STDOUT, "How many games?\n");
$NoOfGamesInMatch = intval(trim(fgets(STDIN)));
for ($NoOfGamesPlayed = 1; $NoOfGamesPlayed <= $NoOfGamesInMatch; $NoOfGamesPlayed++) {
      fwrite(STDOUT, "Did Player One win the game (enter Y or N)?");
      $PlayerOneWinsGame = trim(fgets(STDIN));
      if ($PlayerOneWinsGame == 'Y') {
            $PlayerOneScore++;
      } else {
            $PlayerTwoScore++;
      }
}
fwrite(STDOUT, $PlayerOneScore . "\n");
fwrite(STDOUT, $PlayerTwoScore . "\n");
?&gt;</td><td>9</td></tr>
<tr><td>7</td><td>18</td><td>$Board // $PlayerOneName // $PlayerTwoName // $PlayerOneScore // $PlayerTwoScore // $XCoord // $YCoord // $ValidMove // $NoOfMoves // $GameHasBeenWon // $GameHasBeenDrawn // $CurrentSymbol // $StartSymbol // $PlayerOneSymbol // $PlayerTwoSymbol // $Answer;</td><td>1</td></tr>
<tr><td></td><td>19</td><td>$Row // $Column // $ValidMove // $XorOHasWon // $RandomNumber;</td><td>1</td></tr>
<tr><td></td><td>23</td><td>$NoOfMoves // $Row // $Column;</td><td>1</td></tr>
<tr><td></td><td>24</td><td>$PlayerOneName // $PlayerTwoName // $PlayerTwoSymbol // $StartSymbol // $RandomNumber;</td><td>1</td></tr>
<tr><td></td><td>26</td><td>$RandomNumber = rand(1, 100);</td><td>1</td></tr>
<tr><td>9</td><td>31</td><td>function CheckValidMove($XCoordinate, $Ycoordinate, $Board){
      // check X coordinate is valid
      $ValidMove = true;
      if ($XCoordinate < 1 || $XCoordinate > 3) {
            $ValidMove = false;
      }
      <b>if ($YCoordinate < 1 || $YCoordinate &gt; 3) {</b>
            <b>$ValidMove = false;</b>
      <b>}</b></td><td></td></tr>
</table>

<table>
<tr><td></td><td></td><td>   **if ($ValidMove) {**
                  if ($Board[$XCoordinate][$YCoordinate] != '
')
                          $ValidMove = false;
          }
          return $ValidMove;
  }</td><td>5</td></tr>
</table>

<table>
<tr><td>10</td><td>34</td><td>if ($Board[1][1] == $Board[2][2] &&
                $Board[2][2] == $Board[3][3] &&
                $Board[2][2] != ' ')
        $XorOHasWon = true;
if ($Board[2][2] == $Board[3][1] &&
                $Board[2][2] == $Board[1][3] &&
                $Board[2][2] != ' ')
        $XorOHasWon = true;</td><td>6</td></tr>
</table>

<table>
<tr><td>11</td><td>37</td><td>else {
        fwrite(STDOUT, "A draw this time! \n");
        $PlayerOneScore = $PlayerOneScore + 0.5;
        $PlayerTwoScore = $PlayerTwoScore + 0.5;
}</td><td>2</td></tr>
</table>

<table>
<tr><td>12</td><td>39</td><td>No change necessary (as arrays in PHP dynamic)
$Board = array(array());</td><td>1</td></tr>
<tr><td></td><td>40</td><td>if ($NoOfMoves == **16**)</td><td>1</td></tr>
<tr><td></td><td>41</td><td>for ($Row = 1; $Row <= **4**; $Row++) {
        for ($Column = 1; $Column <= **4**; $Column++) {</td><td>1</td></tr>
<tr><td></td><td>42</td><td>fwrite(STDOUT, " | 1 2 3 **4** \n");
fwrite(STDOUT, "--+---------\n");
for($Row = 1; $Row <= **4**; $Row++) {
        fwrite(STDOUT, $Row . " |");
        for ($Column = 1; $Column <= **4**; $Column++)
        . . . .</td><td>2</td></tr>
<tr><td></td><td>44</td><td>// check X coordinate is valid
$ValidMove = true;
if ($XCoordinate < 1 || $XCoordinate > **4**) {
        $ValidMove = false;
}
// check Y coordinate is valid
if ($YCoordinate < 1 || $YCoordinate > **4**) {
        $ValidMove = false;
}
// check the cell is empty
if ($ValidMove) {
        if ($Board[$XCoordinate][$YCoordinate] != ' ')</td><td></td></tr>
</table>

```
          $ValidMove = false;
}
return $ValidMove;
```

1

45

```
for ($Row = 1; $Row < 5; $Row++) {
     if ($Board[1][$Row] == $Board[2][$Row] &&
               $Board[2][$Row] == $Board[3][$Row] &&
               $Board[2][$Row] != ' ')
          $XorOHasWon = true;
     if ($Board[2][$Row] == $Board[3][$Row] &&
               $Board[3][$Row] == $Board[4][$Row] &&
               $Board[2][$Row] != ' ')
          $XorOHasWon = true;
}
```

4

Python Mark Scheme

<table>
<tr><th>Qu</th><th>Part</th><th>Marking Guidance</th><th>Marks</th></tr>
<tr><td>6</td><td>16</td><td><b>Python 2.5</b>
<pre>
PlayerOneScore = 0
PlayerTwoScore = 0
NoOfGamesPlayed = 0
NoOfGamesInMatch = int(raw_input("How many games?"))
# accept input(("How many games?")
for NoOfGamesPlayed in range(NoOfGamesInMatch):
    PlayerOneWinsGame = raw_input("Did Player One win
the game (enter Y or N)?")
    If PlayerOneWinsGame == 'Y':
        PlayerOneScore = PlayerOneScore + 1
        # accept PlayerOneScore += 1
    else:
        PlayerTwoScore = PlayerTwoScore + 1
        #accept PlayerTwoScore += 1
print PlayerOneScore
print PlayerTwoScore
</pre>
<b>Python 3.0</b>
<pre>
PlayerOneScore = 0
PlayerTwoScore = 0
NoOfGamesInMatch = int(input("How many games?"))
# Accept:
# print("How many games?")
# NoOfGamesInMatch = int(input())
for NoofGamesPlayed in range(NoOfGamesInMatch):
    PlayerOneWinsGame = input("Did Player One win the
game (enter Y or N)?")
    If PlayerOneWinsGame == 'Y':
        PlayerOneScore = PlayerOneScore + 1
        # accept PlayerOneScore += 1
    else:
        PlayerTwoScore = PlayerTwoScore + 1
        # accept PlayerTwoScore += 1
print(PlayerOneScore)
print(PlayerTwoScore)

A. NoOfGamesPlayed = 0
</pre></td><td>9</td></tr>
</table>

<table>
<tr><td>7</td><td>26</td><td><pre>
RandomNo = random.randint(0, 100) //
WhoStarts = 'X' // WhoStarts = 'O';
</pre></td><td>1</td></tr>
</table>

<table>
<tr><td>9</td><td>31</td><td><pre>
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
</pre></td><td></td></tr>
</table>

		** if (ValidMove == True):**
		** if (Board[XCoordinate][YCoordinate] != ' '):**
		** ValidMove = False**
		    return ValidMove
			5

10	**34**	# 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
			6

11	**37**	**Python 2.5**
		        else:
		            print "A draw this time!"
		            PlayerOneScore += 0.5 # accept
		PlayerOneScore = PlayerOneScore + 0.5
		            PlayerTwoScore += 0.5
		
		Python 3.0
		       else:
		            print("A draw this time!")
		            PlayerOneScore += 0.5 # accept
		PlayerOneScore = PlayerOneScore + 0.5
		            PlayerTwoScore += 0.5
			2

12	**39**	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],
		               ]
			1

	40	if NoOfMoves == **16**:	**1**

	41	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.
			1

	42	**Python 2.5**
		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]	

<table>
<tr><td></td><td></td><td><pre>
        print
    print '\n'
</pre>
<b>Python 3.0</b>
<pre>
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')
</pre>
<b>A.</b> range(4) if candidate has used 0 for array position instead of 4.</td><td><b>2</b></td></tr>
<tr><td></td><td><b>44</b></td><td><pre>
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
</pre></td><td><b>1</b></td></tr>
<tr><td></td><td><b>45</b></td><td><pre>
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
</pre></td><td><b>4</b></td></tr>
<tr><td></td><td><b>47</b></td><td>Description of further list nesting (similar to 3d array)</td><td><b>2</b></td></tr>
</table>