AQA

AS
Computing

COMP1 – Problem Solving, Programming, Data Representation and Practical Exercise
Mark scheme

2510
June 2016

Version :1.0 Final

Mark schemes are prepared by the Lead Assessment Writer and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the students' responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation each associate analyses a number of students' scripts. Alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Assessment Writer.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' 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 from aqa.org.uk

Notation used in GCE Computing mark schemes:

; - 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
NE - means not enough

Qu	Part	Marking Guidance	Marks
1	01	180;	1
1	02	5.625 // 5 5/8 // 45/8 **Mark as follows:** 3 bits before binary point correct (5); 5 bits after binary point correct (0.625 or 5/8); **Alternative:** 45/8;;	2
1	03	Easier for people to read/understand; (Can be displayed using) fewer digits; More compact when printed/displayed; **NE.** Takes up less space **NE.** More compact **MAX** 1	1
1	04	B;4;	2
1	05	-;76;	2
1	06	4; **I**. any quotes	1
1	07	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** 2	2
1	08	4;	1

<table>
<tr><td>2</td><td>09</td><td>The number of pixels/dots; per cm/inch/unit of measurement;
//
Width x height; in pixels;</td><td>2</td></tr>
<tr><td>2</td><td>10</td><td>2^4 // 16;</td><td>1</td></tr>
<tr><td>2</td><td>11</td><td>512;;; //
16*16;*16;÷8; //
4096;; ÷8;

MAX 2 if final answer not correct</td><td>3</td></tr>
<tr><td>2</td><td>12</td><td>(For geometric images) less storage space /memory likely to be needed;
NE. less space
(For geometric images) will load faster from secondary storage;
(For geometric images) will download faster;
Can be scaled/resized/zoomed without distortion // can be scaled/resized/zoomed without loss of quality;
Image can be (more easily) searched for particular objects;
Can (more easily) manipulate individual objects in an image;
Can preserve the background so that it can be recreated if an object is deleted;

MAX 2</td><td>2</td></tr>
<tr><td>3</td><td>13</td><td><table><tr><th>Original State</th><th>Input</th><th>New State</th></tr><tr><td>S30</td><td>10</td><td>S40</td></tr><tr><td>S30</td><td>20</td><td>S50</td></tr><tr><td>S30</td><td>50</td><td>S0</td></tr><tr><td>S30</td><td>R</td><td>S0</td></tr></table>
Note: order of rows not important

Mark as follows:
Any 2 rows correct;
All 3 rows correct;</td><td>2</td></tr>
<tr><td>3</td><td>14</td><td>5;</td><td>1</td></tr>
<tr><td>4</td><td>15</td><td>12 (bits);</td><td>1</td></tr>
<tr><td>4</td><td>16</td><td>1600 //
800 * 2;</td><td>1</td></tr>
<tr><td>4</td><td>17</td><td>48000;;; //
12*1600;*20;/8;

A. alternative values to 1600 only if match (incorrect) answer from 16</td><td>3</td></tr>
</table>

4	18	An analogue signal is an electrical signal; that represents analogue/continuous data // that varies in a continuous manner.; Digital signals are electrical signals; (with voltage changes that are) in discrete steps.; **Note:** Only one mark for stating that either of / both of analogue and digital signals are electrical signals	3

5	19	1) Correct variable declarations for `X`, `F`, `L` and `N`; **Note for examiners** If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the four correct variables exist in the program code and the first value they are assigned is of the correct data type. 2) Correct output message `Enter a number:`; 3) `N` assigned the value entered by the user; 4) `F` assigned the value of 16.0; 5) `IF` statement with correct condition; 6) `X` assigned the value stored in `N`; 7) `WHILE` loop, with syntax allowed by the programming language with one correct condition for the termination of the loop; **A.** alternative correct logic for condition 8) `WHILE` loop, has correct 2nd condition and correct logic, `WHILE` is inside the selection structure; **A.** alternative correct logic for condition 9) Two correct assignment statements for `X` and `L` - inside the `WHILE` statement; **R.** If order of assignment statements incorrect 10) 2nd `WHILE` loop, with syntax allowed by the programming language and correct condition for the termination of the loop; **R.** If not inside 1st iterative structure 11) One correct assignment statement for `F` or `X` - inside the 2nd `WHILE` loop; 12) 2nd correct assignment statement for `F` or `X` - inside the 2nd `WHILE` loop; **R.** If order of assignment statements incorrect 13) Value of X outputted – must not be in an iterative structure and must be in the `THEN` part of the selection structure; 14) Correct output message `Not a number greater than or equal to 1` – must be in the `ELSE` part of the selection structure; **I.** Case of variable names and output messages **A.** Minor typos in variable names and output messages **I.** spacing in prompts **A.** initialisation of variables at declaration stage	14

5	20	****SCREEN CAPTURE**** *Must match code from 19, including messages on screen capture matching those in code. Code for 19 must be sensible.* **Mark as follows:** Value of 0.1 entered by the user followed by output message `Not a number greater than or equal to 1`;	1
5	21	****SCREEN CAPTURE**** *Must match code from 19, including messages on screen capture matching those in code. Code for 19 must be sensible.* **Mark as follows:** Value of 4.1 entered by the user followed by output of `2.050025`; **A.** output of any number between 2 and 2.1	1
5	22	(Estimate the) square root (of a number);	1
5	23	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; **NE.** Set of instructions Independent of any programming language // That can be completed in finite time;	2
6	24	`HumanPlayersTurn` // `MoveIsValid` // `FlipStillPossible` // `FlipFound` // `OpponentPieceFound`; **A.** `Piece`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
6	25	`DisplayGameBoard` // `SetUpGameBoard` // `CheckIfMoveIsValid` // `GameOver` // `PlayGame`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1

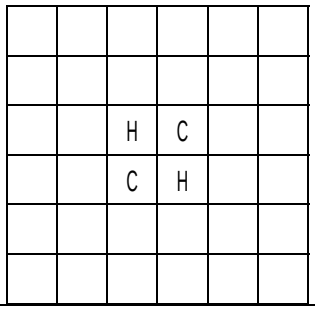
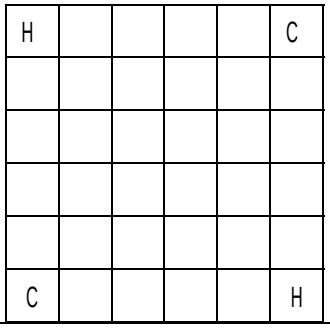
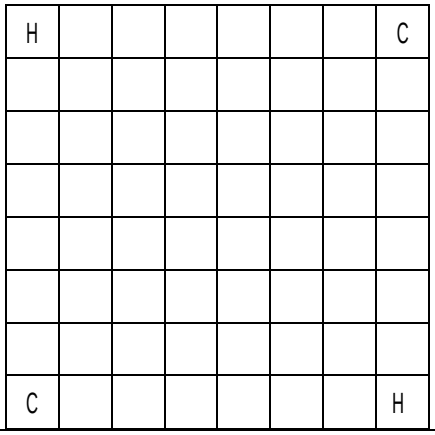
6	26	`Score` // `RowCount` // `ColumnCount`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
6	27	`Count` // `Row` // `Column` // `ColumnCount` // `RowCount`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
6	28	`Row // Column`; **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
6	29	`Choice` // `BoardSize` // `Move`; **A.** `Coordinates` **A.** `PlayerName` **R.** if any additional code **R.** if spelt incorrectly **I.** case	1
6	30	`ChangeBoardSize` // `GetHumanPlayerMove` // `GetComputerPlayerMove` // `GetPlayersName` // `PrintLine` // `DisplayGameBoard` // `DisplayMenu` // `GetMenuChoice`; **A.** `WriteLine` (VB6 only) **A.** `WriteNoLine` (VB6 only) **A.** `ReadLine` (VB6 only) **A.** `CreateBoard` (Python only) **R.** if any additional code **R.** if spelt incorrectly **I.** case	1

6	31	**VB.Net**
		`Row = Move Mod 10 //`
		`MoveIsValid = False //`
		`MoveIsValid = True //`
		`Column = Move \ 10;`
		
		Pascal
		`Row := Move Mod 10 //`
		`Column := Move Div 10 //`
		`MoveIsValid := False //`
		`MoveIsValid := True //`
		`CheckIfMoveIsValid := MoveIsValid;`
		I. semicolons
		
		VB6
		`Row = Move Mod 10 //`
		`MoveIsValid = False //`
		`MoveIsValid = True //`
		`Column = Move \ 10 //`
		`CheckIfMoveIsValid = MoveIsValid;`
		
		Java
		`row = move % 10 //`
		`column = move / 10 //`
		`moveIsValid = false //`
		`moveIsValid = true;`
		I. semicolons
		
		C#
		`Row = Move % 10 //`
		`Column = Move / 10 //`
		`MoveIsValid = false //`
		`MoveIsValid = true;`
		I. semicolons
		
		Python
		`Row = Move % 10 //`
		`Column = Move // 10 //`
		`MoveIsValid = False //`
		`MoveIsValid = True;`
		
		R. if any additional code
		R. if spelt incorrectly
		I. case
			1

6	32	Missing parameter (`Board`) in the calls to `FlipOpponentPiecesInOneDirection`; When `FlipOpponentPiecesInOneDirection` is called with the last two parameters being 0 and 0 this will not be for a diagonal; There are only three calls to FlipOpponentPiecesInOneDirection, there should be 4 // Not all diagonals have been checked; The subroutine calls have been placed before the variable `Row` has been given a value // The subroutine calls have been placed before the variable `Column` has been given a value // The subroutine calls have been placed before the variables have been given values // The subroutine calls have been placed after the if statement;	4
6	33	When row 10 is entered `Row` will be assigned a value of 0; When 10 / a 2-digit row is entered the value of `Column` is always going to be greater than 10 // when 10 / a 2-digit row is entered the value of `Column` will contain part of the row number; // When the user tries to place a piece in 1010 then the program will try to insert a piece in column 101 row 0;; **NE** `row` / `column` would be incorrect **Note for examiners** To get 2 marks the answer must include the inputs that would be used by the program to produce incorrect outputs for `row` and `column` and the incorrect values that would be assigned to both `row` and `column` from these inputs.	2
7	34	Selection structure with correct condition in correct place in program code e.g. by checking for empty string or checking length of string is 0; Correct message `That is not a valid name, using default name instead` displayed – must be inside the selection structure; Default name `Human player` assigned to the `PlayerName` variable – must be inside the selection structure;	3
7	35	****SCREEN CAPTURE**** *Must match code from 34. Code for 34 must be sensible.* No name entered followed by error message from code for 34 being displayed; Message saying `Human player enter the letter of your chosen option` displayed;	2

8	36	Additional parameter in subroutine call to `CheckIfMoveIsValid` subroutine; **R.** If parameters do not match routine interface for code from 37	1
8	37	1) Additional parameter in routine interface for `CheckIfMoveIsValid` subroutine; 2) Selection structure with two correct conditions and correct logic; 3) Selection structure with all four correct conditions and correct logic; 4) Check for square containing a space is only done within their attempt at 2 and/or 3 i.e. when their conditions have been met; 5) Correct Boolean values returned by subroutine under all circumstances; **R.** if check for boundary values are incorrect **A.** answers that obtain correct functionality by just modifying the existing selection structure rather than creating a new selection structure. The check that `Board(Row, Column)` contains a space must be the last condition to award the final two mark points for 37 and short circuit evaluation must be used to connect this condition to the others. If short circuit evaluation has not been used and evidence has been provided showing that the modified program works then examiners should refer the answer to their team leader for advice (in case this functionality could be obtained by altering compiler settings).	5

8	38	****SCREEN CAPTURE**** *Must match code from 36 and 37. Code for 36 and 37 must be sensible.* Coordinates of 74 followed by message saying their name followed by `enter the coordinates for the square that you want to put a piece in:` Coordinates of 47 followed by message saying their name followed by `enter the coordinates for the square that you want to put a piece in:`; **R.** if no/incorrect upper boundary check in code for 37 Coordinates of 10 followed by message saying their name followed by `enter the coordinates for the square that you want to put a piece in:` Coordinates of 01 followed by message saying their name followed by `enter the coordinates for the square that you want to put a piece in:` Coordinates of 66 followed by game board being displayed with a H in the bottom-right corner; **R.** if no/incorrect lower boundary check in code for 37. **R.** both marks if when coordinates entered are out of bounds the program code from 37 would execute the check for an invalid square being empty.	2
9	39	1) Appropriate message displayed to the user; 2) Code that allows the user to enter their choice; 3) Selection structure with suitable condition (comparing user's choice to another value) in correct place in code; 4) Correct code will be executed if the user chooses the four centre square option; 5) Attempt at code for the four corner square option is in the correct part of the selection structure and code will work correctly for at least one size of board; 6) Code for four corner square option works correctly for two corners for all board sizes; 7) Code for four corner square option works correctly for all four corners for all board sizes;	7

9	40	****SCREEN CAPTURE**** *Must match code from 39, including prompts on screen capture matching those in code. Code for 39 must be sensible.* User input for four centre square option followed by board display shown below; 	1
9	41	****SCREEN CAPTURE**** *Must match code from 39, including prompts on screen capture matching those in code. Code for 39 must be sensible.* User input for four corner square option followed by board display shown below; 	1
9	42	****SCREEN CAPTURE**** *Must match code from 39, including prompts on screen capture matching those in code. Code for 39 must be sensible.* User input for four corner square option followed by board display shown below; 	1
10	43	1) Correctly named subroutine `GetFlipSquares` created; **I.** Case and minor typos	10

		2) Two parameters of the correct data type in the routine interface for new subroutine; 3) Code that will cause message `Choosing one of the following squares will mean you flip some pieces:` to be displayed; **A.** Code for message in PlayGame 4) Code will look at every row on the board; **R.** if not for all board sizes 5) Code will look at every column on the board; **R.** if not for all board sizes **Note for examiners** The following apply to mark points 6, 7 and 8 **R.** if will not work correctly for a square on the edge of the board. **R.** if possible flips are checked for a non-empty square. **R.** if squares get "permanently" changed when checking for flips **I.** Out of bounds checks 6) Code correctly checks for flips in one direction; 7) Code correctly checks for flips in a second direction**;** 8) Code correctly checks for flips in all four directions 9) Selection structure that checks if square will result in flips; 10) Coordinates of square displayed – must be inside selection structure that checks if square will result in flips; **A.** alternative code which works correctly **MAX 9** if code will display any incorrect squares **MAX 9** if code will not display any correct square **Note for examiners** Refer unusual answers to Team Leader	

10	44	Correct subroutine call to `GetFlipSquares` subroutine; **R.** if subroutine call would not work **I.** Case Subroutine call added in correct place in program code;	2
10	45	****SCREEN CAPTURE**** *Must match code from 43 and 44, including prompts on screen capture matching those in code. Code for 43 and 44 must be sensible.* Message `Choosing one of the following squares will mean you flip some pieces:` is displayed followed by the values `24`, `35`, `53`, `42`; **I.** order of values displayed	1

Pascal

Qu	Part	Marking Guidance	Marks

5	19	`Program Question5;`
		`  Var`
		`  N : Real;`
		`  X : Real;`
		`  L : Real;`
		`  F : Real;`
		`  Begin`
		`  Write('Enter a number: ');`
		`  Readln(N);`
		`  F := 16;`
		`  If N >= 1`
		`  Then`
		`  Begin`
		`  X := N;`
		`  While (X * X - N > 1) And (F - 1 > 1)`
		`  Do`
		`  Begin`
		`  L := X;`
		`  X := X / F;`
		`  While (X * X <= N)`
		`  Do`
		`  Begin`
		`  F := F - 0.1;`
		`  X := L / F;`
		`  End;`
		`  End;`
		`  Writeln(X);`
		`  End`
		`  Else Writeln('Not a number greater than or equal to 1');`
		`  Readln;`
		`  End.`
			14

7	34	`Function GetPlayersName : String;`
		`  Var`
		`  PlayerName : String;`
		`  Begin`
		`  Write('What is your name? ');`
		`  Readln(PlayerName);`
		`  If PlayerName = ''`
		`  Then`
		`  Begin`
		`  Writeln('That is not a valid name, using default name instead');`
		`  PlayerName := 'Human player';`
		`  End;`
		`  GetPlayersName := PlayerName;`
		`  End;`
			3

8	36	...
		  Else Move := GetComputerPlayerMove(BoardSize);
		  MoveIsValid := CheckIfMoveIsValid(Board, Move, **BoardSize);**
		Until MoveIsValid;
		...
			1

8	37	Function CheckIfMoveIsValid(Board : TBoard; Move, **BoardSize : Integer**) : Boolean;
		  Var
		  Row : Integer;
		  Column : Integer;
		  MoveIsValid : Boolean;
		  Begin
		  Row := Move Mod 10;
		  Column := Move Div 10;
		  MoveIsValid := False;
		  If (Row <= BoardSize) And (Row >= 1) And (Column <= BoardSize) And (Column >= 1)
		  Then
		  If Board[Row, Column] = ' '
		  Then MoveIsValid := True;
		  CheckIfMoveIsValid := MoveIsValid;
		  End;
		
		Alternative answer
		
		Function CheckIfMoveIsValid(Board : TBoard; Move, **BoardSize** : Integer) : Boolean;
		  ...
		  MoveIsValid := False;
		  If (Row > BoardSize) Or (Row < 1) Or (Column > BoardSize) Or (Column < 1)
		  Then MoveIsValid := False
		  Else If Board[Row, Column] = ' '
		  Then MoveIsValid := True;
		  ...
			5

9	39	Procedure SetUpGameBoard(Var Board : TBoard; BoardSize : Integer);
		  Var
		  Row : Integer;
		  Column : Integer;
		  Choice : Char;
		  Begin
		  Write('Do you want to play with the starting pieces being on the four c(e)ntre squares');
		  Write('or on the four c(o)rner squares?');
		  Readln(Choice);
		  If Choice = 'e'
		  Then
		  For Row := 1 To BoardSize
		  Do
		  For Column := 1 To BoardSize
		  Do
		  If (Row = (BoardSize + 1) Div 2) And (Column = (BoardSize + 1) Div 2 + 1)
		  Or (Column = (BoardSize + 1) Div 2) And (Row = (BoardSize + 1) Div 2 + 1)
		  Then Board[Row, Column] := 'C'
		  Else
		  If (Row = (BoardSize + 1) Div 2 + 1) And (Column = (BoardSize + 1) Div 2 + 1)
		  Or (Column = (BoardSize + 1) Div 2) And (Row = (BoardSize + 1) Div 2)
		  Then Board[Row, Column] := 'H'
		  Else Board[Row, Column] := ' '
		  Else
		  For Row := 1 To BoardSize
		  Do
		  For Column := 1 To BoardSize
		  Do
		  If (Row = 1) And (Column = 1)
		  Or (Row = BoardSize) And (Column = BoardSize)
		  Then Board[Row, Column] := 'H'
		  Else
		  If (Row = 1) And (Column = BoardSize)
		  Or (Row = BoardSize) And (Column = 1)
		  Then Board[Row, Column] := 'C'
		  Else Board[Row, Column] := ' ';
		  End;
			7

Alternative answer

```
Procedure SetUpGameBoard(Var Board : TBoard; BoardSize :
Integer);
  Var
    Row : Integer;
    Column : Integer;
    Choice : Char;
  Begin
    Write('Do you want to play with the starting pieces
being on the four c(e)ntre squares');
    Write('or on the four c(o)rner squares?');
    Readln(Choice);
    If Choice = 'e'
      Then
        For Row := 1 To BoardSize
          Do
            For Column := 1 To BoardSize
              Do
                If (Row = (BoardSize + 1) Div 2) And
(Column = (BoardSize + 1) Div 2 + 1)
                  Or (Column = (BoardSize + 1) Div 2) And
(Row = (BoardSize + 1) Div 2 + 1)
                  Then Board[Row, Column] := 'C'
                  Else
                    If (Row = (BoardSize + 1) Div 2 + 1)
And (Column = (BoardSize + 1) Div 2 + 1)
                      Or (Column = (BoardSize + 1) Div 2)
And (Row = (BoardSize + 1) Div 2)
                      Then Board[Row, Column] := 'H'
                      Else Board[Row, Column] := ' '
      Else
        Begin
          For Row := 1 To BoardSize
            Do
              For Column := 1 To BoardSize
                Do
                  Board[Row, Column] := ' ';
          Board[1, 1] := 'H';
          Board[BoardSize, 1] := 'C';
          Board[1, BoardSize] := 'C';
          Board[BoardSize, BoardSize] := 'H';
        End;
  End;
```

10	43	Procedure GetFlipSquares(Board : TBoard; BoardSize : Integer);
		  Var
		  Row, Column: Integer;
		  SquareWillCauseFlips : Boolean;
		  OldValue : Char;
		  Begin
		  Writeln('Choosing one of the following squares will mean you flip some pieces: ');
		  For Row := 1 To BoardSize
		  Do
		  For Column := 1 To BoardSize
		  Do
		  If Board[Row, Column] = ' ' Then
		  Begin
		  OldValue := Board[Row, Column];
		  Board[Row, Column] := 'H';
		  SquareWillCauseFlips := False;
		  If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0 , 1)
		  Then SquareWillCauseFlips := True;
		  If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0 , -1)
		  Then SquareWillCauseFlips := True;
		  If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1 , 0)
		  Then SquareWillCauseFlips := True;
		  If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1 , 0)
		  Then SquareWillCauseFlips := True;
		  If SquareWillCauseFlips
		  Then
		  Begin
		  Write(Column);
		  Writeln(Row);
		  End;
		  Board[Row, Column] := OldValue;
		  End;
		  End;
			10

10	44	...
		  If HumanPlayersTurn
		  Then
		  Begin
		  GetFlipSquares(Board, BoardSize);
		  Move := GetMove(PlayerName);
		  End
		  Else Move := GetComputerPlayerMove(BoardSize);
		  MoveIsValid := CheckIfMoveIsValid(Board, Move, BoardSize);
		Until MoveIsValid;
		...
			2

VB.Net

Qu	Part	Marking Guidance	Marks

5	19	`Dim N As Single`
		`Dim X As Single`
		`Dim L As Single`
		`Dim F As Single`
		`Console.Write("Enter a number: ")`
		`N = Console.ReadLine`
		`F = 16`
		`If N >= 1 Then`
		`  X = N`
		`  While X * X - N > 1 And F - 1 > 1`
		`  L = X`
		`  X = X / F`
		`  While X * X <= N`
		`  F = F - 0.1`
		`  X = L / F`
		`  End While`
		`  End While`
		`  Console.WriteLine(X)`
		`Else`
		`  Console.WriteLine("Not a number greater than or equal to 1")`
		`End If`
		`Console.ReadLine()`
			14

7	34	`Function GetPlayersName() As String`
		`  Dim PlayerName As String`
		`  Console.Write("What is your name? ")`
		`  PlayerName = Console.ReadLine`
		`  If PlayerName = "" Then`
		`  Console.WriteLine("That is not a valid name, using default name instead")`
		`  PlayerName = "Human player"`
		`  End If`
		`  Return PlayerName`
		`End Function`
			3

8	36	`...`
		`  End If`
		`  MoveIsValid = CheckIfMoveIsValid(Board, Move, `**`BoardSize`**`)`
		`Loop Until MoveIsValid`
		`...`
			1

8	37	Function CheckIfMoveIsValid(ByVal Board(,) As Char, ByVal Move As Integer, **ByVal BoardSize As Integer**) As Boolean
		  Dim Row As Integer
		  Dim Column As Integer
		  Dim MoveIsValid As Boolean
		  Row = Move Mod 10
		  Column = Move \ 10
		  MoveIsValid = False
		  If Row > BoardSize Or Row < 1 Or Column > BoardSize Or Column < 1 Then
		  MoveIsValid = False
		  ElseIf Board(Row, Column) = " " Then
		  MoveIsValid = True
		  End If
		  Return MoveIsValid
		End Function
		
		Alternative answer
		
		Function CheckIfMoveIsValid(ByVal Board(,) As Char, ByVal Move As Integer, **ByVal BoardSize As Integer**) As Boolean
		  ...
		  If Row <= BoardSize And Row >= 1 And Column <= BoardSize And Column >= 1 Then
		  If Board(Row, Column) = " " Then
		  MoveIsValid = True
		  End If
		  End If
		  ...
			5

9	39	Sub SetUpGameBoard(ByVal Board(,) As Char, ByVal BoardSize As Integer)
		  Dim Row As Integer
		  Dim Column As Integer
		  Dim Choice As Char
		  Console.Write("Do you want to play with the starting pieces being on the four c(e)ntre squares or on the four c(o)rner squares?")
		  Choice = Console.ReadLine
		  If Choice = "e" Then
		  For Row = 1 To BoardSize
		  For Column = 1 To BoardSize
		  If Row = (BoardSize + 1) \ 2 And Column = (BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And Row = (BoardSize + 1) \ 2 + 1 Then
		  Board(Row, Column) = "C"
		  ElseIf Row = (BoardSize + 1) \ 2 + 1 And Column = (BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And Row = (BoardSize + 1) \ 2 Then
		  Board(Row, Column) = "H"
		  Else
		  Board(Row, Column) = " "
		  End If
		  Next
		  Next
		  Else
		  For Row = 1 To BoardSize
		  For Column = 1 To BoardSize
		  If Row = 1 And Column = 1 Or Row = BoardSize And Column = BoardSize Then
		  Board(Row, Column) = "H"
		  ElseIf Row = 1 And Column = BoardSize Or Row = BoardSize And Column = 1 Then
		  Board(Row, Column) = "C"
		  Else
		  Board(Row, Column) = " "
		  End If
		  Next
		  Next
		  End If
		End Sub
			7

Alternative answer

```
Sub SetUpGameBoard(ByVal Board(,) As Char, ByVal BoardSize
As Integer)
  Dim Row As Integer
  Dim Column As Integer
  Dim Choice As Char
  Console.Write("Do you want to play with the starting
pieces being on the four c(e)ntre squares or on the four
c(o)rner squares?")
  Choice = Console.ReadLine
  If Choice = "e" Then
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
        If Row = (BoardSize + 1) \ 2 And Column =
(BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And
Row = (BoardSize + 1) \ 2 + 1 Then
          Board(Row, Column) = "C"
        ElseIf Row = (BoardSize + 1) \ 2 + 1 And Column =
(BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And
Row = (BoardSize + 1) \ 2 Then
          Board(Row, Column) = "H"
        Else
          Board(Row, Column) = " "
        End If
      Next
    Next
  Else
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
        BoardSize(Row, Column) = " "
      Next
    Next
    Board(1,1) = "H"
    Board(BoardSize, BoardSize) = "H"
    Board(1, BoardSize) = "C"
    Board(BoardSize, 1) = "C"
  End If
End Sub
```

10	43	Sub GetFlipSquares(ByVal Board(,) As Char, ByVal BoardSize As Integer)
		Dim Row As Integer
		Dim Column As Integer
		Dim SquareWillCauseFlips As Boolean
		Dim OldValue As Char
		Console.WriteLine("Choosing one of the following squares will mean you flip some pieces: ")
		For Row = 1 To BoardSize
		For Column = 1 To BoardSize
		If Board(Row, Column) = " " Then
		OldValue = Board(Row, Column)
		Board(Row, Column) = "H"
		SquareWillCauseFlips = False
		If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then
		SquareWillCauseFlips = True
		ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then
		SquareWillCauseFlips = True
		ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then
		SquareWillCauseFlips = True
		ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then
		SquareWillCauseFlips = True
		End If
		If SquareWillCauseFlips Then
		Console.Write(Column)
		Console.WriteLine(Row)
		End If
		Board(Row, Column) = OldValue
		End If
		Next
		Next
		Console.WriteLine()
		End Sub
			10

10	44	...
		Do
		If HumanPlayersTurn Then
		GetFlipSquares(Board, BoardSize)
		Move = GetMove(PlayerName)
		Else
		Move = GetComputerPlayerMove(BoardSize)
		End If
		MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)
		Loop Until MoveIsValid
		...
			2

VB6

Qu	Part	Marking Guidance	Marks

5	19	Private Sub Form_Load()
		  Dim N As Single
		  Dim X As Single
		  Dim L As Single
		  Dim F As Single
		  N = ReadLine("Enter a number: ")
		  F = 16
		  If N >= 1 Then
		  X = N
		  While X * X - N > 1 And F - 1 > 1
		  L = X
		  X = X / F
		  While X * X <= N
		  F = F - 0.1
		  X = L / F
		  Wend
		  Wend
		  WriteLine(X)
		  Else
		  WriteLine("Not a number greater than or equal to 1")
		  End If
		End Sub
		
		Alternative answers could use some of the following instead of WriteLine:
		Console.Text = Console.Text & ...
		WriteLineWithMsg
		WriteWithMsg
		Msgbox
		WriteNoLine
			14

7	34	Private Function GetPlayersName() As String
		  Dim PlayerName As String
		  PlayerName = ReadLine("What is your name? ")
		  If PlayerName = "" Then
		  WriteLine ("That is not a valid name, using default name instead")
		  PlayerName = "Human player"
		  End If
		  GetPlayersName = PlayerName
		End Function
			3

8	36	...
		  End If
		  MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)
		Loop Until MoveIsValid
		...
			1

8	37	Private Function CheckIfMoveIsValid(ByRef Board() As String, ByVal Move As Integer, **ByVal BoardSize As Integer**) As Boolean
		  Dim Row As Integer
		  Dim Column As Integer
		  Dim MoveIsValid As Boolean
		  Row = Move Mod 10
		  Column = Move \ 10
		  MoveIsValid = False
		  If Row > BoardSize Or Row < 1 Or Column > BoardSize Or Column < 1 Then
		  MoveIsValid = False
		  ElseIf Board(Row, Column) = " " Then
		  MoveIsValid = True
		  End If
		  CheckIfMoveIsValid = MoveIsValid
		End Function
		
		Alternative answer
		
		Private Function CheckIfMoveIsValid(ByRef Board() As String, ByVal Move As Integer, **ByVal BoardSize As Integer**) As Boolean
		  ...
		  If Row <= BoardSize And Row >= 1 And Column <= BoardSize And Column >= 1 Then
		  If Board(Row, Column) = " " Then
		  MoveIsValid = True
		  End If
		  End If
		  ...
			5

9	39	Private Sub SetUpGameBoard(ByRef Board() As String, ByVal BoardSize As Integer)
		  Dim Row As Integer
		  Dim Column As Integer
		  Dim Choice As String
		  Choice = ReadLine("Do you want to play with the starting pieces being on the four c(e)ntre squares or on the four c(o)rner squares?")
		  If Choice = "e" Then
		  For Row = 1 To BoardSize
		  For Column = 1 To BoardSize
		  If Row = (BoardSize + 1) \ 2 And Column = (BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And Row = (BoardSize + 1) \ 2 + 1 Then
		  Board(Row, Column) = "C"
		  ElseIf Row = (BoardSize + 1) \ 2 + 1 And Column = (BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And Row = (BoardSize + 1) \ 2 Then
		  Board(Row, Column) = "H"
		  Else
		  Board(Row, Column) = " "
		  End If
		  Next
		  Next
		  Else
		  For Row = 1 To BoardSize
		  For Column = 1 To BoardSize
		  If Row = 1 And Column = 1 Or Row = BoardSize And Column = BoardSize Then
		  Board(Row, Column) = "H"
		  ElseIf Row = 1 And Column = BoardSize Or Row = BoardSize And Column = 1 Then
		  Board(Row, Column) = "C"
		  Else
		  Board(Row, Column) = " "
		  End If
		  Next
		  Next
		  End If
		End Sub
			7

Alternative answer

```
Private Sub SetUpGameBoard(ByRef Board() As String, ByVal
BoardSize As Integer)
  Dim Row As Integer
  Dim Column As Integer
  Dim Choice As String
  Choice = ReadLine("Do you want to play with the starting
pieces being on the four c(e)ntre squares or on the four
c(o)rner squares?")
  If Choice = "e" Then
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
        If Row = (BoardSize + 1) \ 2 And Column =
(BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And
Row = (BoardSize + 1) \ 2 + 1 Then
          Board(Row, Column) = "C"
        ElseIf Row = (BoardSize + 1) \ 2 + 1 And Column =
(BoardSize + 1) \ 2 + 1 Or Column = (BoardSize + 1) \ 2 And
Row = (BoardSize + 1) \ 2 Then
          Board(Row, Column) = "H"
        Else
          Board(Row, Column) = " "
        End If
      Next
    Next
  Else
    For Row = 1 To BoardSize
      For Column = 1 To BoardSize
        BoardSize(Row, Column) = " "
      Next
    Next
    Board(1,1) = "H"
    Board(BoardSize, BoardSize) = "H"
    Board(1, BoardSize) = "C"
    Board(BoardSize, 1) = "C"
  End If
End Sub
```

10	43	Private Sub GetFlipSquares(ByRef Board() As String, ByVal BoardSize As Integer)
		  Dim Row As Integer
		  Dim Column As Integer
		  Dim SquareWillCauseFlips As Boolean
		  Dim OldValue As String
		  WriteLine ("Choosing one of the following squares will mean you flip some pieces: ")
		  For Row = 1 To BoardSize
		  For Column = 1 To BoardSize
		  If Board(Row, Column) = " " Then
		  OldValue = Board(Row, Column)
		  Board(Row, Column) = "H"
		  SquareWillCauseFlips = False
		  If CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1) Then
		  SquareWillCauseFlips = True
		  ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1) Then
		  SquareWillCauseFlips = True
		  ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0) Then
		  SquareWillCauseFlips = True
		  ElseIf CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0) Then
		  SquareWillCauseFlips = True
		  End If
		  If SquareWillCauseFlips Then
		  WriteNoLine (Column)
		  WriteLine (Row)
		  End If
		  Board(Row, Column) = OldValue
		  End If
		  Next
		  Next
		  WriteLine ("")
		End Sub
			10

10	44	...
		Do
		  If HumanPlayersTurn Then
		  Call GetFlipSquares(Board, BoardSize)
		  Move = GetHumanPlayerMove(PlayerName)
		  Else
		  Move = GetComputerPlayerMove(BoardSize)
		  End If
		  MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize)
		Loop Until MoveIsValid
		...
			2

Java

Qu	Part	Marking Guidance	Marks

5	19	`public static void main(String[] args) {`
		`  float n;`
		`  float x;`
		`  float l;`
		`  float f;`
		`  console.print("Enter a number: ");`
		`  n = Float.parseFloat(console.readLine());`
		`  f = 16;`
		`  if (n >= 1) {`
		`  x = n;`
		`  while ((x * x - n > 1) && (f - 1 > 1)) {`
		`  l = x;`
		`  x = (float)(x / f);`
		`  while (x * x <= n) {`
		`  f = (float)(f - 0.1);`
		`  x = (float)(l / f) ;`
		`  }`
		`  }`
		`  console.println(x);`
		`  }`
		`  else {`
		`  console.println("Not a number greater than or equal to 1");`
		`  }`
		`}`
		
		Alternative answer (not using AQAConsole2016) :
		
		`public static void main(String[] args) {`
		`  Scanner in = new Scanner(System.in);`
		`  float n;`
		`  float x;`
		`  float l;`
		`  float f;`
		`  System.out.print("Enter a number: ");`
		`  n = Float.parseFloat(in.next());`
		`  f = 16;`
		`  if (n >= 1) {`
		`  x = n;`
		`  while ((x * x - n > 1) && (f - 1 > 1)) {`
		`  l = x;`
		`  x = (float)(x / f);`
		`  while (x * x <= n) {`
		`  f = (float)(f - 0.1);`
		`  x = (float)(l / f) ;`
		`  }`
		`  }`
			14

```
    System.out.println(x);
  }
  else {
    System.out.println("Not a number greater than or equal
to 1");
  }
}
```

Alternative answer (using double variables)

```
public static void main(String[] args) {
  double n;
  double x;
  double l;
  double f;
  console.print("Enter a number: ");
  n = Double.parseDouble(console.readLine());
  f = 16;
  if (n >= 1) {
    x = n;
    while ((x * x - n > 1) && (f - 1 > 1)) {
      l = x;
      x = x / f;
      while (x * x <= n) {
        f = f - 0.1;
        x = l / f ;
      }
    }
    console.println(x);
  }
  else {
    console.println("Not a number greater than or equal to
1");
  }
}
```

N.B. For n = 4.1 this gives output 2.049999999999958

7	34	String getPlayersName() {
		  String playerName;
		  console.print("What is your name? ");
		  playerName = console.readLine();
		  if (playerName.equals("")){
		  console.println("That is not a valid name, using default name instead");
		  playerName = "Human player";
		  }
		  return playerName;
		}
		
		Alternative answer :
		
		String getPlayersName() {
		  String playerName;
		  console.print("What is your name? ");
		  playerName = console.readLine();
		  if (playerName.isEmpty()) {
		  console.println("That is not a valid name, using default name instead");
		  playerName = "Human player";
		  }
		  return playerName;
		}
		
		Alternative answer :
		
		String getPlayersName() {
		  String playerName;
		  console.print("What is your name? ");
		  playerName = console.readLine();
		  if (playerName.length() == 0) {
		  console.println("That is not a valid name, using default name instead");
		  playerName = "Human player";
		  }
		  return playerName;
		}
			3

8	36	. . .
		  if (humanPlayersTurn) {
		  move = getHumanPlayerMove(playerName);
		  } else {
		  move = getComputerPlayerMove(boardSize);
		  }
		  moveIsValid = checkIfMoveIsValid(board, move, **boardSize**);
		} while (!moveIsValid);
		. . .
			1

8	37	boolean checkIfMoveIsValid(char[][] board, int move, **int boardSize**) {
		  int row;
		  int column;
		  boolean moveIsValid;
		  row = move % 10;
		  column = move / 10;
		  moveIsValid = false;
		  if (row <= boardSize && row >= 1 && column <= boardSize && column >= 1) {
		  if (board[row][column] == ' ') {
		  moveIsValid = true;
		  }
		  }
		  return moveIsValid;
		}
		
		Alternative answer :
		
		boolean checkIfMoveIsValid(char[][] board, int move, **int boardSize**) {
		  int row;
		  int column;
		  boolean moveIsValid;
		  row = move % 10;
		  column = move / 10;
		  moveIsValid = false;
		  if (row > boardSize || row < 1 || column > boardSize || column < 1) {
		  moveIsValid = false;
		  } else {
		  if (board[row][column] == ' ') {
		  moveIsValid = true;
		  }
		  }
		  return moveIsValid;
		}
			5

9	39	void setUpGameBoard(char[][] board, int boardSize) {
		char choice;
		console.print("Do you want to play with the starting pieces being on the four c(e)ntre squares");
		console.print(" or on the four c(o)rner squares? ");
		choice = console.readChar();
		if (choice == 'e') {
		for (int row = 1; row <= boardSize; row++) {
		for (int column = 1; column <= boardSize; column++) {
		if (row == (boardSize + 1) / 2 && column == (boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 && row == (boardSize + 1) / 2 + 1) {
		board[row][column] = 'C';
		} else {
		if (row == (boardSize + 1) / 2 + 1 && column == (boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 && row == (boardSize + 1) / 2) {
		board[row][column] = 'H';
		} else {
		board[row][column] = ' ';
		}
		}
		}
		}
		} else {
		for (int row = 1; row <= boardSize; row++) {
		for (int column = 1; column <= boardSize; column++) {
		if (row == 1 && column == 1 || row == boardSize && column == boardSize) {
		board[row][column] = 'H';
		} else {
		if (row == 1 && column == boardSize || row == boardSize && column == 1) {
		board[row][column] = 'C';
		} else {
		board[row][column] = ' ';
		}
		}
		}
		}
		}
		}
			7

Alternative answer :

```
void setUpGameBoard(char[][] board, int boardSize) {
  char choice;
  console.print("Do you want to play with the starting
pieces being on the four c(e)ntre squares");
  console.print(" or on the four c(o)rner squares? ");
  choice = console.readChar();
  if (choice == 'e') {
    for (int row = 1; row <= boardSize; row++) {
      for (int column = 1; column <= boardSize; column++) {
        if (row == (boardSize + 1) / 2 && column ==
(boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 &&
row == (boardSize + 1) / 2 + 1) {
          board[row][column] = 'C';
        } else {
          if (row == (boardSize + 1) / 2 + 1 && column ==
(boardSize + 1) / 2 + 1 || column == (boardSize + 1) / 2 &&
row == (boardSize + 1) / 2) {
            board[row][column] = 'H';
          } else {
            board[row][column] = ' ';
          }
        }
      }
    }
  } else {
    for (int row = 1; row <= boardSize; row++) {
      for (int column = 1; column <= boardSize; column++) {
        board[row][column] = ' ';
      }
    }
    board[1][1] = 'H';
    board[boardSize][1] = 'C';
    board[1][boardSize] = 'C';
    board[boardSize][boardSize] = 'H';
  }
}
```

10	43	`void getFlipSquares(char[][] board, int boardSize) {`
		`  boolean squareWillCauseFlips;`
		`  char oldValue;`
		`  console.println("Choosing one of the following squares will mean you flip some pieces");`
		`  for (int row = 1; row <= boardSize; row++) {`
		`  for (int column = 1; column <= boardSize; column++) {`
		`  if (board[row][column] == ' ') {`
		`  oldValue = board[row][column];`
		`  board[row][column] = 'H';`
		`  squareWillCauseFlips = false;`
		`  if (checkIfThereArePiecesToFlip(board, boardSize, row, column, 0, 1)) {`
		`  squareWillCauseFlips = true;`
		`  }`
		`  if (checkIfThereArePiecesToFlip(board, boardSize, row, column, 0, -1)) {`
		`  squareWillCauseFlips = true;`
		`  }`
		`  if (checkIfThereArePiecesToFlip(board, boardSize, row, column, 1, 0)) {`
		`  squareWillCauseFlips = true;`
		`  }`
		`  if (checkIfThereArePiecesToFlip(board, boardSize, row, column, -1, 0)) {`
		`  squareWillCauseFlips = true;`
		`  }`
		`  if (squareWillCauseFlips) {`
		`  console.print(column);`
		`  console.println(row);`
		`  }`
		`  board[row][column] = oldValue;`
		`  }`
		`  }`
		`  }`
		`  console.println();`
		`}`
			10

10	44	`. . .`
		`do {`
		`  if (humanPlayersTurn) {`
		`  **getFlipSquares(board, boardSize);**`
		`  move = getHumanPlayerMove(playerName);`
		`  } else {`
		`  move = getComputerPlayerMove(boardSize);`
		`  }`
		`  moveIsValid = checkIfMoveIsValid(board, move, boardSize);`
		`} while (!moveIsValid);`
		`. . .`
			2

C#

Qu	Part	Marking Guidance	Marks

5	19	`double  X, F, L, N;`
		`Console.WriteLine("Enter a number:");`
		`N = Convert.ToDouble(Console.ReadLine());`
		`F = 16.0;`
		`if (N >= 1.0)`
		`{`
		`  X = N;`
		`  while (X * X - N > 1.0 && F - 1.0 > 1.0)`
		`  {`
		`    L = X;`
		`    X = X / F;`
		`    while (X * X <= N )`
		`    {`
		`      F = F - 0.1;`
		`      X = L / F;`
		`    }`
		`  }`
		`  Console.WriteLine(X);`
		`}`
		`else`
		`{`
		`  Console.WriteLine("Not a number greater than or equal to 1");`
		`}`
		`Console.ReadLine();`
			14

7	34	`static string GetPlayersName()`
		`{`
		`  string PlayerName;`
		`  Console.Write("What is your name? ");`
		`  PlayerName = Console.ReadLine();`
		`  if (PlayerName == "")`
		`  {`
		`    Console.WriteLine("That is not a valid name, using default name instead");`
		`    PlayerName = "Human Player";`
		`  }`
		`  return PlayerName;`
		`}`
			3

8	36	`...`
		`    Move = GetComputerPlayerMove(BoardSize);`
		`  }`
		`  **MoveIsValid = CheckIfMoveIsValid(Board, Move, BoardSize);**`
		`}`
		`if (!HumanPlayersTurn)`
		`...`
			1

8	37	static bool CheckIfMoveIsValid(char[,] Board, int Move, **int BoardSize**)
		{
		  int Row;
		  int Column;
		  bool MoveIsValid;
		  Row = Move % 10;
		  Column = Move / 10;
		  MoveIsValid = false;
		  if (Row < 1 || Row > BoardSize || Column < 1 || Column > BoardSize)
		  {
		  MoveIsValid = false;
		  }
		  else if (Board[Row, Column] == ' ')
		  {
		  MoveIsValid = true;
		  }
		  return MoveIsValid;
		}
		
		Alternative answer
		
		static bool CheckIfMoveIsValid(char[,] Board, int Move, **int BoardSize**)
		{
		  ...
		  if (Row >= 1 && Row <= BoardSize && Column >= 1 && Column <= BoardSize)
		  {
		  if (Board[Row, Column] == ' ')
		  {
		  MoveIsValid = true;
		  }
		  }
		  ...
		}
			5

9	39	static void SetUpGameBoard(char[,] Board, int BoardSize)
		{
		  char Choice;
		  Console.Write("Do you want to play with the starting pieces being on the four c(e)ntre squares or the four c(o)rner squares?");
		  Choice = Convert.ToChar(Console.ReadLine());
		  if (Choice == 'e')
		  {
		  for (int Row = 1; Row <= BoardSize; Row++)
		  {
		  for (int Column = 1; Column <= BoardSize; Column++)
		  {
		  if (Row == (BoardSize + 1) / 2 && Column == (BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2 && Row == (BoardSize + 1) / 2 + 1)
		  {
		  Board[Row, Column] = 'C';
		  }
		  else if (Row == (BoardSize + 1) / 2 + 1 && Column == (BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2 && Row == (BoardSize + 1) / 2)
		  {
		  Board[Row, Column] = 'H';
		  }
		  else
		  {
		  Board[Row, Column] = ' ';
		  }
		  }
		  }
		  }
		  else
		  {
		  for (int Row = 1; Row <= BoardSize; Row++)
		  {
		  for (int Column = 1; Column <= BoardSize; Column++)
		  {
		  if (Row == 1 && Column == 1 || Row == BoardSize && Column == BoardSize)
		  {
		  Board[Row, Column] = 'H';
		  }
		  else if (Row == 1 && Column == BoardSize || Row == BoardSize && Column == 1)
		  {
		  Board[Row, Column] = 'C';
		  }
		  else
		  {
		  Board[Row, Column] = ' ';
		  }
		  }
		  }
		  }
		}
			7

Alternative answer

```
static void SetUpGameBoard(char[,] Board, int BoardSize)
{
  char Choice;
  Console.Write("Do you want to play with the starting
pieces being on the four c(e)ntre squares or the four
c(o)rner squares?");
  Choice = Convert.ToChar(Console.ReadLine());
  if (Choice == 'e')
  {
    for (int Row = 1; Row <= BoardSize; Row++)
    {
      for (int Column = 1; Column <= BoardSize; Column++)
      {
        if (Row == (BoardSize + 1) / 2 && Column ==
(BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2 &&
Row == (BoardSize + 1) / 2 + 1)
        {
          Board[Row, Column] = 'C';
        }
        else if (Row == (BoardSize + 1) / 2 + 1 && Column
== (BoardSize + 1) / 2 + 1 || Column == (BoardSize + 1) / 2
&& Row == (BoardSize + 1) / 2)
        {
          Board[Row, Column] = 'H';
        }
        else
        {
          Board[Row, Column] = ' ';
        }
      }
    }
  }
  else
  {
    for (int Row = 1; Row <= BoardSize; Row++)
    {
      for (int Column = 1; Column <= BoardSize; Column++)
      {
        Board[Row, Column] = ' ';
      }
      Board[1, 1] = 'H';
      Board[BoardSize, BoardSize] = 'H';
      Board[1, BoardSize] = 'C';
      Board[BoardSize, 1] = 'C';
    }
  }
}
```

10	43	static void GetFlipSquares(char[,] Board, int BoardSize)
		{
		  bool SquareWillCauseFlips;
		  char OldValue;
		  Console.WriteLine("Choosing one of the following squares will mean you flip some pieces: ");
		  for (int Row = 1; Row <= BoardSize; Row++)
		  {
		  for (int Column = 1; Column <= BoardSize; Column++)
		  {
		  if (Board[Row, Column] == ' ')
		  {
		  OldValue = Board[Row, Column];
		  Board[Row, Column] = 'H';
		  SquareWillCauseFlips = false;
		  if (CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1))
		  {
		  SquareWillCauseFlips = true;
		  }
		  else if (CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1))
		  {
		  SquareWillCauseFlips = true;
		  }
		  else if (CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0))
		  {
		  SquareWillCauseFlips = true;
		  }
		  else if (CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0))
		  {
		  SquareWillCauseFlips = true;
		  }
		  if (SquareWillCauseFlips)
		  {
		  Console.Write(Column);
		  Console.WriteLine(Row);
		  }
		  Board[Row, Column] = OldValue;
		  }
		  }
		  }
		  Console.WriteLine();
		}
			10

10	44	...
		do
		{
		  if (HumanPlayersTurn)
		  {
		  GetFlipSquares(Board, BoardSize);
		  Move = GetHumanPlayerMove(PlayerName);
		  }
		  else
		  {
		  Move = GetComputerPlayerMove(BoardSize);
		  }
		  MoveIsValid = CheckIfMoveIsValid(Board, Move);
		} while (!MoveIsValid);
		...
			2

Python 2

Qu	Part	Marking Guidance	Marks

5	19	`N = float(raw_input("Enter a number:"))`
		`F = 16.0`
		`if N >= 1.0:`
		`  X = N`
		`  while X * X - N > 1.0 and F - 1.0 > 1.0:`
		`  L = X`
		`  X = X / F`
		`  while X * X <= N:`
		`  F = F - 0.1`
		`  X = L / F`
		`  print X`
		`else:`
		`  print "Not a number greater than or equal to 1"`
			14

7	34	`def GetPlayersName():`
		`  PlayerName = raw_input("What is your name? ")`
		`  if PlayerName == "":`
		`  print "That is not a valid name, using default name instead"`
		`  PlayerName = "Human player"`
		`  return PlayerName`
		
		Alternative answer
		
		`def GetPlayersName():`
		`  PlayerName = raw_input("What is your name? ")`
		`  if len(PlayerName) == 0:`
		`  print "That is not a valid name, using default name instead"`
		`  PlayerName = "Human player"`
		`  return PlayerName`
			3

8	36	`...`
		`else:`
		`  Move = GetComputerPlayerMove(BoardSize)`
		`MoveIsValid = CheckIfMoveIsValid(Board, Move, `**`BoardSize)`**
		`...`
			1

8	37	def CheckIfMoveIsValid(Board, Move, **BoardSize**):
		  Row = Move % 10
		  Column = Move / 10
		  MoveIsValid = False
		  if Row > BoardSize or Row < 1 or Column > BoardSize or Column < 1:
		  MoveIsValid = False
		  elif Board[Row][Column] == " ":
		  MoveIsValid = True
		  return MoveIsValid
		
		Alternative answer
		
		def CheckIfMoveIsValid(Board, Move, **BoardSize**):
		  Row = Move % 10
		  Column = Move / 10
		  MoveIsValid = False
		  if Row <= BoardSize and Row >= 1 and Column <= BoardSize and Column >= 1:
		  if Board[Row][Column] == " ":
		  MoveIsValid = True
		  return MoveIsValid
		
		Alternative answer
		
		def CheckIfMoveIsValid(Board, Move, **BoardSize**):
		  Row = Move % 10
		  Column = Move / 10
		  MoveIsValid = False
		  if Row <= BoardSize and Row >= 1 and Column <= BoardSize and Column >= 1 and Board[Row][Column] == " **":**
		  MoveIsValid = True
		  return MoveIsValid
			5

9	39	def SetUpGameBoard(Board, BoardSize):
		Choice = raw_input("Do you want the starting pieces being on the four c(e)ntre squares or on the four c(o)rner squares?")
		if Choice == "e":
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if (Row == (BoardSize + 1) / 2 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2 + 1):
		Board[Row][Column] = "C"
		elif (Row == (BoardSize + 1) / 2 + 1 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2):
		Board[Row][Column] = "H"
		else:
		Board[Row][Column] = " "
		else:
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if Row == 1 and Column == 1 or Row == BoardSize and Column == BoardSize:
		Board[Row][Column] = "H"
		elif Row == 1 and Column == BoardSize or Row == BoardSize and Column == 1:
		Board[Row][Column] = "C"
		else:
		Board[Row][Column] = " "
		
		Alternative answer
		
		def SetUpGameBoard(Board, BoardSize):
		Choice = raw_input("Do you want the starting pieces being on the four c(e)ntre squares or on the four c(o)rner squares?")
		if Choice == "e":
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if (Row == (BoardSize + 1) / 2 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2 + 1):
		Board[Row][Column] = "C"
		elif (Row == (BoardSize + 1) / 2 + 1 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2):
		Board[Row][Column] = "H"
		else:
		Board[Row][Column] = " "
			7

		else: **for Row in range(1, BoardSize + 1):** **for Column in range(1, BoardSize + 1):** **Board[Row][Column] = " "** **Board[1][1] = "H"** **Board[BoardSize][BoardSize] = "H"** **Board[1][BoardSize] = "C"** **Board[BoardSize][1] = "C"**	

10	43	def GetFlipSquares(Board, BoardSize):
		print "Choosing one of the following squares will mean you will flip some pieces: "
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if Board[Row][Column] == " ":
		OldValue = Board[Row][Column]
		Board[Row][Column] = "H"
		SquareWillCauseFlips = False
		if CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1):
		SquareWillCauseFlips = True
		elif CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1):
		SquareWillCauseFlips = True
		elif CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0):
		SquareWillCauseFlips = True
		elif CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0):
		SquareWillCauseFlips = True
		if SquareWillCauseFlips:
		print str(Column) + str(Row)
		Board[Row][Column] = OldValue
		print
			10

10	44	. . .
		while not MoveIsValid:
		if HumanPlayersTurn:
		GetFlipSquares(Board, BoardSize)
		Move = GetHumanPlayerMove(PlayerName)
		else:
		. . .
			2

Python 3

Qu	Part	Marking Guidance	Marks

5	19	`N = float(input("Enter a number:"))`
		`F = 16.0`
		`if N >= 1.0:`
		`  X = N`
		`  while X * X - N > 1.0 and F - 1.0 > 1.0:`
		`  L = X`
		`  X = X / F`
		`  while X * X <= N:`
		`  F = F - 0.1`
		`  X = L / F`
		`  print(X)`
		`else:`
		`  print("Not a number greater than or equal to 1")`
			14

7	34	`def GetPlayersName():`
		`  PlayerName = input("What is your name? ")`
		`  **if PlayerName == "":**`
		`  **print("That is not a valid name, using default name instead")**`
		`  **PlayerName = "Human player"**`
		`  return PlayerName`
		
		Alternative answer
		
		`def GetPlayersName():`
		`  PlayerName = raw_input("What is your name? ")`
		`  **if len(PlayerName) == 0:**`
		`  **print("That is not a valid name, using default name instead")**`
		`  **PlayerName = "Human player"**`
		`  return PlayerName`
			3

8	36	`...`
		`else:`
		`  Move = GetComputerPlayerMove(BoardSize)`
		`MoveIsValid = CheckIfMoveIsValid(Board, Move, **BoardSize)**`
		`...`
			1

8	37	def CheckIfMoveIsValid(Board, Move, **BoardSize**):
		  Row = Move % 10
		  Column = Move // 10
		  MoveIsValid = False
		  if Row > BoardSize or Row < 1 or Column > BoardSize or Column < 1:
		  MoveIsValid = False
		  elif Board[Row][Column] == " ":
		  MoveIsValid = True
		  return MoveIsValid
		
		Alternative answer
		
		def CheckIfMoveIsValid(Board, Move, **BoardSize**):
		  Row = Move % 10
		  Column = Move // 10
		  MoveIsValid = False
		  if Row <= BoardSize and Row >= 1 and Column <= BoardSize and Column >= 1:
		  if Board[Row][Column] == " ":
		  MoveIsValid = True
		  return MoveIsValid
		
		Alternative answer
		
		def CheckIfMoveIsValid(Board, Move, **BoardSize**):
		  Row = Move % 10
		  Column = Move // 10
		  MoveIsValid = False
		  if Row <= BoardSize and Row >= 1 and Column <= BoardSize and Column >= 1 and Board[Row][Column] == " **":**
		  MoveIsValid = True
		  return MoveIsValid
			5

9	39	def SetUpGameBoard(Board, BoardSize):
		Choice = input("Do you want the starting pieces being on the four c(e)ntre squares or on the four c(o)rner squares?")
		if Choice == "e":
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if (Row == (BoardSize + 1) / 2 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2 + 1):
		Board[Row][Column] = "C"
		elif (Row == (BoardSize + 1) / 2 + 1 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2):
		Board[Row][Column] = "H"
		else:
		Board[Row][Column] = " "
		else:
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if Row == 1 and Column == 1 or Row == BoardSize and Column == BoardSize:
		Board[Row][Column] = "H"
		elif Row == 1 and Column == BoardSize or Row == BoardSize and Column == 1:
		Board[Row][Column] = "C"
		else:
		Board[Row][Column] = " "
		
		Alternative answer
		
		def SetUpGameBoard(Board, BoardSize):
		Choice = input("Do you want the starting pieces being on the four c(e)ntre squares or on the four c(o)rner squares?")
		if Choice == "e":
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if (Row == (BoardSize + 1) / 2 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2 + 1):
		Board[Row][Column] = "C"
		elif (Row == (BoardSize + 1) / 2 + 1 and Column == (BoardSize + 1) / 2 + 1) or (Column == (BoardSize + 1) / 2 and Row == (BoardSize + 1) / 2):
		Board[Row][Column] = "H"
		else:
		Board[Row][Column] = " "
			7

		else: **for Row in range(1, BoardSize + 1):** **for Column in range(1, BoardSize + 1):** **Board[Row][Column] = " "** **Board[1][1] = "H"** **Board[BoardSize][BoardSize] = "H"** **Board[1][BoardSize] = "C"** **Board[BoardSize][1] = "C"**	

10	43	def GetFlipSquares(Board, BoardSize):
		print("Choosing one of the following squares will mean you will flip some pieces: ")
		for Row in range(1, BoardSize + 1):
		for Column in range(1, BoardSize + 1):
		if Board[Row][Column] == " ":
		OldValue = Board[Row][Column]
		Board[Row][Column] = "H"
		SquareWillCauseFlips = False
		if CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, 1):
		SquareWillCauseFlips = True
		elif CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 0, -1):
		SquareWillCauseFlips = True
		elif CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, 1, 0):
		SquareWillCauseFlips = True
		elif CheckIfThereArePiecesToFlip(Board, BoardSize, Row, Column, -1, 0):
		SquareWillCauseFlips = True
		if SquareWillCauseFlips:
		print(str(Column) + str(Row))
		Board[Row][Column] = OldValue
		print()
			10

10	44	...
		while not MoveIsValid:
		if HumanPlayersTurn:
		GetFlipSquares(Board, BoardSize)
		Move = GetHumanPlayerMove(PlayerName)
		else:
		...
			2

>'