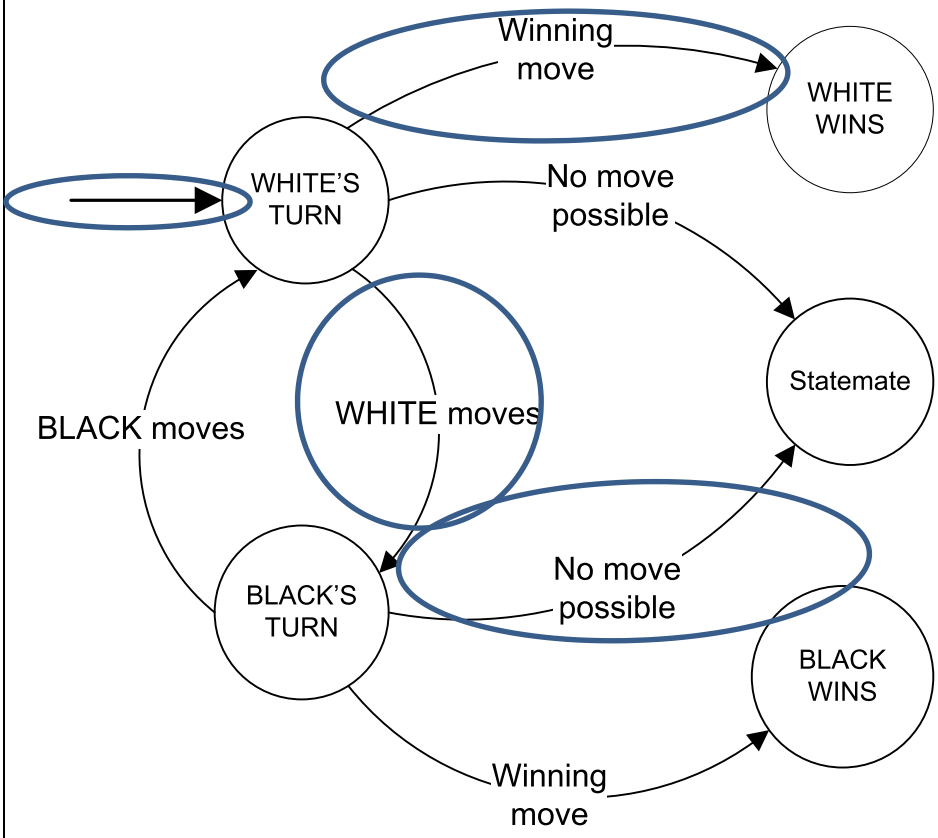
QUESTION 1.

<table>
<tr><td>5 (a) (i)</td><td>
<table>
<tr><th>Mark</th><th>Description</th><th>Expected result (Grade)</th></tr>
<tr><td></td><td>Normal</td><td>FAIL/PASS/MERIT/DISTINCTION</td></tr>
<tr><td></td><td>Abnormal</td><td>Error</td></tr>
<tr><td></td><td>Extreme/Boundary</td><td>FAIL/PASS/MERIT/DISTINCTION</td></tr>
</table>
3 × (mark + matching grade)
for abnormal data accept negative values, non-integer values, Expected Result: Error
0 and marks above 100 are still acceptable values
Do not accept FAIL in expected result column for Abnormal data
</td><td>3</td></tr>
<tr><td>(ii)</td><td>(The programmer is) concerned only with the input (i.e. the mark) to the function and monitoring the expected output (i.e. the grade)
// can compare expected result and actual result</td><td>1</td></tr>
<tr><td>(b)</td><td>Exception:
1. situation causing a crash / run-time error / fatal error 1

Exception handling:
2. code which is called when a run-time error occurs 1
3. … to avoid the program terminating/crashing 1</td><td>3</td></tr>
</table>

Question	Answer	
(c)	1 Open a non-existent file 2 Directory path does not exist 3 Attempt to read past the end of the file // attempt to read an empty file 4 Array subscript is out of range 5 Non-integer value / corrupt data read 6 File already open in a different mode // wrong file permissions	
(d) (i)	`09` // `9`	1
(ii)	1 Line 11 catches exceptions (only) between lines 05 and 10 1 2 Line 11 stops the program from crashing 1 3 Different exception types recognised 1 4 Each exception type has an appropriate message output 1 5 The program language has an (object) type `EXCEPTION` 1 6 `ThisException` is the instance of `EXCEPTION` which has been raised 1 7 `EXCEPTION` objects have a 'Message' property // the message property for `ThisException` is "Arithmetic operation resulted in an overflow" 1	Max 3
6 (a)	 Max 3 marks if extra states/transitions added.	4

Question	Answer	
(b) (i)	Mark as follows: 1 Declaration for array (character or string data type) 2 FOR loop for x going from 1 to 8, generating column index used in array 3 FOR loop for y going from 1–2, 3–6, 7–8 (Accept all squares being set to `'E'` and then overwritten with `'B'`, `'W'` respectively) 4 Setting squares to `'B', 'E', 'W'` (must be in quotes, accept single or double)	
(ii)	Mark as follows: 1 Procedure heading and declaration of 2 local variables 1 2 Establishing the stopper colour – opposite to the mover 1 3 Test for piece in column 1 (x>1) // column 8 (x<8) 1 4 Test for 'E' 1 5 Correct method for moving left // for moving right 1 6 until edge of board reached 1 7 until other colour (stopper colour) encountered 1 8 until own colour encountered (PieceColour) 1 9 Correct output for cell indexes 1 (accept for moving in 1 direction only) 10 including the 'REMOVE' message 1 Note: must use given parameter identifiers: PieceColour, xCurrent, yCurrent	**Max 5**
(c) (i)	Classes could be designed for : • the board • a piece Containment (Board contains Pieces) The pieces are <u>instances/objects</u> (of the Piece class)	**Max 2**

Question	Answer	
(ii)	Accept any reasonable answer, for example: `BOARD` class: Properties: • Number of squares / size / dimensions • Current state of all squares Methods: – • Set the starting board • Capture the finishing state of the board • Display the state of the board after each move `PIECE` class: Properties: • Starting x position • Starting y position • Current x position • current y position • Colour • State / Removed / Active Methods: • Move piece • Remove piece Mark as follows: two correct responses are worth 1 mark Accept other classes: Game, Player	

Programming code

6 (b) (i)

VB.NET

```
Dim Board(8, 8) As Char
Dim Row, Column As Integer
For Row = 1 To 2
    For Column = 1 To 8
       Board(Row, Column) = "B"
    Next
Next
For Row = 3 To 6
      For Column = 1 To 8
       Board(Row, Column) = "E"
    Next
Next
For Row = 7 To 8
    For Column = 1 To 8
       Board(Row, Column) = "W"
    Next
Next
```

PASCAL

```
var Row, Column : integer;
    Board : array[1..8, 1..8] of char;
begin
    for Row := 1 to 2 do
       for Column := 1 to 8 do
          Board[Row, Column] := 'B';
    for Row := 3 to 6 do
       for Column := 1 to 8 do
           Board[Row, Column] := 'E';
    for Row := 7 to 8 do
       for Column := 1 to 8 do
          Board[Row, Column] := 'W';
end.
```

PYTHON

```
Board = [["" for j in range(9)] for i in range(9)]
for Row in range(1, 3) :
    for Column in range(1, 9) :
        Board[Row][Column] = "B"
for Row in range(3, 7) :
    for Column in range(1, 9) :
        Board[Row][Column] = "E"
for Row in range(7, 9) :
    for Column in range(1, 9) :
        Board[Row][Column] = "W"
```

Alternative declarations of Board array :

```
Board = [[""] * 9 for i in range(9)]
```

```
Board = [[]]
for i in range(9) :
    for j in range(9) :
        Board.append("")
```

Instead of initialising with empty string, could initialise with ‘E’. this would then only require ‘B’ and ‘W’ loops later.

For example:

```
Board = [["E"] * 9 for i in range(9)] // Board =[["E"]*9]*9
for Row in range(1, 3) :
    for Column in range(1, 9) :
        Board[Row][Column] = "B"
for Row in range(7, 9) :
    for Column in range(1, 9) :
        Board[Row][Column] = "W"
```

```
Board =[]
for i in range(9):
    Board.append(["E"]*9)
```