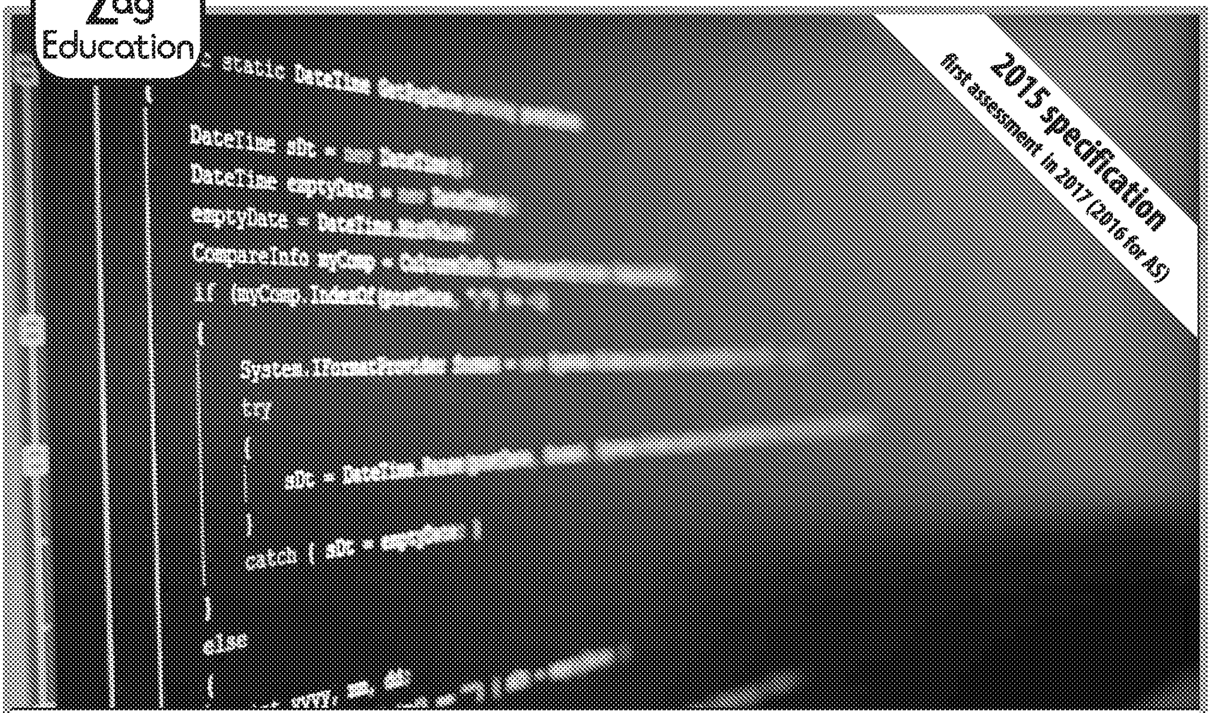
Zig Zag Education

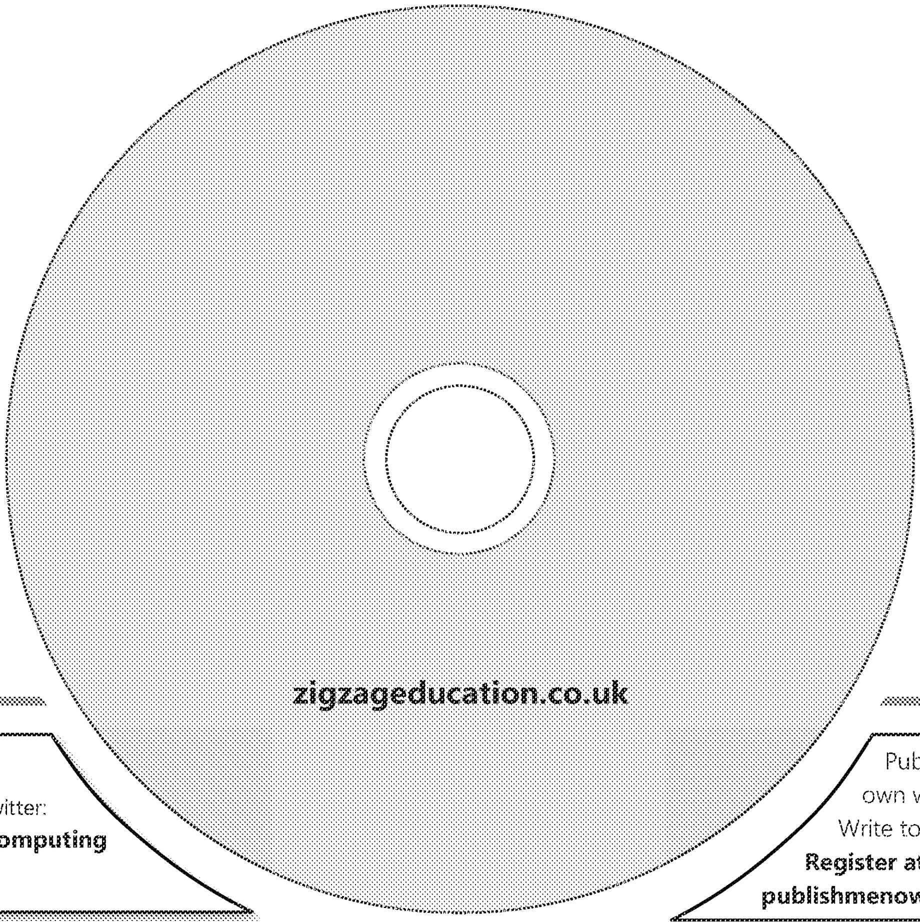
Computer Science AS & A Level | AQA | 7516 & 7517



2015 specification
first assessment in 2017 (2016 for AS)

C# Exercises

for AS & A Level AQA Computer Science



DH3/ 10508

POD 10508

zigzageducation.co.uk

Follow on Twitter:
@ZigZagComputing

Publish your
own work...
Write to a brief...
Register at
publishmenow.co.uk

Contents

Teacher's Introduction

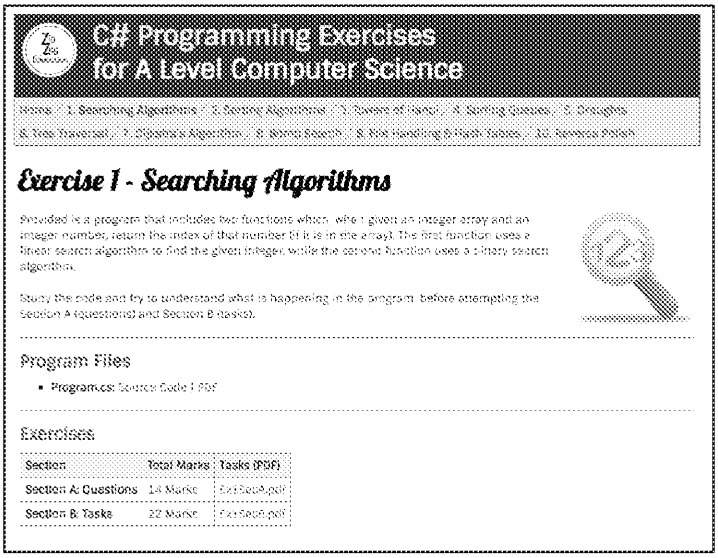
This resource has been designed to support the development of students' programming skills at KS5.

It contains 10 unique exercises, featuring a range of scenarios that develop the core programming principles, as well as bringing to life a number of important concepts across the A Level AQA specification – including programming constructs, recursion, global/local variables, modularity, debugging programs, object-oriented techniques, divide-and-conquer algorithms, data structures and standard algorithms.

Each exercise contains a combination of questions and tasks, and consists of two sections – Section A and Section B.

Section A
Section A provides a series of questions that test theoretical understanding of the code. The code that these questions are based on is provided as .cs files and as PDFs for convenient printing. These questions require written responses only – **no programming is required.**

Section B
Section B provides a series of tasks that require the programs to be modified in order to make improvements and/or develop their functionality – **programming is required.**



C# Programming Exercises for A Level Computer Science

Home | 1. Searching Algorithms | 2. Sorting Algorithms | 3. Towers of Hanoi | 4. Sorting Queues | 5. Draughts | 6. Tree Traversal | 7. Dijkstra's Algorithm | 8. Word Search | 9. File Handling & Hash Tables | 10. Reverse Polish

Exercise 1 - Searching Algorithms

Provided is a program that includes two functions which, when given an integer array and an integer number, return the index of that number (if it is in the array). The first function uses a linear search algorithm to find the given integer, while the second function uses a binary search algorithm.

Study the code and try to understand what is happening in the program before attempting the Section A (questions) and Section B (tasks).

Program Files

- **Program.cs:** Source Code | PDF

Exercises

Section	Total Marks	Tasks (PDF)
Section A: Questions	14 Marks	Ex1SecA.pdf
Section B: Tasks	22 Marks	Ex1SecB.pdf

For every exercise, there is source code, which students will need to save to their computer (or other local directory) and open in a code editor before they can complete the tasks.

Section B should take longer than Section A to complete and will aid preparation for the non-exam assessment (NEA) and any other practical assessments.

The question/task sheets can be photocopied or, if using the PDF versions on the CD, can be printed or simply followed on-screen (for Section B especially).

Suggested solutions and mark schemes for all questions/tasks are provided (in print and as electronic copies on the CD). Note that credit should be given for any valid responses that are not explicitly included in this resource. Exemplar C# files demonstrating all of the Section B changes made for every exercise are also included on the CD.

Note that the mark schemes and solution files are not directly accessible to the students via the HTML interface (index.html), but can be accessed directly from the resource folder.

A Level AQA Computer Science Specification Map

	1	2	3	4	5	6	7	8	9	10
	Searching Algorithms	Sorting Algorithms	Towers of Hanoi	Sorting Queues	Draughts	Tree Traversal	Dijkstra's Algorithm	Bomb Search	File Handling & Hash Tables	Reverse Polish
1.1.1 – Data types	✓	✓	✓	✓	✓		✓	✓	✓	✓
1.1.2 – Programming concepts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.1.3 – Arithmetic operations	✓	✓		✓	✓	✓	✓	✓	✓	✓
1.1.4 – Relational operations	✓	✓	✓	✓	✓		✓	✓	✓	✓
1.1.5 – Boolean operations	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.1.6 – Constants and variables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.1.7 – String handling	✓	✓	✓		✓		✓	✓	✓	✓
1.1.8 – Random numbers	✓							✓		
1.1.9 – Exception handling	✓	✓			✓			✓		✓
1.1.10 – Subroutines		✓	✓	✓	✓	✓	✓	✓	✓	✓
1.1.11 – Parameters	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.1.12 – Returning values	✓	✓	✓		✓	✓	✓	✓	✓	✓
1.2.1 – Programming paradigms	✓	✓	✓	✓	✓		✓	✓	✓	✓
1.2.2 – Procedural paradigms	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.2.3 – Object-oriented paradigms	✓	✓		✓	✓	✓	✓	✓	✓	✓

(Continues on next page)

(Continued from previous page)

	1	2	3	4	5	6	7	8	9	10
	Searching Algorithms	Sorting Algorithms	Towers of Hanoi	Sorting Queues	Draughts	Tree Traversal	Dijkstra's Algorithm	Bomb Search	File Handling & Hash Tables	Reverse Polish
2.1.2 – Arrays	✓	✓	✓	✓	✓		✓	✓	✓	✓
2.1.4 – Data structures			✓	✓		✓	✓			✓
2.2.1 – Queues				✓						
2.3.1 – Stacks			✓							✓
2.4.1 – Graphs							✓			
2.5.1 – Trees						✓				✓
2.6.1 – Hash tables									✓	
3.1.1 – Graph-traversal						✓				
3.2.1 – Tree-traversal						✓				✓
3.3.1 – Reverse Polish Notation										✓
3.4.1 – Linear search	✓								✓	
3.4.2 – Binary search	✓									
3.5.1 – Bubble sort		✓								
3.5.2 – Merge sort		✓								
3.6.1 – Dijkstra's shortest path							✓			

Exercise 1 – Searching Algorithms

Section A

A 1 Give a line number from the program that contains a function call.

A 2 Give the line number from the program that contains a DIV operatio

A 3 Explain why, given a choice of both, a binary search is often preferabl

A 4 Explain why some arrays are not searchable with a binary search algo

A 5 The program uses a constant.

Explain why a constant was a suitable choice in this case.

A 6 Both functions return -1 if no value is found.

Explain why -1 is a suitable choice of value in this case.

A 7 The `linearSearch` function does not use an ELSE statement as par

Explain why it is not essential to use an ELSE statement in this case.

A 8 Explain what is meant by the time complexity of an algorithm.

....

....

....

A 9 State the time complexity of the linear search and binary search algo

....

....

....

A 10 The binary search algorithm can be implemented using recursion.

Explain why a recursive version of the binary search algorithm may no

....

....

....

Exercise 1 – Searching Algorithms

Section B

B 1

Identify and correct the syntax error in this version of the linear searc

```
private static int linearSearch(int[] searchList, int se
    for(int i=0; i < searchList.Length; i++) {
        if (searchList[i] = searchVal) {
            return i;
        }
    }
    Console.WriteLine("Value not found!");
    return VALUE_NOT_FOUND;
}
```

Program in question has been successfully corrected □

B 2

Modify the program so that when the functions do not find the value
part of the message shown on the console.

Program updated □

B 3

Modify the program so that the identifier `x` is replaced with a better v

Program updated □

B 4

Modify the program to add a `recursiveBinarySearch` function th
and indices [illegible] at the start and end of the portion of the array wh
recursive binary search to return the index of the search value if it is i
"`Value not found`" if the value is found not to be in the array. Th
The main program procedure should be updated to call this procedu

Program updated □

B 5

Modify the program to add a `getVal` function that asks the user for
integer. This function should take no arguments and be able to handl
input. The main program procedure should be updated to call this fu
will end up being passed to each of the search methods.

Program updated □

B 6

Modify the program to add a `generateList` function that is given
returns an ordered array of all positive integers from 1 to the given v
procedure should be updated to call this procedure to create the `se`
given by the user.

Program updated □

B 7

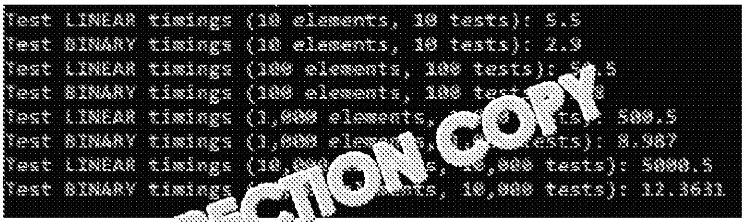
Create duplicate copies of the methods `linearSearch` and `binar`
be on comparing their time efficiency. The duplicate methods should
`timedLinearSearch` and `timedBinarySearch`. The new functio
a `count` variable that increments by 1 every time a new element is v
be returned when the search ends (either if the value is found or whe
search value is not in the array).

A `testLinearTimings` function should now be added that accepts
- `n` for the length of the array to be used for testing purposes
- `tests` for the number of tests that should be carried out on

The function should generate an array of length `n` and then repeated
array of `n` elements. It should then return the average (i.e. the m
`timedLinearSearch` method. It is advisable to search for values 1,
elements in the array are searched for.

Next, a similar method called `testBinaryTimings` needs to be cre
on arrays using binary searches and then find the average amount of
`testLinearTimings` function and modify the copy to fulfil the sa
searches.

The main program procedure should now have test code added to it
1,000 and 10,000 are created and the same numbers of searches are
each array is searched for once). The main program should output th
searching alongside the average time taken for binary searching, suc



```
Test LINEAR timings (10 elements, 10 tests): 5.5
Test BINARY timings (10 elements, 10 tests): 2.9
Test LINEAR timings (100 elements, 100 tests): 50.5
Test BINARY timings (100 elements, 100 tests): [illegible]
Test LINEAR timings (1,000 elements, 1,000 tests): 500.5
Test BINARY timings (1,000 elements, 1,000 tests): 8.987
Test LINEAR timings (10,000 elements, 10,000 tests): 5000.5
Test BINARY timings (10,000 elements, 10,000 tests): 12.3631
```

Program [illegible]

Exercise 2 – Sorting Algorithms

Section A

A 1 Explain how the integer values for the array are read in on line 9.

....

....

....

....

A 2 State the line number from the `mergeSort()` method where recursi

....

A 3 Define 'recursion'.

....

....

A 4 The character '\t' is used in the `Main()` method.

Explain what it is and how it has been used in this case.

....

....

....

A 5 When the `bubbleSort` function is called, the program uses a Boolea

Explain the role played by this variable.

....

....

A 6 A FOR loop is used in the `bubbleSort()` method. The value of the c
as far as the value before SIZE-1 instead of going all the way up to th

Explain why this is the correct approach to use in this case.

....

....

....

....

A 7 The purpose of the `split()` method is to separate one array into tw elements in the original array is even, the two arrays will be of equal le

Determine what will happen to the middle element of the original arr array contains an odd number of elements. Explain the role of the DIV

A 8 The merge sort algorithm is an example of a divide-and-conquer algo Explain what a divide-and-conquer algorithm is.

A 9 the time complexity of the bubble sort and merge sort algorith

A 10 Another method of sorting an array of numbers is known as an insert Describe how the insertion sort algorithm works.

Exercise 2 – Sorting Algorithms

Section B

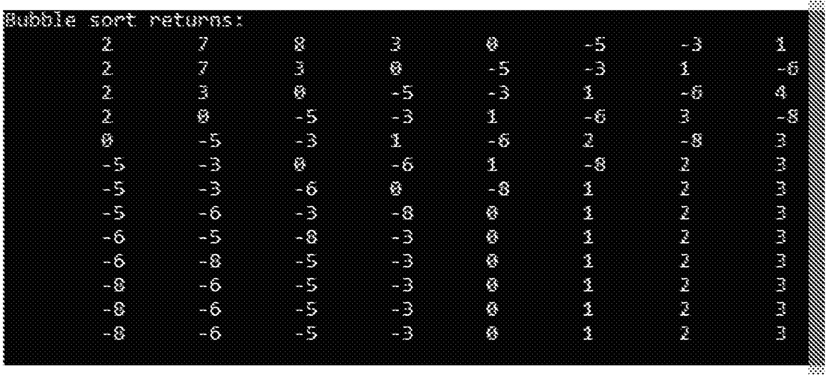
B 1

Modify the program to allow the user to enter a list of 12 integers.

Program updated □

B 2

Modify the program so that the bubbleSort function outputs the fu
has stepped through all elements from left to right and before it resta
it by passing a copy of the current array to a newly written metho
this has been achieved, modify the Main() method to make use of t
so many FOR loops which do the same task.



```
Bubble sort returns:
        2       7       8       3       0      -5      -3       1
        2       7       3       0      -5      -3       1      -6
        2       3       0      -5      -3       1      -6       4
        2       0      -5      -3       1      -6       3      -8
        0      -5      -3       1      -6       2      -8       3
       -5      -3       0      -6       1      -8       2       3
       -5      -3      -6       0      -8       1       2       3
       -5      -6      -3      -8       0       1       2       3
       -6      -5      -8      -3       0       1       2       3
       -6      -8      -5      -3       0       1       2       3
       -8      -6      -5      -3       0       1       2       3
       -8      -6      -5      -3       0       1       2       3
       -8      -6      -5      -3       0       1       2       3
```

Program updated □

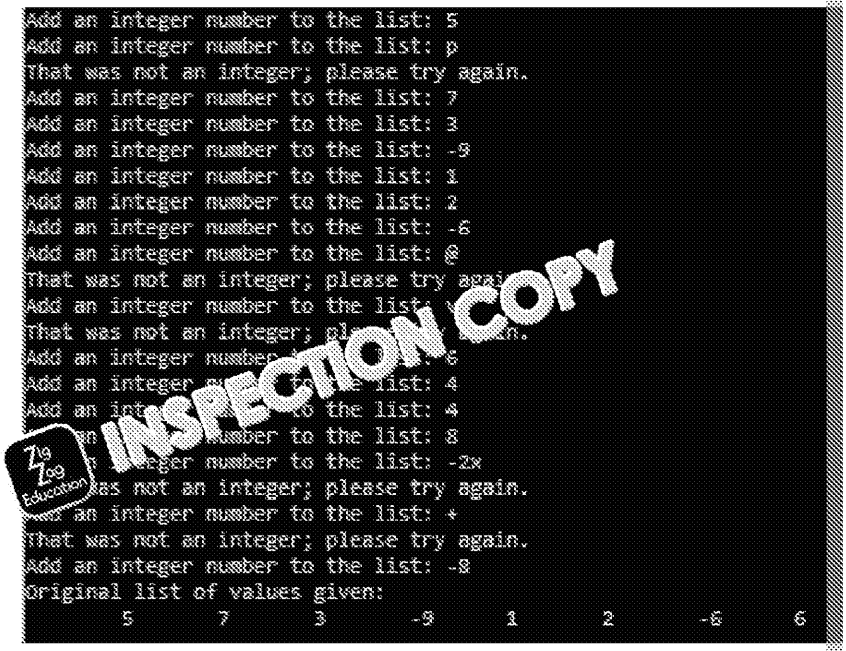
B 3

Modify the FOR loop of the bubbleSort function so that it doesn't
elements that are known to already be sorted, i.e. those that have be
after each pass. To achieve this, stop the nested loop from going all t

Program updated □

B 4

Modify the program so that the program does not crash if the user e
prompted to add a number to the array. Your solution should display
them when they have entered a non-integer number and keep asking
a valid integer.



```
Add an integer number to the list: 5
Add an integer number to the list: p
That was not an integer; please try again.
Add an integer number to the list: 7
Add an integer number to the list: 3
Add an integer number to the list: -9
Add an integer number to the list: 1
Add an integer number to the list: 2
Add an integer number to the list: -6
Add an integer number to the list: @
That was not an integer; please try again.
Add an integer number to the list: v
That was not an integer; please try again.
Add an integer number to the list: 6
Add an integer number to the list: 4
Add an integer number to the list: 4
Add an integer number to the list: 8
Add an integer number to the list: -2x
That was not an integer; please try again.
Add an integer number to the list: +
That was not an integer; please try again.
Add an integer number to the list: -8
Original list of values given:
        5       7       3      -9       1       2      -6       6
```

Program updated □

B 5

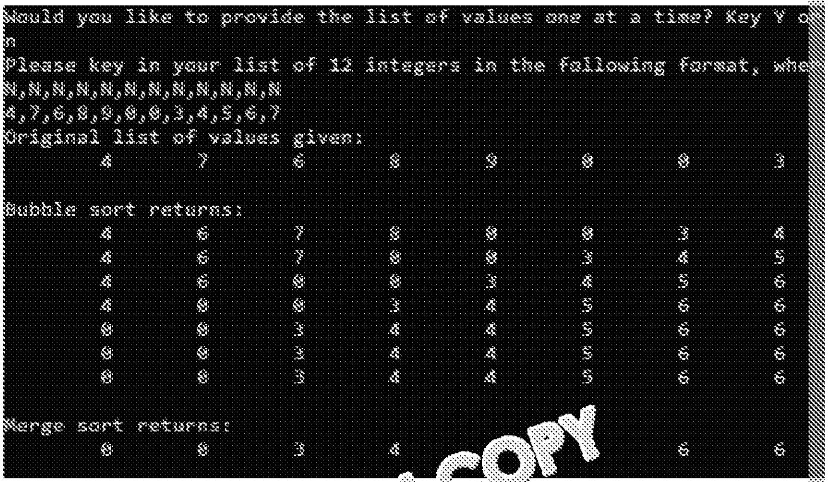
Currently, the code (which has just been improved in Task B4) that as into the array is hard-coded into the main program procedure, and s

Modify the program so that this code is moved into a new `GetList` f array as a parameter and returns the array fully populated with value the main program procedure to overwrite the empty array `numList` as described in Task B4.

Program updated □

B 6

Modify the `GetList` function so that the user can also choose to en separated by commas (e.g. 4, 4, 2, 17, 14, 12) to give their entire array individually. The `Split()` method of the `String` class may be used They should still have the option to enter numbers individually if they requirement in this task to make the list option robust enough to de



```
Would you like to provide the list of values one at a time? Key Y o
n
Please key in your list of 12 integers in the following format, whe
N,N,N,N,N,N,N,N,N,N,N,N
4,7,6,8,9,0,0,3,4,5,6,7
Original list of values given:
        4       7       6       8       9       0       0       3

Bubble sort returns:
        4       6       7       8       0       0       3       4
        4       6       7       0       0       3       4       5
        4       6       0       0       3       4       5       6
        4       0       0       3       4       5       6       6
        0       0       3       4       4       5       6       6
        0       0       3       4       4       5       6       6
        0       0       3       4       4       5       6       6

Merge sort returns:
        0       0       3       4                       6       6
```

Program updated □

B 7

Modify the bubbleSort function so that it passes the array and the to a function called `Swap()` which uses the parameters passed to it to re values correctly swapped around. The `bubbleSort()` function s accordingly to use this function.

Program updated □

B 8

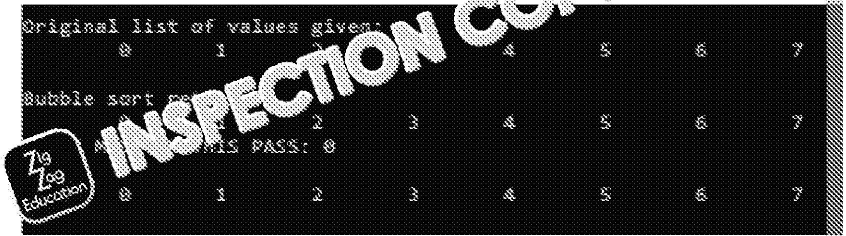
Thoroughly comment the entire `merge()` function to aid future prog

Program updated □

B 9

Modify the program's output to observe the behaviour of the `bubbl` `bubbleSort()` function should now include a `swaps` variable that are made on each pass. The program should display the value of `swa` performing the next pass.

Test it with a sorted list and it should produce output along the follo



```
Original list of values given:
        0       1       2       3       4       5       6       7

Bubble sort returns:
        0       1       2       3       4       5       6       7
SWAPS THIS PASS: 0

        0       1       2       3       4       5       6       7
```

Program updated □

Exercise 3 – Towers of Hanoi

Section A

A1 Give a line number from the `Game` class where an instance variable is

A2 What percentage of the methods in the `Tower` class are constructors?

A3 Discs can only be removed from or added to the end of a tower's arr

State the data structure which represents this behaviour and describe

of that data structure.

A4 The program requires less information when the `Game` class tries to in

`towerThree`

Explain how all three towers get successfully instantiated despite the f

ements.

A5 The program does not accept "ONE", "TWO" or "THREE" as valid inpu

Explain what would happen if such inputs were given and how C# wo

A6 A less robust program could crash if the player tried to move a disc fr

Explain how the `Move()` method handles this eventuality.

A 7 The OR operator is used to perform a check before a valid move occu

Explain the role of the OR operator in this case.

A 8 Explain the operation of the `RemoveDisc()` method of the `Tower` c

A 9 The program uses multiple classes for encapsulation.

State the meaning of encapsulation and why it is useful.

A 10 Outline the differences between an array and a list.

Exercise 3 – Towers of Hanoi

Section B

B 1

Modify the program to output an introductory line at the start of a n show the name of the game.

Program updated □

B 2

Modify the program so that it accepts "ONE", "TWO" and "THREE" as tower. Implement this so that it is robust enough to accept the letters i B2, e.g. it accepts "One", "ONE", "oNe" and "one".

Program updated □

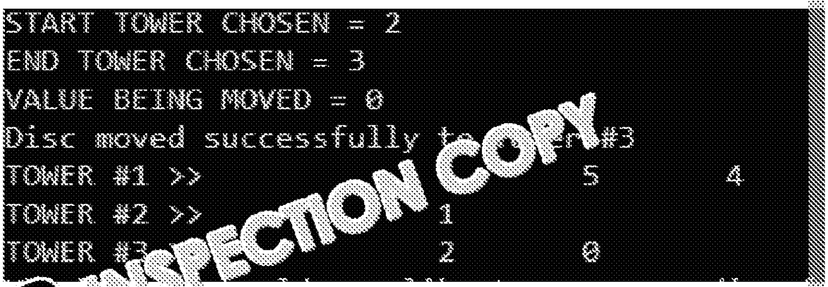
B 3

The instance variable `Number` in the `Tower` class currently has public mutator methods for this variable and set its visibility to private, mod program so that they call the relevant accessor/getter method instea

Program updated □

B 4

Modify the program so that it displays a simple visual representation each move is played.



```
START TOWER CHOSEN = 2
END TOWER CHOSEN = 3
VALUE BEING MOVED = 0
Disc moved successfully to tower#3
TOWER #1 >>                    5       4
TOWER #2 >>            1
TOWER #3 >>            2       0
```

Program updated □

B 5

Modify the program to add a `CheckWon` function in the `Game` class t successfully completed the game, or otherwise returns *False*. (*Hint:* T line of code inside a method body!). The `Main` procedure should be function to end the game once it has been won, and then displays th prints a message to tell the player that they have completed the gam have the `while True:` loop changed to use a more appropriate co as `while True` loops should be reserved for testing purposes or inc

Program updated □

B 6

The minimum number of moves needed to complete the game is 2^n – So a game with three discs can be completed in seven moves, a game wi moves, etc.

Modify the program to allow the user to choose the number of disc and the program should then output the minimum number of move inputs of the wrong data type and integers that are out of range. Int variable which will keep track of how many successful moves the use game, output a message congratulating them on achieving their of moves, or else encouraging them to try again to see if they can ma

Program updated □

Exercise 4 – Sorting Queues

Section A

A 1 How many private members (i.e. attributes and methods) are there in

....

A 2 Give a line number from the `Node` class that demonstrates parameter

....

A 3 A queue is one type of data structure; a stack is another type.

Explain the difference between a queue and a stack.

....

....

....

....

A 4 The method `IdentifyQueueTail()` is used to find the tail node in

State why this method must therefore be called as part of the `Enque`

....

....

A 5 The `PrintQueue()` procedure uses a FOR loop.

Explain the purpose of the FOR loop.

....

....

A 6 The `Node` constructor sets a new node's pointer to be null. Explain wh

....

....

A 7 The queue is implemented as a linked list.

Explain one advantage of using a linked list instead of an array to imp

....

....

....

....

A 8 Having read and understood the code, describe in words how to enq
that is implemented as a linked list.

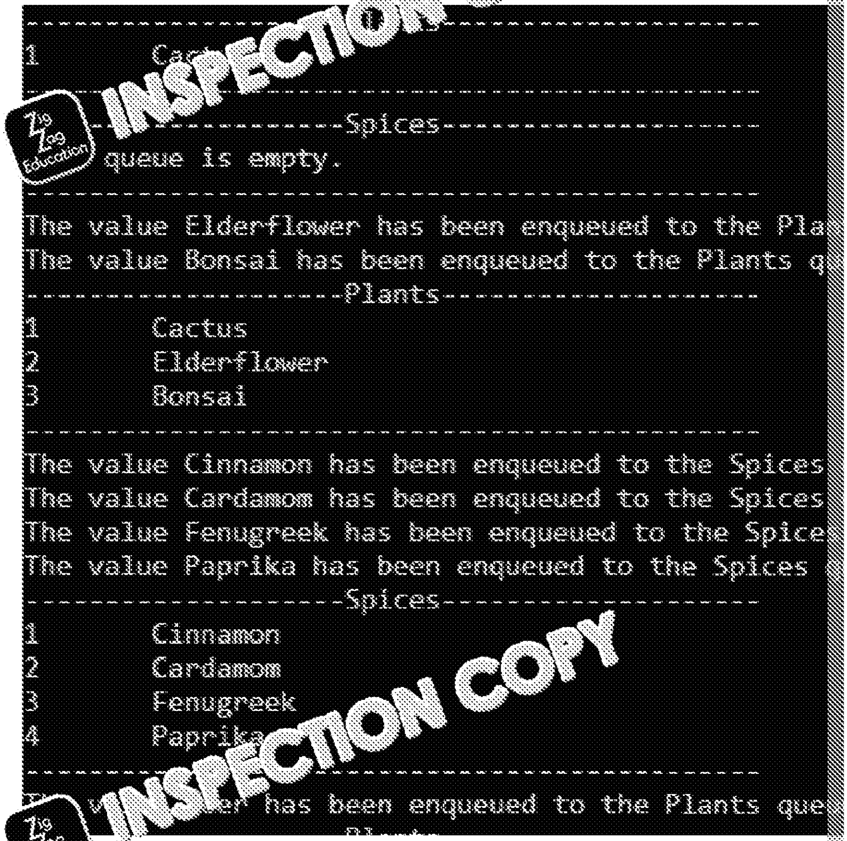
A 9 Queues can be implemented in different ways; for example, as a circu
Explain what a circular queue is.

A 10 Explain what type of queue is used in the program and how you can

Exercise 4 – Sorting Queues

Section B

B 1 Modify the program so that it outputs a meaningful message to the c enqueued (added to the tail of the queue) along the lines of the exam



```
--------------------------------------------
1       Cactus
--------------------------------------------
------------------Spices--------------------
The queue is empty.
--------------------------------------------
The value Elderflower has been enqueued to the Pla
The value Bonsai has been enqueued to the Plants q
------------------Plants--------------------
1       Cactus
2       Elderflower
3       Bonsai
--------------------------------------------
The value Cinnamon has been enqueued to the Spices
The value Cardamom has been enqueued to the Spices
The value Fenugreek has been enqueued to the Spice
The value Paprika has been enqueued to the Spices
------------------Spices--------------------
1       Cinnamon
2       Cardamom
3       Fenugreek
4       Paprika
--------------------------------------------
The value  er has been enqueued to the Plants que
```

Program updated □

B 2 Modify the program so that it outputs a meaningful message to the c dequeued (removed from the head of the queue). Be careful where yo

Program updated □

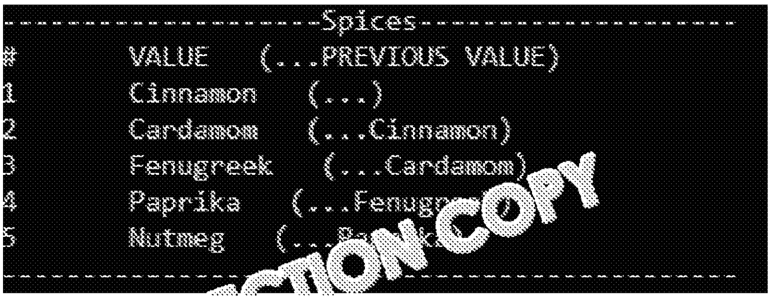
B 3 Add a method called `GetSize()` to the `Queue` class which returns th The method should work with empty queues (size = 0). Choose mean name and any variables used within it. Write code in `Program.cs` to on all newly added code.

Program updated □

B 4 A 'doubly linked' list (also known as a 'two-way' linked list) uses point and pointers that point to the previous value in the list. Modify the N `PreviousNode` attribute that indicates the node that comes before t `GetPrevious()` function and the `SetPrevious()` procedure s

The methods `Enqueue()` and `Dequeue()` should be amended acco new attribute. Finally, the constructor for a queue which takes a node amended so that it calls the `Enqueue()` method from now on. Suitab into `Queue.PrintQueue()` to ensure that it has succeeded.

Exemplar output is shown below.



```
--------------------Spices---------------------
#       VALUE   (...PREVIOUS VALUE)
1       Cinnamon   (...)
2       Cardamom   (...Cinnamon)
3       Fenugreek   (...Cardamom)
4       Paprika   (...Fenugreek)
5       Nutmeg   (...Paprika)

-----------------------------------------------
```

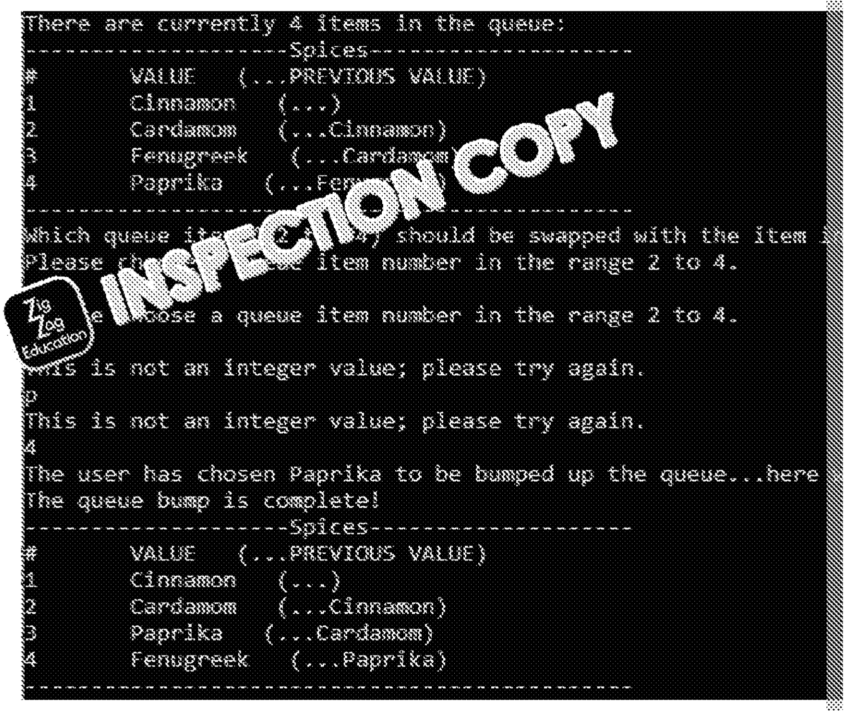
Program updated

B 5

Add a `GetNodeAt(n)` function to the `Queue` class which gets the n
indexing should NOT be used. Assume that the first node in the queu

Add a `Bump()` procedure to the `Queue` class.

When the function is called, as long as there are two or more items in
shown an on-screen printout of the queue and they should then be a
indicate a particular position in the queue (#1 represents the head of
head of the queue will be regarded as #1, the only valid inputs are in
have no previous node with which to swap) and ranging all the way u
queue. Suitable validation should be included to make this procedure



```
There are currently 4 items in the queue:
--------------------Spices--------------------
#       VALUE   (...PREVIOUS VALUE)
1       Cinnamon   (...)
2       Cardamom   (...Cinnamon)
3       Fenugreek   (...Cardamom)
4       Paprika   (...Fenugreek)
----------------------------------------------
Which queue item (2 to 4) should be swapped with the item i
Please choose a queue item number in the range 2 to 4.

Please choose a queue item number in the range 2 to 4.

This is not an integer value; please try again.
p
This is not an integer value; please try again.
4
The user has chosen Paprika to be bumped up the queue...here
The queue bump is complete!
--------------------Spices--------------------
#       VALUE   (...PREVIOUS VALUE)
1       Cinnamon   (...)
2       Cardamom   (...Cinnamon)
3       Paprika   (...Cardamom)
4       Fenugreek   (...Paprika)
----------------------------------------------
```

The program should then swap that node with the previous node in t
of 'bumping' the user's chosen node closer to the front of the queue.
`Program.cs` to ensure that the `Bump()` procedure is working effecti

Here is some structured hints to guide the latter half of this task on
been received:

- Iterate through the queue until the desired node is reached an
- Work through each of the three `Pointer` variables and modify th
  queue).
- Work through each of the three previous pointers and modify
  the tail towards the head of the queue).
- Output the newly sequenced queue.

Program updated ☐

B 6

Bubble sort is a sorting algorithm where values held in a linear data st
compared and potentially swapped with adjacent values. On each pas
value 'bubbles' to the head and takes its place as the maximum value
successive passes to ignore that node.

Having developed a swapping procedure, make a copy of it and name
and output statements can be deleted from Swap(). Modify Swap()
position of an item in the queue (as an integer) and the Swap() pro
value that is currently ahead of it in the queue.

Next, build a procedure called BubbleSort() that uses the Swap()
to bubble sort a queue held as a linked list into alphabetical order. Thi
from the tail of the queue and keep 'bubbling' the earlier values

The BubbleSort() procedure should return straight away if there a
queue.

Program updated □

Exercise 5 – Draughts

Section A

A 1 Give the class name and the line number from the program where a 2

....

A 2 Give a class name and line number from the program where a priva

is declared ....

gets initialised...

has its value read in full ....

A 3 Explain why the `PlacePieces` method is private instead of public.

....

....

....

....

A 4 Explain why the constructor of the `Piece` class requires one and only to it.

....

....

....

....

A 5 Which of the following is true?

1. 'Each piece stores its position on the board.'
2. 'The board stores the piece positioned on each square.'

Explain your answer.

....

....

....

....

A 6

Explain how the program generates the output shown below when it [illegible] methods involved and describe what they do, but you do not need to carry out their roles.

....

....

....

....

....

....

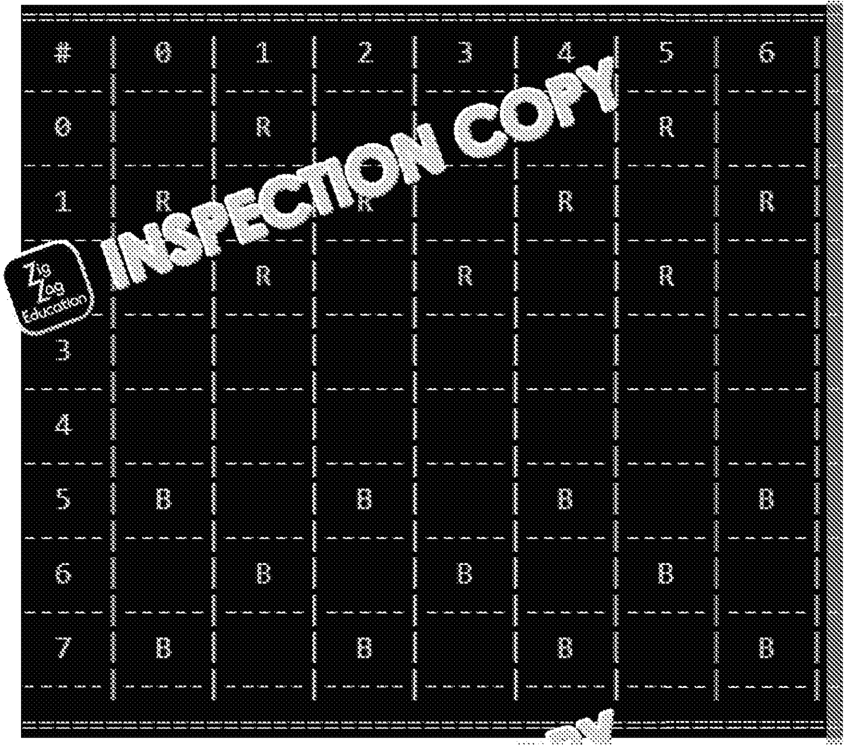
....

....

....

....

....



```
  #  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |
-----|-----|-----|-----|-----|-----|-----|-----|
  0  |     |  R  |     |  R  |     |  R  |     |
-----|-----|-----|-----|-----|-----|-----|-----|
  1  |  R  |     |  R  |     |  R  |     |  R  |
-----|-----|-----|-----|-----|-----|-----|-----|
  2  |     |  R  |     |  R  |     |  R  |     |
-----|-----|-----|-----|-----|-----|-----|-----|
  3  |     |     |     |     |     |     |     |
-----|-----|-----|-----|-----|-----|-----|-----|
  4  |     |     |     |     |     |     |     |
-----|-----|-----|-----|-----|-----|-----|-----|
  5  |  B  |     |  B  |     |  B  |     |  B  |
-----|-----|-----|-----|-----|-----|-----|-----|
  6  |     |  B  |     |  B  |     |  B  |     |
-----|-----|-----|-----|-----|-----|-----|-----|
  7  |  B  |     |  B  |     |  B  |     |  B  |
-----|-----|-----|-----|-----|-----|-----|-----|
```

A 7

The value *8* is hard-coded int[illegible] Bo[illegible]d class to represent the size [illegible] Explain why this is [illegible] bad practice and what should be used [illegible]

....

....

....

....

A 8 Explain the use of the MOD operator in placing the pieces on the boa

A 9 A new `King` class could be created that inherits the `Piece` class.

Explain what inheritance is and why it is useful.

A 10 Explain the difference between functions, procedures and methods.

A 11 A `Board` object is created to represent one game's board.

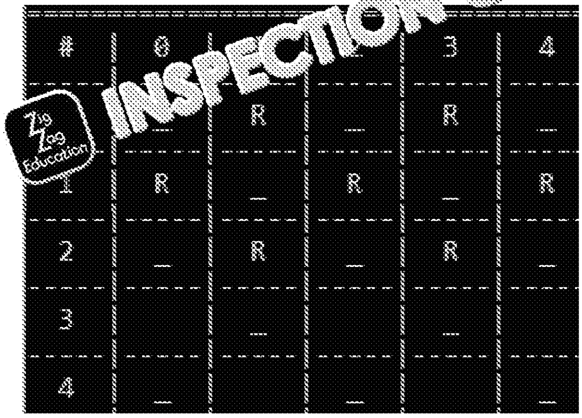
Explain the difference between an object and a class in this case.

Exercise 5 – Draughts

Section B

B 1

Modify the program so that the white squares are indicated by an un
is displayed.



```
  #  |  0  |  1  |  2  |  3  |  4
-----|-----|-----|-----|-----|-----
  0  |  _  |  R  |  _  |  R  |  _
-----|-----|-----|-----|-----|-----
  1  |  R  |  _  |  R  |  _  |  R
-----|-----|-----|-----|-----|-----
  2  |  _  |  R  |  _  |  R  |  _
-----|-----|-----|-----|-----|-----
  3  |     |  _  |     |  _  |
-----|-----|-----|-----|-----|-----
  4  |  _  |     |  _  |     |  _
```

Program updated ☐

B 2

Modify the program to add a `BoardSize` attribute to the `Board` cla
should be set to *8* in the constructor, and the `BoardSize` attribute s
hard-coded values throughout the `Board` class.

Program updated ☐

B 3

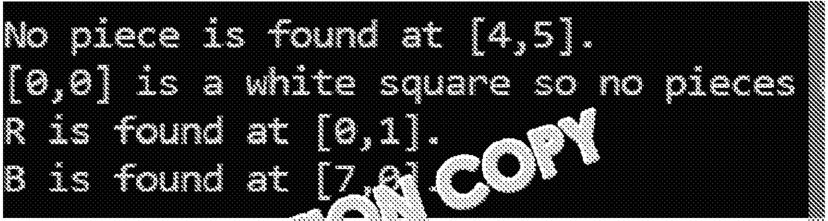
Add an accessor method for the `BoardSize` attribute.

Program updated ☐

B 4

Modify the program to add a public `PieceAt` function into the `Boa`
integers (a row and a column from the board) as input and returns th
board. Zero-based indexing should be used. Validate the user's input
range, and return a *Null* value if either is out of range.

Add a procedure called `DisplayPieceAt` into the `Board` class. It sh
and return the character representation of the piece found there, trea
differently from all white squares. Add code to the `Program` class to



```
No piece is found at [4,5].
[0,0] is a white square so no pieces
R is found at [0,1].
B is found at [7,0].
```

Program updated ☐

B 5

Add a private integer attribute called `TurnNumber` to the `Board` cla
moves made by the players. When whoever is playing as black (B) ma
of turns should be incremented from 1 to 2. Amend the constructor a
method and a mutator method that simply increments it.

Modify the program to add a `ValidMove` function in the `Board` clas
(representing a start position and an end position on the board) as in
of the given colour can move a piece from the given start position to
move. The ability to do repetitive jumps in a game of draughts can be
whether the correct colour of piece has been chosen by the player wi

The rules of single movement which must now be programmed are
- Pieces can move in a straight diagonal line either one space
  spaces (if the tile one space diagonally on from the start pos
  player's piece and the end position is empty).
- Non-king pieces can move only towards the opponent's side
- King pieces can move in any diagonal direction.
- Pieces cannot move to a position that is not on the board.

Here is a pseudocode structure you can follow:

```
Store the PieceToBeMoved as a Piece variable
Return False if PieceToBeMoved isn't an actual Piece
Return False if the destination is off the board

IF PieceToBeMoved is not a king:
    •   IF it's Black's move:
            o   IF Black is moving only 1 row 'forwards' AND 1 colum
                    ▪   IF the destination is empty:
                            •   ALLOW
                    ▪   ELSE
                            •   DISALLOW
            o   ELSE IF Black is moving 2 rows away:
                    ▪   IF Black is moving 2 columns left or right:
                            •   IF the destination is empty:
                                    o   IF there is a Red in between:
                                            ▪   ALLOW
                                    o   ELSE
                                            ▪   DISALLOW
                            •   ELSE
                                    o   DISALLOW
                    ▪   ELSE
                            •   DISALLOW
            o   ELSE
                    ▪   DISALLOW
    •   ELSE
            o   ... as above but reinterpreting the idea of 'forwards' and t
ELSE:
    •   ... as ALL the above but with more freedom of movement be
```

You can test your work by including this code in the `Program` class a
`TurnNumber` is set to *1*:

```
// B5
Console.WriteLine("\n\n>>>>>>>> TESTING <<<<<<<<");
Console.WriteLine("These should all be FALSE regardless of wh
// Invalid piece
Console.WriteLine(GameBoard.ValidMove(-3, 0, 5, 1));
Console.WriteLine(GameBoard.ValidMove(0, -3, 5, 1));
// Invalid destination
Console.WriteLine(GameBoard.ValidMove(0, 5, -3, 1));
Console.WriteLine(GameBoard.ValidMove(0, 5, 1, -3));

Console.WriteLine("\nBLACK'S TESTING = TURN NUMBER = 1");
Console.WriteLine("\nThese should all be TRUE when it is BLAC
Console.WriteLine(GameBoard.ValidMove(5, 0, 4, 1));
Console.WriteLine(GameBoard.ValidMove(5, 4, 4, 3));
Console.WriteLine(GameBoard.ValidMove(5, 6, 4, 7));
Console.WriteLine("\nThese should all be FALSE and produce er
// Move to an occupied square
Console.WriteLine(GameBoard.ValidMove(6, 7, 5, 6));
// Not a diagonal move
Console.WriteLine(GameBoard.ValidMove(5, 0, 4, 0));
// Off the board
Console.WriteLine(GameBoard.ValidMove(5, 0, 5, -1));
// Black jumping black into a vacant square
Console.WriteLine(GameBoard.ValidMove(6, 1, 4, 3));
// Black jumping black into an occupied square
Console.WriteLine(GameBoard.ValidMove(7, 0, 5, 2));

GameBoard.UpdateTurnNumber(); // switch to RED's turn

Console.WriteLine("\nRED'S TESTING = TURN NUMBER = 2");
Console.WriteLine("\nThese should all be TRUE when it is RED's
Console.WriteLine(GameBoard.ValidMove(2, 1, 3, 0));
Console.WriteLine(GameBoard.ValidMove(2, 3, 3, 2));
Console.WriteLine(GameBoard.ValidMove(2, 7, 3, 6));
Console.WriteLine("\nThese should all be FALSE and produce er
// Move to an occupied square
Console.WriteLine(GameBoard.ValidMove(1, 0, 2, 1));
// Not a diagonal move
Console.WriteLine(GameBoard.ValidMove(1, 2, 2, 2));
// Off the board
Console.WriteLine(GameBoard.ValidMove(0, 7, 1, 8));
// Red jumping red into a vacant square
Console.WriteLine(GameBoard.ValidMove(1, 0, 3, 2));
// Red jumping red into an occupied square
Console.WriteLine(GameBoard.ValidMove(0, 1, 2, 3));
Console.WriteLine(">>>>>>>> END OF TESTING <<<<<<<<\n\n");
```

Program updated □

B 6 Modify the program to include a `ValidColour()` method which ta
as parameters and returns *True* if the user lifts their correct colour of
an incorrect colour. To achieve this, firstly check if the playing piece c
proceed to check the colour against the turn number.

Program updated □

B 7

Modify the program to add a GetMove procedure that asks the user at a time) and then checks if the playing piece lifted by the user is of (using the turn number that the game is currently at). If the colour is proceed to read in an end position (one integer at a time) and check given by calling the ValidMove method.

Assuming all checks pass, the move should be updated accordingly a amended accordingly. If ValidMove fails, the user should be told "I the colour test fails, the user should be told:
"The board has not been changed. Try again; it is s

All four integers read in from the user should be fully validated using [illegible] should be added to the Program class so that six moves can be some invalid.

Program updated □

B 8

Modify the program to add a CheckWon function that returns the pl as a string (if the game has been won) or returns an empty string if n support the development of this method, add two instance variables pieces of each colour that have been removed from the board.

Add lines of code to the GetMove method so that when a piece is ju board.

The main method of the program should be modified to run in a loop from each player until one of the players has won, displaying the star At the end of the game, a message should be displayed to say which

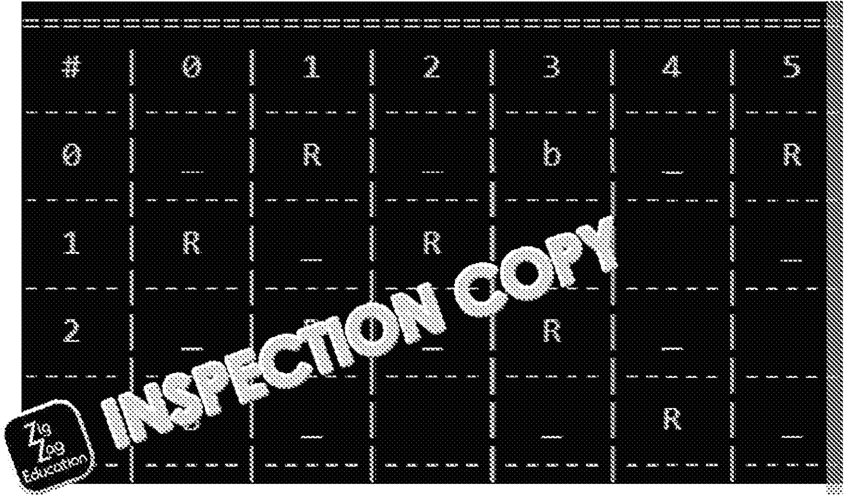
Program updated □

B 9

Modify the GetMove method so that when a piece reaches the oppo promoted to a King. Write the code for awarding King status for Red still work if the board size was increased or decreased.

Amend the Piece.cs class so that when a piece is kinged, its letter this approach so that if the kinged piece re-enters the far row, it will any way:

```
("" + Colour).ToLower()[0]
```



```
==========================================
  #  |  0  |  1  |  2  |  3  |  4  |  5
-----|-----|-----|-----|-----|-----|-----
  0  |  _  |  R  |  _  |  b  |  _  |  R
-----|-----|-----|-----|-----|-----|-----
  1  |  R  |  _  |  R  |     |     |  _
-----|-----|-----|-----|-----|-----|-----
  2  |  _  |     |  _  |  R  |  _  |
-----|-----|-----|-----|-----|-----|-----
     |     |  _  |     |  _  |  R  |  _
-----|-----|-----|-----|-----|-----|-----
```

Program updated □

Exercise 6 – Tree Traversal

Section A

A 1 Give a class name and line number from the program that contains a

call to a constructor ....

private attribute declaration ....

[illegible] a null pointer ....

A 2 Give the class name and line number from the program that shows t

....

A 3 Draw the tree that is created by the program. You only have to show

A 4 The tree created by the program is a binary tree.

Explain the difference between a binary tree and a multi-branch tree.

....

....

....

....

A 5 The program uses encapsulation. Explain with reference to the Node c

....

....

....

....

A 6 The program uses just four data types. Name all four of them.

1 ....

2 ....

3 ....

4 ....

A 7 Write a line of code that would attempt to display the value found at [illegible] as [illegible] would not work.

....

....

A 8 Explain how C# would handle the error that would occur in Task A7 a

....

....

....

....

A 9 Write the tree values as they would be returned in a depth-first, post

....

A 10 Write the tree values as they would be returned in a depth-first, pre-o

....

A 11 Write the tree values as they would be returned in a depth-first, in-or

....

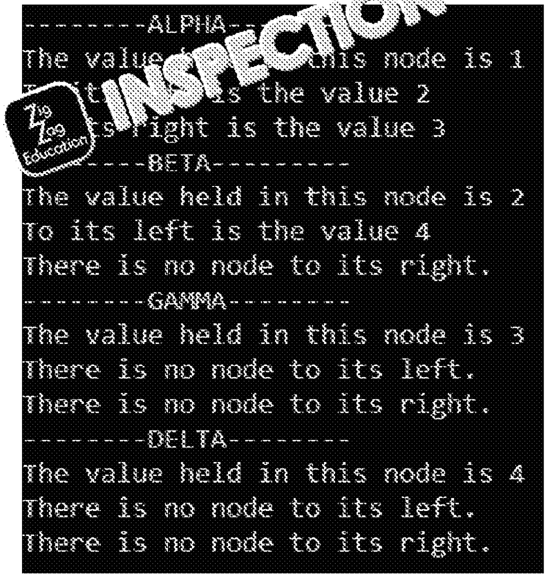
A 12 Write the tree values as they would be returned in a breadth-first tree

....

Exercise 6 – Tree Traversal

Section B

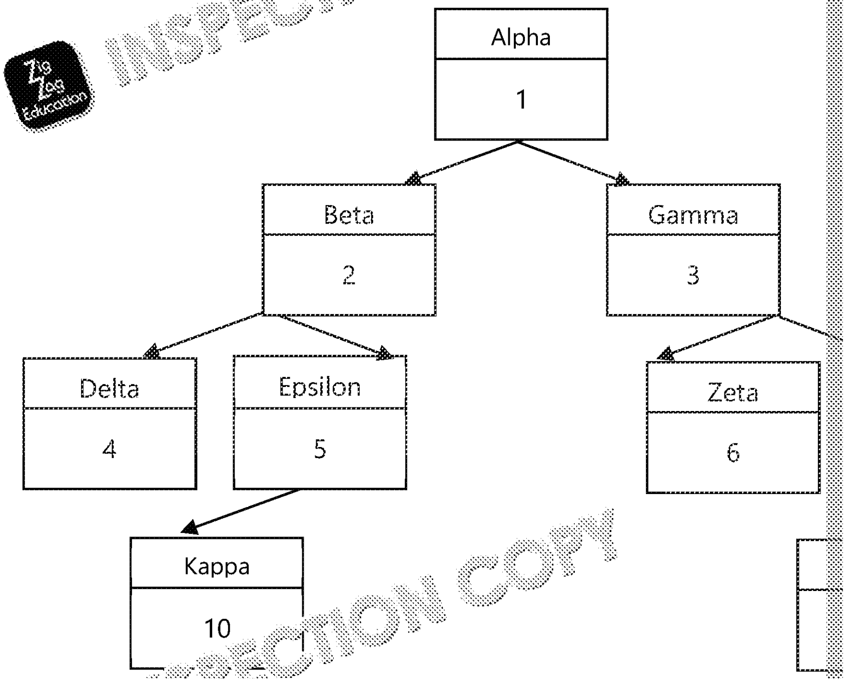
B 1 Modify `Node.cs` so that it contains a procedure that can print a full s
node, as per the example output below. Insert test code into `Progra`
all four existing nodes of `GreekTree`.



```
--------ALPHA--------
The value held in this node is 1
To its left is the value 2
To its right is the value 3
--------BETA---------
The value held in this node is 2
To its left is the value 4
There is no node to its right.
--------GAMMA--------
The value held in this node is 3
There is no node to its left.
There is no node to its right.
--------DELTA--------
The value held in this node is 4
There is no node to its left.
There is no node to its right.
```

Program updated ☐

B 2 Modify `Program.cs` so that `GreekTree` represents this binary tree
end of any existing code from task B1.



Program updated ☐

B 3 Modify the program so that it creates a new tree called `AssortedTr`
called Gold and have the value *24*, and there should be only a right n
whose value is *8*. Add a line of code to display the full set of informati

Program updated ☐

B 4

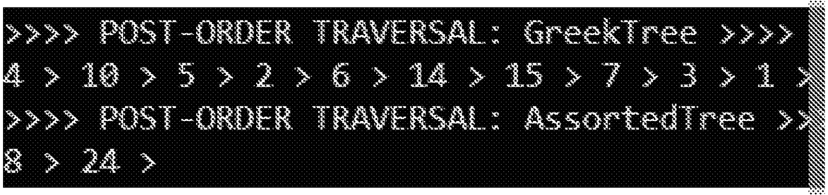
Modify Program.cs to add a recursive PostOrderTraversal pro
input and performs a depth-first, post-order tree traversal from that
separated by > symbols as it progresses.

To achieve this, here is a suitable algorithm for the body of the proc

If NOT Null:

- *Traverse the Left subtree via a recursive call that passes in*
- *Traverse the Right subtree via a recursive call that passes*
  node
- *Output the Value followed by a > symbol without taking a*

The Main procedure should be modified to test the procedure twice
PostOrderTraversal procedure using the Root value from both
the program. Each value should be displayed in the order in which it
and this should be manually checked for correctness.



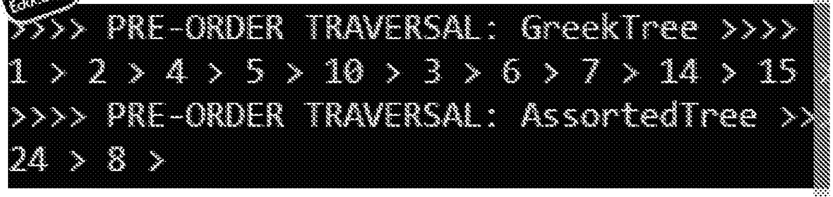
```
>>>> POST-ORDER TRAVERSAL: GreekTree >>>>
4 > 10 > 5 > 2 > 6 > 14 > 15 > 7 > 3 > 1 >
>>>> POST-ORDER TRAVERSAL: AssortedTree >>
8 > 24 >
```

Program updated □

B 5

Modify the program to add a recursive PreOrderTraversal proce
input and performs a depth-first pre-order tree traversal from that ro
separated by > symbols as it progresses.

The Main procedure should be modified to test the procedure twice
PreOrderTraversal procedure using the Root value from both o
program. Each value should be displayed in the order in which it is c
this should be manually checked for correctness.



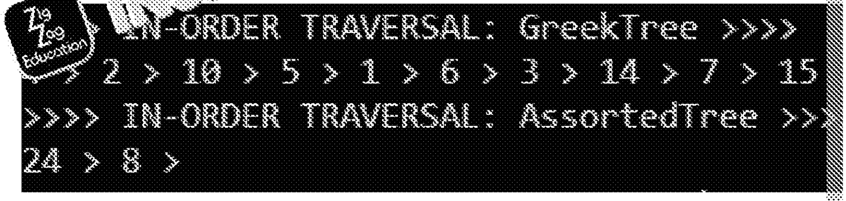
```
>>>> PRE-ORDER TRAVERSAL: GreekTree >>>>
1 > 2 > 4 > 5 > 10 > 3 > 6 > 7 > 14 > 15
>>>> PRE-ORDER TRAVERSAL: AssortedTree >>
24 > 8 >
```

Program updated □

B 6

Modify the program to add a recursive InOrderTraversal proced
input and performs a depth-first, in-order tree traversal from that ro
separated by > symbols as it progresses.

The Main procedure should be modified to test the procedure twice
InOrderTraversal procedure using the Root value from both of
program. Each value should be displayed in the order in which it is c
this should be manually checked for correctness.



```
>>>> IN-ORDER TRAVERSAL: GreekTree >>>>
4 > 2 > 10 > 5 > 1 > 6 > 3 > 14 > 7 > 15
>>>> IN-ORDER TRAVERSAL: AssortedTree >>
24 > 8 >
```

Program updated □

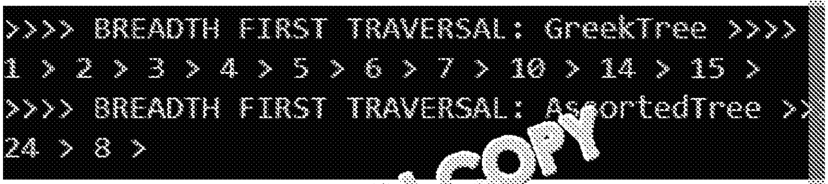
B 7

Modify the program to add a `BreadthFirstTraversal` procedur and performs a breadth-first tree traversal from that root node, outp symbols as it progresses.

Here is an algorithm for guidance:

- *Create two lists of Node objects: one for the current level and one f*
- *Handle empty trees to improve this method's robustness*
    - *Output a message*
    - *Return/Exit*
- *Initialise the list of nodes at this level with the root of the tree pass*
- *While a level with no nodes has not yet been encountered...*
    - *Form a new empty list for the next level's nodes*
    - *Visit all the nodes in the list*
        - *Output the value found at each node*
        - *If the node has a left/right node further down the t*
    - *Having visited all this level's nodes, set the next level's list t*

The `Main` procedure should be modified to test the procedure twice `BreadthFirstTraversal` procedure using the `Root` value from b in the program. Each value should be displayed in the order in which and this should be manually checked for correctness.



```
>>>> BREADTH FIRST TRAVERSAL: GreekTree >>>>
1 > 2 > 3 > 4 > 5 > 6 > 7 > 10 > 14 > 15 >
>>>> BREADTH FIRST TRAVERSAL: AssortedTree >>
24 > 8 >
```

Program updated □

B 8

A special case of a binary tree is called a BINARY SEARCH TREE. It has

- It is a binary tree.
- When any node is chosen, its left subtree contains only value
- When any node is chosen, its right subtree contains only valu

Modify the `Program` class by adding a recursive `CreateBinarySea` a sorted array of integers and create a binary search tree from it whic object (its root node). For robustness, any null or zero-length arrays s Here is an algorithm to guide the development process:

- *Set up a method that returns a Node and takes in 3 paramete boundary and a pointer to the upper boundary of the array. Th the recursive calls, hence the need for 3 parameters.*
- *If any of the following 3 cases arise, return Null immediately:*
    - *The array supplied is a null pointer*
    - *The array supplied has 0 values in it*
    - *The lower boundary is greater than the upper bounda*
- *Calculate and store the index of the midpoint of the array*
- *Determine the value held at the midpoint of the array*
- *Declare and initialise a new node using the value just found a*
- *Set the left pointer of this new node by recursively calling the m portion of the sorted array (by setting the parameters to descri left of the mid-value)*

- *Set the right pointer of this new node by recursively calling the portion of the sorted array (by setting the parameters to descri right of the mid-value)*
- *Return the newly created node*

To make it easier to use this recursive method, overload the method w require the programmer to supply the lower and upper boundaries w Main method. This new method will then generate the relevant point defined method that was just built if either of the following two cases

- The array supplied is a null pointer
- The array supplied has 0 values in it

The Main procedure should be modified to create the array Number {1,2,3,4,5,6,7,8}. Construct a Tree object using this array and traversals on this tree.

Program updated □

B 9

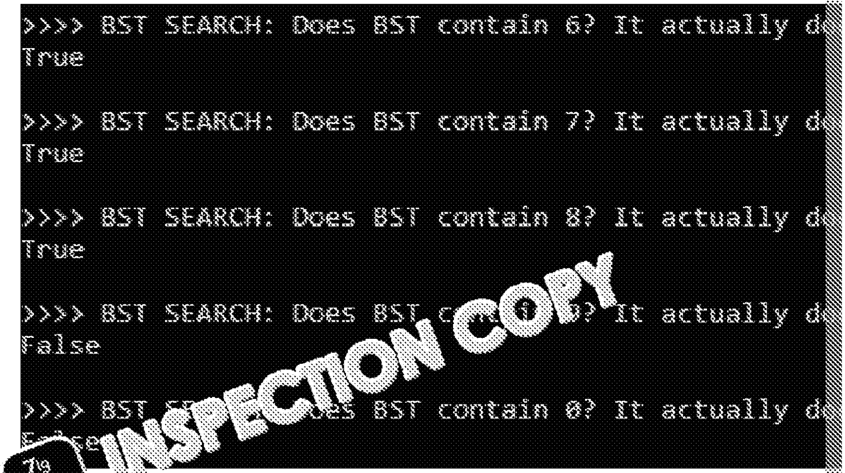
Modify the program to add a SearchBST function in the Program c search tree's root node and an integer value as parameters. The meth the relevant subtrees emanating from the root node until it can disce not found in the tree. It should return a single Boolean result.

Note: This method cannot be used with unsorted trees such as Gre

Here is an algorithm to aid the development of this method:

- *If the node is a null pointer, return False*
- *If the sought value is found at that node, return True*
- *If the sought value is less than the value found at that node, s by performing a recursive call*
- *If the sought value is greater than the value found at that nod node by performing a recursive call*

The Main procedure should be modified to call the SearchBST func created from the list {1,2,3,5,6,7,8} to search for the values 6, 7, displayed. An example is shown below as a guideline only.



```
>>>> BST SEARCH: Does BST contain 6? It actually d
True

>>>> BST SEARCH: Does BST contain 7? It actually d
True

>>>> BST SEARCH: Does BST contain 8? It actually d
True

>>>> BST SEARCH: Does BST contain [illegible]? It actually d
False

>>>> BST SEARCH: Does BST contain 0? It actually d
False
```

Program updated □

B 10

Here are the lines of code required to traverse an unordered tree usi unsorted list of values found. Put the lines of code into the correct or `Program.cs` as a new method.

A	`InOrderListBuilder(SubtreeRoot.GetLeft(), CurrentList`
B	`}`
C	`return CurrentList;`

D	`public static List<int> InOrderListBuilder(Node Subtr`
	`CurrentList)`

E	`InOrderListBuilder(SubtreeRoot.GetRight(), CurrentLis`
F	`}`
G	`{`
H	`if (SubtreeRoot != null)`
I	`{`
J	`CurrentList.Add(SubtreeRoot.GetValue());`

Now that a function exists that can convert an unsorted tree into a li `Sort()` method for sorting lists in C#.

The following method takes a list, sorts it and constructs a binary sea converting an unordered binary tree into a binary search tree. As bef into the correct order, adding the method to `Program.cs` so that it

A	`// [2] Sort the list`
B	`return NewBST;`
C	`// [1] Turn the tree into a list using in-order traver`
D	`ListFormat.Sort();`
E	`Node NewTreeRoot = ConstructBinarySearchTree(ListForm`
F	`{`
G	`ListFormat = InOrderListBuilder(RootOfUnsortedTree, Li`
H	`}`
I	`public static Tree ConvertToBST(Node RootOfUnsortedTr`
J	`Tree NewBST = new Tree(NewTreeRoot);`
K	`List<int> ListFormat = new List<int>();`
L	`// [3] Convert it into a BST`

Program updated □

B 11

Modify the program by making a copy of the `BreadthFirstTrave` `AddNode` in the `Program` class. This version of the method should ta parameters and add that node to that tree in the first available slot us approach. It should return a `Tree`.

The main program should be modified to add some randomly chose `AssortedTree` using this new method, and then create a copy of th a binary search tree called `AssortedBST`.

Finally, the program should perform all four traversals on the `Assor` apparent that performing in-order traversal on a BST produces a sor

Program updated □

Exercise 7 – Dijkstra's Shortest Path

Section A

A 1 On how many lines of `Program.cs` are other classes' constructors ca

A 2 Give a line number of `Graph.cs` that uses all three of the Boolean op

A 3 The attributes in the `Graph` class are private.

Define what a public attribute is and explain why an attribute may be m

A 4 The code contains very few comments, and the purpose of some of th be immediately clear to anyone who sees it.

Explain what is happening on `Program.cs` line 30.

A 5 Explain what is happening on `Graph.cs` lines 16–19.

A 6

The 'closest node' is the node that you can advance to directly in one at. If a node is at a 'dead end' and no nodes emanate from it, there wi

Using the comments given and by looking in detail at the lines of code GetClosestNode in Graph.cs works.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

A 7

Sketch a UML class diagram for Edge.cs.

A 8 The program defines a graph data structure. A tree is a specific type o

Explain what a graph data structure is and what the features of a tree

....

....

....

....

A 9 The GetClosestNode function can be used as part of an implementati hm.

State the purpose of Dijkstra's shortest path algorithm.

....

....

A 10 Describe how Dijkstra's shortest path algorithm works.

....

....

....

....

....

....

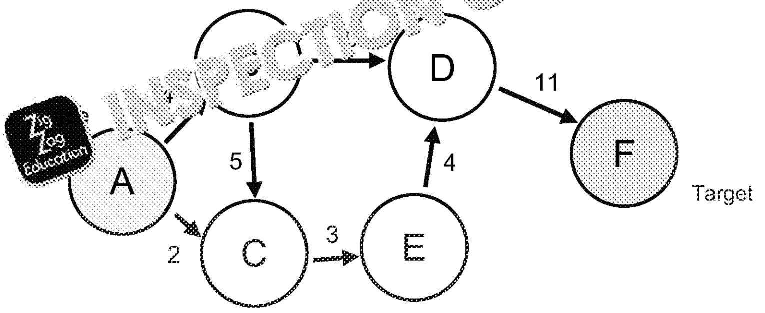
....

....

Exercise 7 – Dijkstra's Shortest Path

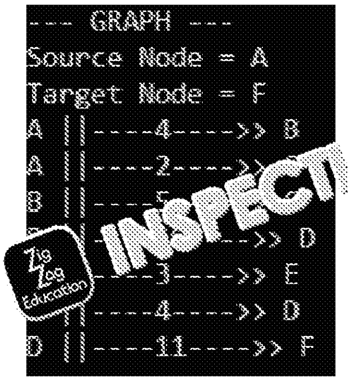
Section B

B 1 Modify the program by commenting out the existing graph creation c
replacing it with code which will model the graph shown here:



Program updated □

B 2 Modify the program so that when it is run, the graph above is visualise



```
--- GRAPH ---
Source Node = A
Target Node = F
A ||----4---->> B
A ||----2---->>
B ||----5
      ---->> D
   ----3---->> E
   ----4---->> D
D ||----11---->> F
```

Program updated □

B 3 Modify the program to output the closest node that comes after each
was used in the given code from Section A. This should expose a bug
end' nodes, which have no nodes emanating from them. Diagnose th

To debug the problem, add a new private method to `Program.cs` c
which receives two parameters (the graph and the node being investi
cases such as node F.

Finally, replace the code you have just added so that it uses the `Outp`
instead to output the closest node to each node.

Program updated

B 4

When manually performing Dijkstra's shortest path algorithm, a table

- the name of each node
- whether that node has been visited (i.e. all edges emanating
- the length of the shortest known path from that node back to
- the node that sits prior to that node along the shortest path

Before advancing to making further modifications to the program, try Dijkstra's algorithm on paper for both directed graphs shown previous

Section A		
Node	**Visited?**	**Shortest Distance to Start Node**
A		**0**
B		∞
C		∞
D		∞
E		∞
F		∞
G		∞
H		∞
I		∞

Section B		
Node	**Visited?**	**Shortest Distance to Start Node**
A		**0**
B		∞
C		∞
D		∞
E		∞
F		∞

Section A: Shortest path from A to I = ______________________

Section B: Shortest path from A to F = ______________________

Modify the program to model the table structure as a list of objects th class to perform Dijkstra's algorithm:

1) Build a new class called `TableRow` which contains four attrib columns.
2) Add accessor and mutator methods for all four attributes in t
3) In the `Graph` class, add a new private [illegible]`tance` method ca returns the current graph described as a blank table (the retu objects). It should be implemented by following the algorithm

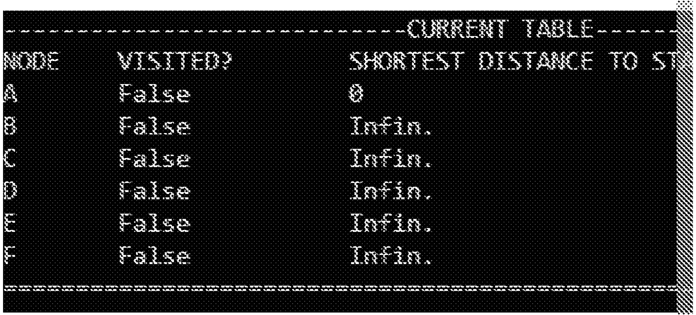
- *Create an empty list of nodes called TableNodes*
- *Create a node variable called NodeMightBeAdded to temp edges*
- *For all edges in the graph:*
    - *If the node at the start of that edge is not already i*
    - *If the node at the end of that edge is not already i*
- *Make a new empty list of TableRow objects*

- *For all nodes in TableNodes:*
    - *Create a new TableRow object by passing the node*
    - *Add it to the list of TableRow objects*
    - *If the node is also the SourceNode for this graph, se this, use the fact that it will currently be the last ele been added, and bear in mind that SourceNode is a object)*
- *Return the list of TableRow objects*

4) Add a new instance variable to `Graph.cs` called `Dijkstra` type `List<TableRow>`. Amend the existing constructors and the last line of each method, after something has changed it method to update the instance variable `DijkstraTable`. Am `AddEdge()` too so that the table version of the digraph is al edge is added. Finally, add a public accessor method to provi `DijkstraTable` variable.

5) Add a public procedure called `PrintTable` to the `Graph` cl console. It should work robustly whether Dijkstra's algorithm underway or is fully complete. This will involve watching out f \u00D8) and unusual values (e.g. `int.MaxValue`) when outp `PrintTable` using the following pseudocode, then add one test that it works:

- *Output suitable table column headings and an overarching*
- *For each row of the table:*
    - *If the row node is null, output an underscore; other*
    - *Output "True"/"False" for whether the node has bee*
    - *If the shortest distance is int.MaxValue, output "Inf*
    - *If the previous node is null, output the Null symbol*
    - *Make a new line*
- *Output a footer of continuous equal signs to end the table*



```
---------------------------CURRENT TABLE-----
NODE    VISITED?       SHORTEST DISTANCE TO ST
A       False          0
B       False          Infin.
C       False          Infin.
D       False          Infin.
E       False          Infin.
F       False          Infin.
==============================================
```

Program updated □

B 5

When iteratively processing the table rows during Dijkstra's algorithm stopping condition is that when you look at all of the rows for unvisit target/destination node is the one with the minimum distance from t

Modify the `Graph` class to add a public function called `TargetNode` function returns a Boolean value to indicate whether the target node

Here is a pseudocode outline of how this should be implemented:

- *Declare and initialise an integer variable called MinimumDista distance (of the unvisited nodes) in the table and set its value*
- *Declare an integer variable called CurrentDistance for tempora value*
- *Declare an integer variable called NotedPositionNumber to re where the TargetNode is found. Initialise this to -1 to help iden*
- *FOR all table rows in DijkstraTable (tip: use a FOR loop):*
    - *If it is an unvisited node:*
        - *Overwrite CurrentDistance with that node's c source node*
        - *If the CurrentDistance < MinimumDistanceIn MinimumDistanceInTable to hold the value o*
    - *If it is the target node:*
        - *Record the position in the list that you are cu*
- *IF NotedPositionNumber is still -1, output an error message a*
- *ELSE IF the target node's shortest distance to the source/start n MinimumDistanceInTable value, return "True"*
- *ELSE return "False"*

Test the new method by adding the necessary code to `Program.cs` and m in `Program.cs` so that the source and target nodes are the same.

Item completed ☐

B 6

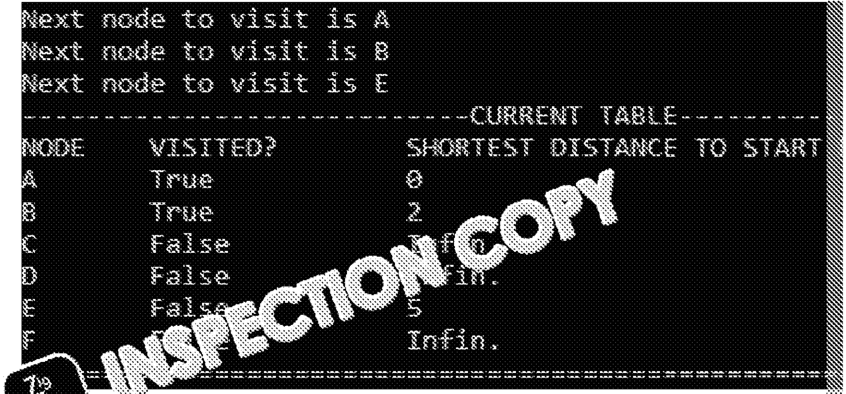
Each time a new row of the table is to be inspected, it has to be chos unvisited nodes. The row with the shortest distance value in column 3 node to inspect. Modify the `Graph` class so that it has a new method `GetNextUnvisitedNode()` which returns the node held at the tab will be zero-based) containing the unvisited node with the shortest d ignored.

Here is some test code and the expected outcomes that can be used

`Program.cs` should have the following added to it:

```
// B6
Console.WriteLine("Next node to visit is " + Map.GetNext
Map.GetDijkstraTable()[1].SetShortestDistanceToStart(2);
Map.GetDijkstraTable()[5].SetVisited();
Console.WriteLine("Next node to visit is " + Map.GetNext
Map.GetDijkstraTable()[4].SetShortestDistanceToStart(5);
Map.GetDijkstraTable()[1].SetVisited();
Console.WriteLine("Next node to visit is " + Map.GetNext
Map.PrintTable();
```

Expected outcome:



```
Next node to visit is A
Next node to visit is B
Next node to visit is E
---------------------------CURRENT TABLE---------
NODE    VISITED?         SHORTEST DISTANCE TO START
A       True             0
B       True             2
C       False            Infin.
D       False            Infin.
E       False            5
F       False            Infin.
```

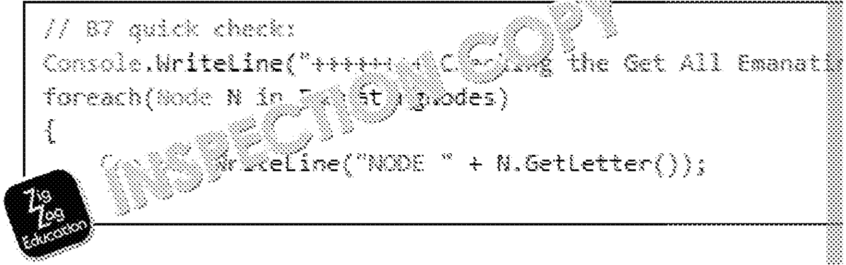
Program updated □

B 7 Modify the program by adding a method called `GetAllEmanating` in a node and consults the table and the diagram to identify any unv (i.e. come after) that node in the diagram.

It should return a list of suitable nodes.

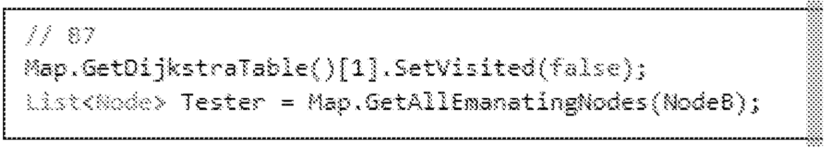
This task can be achieved by looking for existing similar code in the

Immediately before the return statement, insert a block of code to o that are in the list that is about to be returned:

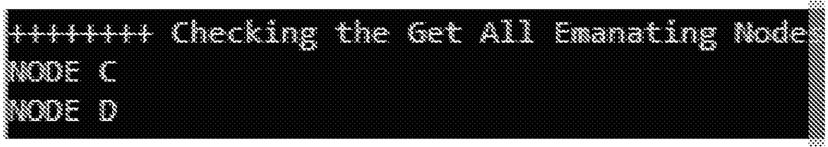


```
// B7 quick check:
Console.WriteLine("++++++++ Checking the Get All Emanati
foreach(Node N in T[illegible]odes)
{
    Console.WriteLine("NODE " + N.GetLetter());
```

Here is some test code that can thus be added to `Program.cs`, as w



```
// B7
Map.GetDijkstraTable()[1].SetVisited(false);
List<Node> Tester = Map.GetAllEmanatingNodes(NodeB);
```



```
++++++++ Checking the Get All Emanating Node
NODE C
NODE D
```

Notice that the 'visited' property of NodeB is set to *false* before testi *true* for testing purposes.

Program updated □

B 8 Modify the program by adding a `ConvertNodeToRowNumber` func [illegible]s the row index where it exists in the `DijkstraTable` list.

Program updated □

B 9

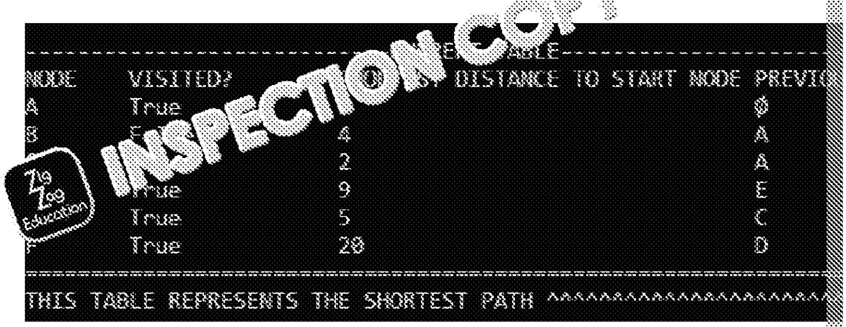
Modify the program by adding a `GetShortestPath` function that u algorithm to construct and output a table describing the shortest pa node of that digraph. (NB. The table will thus also describe the short the same starting node, but the algorithm will terminate based on th

Here is the algorithm that should be implemented:

```
•  Declare the following 5 variables:
      o  A list of Node objects called StillNotVisited
      o  Integers called NewDistanceFromSource and TableRo
      o  Set TableRowIndex = -1
      o  A Node called Current which should be initialised by
         GetNextUnvisitedNode() method
      o  A Boolean called TargetNodeNotVisited, initialised to
•  WHILE the target node doesn't hold the shortest distance AND
      o  TableRowIndex = the row index of Current (using an
      o  IF TableRowIndex== -1, output an error string and re
      o  Populate StillNotVisited by passing Current to an exis
      o  Initialise a list of Edge objects called RelevantEdges b
      o  FOREACH Edge in Diagram:
            ▪  If Current == the StartNode of that Edge:  Ad
      o  END OF FOREACH LOOP
      o  Declare int FirstLegShortDistance and retrieve the dis
         TableRowIndex points to in the table, storing it here
      o  Declare int LastLegDistance (to be initialised later)
      o  Declare int RowInTableGettingUpdated
      o  FOREACH Edge in RelevantEdges:
            ▪  Obtain the LastLegDistance from the Edge
            ▪  Update the value of NewDistanceFromSource
            ▪  Use the EndNode of that Edge to work out the
               RowInTableGettingUpdated (call an existing
            ▪  IF the distance in the table is larger than the
                  •  Update the distance in the table
                  •  Update the "previous node" column
      o  END OF FOREACH LOOP
      o  Set the Visited? property of that row of the table to Tr
      o  IF the Current node is actually the TargetNode, set Ta
      o  Update Current to the closest node to Current (use an
      o  Print the table for testing purposes with some blank l
•  END OF WHILE LOOP
•  RETURN "This table represents the shortest path ^^^^^^^^
```

Finally, update the `Program.cs` code to call the method as follows:

```
Console.WriteLine(Map.GetShortestPath());
```



```
---------------------------------------TABLE---------------------------
NODE    VISITED?         SHORTEST DISTANCE TO START NODE PREVIO
A       True                                             Ø
B       F                4                               A
C                        2                               A
D       True             9                               E
E       True             5                               C
F       True             20                              D
=======================================================================
THIS TABLE REPRESENTS THE SHORTEST PATH ^^^^^^^^^^^^^^^^^^^^^^
```

Program updated □

Exercise 8 – Bomb Search

Section A

A 1 Which one of the three existing method stubs in `Board.cs` is suite

....

A 2 Give a class name and line number from the program where a constru

....

A 3 Explain the purpose of the code `Bombs = R*C / 3;` in the constru

....

....

....

....

....

....

A 4 Explain why the constructor of `Tile.cs` only takes one parameter wh
variables that have to be initialised.

....

....

....

....

A 5 There is an extra, redundant method in `Tile.cs` called `Reveal()`.
Explain why it is convenient to include this method.

....

....

....

....

A 6 The program could be crashed by passing negative numbers to the co
Name the type of exception that would be thrown in this case.

....

A 7

The constructor of `Board.cs` could be made more robust by using a
the unusual event that negative numbers are passed in as parameters

Explain how try–catch statements work and how one can be useful in

A 8

The `Arena` variable is an instance variable of the `Board` class. It is a 2

State the key difference between a list and an array.

A 9

n whether or not it would be suitable to use a list instead of an a

A 10

The `Board` class stores the number of rows and columns integers for

Write a line of code that could be used to find the number of column

Exercise 8 – Bomb Search

Section B

B 1

Modify the program to implement the `Explain` method of `Tile.c` output a description of a particular tile to the console.

Program updated □

B 2

Modify the Tile class to include accessor and mutator methods for ins

Program updated □

B 3

Modify the `SetUpBoard()` procedure to add tiles to the board. The random positions on the board. Here is an algorithm to guide the im

- *Build a list of Bomb tiles*
- *Build a list of Safe tiles*
- *Build an empty list of tiles to hold the newly shuffled list*
- *WHILE both lists contain elements:*
    - *Choose at random which list to remove a tile from: Bomb o*
    - *Add the first tile from the chosen list to the shuffled list*
    - *Remove the first tile from the chosen list*
- *ENDWHILE*
- *Determine which list has still got elements in it and add the rema shuffled list*
- *Place the tiles from the shuffled list onto the board*
- *Amend each tile's adjacent 'bomb' value now that it is in place ← many!*

Here is an algorithm to guide the implementation of the method for

- *Take the row and column position indices in as parameters*
- *Declare a counter variable to keep a running total of bombs found*
- *IF the row is not the top row, proceed to check the 3 squares abov*
- *IF the row is not the bottom row, proceed to check the 3 squares b*
- *IF the tile isn't in the first column, proceed to check the square im*
- *IF the tile isn't in the final column, proceed to check the square im*
- *Return the total number of bombs found*

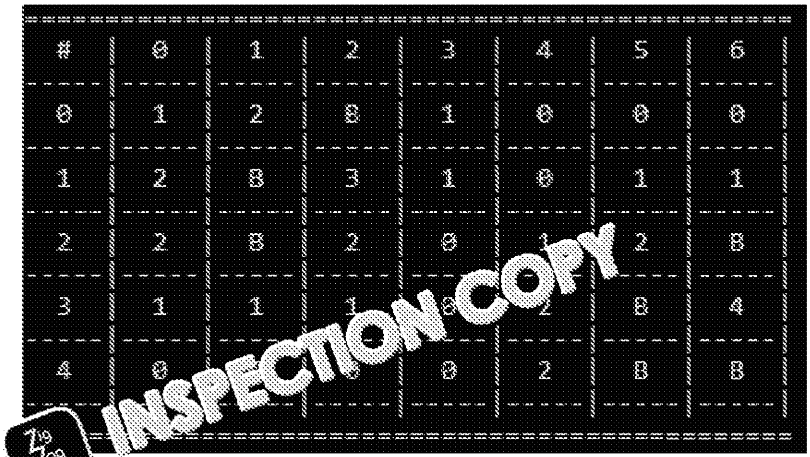
Solving this part of the problem is faster if you understand 'short circ expressions, i.e. when you use a logical AND, the expression on the ri evaluated if the expression on the left evaluates to False. Deploying th that you aren't about to attempt to access an array index that is out o

Program updated □

B 4

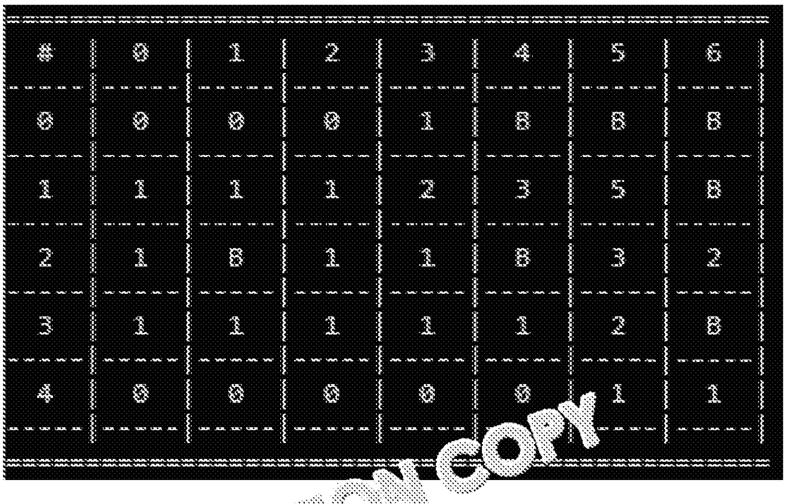
Modify the `DisplayBoard()` procedure so that the board is visualised contents of all the tiles. The visualisation should show the row and board numbers, and there should be lines partitioning the rows and c

Having just developed a method for setting up a board, it would be u visually see the positions of the bombs and the numeric values of the first set up. Any bombs should be displayed as B. Any revealed numb single-digit numeric value.



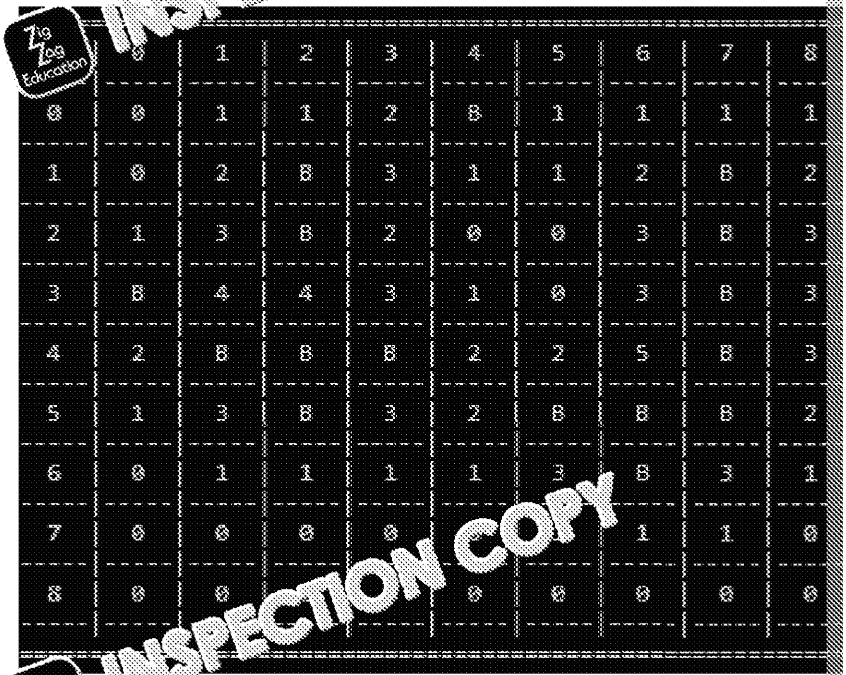
```
==================================================================
  #  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |
-----|-----|-----|-----|-----|-----|-----|-----|
  0  |  1  |  2  |  8  |  1  |  0  |  0  |  0  |
-----|-----|-----|-----|-----|-----|-----|-----|
  1  |  2  |  8  |  3  |  1  |  0  |  1  |  1  |
-----|-----|-----|-----|-----|-----|-----|-----|
  2  |  2  |  8  |  2  |  0  |  1  |  2  |  8  |
-----|-----|-----|-----|-----|-----|-----|-----|
  3  |  1  |  1  |  1  |  8  |  2  |  8  |  4  |
-----|-----|-----|-----|-----|-----|-----|-----|
  4  |  0  | [illegible] |  0  |  0  |  2  |  8  |  8  |
-----|-----|-----|-----|-----|-----|-----|-----|
==================================================================
```

Running the program several times will always produce randomised r

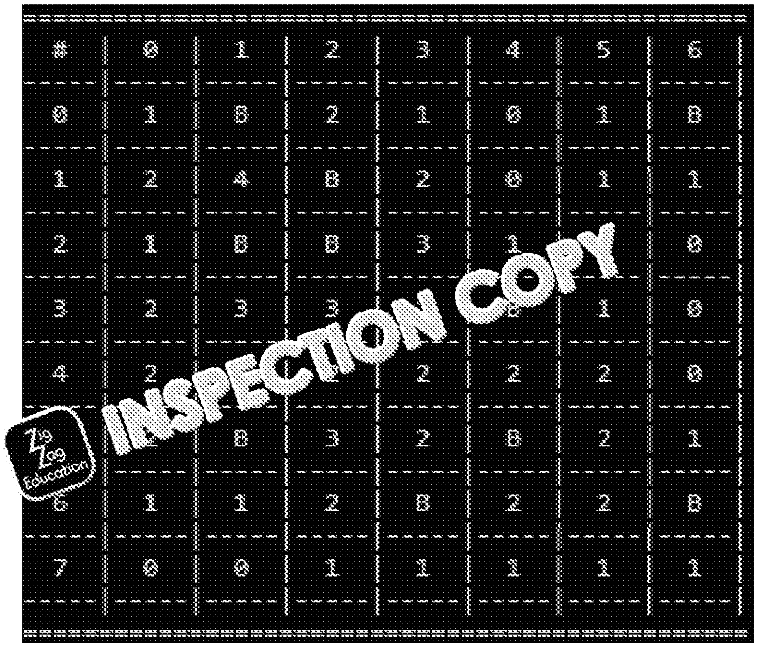


```
==================================================================
  #  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |
-----|-----|-----|-----|-----|-----|-----|-----|
  0  |  0  |  0  |  0  |  1  |  8  |  8  |  8  |
-----|-----|-----|-----|-----|-----|-----|-----|
  1  |  1  |  1  |  1  |  2  |  3  |  5  |  8  |
-----|-----|-----|-----|-----|-----|-----|-----|
  2  |  1  |  8  |  1  |  1  |  8  |  3  |  2  |
-----|-----|-----|-----|-----|-----|-----|-----|
  3  |  1  |  1  |  1  |  1  |  1  |  2  |  8  |
-----|-----|-----|-----|-----|-----|-----|-----|
  4  |  0  |  0  |  0  |  0  |  0  |  1  |  1  |
-----|-----|-----|-----|-----|-----|-----|-----|
==================================================================
```

You can also try varying the dimensions of the game board in Progr



```
=====================================================================
  #  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  0  |  0  |  1  |  1  |  2  |  8  |  1  |  1  |  1  |  1
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  1  |  0  |  2  |  8  |  3  |  1  |  1  |  2  |  8  |  2
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  2  |  1  |  3  |  8  |  2  |  0  |  0  |  3  |  8  |  3
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  3  |  8  |  4  |  4  |  3  |  1  |  0  |  3  |  8  |  3
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  4  |  2  |  8  |  8  |  8  |  2  |  2  |  5  |  8  |  3
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  5  |  1  |  3  |  8  |  3  |  2  |  8  |  8  |  8  |  2
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  6  |  0  |  1  |  1  |  1  |  1  |  3  |  8  |  3  |  1
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  7  |  0  |  0  |  0  |  0  | [illegible] | [illegible] |  1  |  1  |  0
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
  8  |  0  |  0  | [illegible] | [illegible] |  0  |  0  |  0  |  0  |  0
-----|-----|-----|-----|-----|-----|-----|-----|-----|----
=====================================================================
```



```
=======================================================
  #  |  0  |  1  |  2  |  3  |  4  |  5  |  6  |
-----|-----|-----|-----|-----|-----|-----|-----|
  0  |  1  |  B  |  2  |  1  |  0  |  1  |  B  |
-----|-----|-----|-----|-----|-----|-----|-----|
  1  |  2  |  4  |  B  |  2  |  0  |  1  |  1  |
-----|-----|-----|-----|-----|-----|-----|-----|
  2  |  1  |  B  |  B  |  3  |  1  |  1  |  0  |
-----|-----|-----|-----|-----|-----|-----|-----|
  3  |  2  |  3  |  3  |[illegible]|  2  |  1  |  0  |
-----|-----|-----|-----|-----|-----|-----|-----|
  4  |  2  |[illegible]|[illegible]|  2  |  2  |  2  |  0  |
-----|-----|-----|-----|-----|-----|-----|-----|
[illegible]|[illegible]|  B  |  3  |  2  |  B  |  2  |  1  |
-----|-----|-----|-----|-----|-----|-----|-----|
  6  |  1  |  1  |  2  |  B  |  2  |  2  |  B  |
-----|-----|-----|-----|-----|-----|-----|-----|
  7  |  0  |  0  |  1  |  1  |  1  |  1  |  1  |
-----|-----|-----|-----|-----|-----|-----|-----|
=======================================================
```

If you are not happy with how well-distributed the bombs are, the issu randomisation was used. Try to vary the random number selection pro

Once this is proven to work, create a second copy of the method call that it displays as a '?' character any tile that has not yet been reveale zero value should be blank, not zero-valued.

Add test code for both methods to `Program`[illegible] Remember that yo `SetUpBoard()` first or there will be no tile objects in each of the pos

Program updated □

B 5

Modify the program to add code to the `GetMove()` method of the B [illegible], dealing with incorrect data types being entered. It should re

Program updated □

B 6

Modify the `Board` class to add a `Reveal()` function that gets a mov `GetMove` function) and reveals the chosen tile. The user's selection sh ensure that the tile has not yet been revealed, and if it has, the functi message and return *False*. The function should output a "`Game Ove` bomb has been revealed. Otherwise, it should simply return a *False* va

The main program procedure should be modified to continually call th the result in a variable, and then display the game board. The result o be used to decide whether to continue iterating.

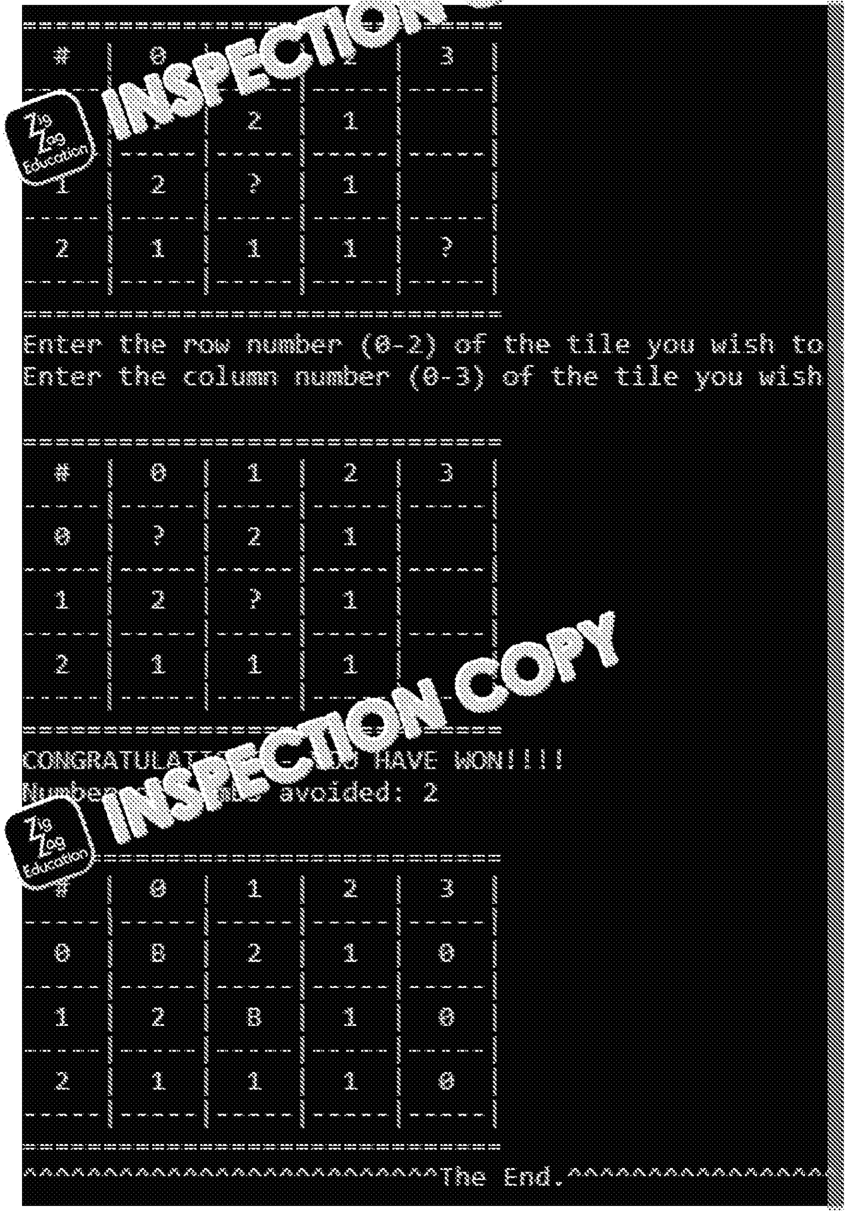
Program updated □

B 7

Modify the `Main` program to display a celebratory message followed player has revealed every tile except those containing bombs. Add a `Board.cs` to store the number of safe (non-bomb) tiles that have al constructor and add suitable getter and setter methods.

Modify `Program.cs` so that it checks the number of safe tiles found the game.

For example, the result of winning would be:



```
==============================
  #  |  0  |  1  |  2  |  3  |
-----|-----|-----|-----|-----|
  0  |  ?  |  2  |  1  |     |
-----|-----|-----|-----|-----|
  1  |  2  |  ?  |  1  |     |
-----|-----|-----|-----|-----|
  2  |  1  |  1  |  1  |  ?  |
-----|-----|-----|-----|-----|
==============================
Enter the row number (0-2) of the tile you wish to
Enter the column number (0-3) of the tile you wish


==============================
  #  |  0  |  1  |  2  |  3  |
-----|-----|-----|-----|-----|
  0  |  ?  |  2  |  1  |     |
-----|-----|-----|-----|-----|
  1  |  2  |  ?  |  1  |     |
-----|-----|-----|-----|-----|
  2  |  1  |  1  |  1  |     |
-----|-----|-----|-----|-----|
==============================
CONGRATULATIONS YOU HAVE WON!!!!
Number of bombs avoided: 2


==============================
  #  |  0  |  1  |  2  |  3  |
-----|-----|-----|-----|-----|
  0  |  B  |  2  |  1  |  0  |
-----|-----|-----|-----|-----|
  1  |  2  |  B  |  1  |  0  |
-----|-----|-----|-----|-----|
  2  |  1  |  1  |  1  |  0  |
-----|-----|-----|-----|-----|
==============================
^^^^^^^^^^^^^^^^^^^^^^^^^^The End.^^^^^^^^^^^^^^^^
```

Program updated □

Exercise 9 – File Handling and Hash Tabl

Section A

A 1 Give a line number from the program that contains a literal long valu

A 2 Give a line number from the program where an array of longs is decla

A 3 Explain how the program ensures that the newly generated product c single digit provided by the user.

A 4 The array could instead be implemented as a tuple.

Describe what a tuple is.

A 5 Explain why you might choose to avoid tuples for representing produ

A 6 The program does not currently access the text file that stores multipl

Name the class that is used to represent the text file as an object in a

A 7 A list variable called `Table` is to be set up so that it can be used to c is to be populated with data held in the current text file.

Explain how data is stored in a hash table.

....

....

....

....

....

....

A 8 The unique product codes can be used as keys (inputs) for the hash fu

a) Explain what a hash function is.

....

....

....

b) Even when keys for data are unique, it is often the case that hash values that have been generated before. Name this phenomenon values of addresses to be limited, even though this may lead to

....

....

....

....

A 9 Compare and contrast the use of a serial text file with the use of a seq

....

....

....

....

....

....

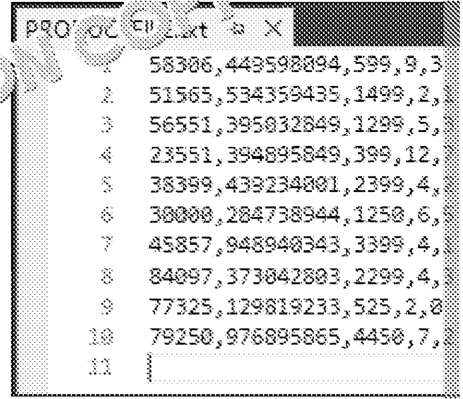
A 10

The records of each product will be organised as a hash table using th

```
(ProductCode * (ProductCode + ProductCode / 29)) % 
```

... and then the resulting table will be stored as a new text file called H

Compute the contents of this hash table manually using the data from PRODUCTFILE.txt.



```
58306,443598894,599,9,3
51565,534359435,1499,2,
56551,395032849,1299,5,
23551,394895849,399,12,
38399,438234801,2399,4,
38000,284738944,1250,6,
45857,948940343,3399,4,
84097,373042803,2299,4,
77325,129819233,525,2,0
79250,976895865,4450,7,
```

Hash Table Location	First Entry	Oth
Table[0]		
Table[1]		
Table[2]		
Table[3]		
Table[4]		
Table[5]		
Tabl[illegible]		
[illegible][7]		
Table[8]		
Table[9]		
Table[10]		
Table[11]		
Table[12]		
Table[13]		
Table[14]		
Table[15]		
Table[16]		
Table[17]		
Table[18]		

Exercise 9 – File Handling & Hash Tables

Section B

B 1 Add a new text file to the project called HASHFILE.txt and leave it bla
Comment out all contents of `Main()` except the `Console.ReadKey`
open on the screen.

Program updated □

B 2 Add a private class method called `GenerateHashValue` which is a t
as a parameter and returns its hash value.

Program updated □

B 3 Add a line of code to the `ShowFactFile()` method so that the has
displayed at the end of the fact file when it is output.

Program updated □

B 4 Add a private class function called `ReadInOldTextFile()` which r
PRODUCTFILE.txt. Each line of the file should be transformed into an a
whole should be represented as a list of these arrays when it is return
handling should be used as part of opening the file.

Program updated □

B 5 Add a private class procedure called `ShowTextFileOfWholeTabl`
arrays generated from the previous task's function `ReadInOldTextF`
the `ShowFactFile()` method to display it in full. Add code to the `M`
everything so far is working.

Program updated □

B 6 Add a function called `InitiallyPopulateHashFile()` which tak
extracted from the old products file called PRODUCTSFILE.txt and ret
The hash table is a list of 19 lists of arrays, and the decision as to whi
array gets added to is determined by its hash value. Use the identifie
returned by the function. Add code to the `Main()` method to call th
result in a variable.

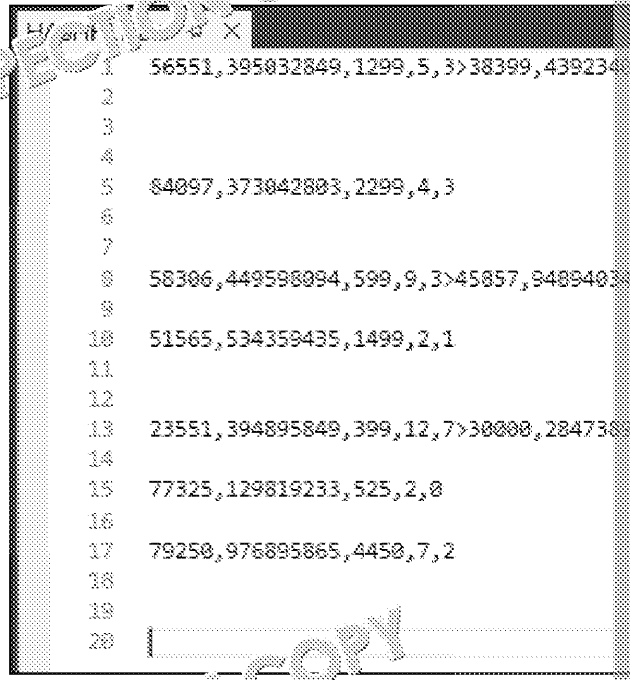
Program updated □

B 7 Add a procedure called `WriteMigratedData()` to write the curren
called HASHFILE.txt, separating the arrays from each other with the '>' c
same line. The file will have 19 lines for storing data, one per list of arra
a list). Lines with no data should stay blank. The first few lines can be re
method (which currently opens this file in read mode and builds a hash
single call to `InitiallyPopulateHashFile` can be made from wi
Replace the `Main()` method code with a call to this new method to te
It should overwrite the current file's entire contents (write mode, n

Tip: While it is not essential, it is advisable to write a separate function c
`ConvertArrayToString()` to reformat an array of long integers in
separated list.

It is intended that this method will be used only once as part of data co and the new system, so once the file is successfully populated you can which accesses the old text file. Moving forward, the features that this p solely around the ability to work with the new HASHFILE.txt file.

Note: In the diagram below, the line numbers are 1 higher than the has line numbering in Visual Studio and the hash v lue is not stored with th range from *0* to *19*.



```
1   56551,395032849,1299,5,3>38399,439234
2
3
4
5   84097,373042803,2299,4,3
6
7
8   58306,449598094,599,9,3>45857,9489403
9
10  51565,534359435,1499,2,1
11
12
13  23551,394895849,399,12,7>30000,284738
14
15  77325,129819233,525,2,0
16
17  79250,976895865,4450,7,2
18
19
20
```

Program updated □

B 8 a second copy of the `WriteMigratedData()` called `Update` as a parameter and overwrites the current text file with it. This sh first few lines from this copy of the method and inserting a parameter.

Program updated □

B 9 Create a `ReadHashFile()` function which reads the entire contents into the hash table format used previously: a list of lists of long integer construct this function if you also construct a function called `Convert` can convert one product's details from a string representation to an ar

Program updated □

EXERCISE 10 – REVERSE POLISH

SECTION A

A 1 Line 9 references the bitwise ^ operator, which in some other languag
Explain what it achieves in C#.

A 2 Give a line number from the program that contains a class variable.

A 3 Explain how the `IsInt` function determines whether or not the given

A 4 Write the RPN form of the following infix expression: (3 + 2) * (4 - 1) /

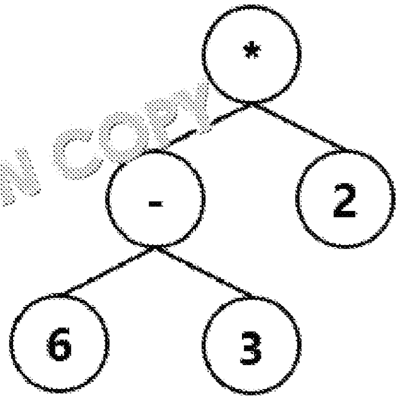
A 5 Write the infix form of the following RPN expression: 4 5 + 3 2 1 / - *

A 6 A stack is a data structure that behaves like a list but with restrictions.

Describe the main features of a stack.

A 7 Mathematical expressions can be represented as a binary tree, where produce the RPN expression, and an in-order tree traversal will produ

Write the RPN expression produced by the following binary tree:



....

....

A 8 Write the infix expression produced by the binary tree in A7.

....

....

A 9 Draw the binary tree that is created by the following infix expression:

A 10 Draw the binary tree that is created by the following RPN expression:

Exercise 10 – Reverse Polish

Section B

B 1

Modify the `ConvertToPostfix` function so that it accepts a list of
parameter and returns a list of strings

Program updated □

B 2

Modify the `ConvertToPostfix` function so that, at the very begin
declared and initialised: `Stack` and `OpStack`.

Program updated □

B 3

Modify the `ConvertToPostfix` function so that the major iterative
controlled by a FOREACH loop that iterates through all items in the li
parameter. Use the identifier `Item` when setting up the loop.

Program updated □

B 4

Modify the `ConvertToPostfix` function so that the first task carri
is a check to see whether the value of `Item` is an integer, and if it is, i
`Stack`. While Stack is technically a list and not formally defined as a s
to behave like a stack and so items pushed to it should be appended
position.

Program updated □

B 5

Modify the `ConvertToPostfix` function so that the ELSE block wi
begins with an inner (nested) IF statement which checks that the `OpS`
long as it isn't empty, the variable `LastOp` should be set to store a cop
[illegible] the item at the top of the stack of operators (but it sho
`OpStack`!).

Program updated □

B 6

Modify the `ConvertToPostfix` function so that after the IF statem
there is a new and separate IF-ELSEIF-ELSE structure that implements
nested IF structure must also sit inside the same ELSE structure as the

```
IF any of these 3 criteria is true, enter the IF block:
    1.  OpStack is empty
    2.  Item is an opening parenthesis
    3.  LastOp is either + or –, and at the same time, Item is either * o
ELSE IF Item is a closing parenthesis:
    Set the value of Operator to null
    WHILE Operator is not an opening parenthesis AND OpStack conta
            Pop the value of the top of OpStack and store it in the var
            IF Operator is not an opening parenthesis, push it to Stack
ELSE
    IF LastOp is not an opening parenthesis:
            Push LastOp to Stack
            Overwrite the top value of OpStack with Item
    ELSE
            Push Item on to OpStack
```

Program updated □

B 7

Modify the `ConvertToPostfix` function so that after the FOREACH iterates from right to left through the `OpStack` list and successively:

- pushes each item to Stack
- pops each item from `OpStack` (without storing or inspecting

Program updated □

B 8

Modify the `Main()` procedure so that it tests the working of the Con valid strings.

Program updated □

Suggested Solutions & Mark Scheme

***NB.** When studying the suggested answers for Section B tasks, it is important to rem ways of achieving the same outcome, and credit should be given for alternative solu*

Exercise 1 – Searching Algorithms

Section A

A1

1 mark for giving a suitable example:

Line 7/8/9

A2

1 mark for giving a suitable example:

Line 20

A3

1 mark for explaining that binary search is more efficient / faster than linear search:

Binary search is usually more time-efficient (takes less time to run) than linear sea

A4

1 mark for explaining that binary search can be performed only on sorted lists:

The list might be unsorted – a binary search requires the list to be sorted.

A5

1 mark for explaining why it was used, not defining what it is:

It aided readability as it was obvious when it was used.
The same rogue value (-1) was used throughout the program.
There was no risk of accidentally overwriting it (i.e. no logic error was possible).

A6

1 mark for explaining the suitability:

Array indexing starts at 0, so -1 is an obviously invalid number.

A7

Up to 2 marks for explaining the reason why ELSE is optional; award 2 marks for cle

The `linearSearch` function will be exited when it hits the return statement withi happen because the Boolean expression evaluates to *False*, then the program adva immediately after the IF block anyway. Using ELSE is thus optional.

A8

2 marks (1 mark for explaining that time complexity describes number of operation mark for explaining how time complexity relates to varying input sizes):

The time complexity of an algorithm is a description of the number of operations t complete in relation to the size of the input given to the algorithm.

A9

2 marks (1 mark for stating the time complexity of linear search; 1 mark for binary s

Linear search has a time complexity of **O(n)**. Binary search has a time complexity o

A10

Up to 2 marks for explaining why recursion may not be suitable. For example:

Recursion may not be suitable for searching large arrays because each recursive ca frame which includes a copy of the list (or part of the list), return addresses, and th variables to create, which could lead to the computer running out of memory many langu[illegible] combinations have imposed recursion limits for this reason. Al recursive solutions less time-efficient than iterative solutions.

Section B

B1

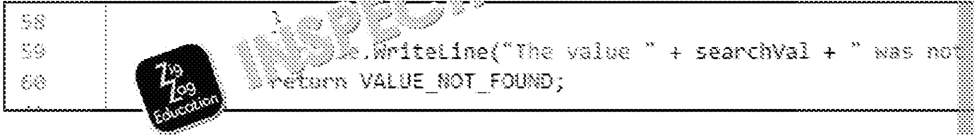
1 mark available for providing this correction:

= should be replaced with == on Line 3

B2

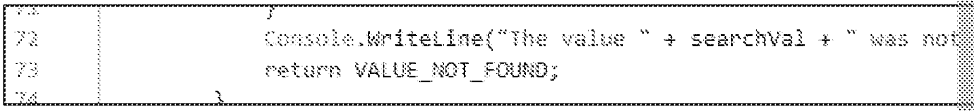
1 mark available for modifying the code as shown (or equivalent code):

In linear search:



```
            }
            Console.WriteLine("The value " + searchVal + " was not
            return VALUE_NOT_FOUND;
```

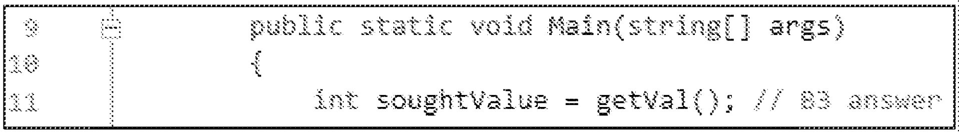
In binary search:



```
            }
            Console.WriteLine("The value " + searchVal + " was not
            return VALUE_NOT_FOUND;
        }
```

B3

1 mark available for modifying the code as shown (or equivalent code):



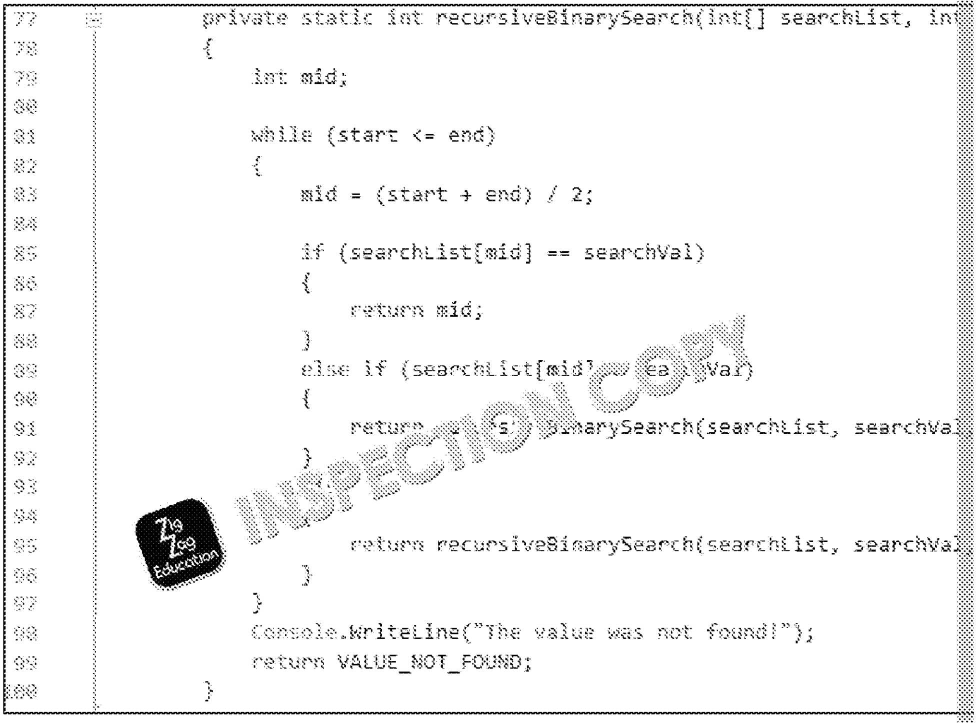
```
        public static void Main(string[] args)
        {
            int soughtValue = getVal(); // B3 answer
```

B4

5 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- creating a recursiveBinarySearch function that takes an array, search
- returning the correct index (mid) when the element is found
- returning "VALUE_NOT_FOUND" if the element is not in the array
- recursively calling recursiveBinarySearch if another stage of searching
- modifying the main program procedure to display the result of recursi



```
        private static int recursiveBinarySearch(int[] searchList, int
        {
            int mid;

            while (start <= end)
            {
                mid = (start + end) / 2;

                if (searchList[mid] == searchVal)
                {
                    return mid;
                }
                else if (searchList[mid] < searchVal)
                {
                    return recursiveBinarySearch(searchList, searchVal
                }
                else
                {
                    return recursiveBinarySearch(searchList, searchVal
                }
            }
            Console.WriteLine("The value was not found!");
            return VALUE_NOT_FOUND;
        }
```

In the main program:

```
            // B4 answer part 2 of 2:
            Console.WriteLine("RECURSIVE BINARY TEST (-5): " +
            Console.WriteLine("RECURSIVE BINARY TEST ( 1): " +
            Console.WriteLine("RECURSIVE BINARY TEST ( 0): " +
            Console.WriteLine("RECURSIVE BINARY TEST (10): " +
            Console.WriteLine("RECURSIVE BINARY TEST (11): " +
```

■ B5

4 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- creating a `getVal` function that repeats until a valid input is given
- handling (but not accepting) invalid input
- using appropriate messages
- returning the resulting value as an integer
- modifying the main program procedure to use `getVal` to set the value of

```
// B5 answer part 1 of 2:
private static int getVal()
{
    Console.Write("Enter an integer value to search for: ");
    string userInput = Console.ReadLine();
    int userValue = 0;
    bool successful = false;

    do
    {
        try
        {
            userValue = int.Parse(userInput);
            successful = true;
        }
        catch (Exception e)
        {
            Console.Write("NOT AN INTEGER! TRY AGAIN...\nEnter an int
            userInput = Console.ReadLine();
        }
    } while (!successful);

    return userValue;
}
```

In the main program:

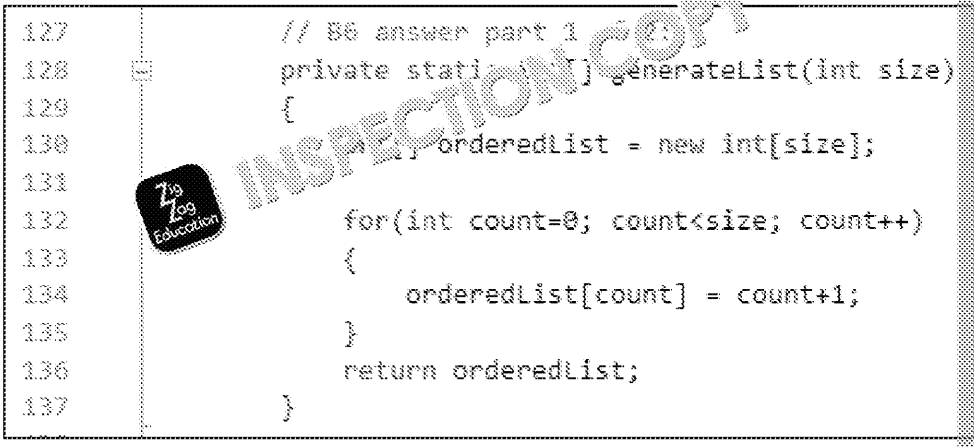
```
        {
            int soughtValue = getVal(); // B3 answer
```

▩ B6

2 marks available for modifying the code as shown (or equivalent code):

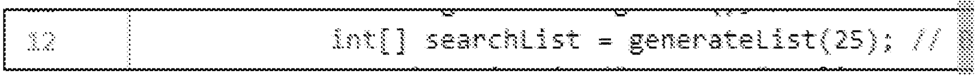
Marks could be awarded for:

- creating a `generateList` function that correctly generates and returns a length
- modifying the main program procedure to use `generateList` to create



```
        // B6 answer part 1 of 2:
        private static int[] generateList(int size)
        {
            int[] orderedList = new int[size];

            for(int count=0; count<size; count++)
            {
                orderedList[count] = count+1;
            }
            return orderedList;
        }
```

In the main program:



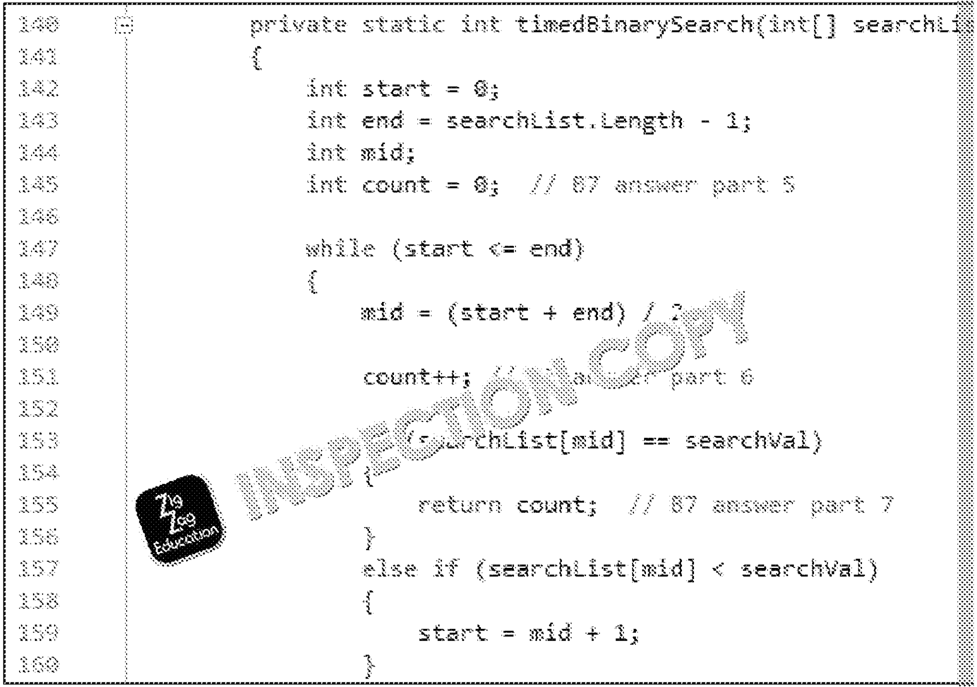
```
            int[] searchList = generateList(25); //
```

▩ B7

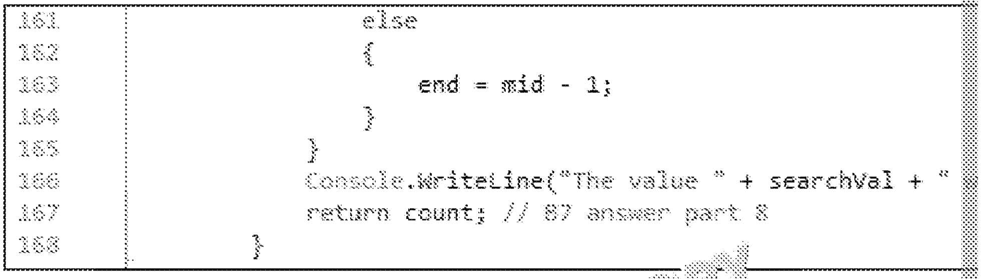
5 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

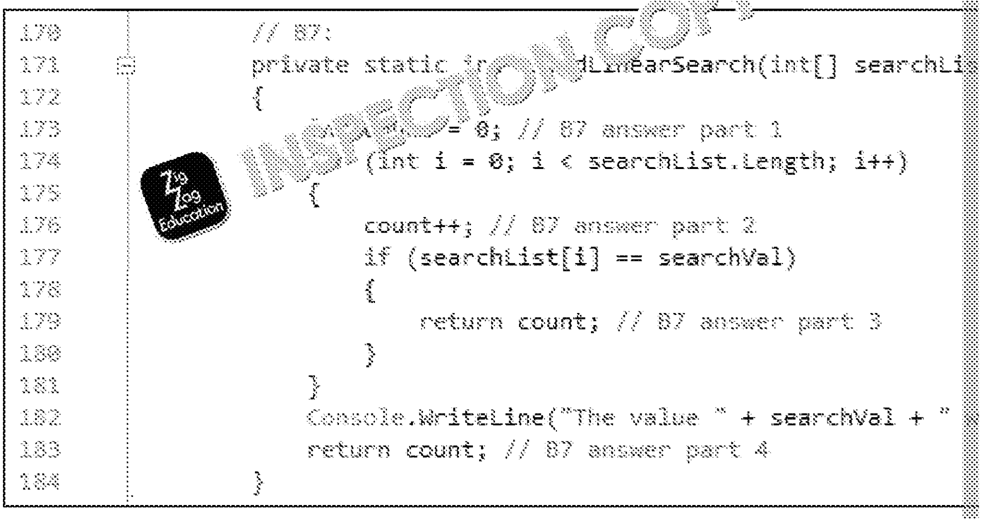
- correctly counting and returning the number of steps made by the `line`
- correctly counting and returning the number of steps made by the `bina`
- creating a `test` function that returns the average number of operations of a given length
- modifying the main program to use `test` to perform 1,000 tests on lists and [illegible]00
- modifying the main program to display how many more operations `lin` `binarySearch` for each of the list lengths tests



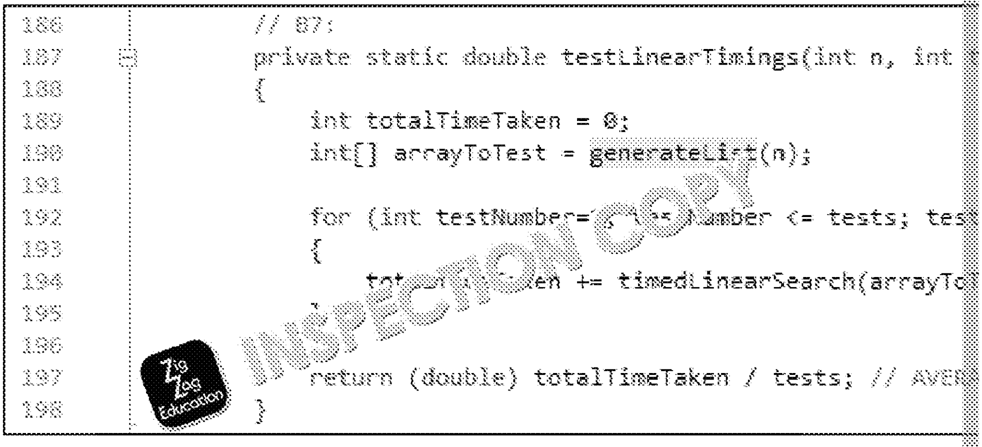
```
        private static int timedBinarySearch(int[] searchLi
        {
            int start = 0;
            int end = searchList.Length - 1;
            int mid;
            int count = 0;  // B7 answer part 5

            while (start <= end)
            {
                mid = (start + end) / 2;

                count++; // B7 answer part 6

                if (searchList[mid] == searchVal)
                {
                    return count;  // B7 answer part 7
                }
                else if (searchList[mid] < searchVal)
                {
                    start = mid + 1;
                }
```



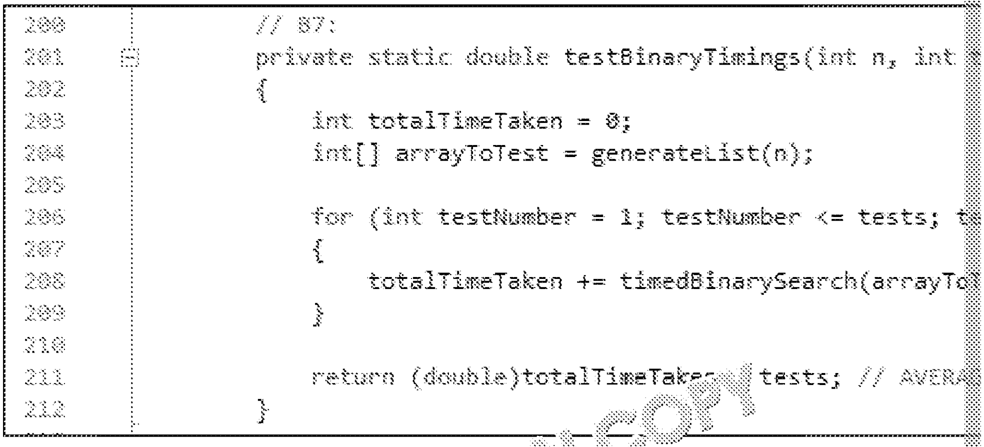
```
                else
                {
                    end = mid - 1;
                }
            }
            Console.WriteLine("The value " + searchVal + "
            return count; // 87 answer part 8
        }
```



```
        // 87:
        private static int timedLinearSearch(int[] searchLi
        {
            int count = 0; // 87 answer part 1
            for (int i = 0; i < searchList.Length; i++)
            {
                count++; // 87 answer part 2
                if (searchList[i] == searchVal)
                {
                    return count; // 87 answer part 3
                }
            }
            Console.WriteLine("The value " + searchVal + "
            return count; // 87 answer part 4
        }
```

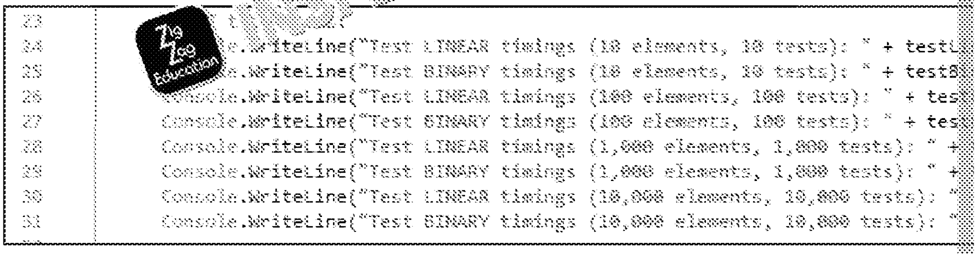


```
        // 87:
        private static double testLinearTimings(int n, int
        {
            int totalTimeTaken = 0;
            int[] arrayToTest = generateList(n);

            for (int testNumber=1; testNumber <= tests; tes
            {
                totalTimeTaken += timedLinearSearch(arrayTo
            }

            return (double) totalTimeTaken / tests; // AVER
        }
```



```
        // 87:
        private static double testBinaryTimings(int n, int
        {
            int totalTimeTaken = 0;
            int[] arrayToTest = generateList(n);

            for (int testNumber = 1; testNumber <= tests; t
            {
                totalTimeTaken += timedBinarySearch(arrayTo
            }

            return (double)totalTimeTaken / tests; // AVERA
        }
```

In the main program:



```
            // 87:
            Console.WriteLine("Test LINEAR timings (10 elements, 10 tests): " + testL
            Console.WriteLine("Test BINARY timings (10 elements, 10 tests): " + testB
            Console.WriteLine("Test LINEAR timings (100 elements, 100 tests): " + tes
            Console.WriteLine("Test BINARY timings (100 elements, 100 tests): " + tes
            Console.WriteLine("Test LINEAR timings (1,000 elements, 1,000 tests): " +
            Console.WriteLine("Test BINARY timings (1,000 elements, 1,000 tests): " +
            Console.WriteLine("Test LINEAR timings (10,000 elements, 10,000 tests): "
            Console.WriteLine("Test BINARY timings (10,000 elements, 10,000 tests): "
```

Exercise 2 – Sorting Algorithms

Section A

A1

1 mark for giving a suitable example:

They are accepted in string format... [1]
... then the string is parsed to read its value in as an integer [1]

A2

1 mark:

Line 66

A3

1 mark for a suitable definition:

Recursion is when a subroutine is defined in terms of itself or calls itself.

A4

2 marks for any two of these points:

\t is an escape sequence.
In this case it represents the Tab character.
It is used here so that lists get output with their values all starting at a new tab sto
in a human-friendly format, horizontally across the screen without overlapping.

A5

Any 2 marks drawn from any of the following points:

When its value eventually gets set to *True*...
... this represents the event where an entire pass has been made through the array
... and when this occurs, the sorting can be stopped immediately...
... as the array is sorted.
This improves the overall efficiency of the algorithm in cases where it would o
the array ch values that are in order.

A6

2 marks for any two of these ideas:

During bubble sort, the index of the value on the left is indicated by the pointer.
The value to the right of the pointer is thus compared with the value at the pointe
A pointer value of SIZE-1 would point at the last element in the array due to the u
arrays.
If the pointer were permitted to point at the last element, it would attempt to com
immediately to its right.
This would be an 'array out of bounds' exception / a logic error / a run-time error.

A7

3 marks (1 mark for explaining DIV; 1 mark for explaining the impact on BOTH arra

The DIV operation takes the odd length and divides it by two, discarding the remai
mean that the new left array would contain only six elements.
The right array will have the remaining elements, calculated from the original array
length, so the middle element will always end up occurring as the first element of
have one extra element.

A8

2 marks (1 mark for explaining that divide-and-conquer algorithms break a problem (divide); 1 mark for explaining that these problems can then be further divided until (conquer)):

A divide-and-conquer algorithm is an algorithm that breaks down a problem into
that can be individually solved and then recombined to solve the original problem

▩ A9

2 marks (1 mark for giving the time complexity of a bubble sort; 1 mark for giving the

Bubble sort has a time complexity of **$O(n^2)$**. Merge sort has a time complexity of O

▩ A10

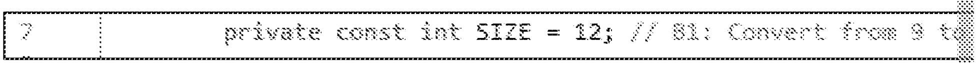
2 marks (1 mark for describing how an insertion sort uses a sorted list and an unsorte each element in the unsorted list is placed into the correct position in the sorted list):

An insertion sort creates an empty sorted list and an unsorted list. Each element o the correct position in the sorted list until the unsorted list is empty and all the nu correct order in the new sorted list.

SECTION B

▩ B1

1 mark available for modifying the code as shown:

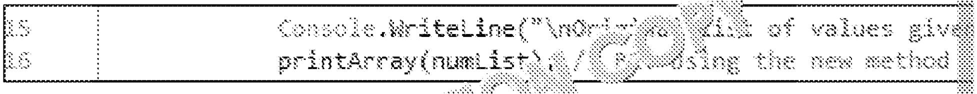


```
        private const int SIZE = 12; // B1: Convert from 9 to
```

▩ B2

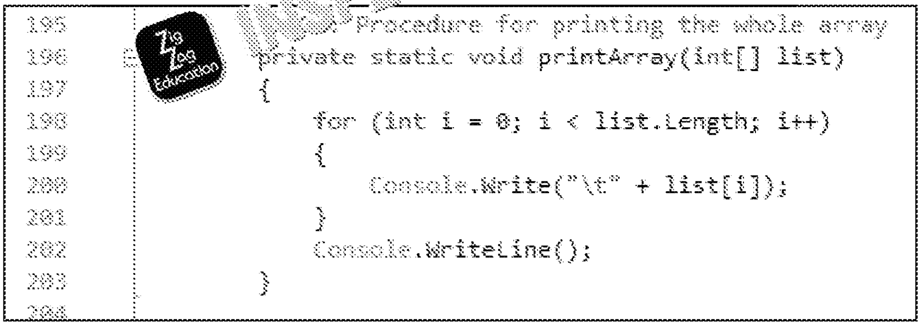
1 mark available for modifying the code as shown (or equivalent code):

Main method (1 mark):



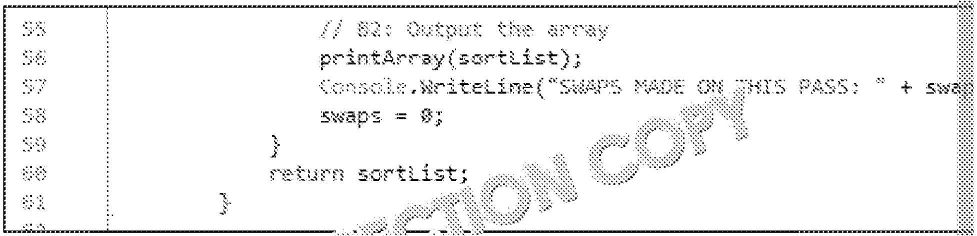
```
            Console.WriteLine("\nOriginal list of values give
            printArray(numList); // B2: Using the new method
```

`printArray` method (2 marks):



```
        // B2: Procedure for printing the whole array
        private static void printArray(int[] list)
        {
            for (int i = 0; i < list.Length; i++)
            {
                Console.Write("\t" + list[i]);
            }
            Console.WriteLine();
        }
```

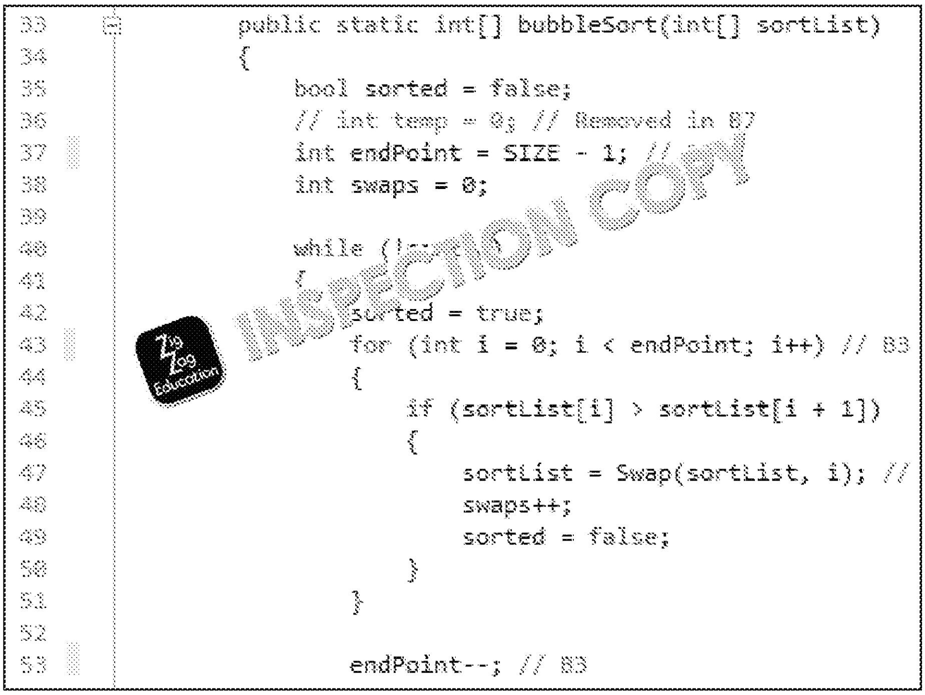
Bubble sort method (1 mark):



```
                // B2: Output the array
                printArray(sortList);
                Console.WriteLine("SWAPS MADE ON THIS PASS: " + swa
                swaps = 0;
            }
            return sortList;
        }
```

▩ B3

1 mark deducted per earmarked part missing or improperly used:



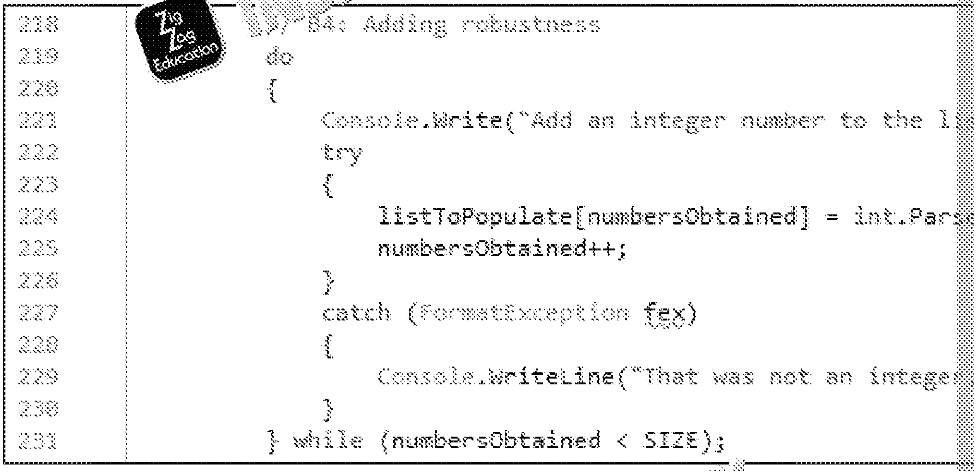
```
        public static int[] bubbleSort(int[] sortList)
        {
            bool sorted = false;
            // int temp = 0; // Removed in B7
            int endPoint = SIZE - 1; // B3
            int swaps = 0;

            while (!sorted)
            {
                sorted = true;
                for (int i = 0; i < endPoint; i++) // B3
                {
                    if (sortList[i] > sortList[i + 1])
                    {
                        sortList = Swap(sortList, i); //
                        swaps++;
                        sorted = false;
                    }
                }

                endPoint--; // B3
```

▩ B4

4 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- setting up a loop that ends only once enough valid inputs are given
- successfully adding valid values to the array at the right location
- displaying a clear error message if an invalid input is given
- correctly allowing re-entry of an attempt



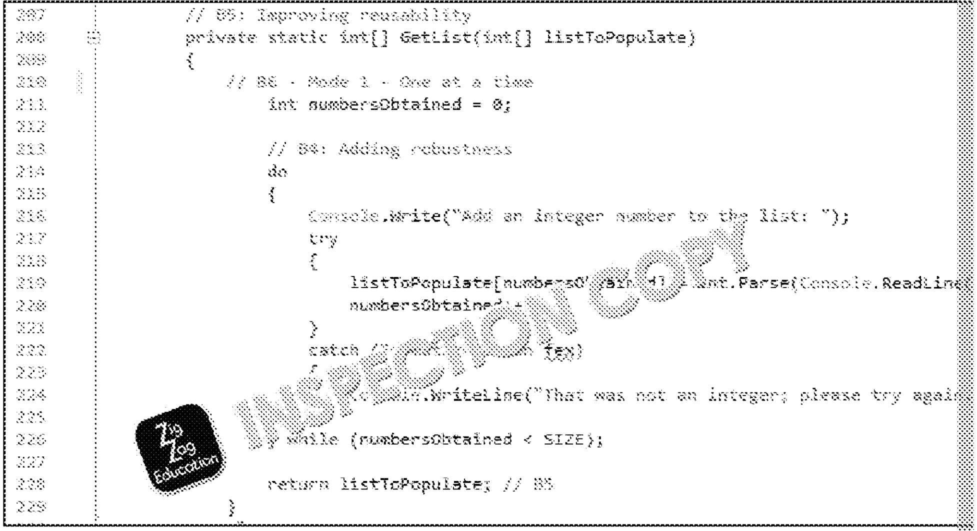
```
            // B4: Adding robustness
            do
            {
                Console.Write("Add an integer number to the li
                try
                {
                    listToPopulate[numbersObtained] = int.Pars
                    numbersObtained++;
                }
                catch (FormatException fex)
                {
                    Console.WriteLine("That was not an intege
                }
            } while (numbersObtained < SIZE);
```

▩ B5

4 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- creating a getList function that returns a list
- using a while loop followed by a selection structure (or equivalent) to con
  unless a blank is given
- using try–except structure to detect invalid input and having an appropri
  another value to be added under these circumstances
- modifying the main program procedure to use the `getList` function ap



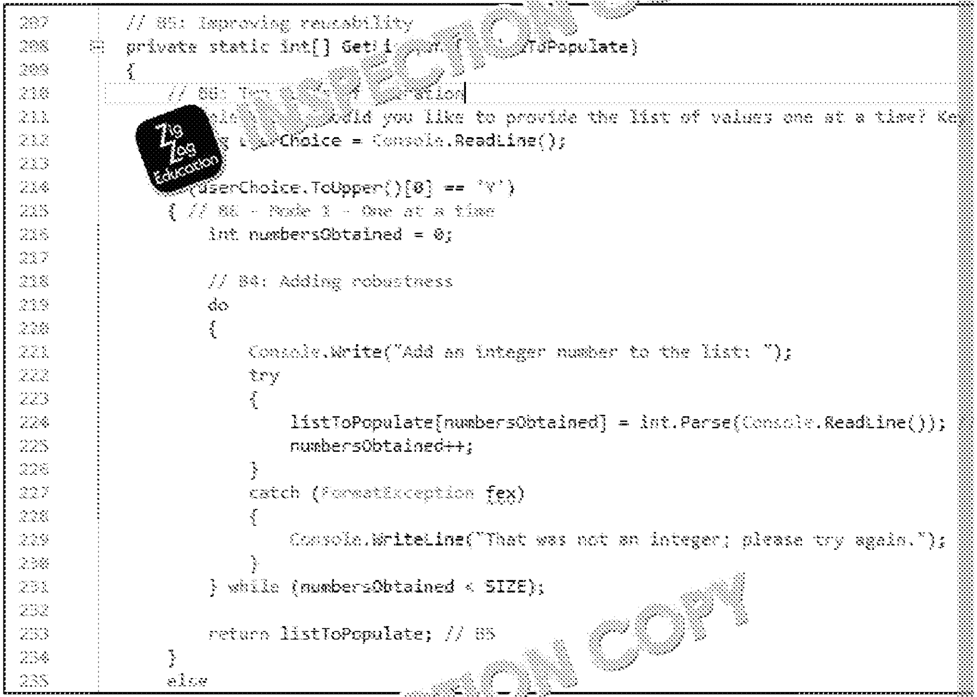
```
        // B5: Improving reusability
        private static int[] GetList(int[] listToPopulate)
        {
            // B6 - Mode 1 - One at a time
                int numbersObtained = 0;

                // B4: Adding robustness
                do
                {
                    Console.Write("Add an integer number to the list: ");
                    try
                    {
                        listToPopulate[numbersObtained] = int.Parse(Console.ReadLine
                        numbersObtained++;
                    }
                    catch (FormatException fex)
                    {
                        Console.WriteLine("That was not an integer; please try agai
                    }
                } while (numbersObtained < SIZE);

                return listToPopulate; // B5
            }
```

B6

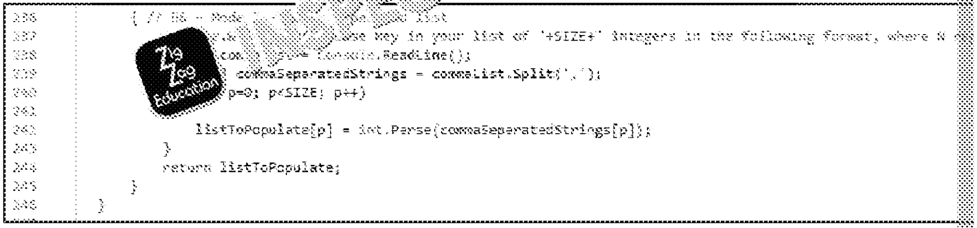
7 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- [1] reading in the user's preferred mode of entry
- [1] identifying if they have chosen option #1
- [1] embedding previously written code successfully into the new selection
- [1] making it clear to the user how to format their comma-separated list
- [1] separating the list into values
- [1] adding the values correctly to the list
- [1] returning the comma-separated list as an array



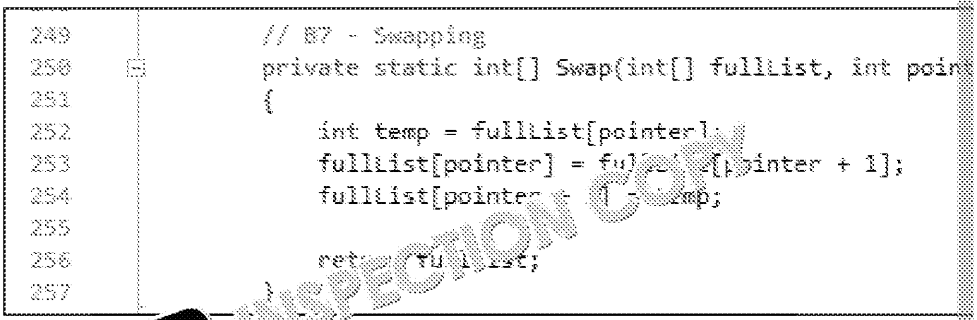
```
    // B5: Improving reusability
    private static int[] GetList(int[] listToPopulate)
    {
        // B6: Two modes of operation
        Console.Write("Would you like to provide the list of values one at a time? Ke
        string userChoice = Console.ReadLine();

        if (userChoice.ToUpper()[0] == 'Y')
        { // B6 - Mode 1 - One at a time
            int numbersObtained = 0;

            // B4: Adding robustness
            do
            {
                Console.Write("Add an integer number to the list: ");
                try
                {
                    listToPopulate[numbersObtained] = int.Parse(Console.ReadLine());
                    numbersObtained++;
                }
                catch (FormatException fex)
                {
                    Console.WriteLine("That was not an integer; please try again.");
                }
            } while (numbersObtained < SIZE);

            return listToPopulate; // B5
        }
        else
```



```
        { // B6 - Mode 2 - Comma-separated list
            Console.WriteLine("Please key in your list of '+SIZE+' integers in the following format, where N 
            string commaList = Console.ReadLine();
            string[] commaSeparatedStrings = commaList.Split(',');
            for (int p=0; p<SIZE; p++)

                listToPopulate[p] = int.Parse(commaSeparatedStrings[p]);
            }
            return listToPopulate;
        }
    }
```

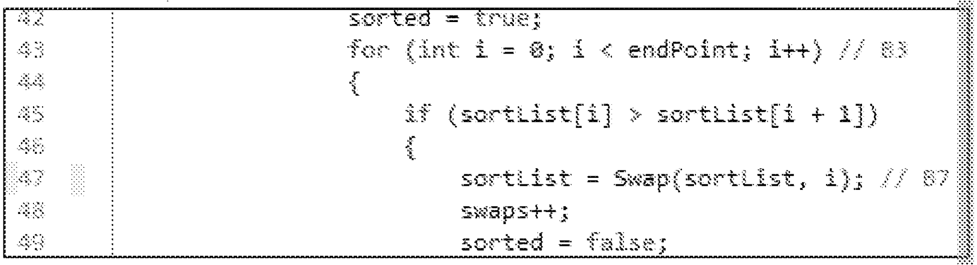
▩ B7

2 marks (as shown below, or equivalent code):



```
        // B7 - Swapping
        private static int[] Swap(int[] fullList, int poin
        {
            int temp = fullList[pointer];
            fullList[pointer] = fullList[pointer + 1];
            fullList[pointer + 1] = temp;

            return fullList;
        }
```

1 mark (called correctly from within the bubble sort method):



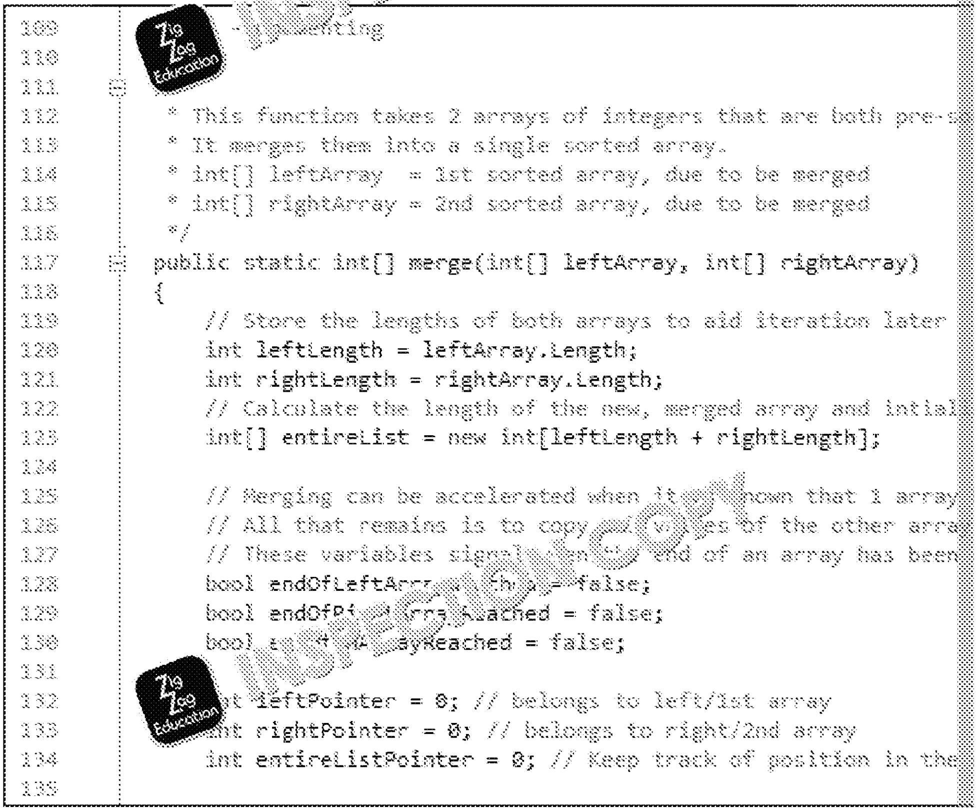
```
                sorted = true;
                for (int i = 0; i < endPoint; i++) // B3
                {
                    if (sortList[i] > sortList[i + 1])
                    {
                        sortList = Swap(sortList, i); // B7
                        swaps++;
                        sorted = false;
```

▩ B8

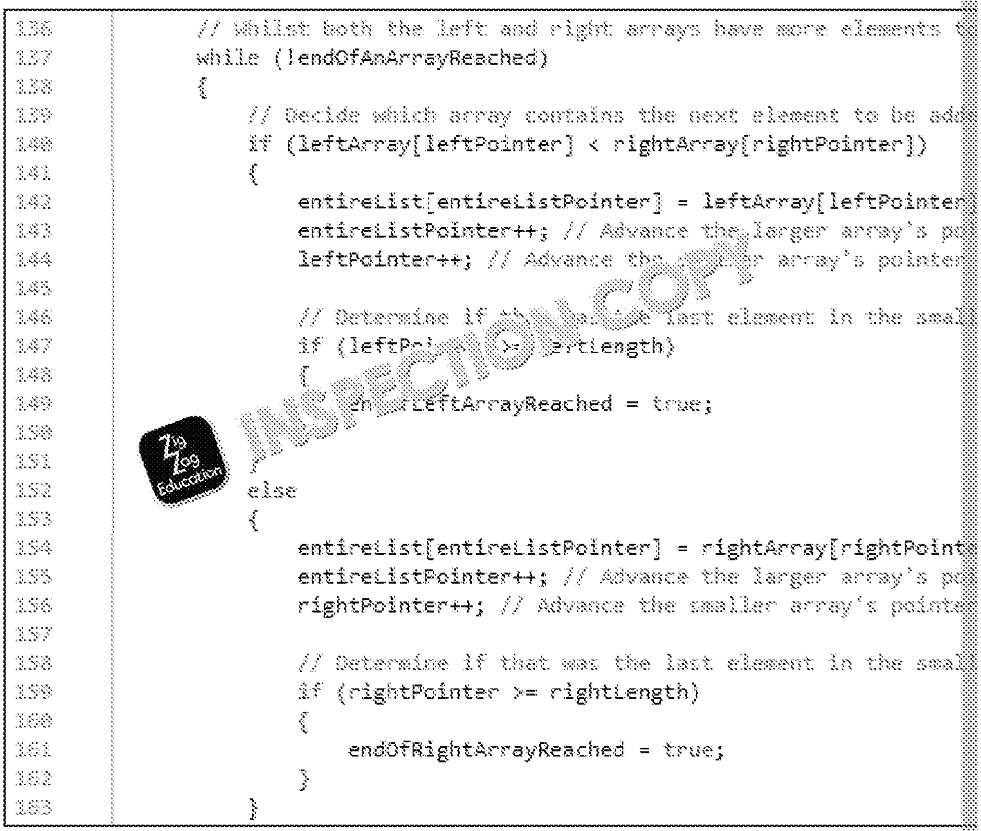
6 marks for overall quality:

Award marks as follows:

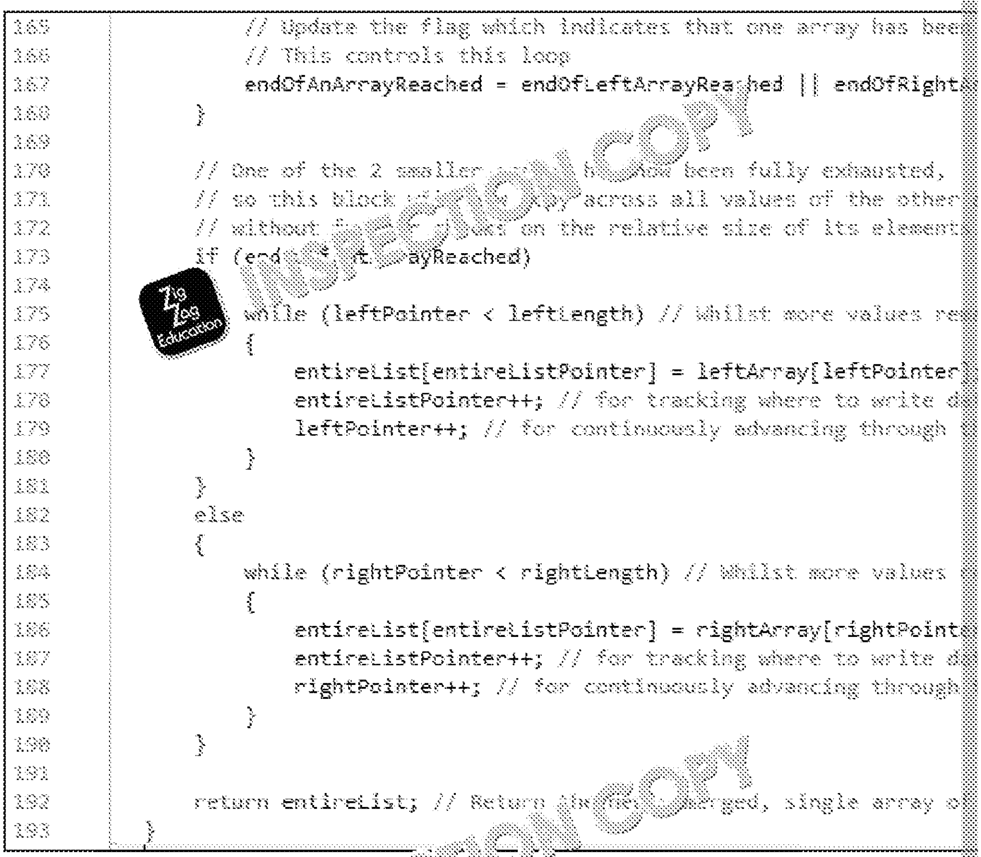
- [0] for minimal commenting
- [2] for clear comments but not many of them
- [4] for medium-level volumes of comments or inaccurate/wasteful comme
- [6] for excellent comments that truly aid maintainability and readability



```
// B8 - Commenting

/*
 * This function takes 2 arrays of integers that are both pre-s
 * It merges them into a single sorted array.
 * int[] leftArray  = 1st sorted array, due to be merged
 * int[] rightArray = 2nd sorted array, due to be merged
 */
public static int[] merge(int[] leftArray, int[] rightArray)
{
    // Store the lengths of both arrays to aid iteration later
    int leftLength = leftArray.Length;
    int rightLength = rightArray.Length;
    // Calculate the length of the new, merged array and intial
    int[] entireList = new int[leftLength + rightLength];

    // Merging can be accelerated when it is known that 1 array
    // All that remains is to copy all values of the other arra
    // These variables signal when the end of an array has been
    bool endOfLeftArrayReached = false;
    bool endOfRightArrayReached = false;
    bool endOfAnyArrayReached = false;

    int leftPointer = 0; // belongs to left/1st array
    int rightPointer = 0; // belongs to right/2nd array
    int entireListPointer = 0; // Keep track of position in the

```



```
        // Whilst both the left and right arrays have more elements t
        while (!endOfAnArrayReached)
        {
            // Decide which array contains the next element to be add
            if (leftArray[leftPointer] < rightArray[rightPointer])
            {
                entireList[entireListPointer] = leftArray[leftPointer
                entireListPointer++; // Advance the larger array's po
                leftPointer++; // Advance the smaller array's pointer

                // Determine if that was the last element in the smal
                if (leftPointer >= leftLength)
                {
                    endOfLeftArrayReached = true;
                }
            }
            else
            {
                entireList[entireListPointer] = rightArray[rightPoint
                entireListPointer++; // Advance the larger array's po
                rightPointer++; // Advance the smaller array's pointe

                // Determine if that was the last element in the smal
                if (rightPointer >= rightLength)
                {
                    endOfRightArrayReached = true;
                }
            }
```



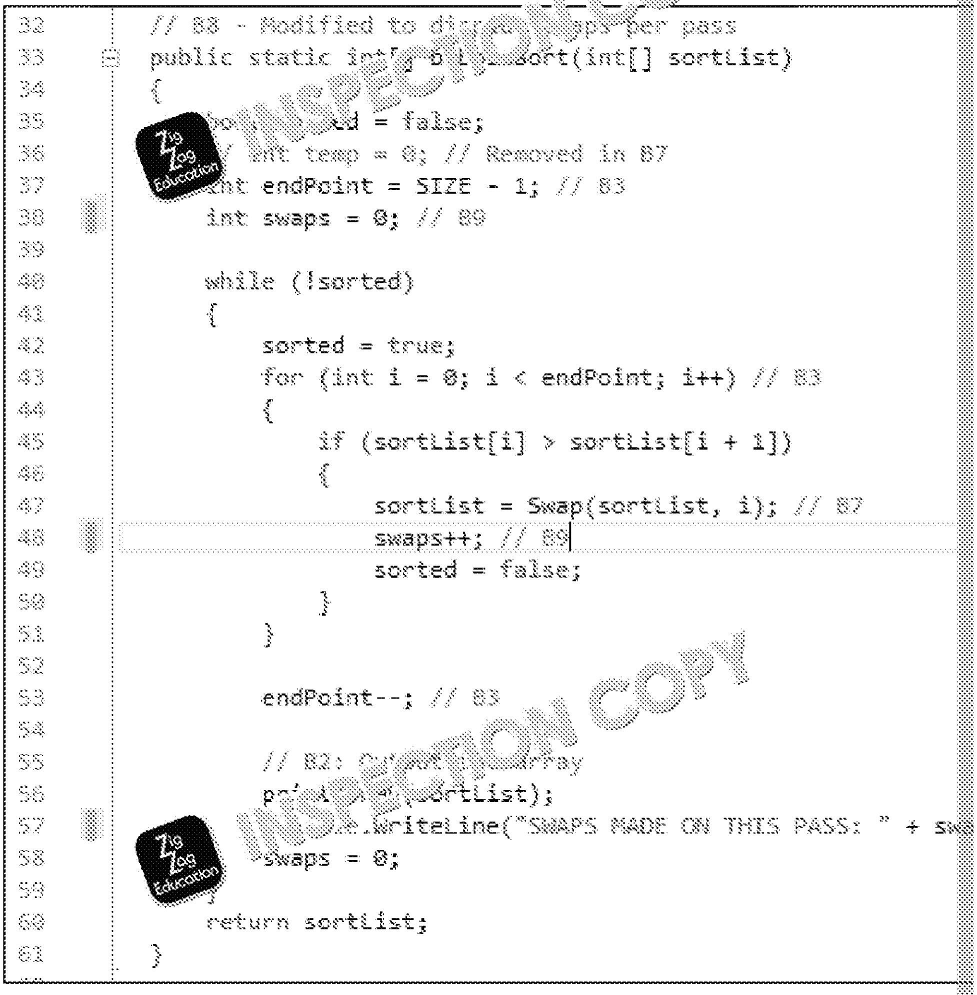
```
            // Update the flag which indicates that one array has bee
            // This controls this loop
            endOfAnArrayReached = endOfLeftArrayReached || endOfRight
        }

        // One of the 2 smaller arrays has now been fully exhausted,
        // so this block will now copy across all values of the other
        // without further checks on the relative size of its element
        if (endOfRightArrayReached)
        {
            while (leftPointer < leftLength) // Whilst more values re
            {
                entireList[entireListPointer] = leftArray[leftPointer
                entireListPointer++; // for tracking where to write d
                leftPointer++; // for continuously advancing through
            }
        }
        else
        {
            while (rightPointer < rightLength) // Whilst more values
            {
                entireList[entireListPointer] = rightArray[rightPoint
                entireListPointer++; // for tracking where to write d
                rightPointer++; // for continuously advancing through
            }
        }

        return entireList; // Return the fully merged, single array o
    }
```

■ B9

4 marks available for modifying the code as described:

Marks could be awarded for the swaps variable being:

- correctly created as a local variable
- correctly
- correctly changed during iterations
- output within a meaningful statement



```
// B8 - Modified to display swaps per pass
public static int[] BubbleSort(int[] sortList)
{
    bool sorted = false;
    // int temp = 0; // Removed in B7
    int endPoint = SIZE - 1; // B3
    int swaps = 0; // B9

    while (!sorted)
    {
        sorted = true;
        for (int i = 0; i < endPoint; i++) // B3
        {
            if (sortList[i] > sortList[i + 1])
            {
                sortList = Swap(sortList, i); // B7
                swaps++; // B9
                sorted = false;
            }
        }

        endPoint--; // B3

        // B2: Output the array
        PrintArray(sortList);
        Console.WriteLine("SWAPS MADE ON THIS PASS: " + swa
        swaps = 0;
    }
    return sortList;
}
```

Exercise 3 – Towers of Hanoi

Section A

▩ A1

1 mark:
Line 9

▩ A2

1 mark:
40%

▩ A3

3 marks (1 mark for identifying data structure as a stack; 1 mark for describing a sta
1 mark for describing what it means for a data structure to be FILO):

This behaviour is represented by a stack data structure.
A stack is a First-In, Last-Out (FILO) data structure (can also say LIFO),
meaning that only the most recently stored data can be accessed.

▩ A4

1 mark per relevant point in the explanation (up to 3):

This is achieved through the use of multiple constructors.
All three require a constructor to build a new object, but by default any newly buil
integers, as well as a tower number.
It is only Tower #1 that needs further information, so it is built using a different co
This is possible in OOP thanks to method overloading.

▩ A5

1 mark for each part of the explanation (up to 4):

This would lead to a string being entered which cannot be turned into digits...
... resulting in an exception being thrown,
... specifically a FormatException,
... which would crash the program as no exception handling has been built in.

▩ A6

2 marks for quoting code and giving an explanation in prose; limit to 1 if no referen

Line 45 solves this:

```
if (StartTower.CheckTower().Count != 0)
```

...and the corresponding ELSE block on Line 64 absorbs the cases where there are

```
    else
            {
                Console.WriteLine("Invalid move: There are no disc
}
```

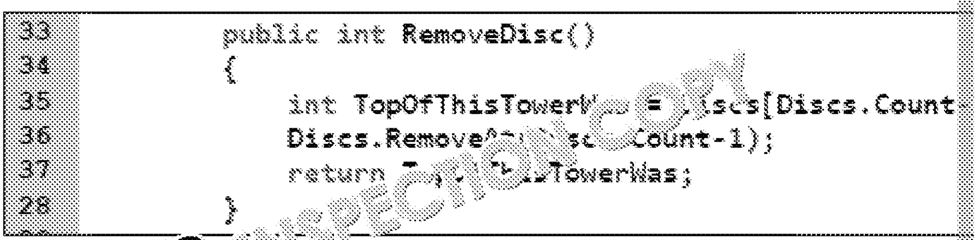
▩ A7

3 marks for communicating the idea that the OR is needed to avoid an exception being

If the tower is EMPTY...
... or if the value being added to it is smaller than the current top of the tower...
... then proceed with the move.
This is required to implement the key rule of the Towers of Hanoi.
The first part of the OR is required because no index notation can be used to read
tower is EMPTY, the OR expression 'short circuits' and the part on the right is neve
TRUE.

▩ A8

4 marks (1 mark for explaining that the value of the top disc is returned; 1 mark for removed from the tower; 1 mark for explaining that -1 refers to the right of the list, mark for perfect accuracy of explanation, with no ambiguity):



```
33          public int RemoveDisc()
34          {
35              int TopOfThisTowerWas = Discs[Discs.Count
36              Discs.RemoveAt(Discs.Count-1);
37              return TopOfThisTowerWas;
28          }
```

The code on line 37 on the above screenshot of the program is used to return the
The code on line 36 removes that disc from the tower…
… using the top of the stack as a pointer, generated from the size of the stack but
The index value of -1 is used to remove the rightmost element of the list, which is
right-hand side being the top of the stack (or tower).

▩ A9

Up to 2 marks:

Encapsulation means grouping together related data and subroutines and controll by which parts of the program by hiding the details of implementation. Encapsula to be modified without affecting the entire program, as the implementation of me changing how the methods are used.

▩ A10

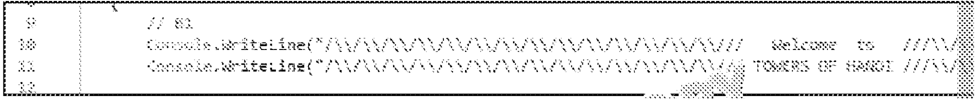
Up to 3 marks for full explanation; limit to 2 if the word (im)mutability is not used:

[0] Arrays & lists use numeric indexes.
[0] Arrays & lists hold values with consistent/matching/same data types.
[1] Arrays have an immutable structure BUT…
[1] … lists can grow and shrink.

Section B

B1

1 mark available for modifying the code as shown (or equivalent code):

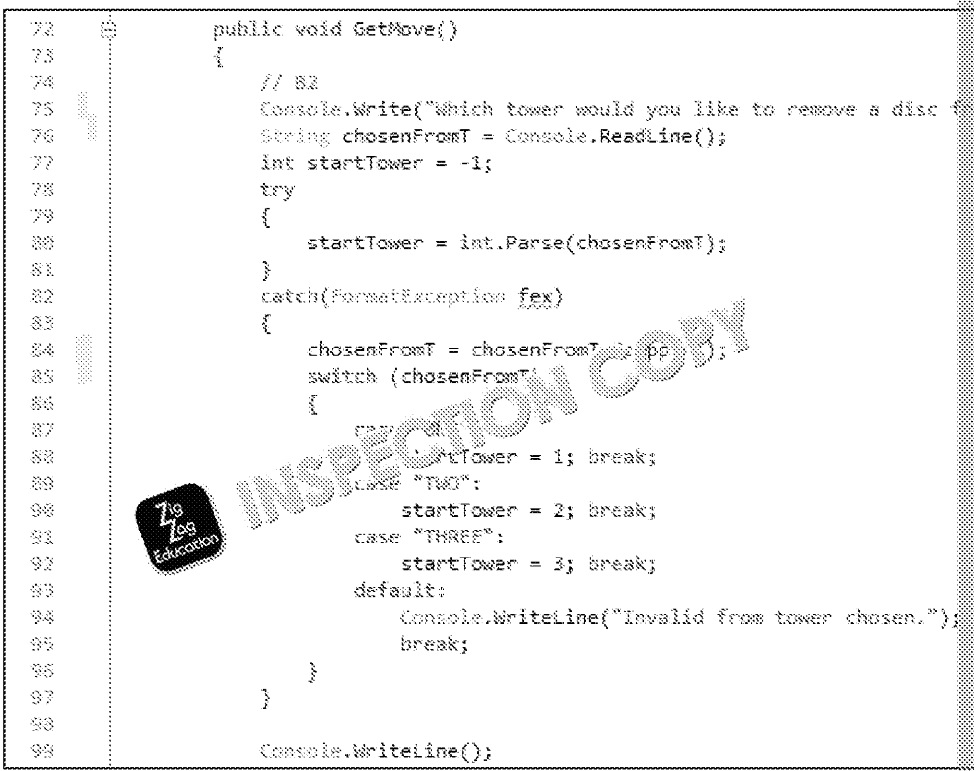


```
// B1
Console.WriteLine("/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\//   Welcome  to   ///\\/
Console.WriteLine("/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\// TOWERS OF HANOI ///\\/
```

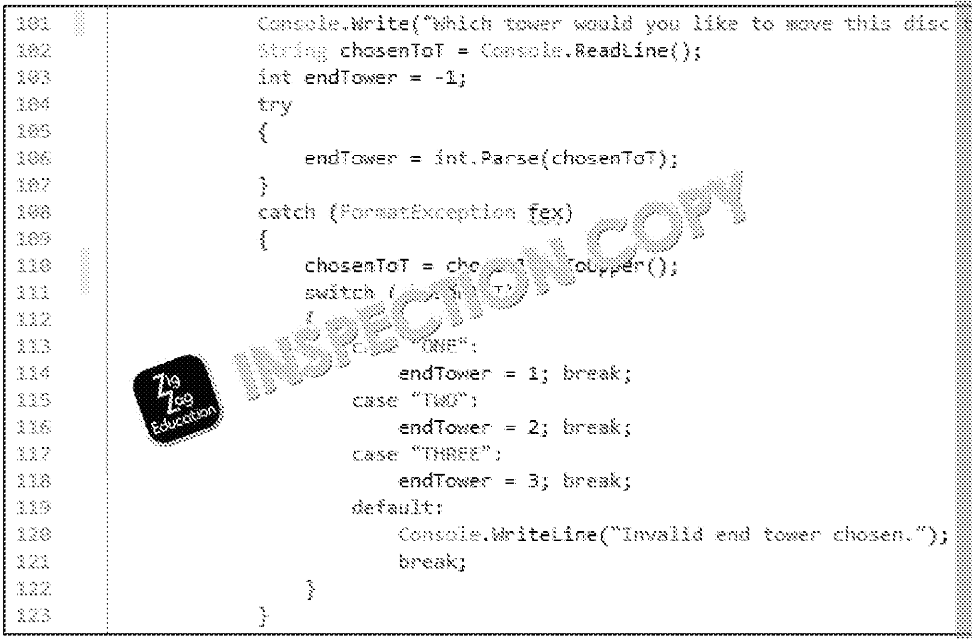
B2

1 mark available for:

- meaningful prompts to the user
- trying to parse but using try–catch OR alternative approach, e.g. looking f
- coping with inputs given in upper case / lower case
- successful conversion from words to numbers
- robust defence against invalid inputs
- replicating the work for both inputs



```
        public void GetMove()
        {
            // B2
            Console.Write("Which tower would you like to remove a disc f
            String chosenFromT = Console.ReadLine();
            int startTower = -1;
            try
            {
                startTower = int.Parse(chosenFromT);
            }
            catch(FormatException fex)
            {
                chosenFromT = chosenFromT.ToUpper();
                switch (chosenFromT)
                {
                    case "ONE":
                        startTower = 1; break;
                    case "TWO":
                        startTower = 2; break;
                    case "THREE":
                        startTower = 3; break;
                    default:
                        Console.WriteLine("Invalid from tower chosen.");
                        break;
                }
            }

            Console.WriteLine();
```

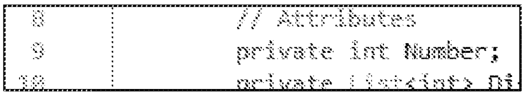


```
            Console.Write("Which tower would you like to move this disc 
            String chosenToT = Console.ReadLine();
            int endTower = -1;
            try
            {
                endTower = int.Parse(chosenToT);
            }
            catch (FormatException fex)
            {
                chosenToT = chosenToT.ToUpper();
                switch (chosenToT)
                {
                    case "ONE":
                        endTower = 1; break;
                    case "TWO":
                        endTower = 2; break;
                    case "THREE":
                        endTower = 3; break;
                    default:
                        Console.WriteLine("Invalid end tower chosen.");
                        break;
                }
            }
```

B3

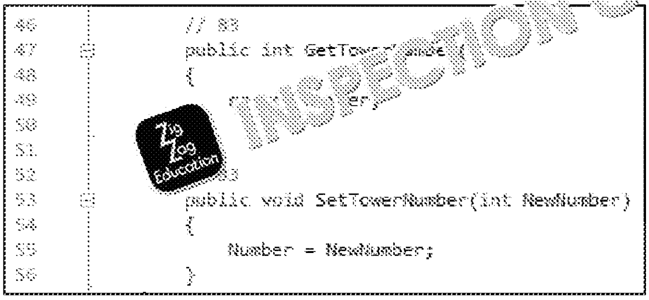
1 mark available from each method (getter/accessor and setter/mutator):

Private visibility [0 marks in itself]:



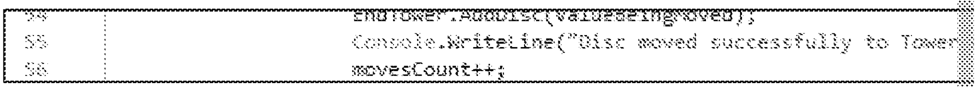
```
        // Attributes
        private int Number;
        private List<int> Di
```

Methods required [1 mark each]:



```
        // B3
        public int GetTowerNumber()
        {
            return Number;
        }

        // B3
        public void SetTowerNumber(int NewNumber)
        {
            Number = NewNumber;
        }
```

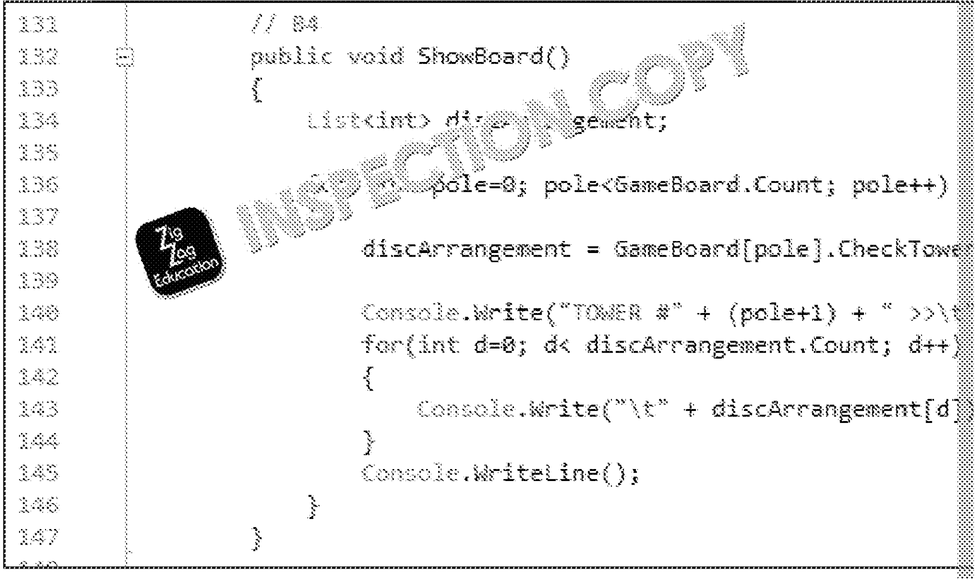
For example, in `Game.cs` it has been used [1 mark]:



```
                    EndTower.AddDisc(valueBeingMoved);
                    Console.WriteLine("Disc moved successfully to Tower
                    movesCount++;
```

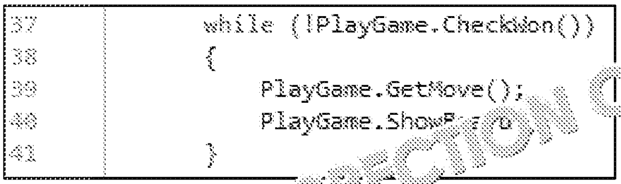
B4

4 marks available for developing the method, +1 mark for calling it:



```
        // B4
        public void ShowBoard()
        {
            List<int> discArrangement;

            for(int pole=0; pole<GameBoard.Count; pole++)

                discArrangement = GameBoard[pole].CheckTowe

                Console.Write("TOWER #" + (pole+1) + " >>\t
                for(int d=0; d< discArrangement.Count; d++)
                {
                    Console.Write("\t" + discArrangement[d]
                }
                Console.WriteLine();
            }
        }
```

Method call in `Program.cs`:

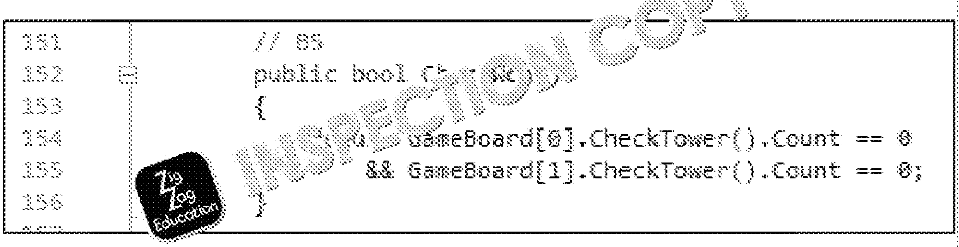


```
            while (!PlayGame.CheckWon())
            {
                PlayGame.GetMove();
                PlayGame.ShowBoard();
            }
```

▩ B5

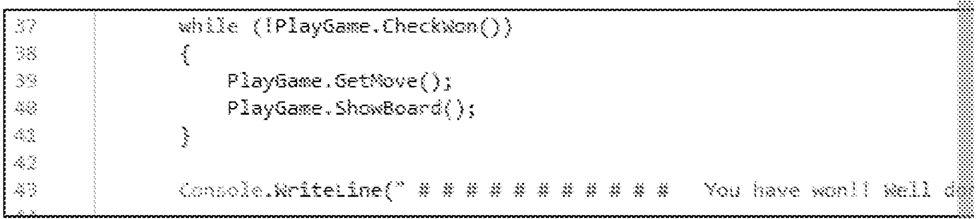
4 marks available for:

- checking if the game has been won (all discs on Peg #3 OR both other pe
- returning a suitable value (MUST be Boolean)
- calling the method iteratively using NOT
- suitable messages in `Program.cs`



```
        // B5
        public bool CheckWon()
        {
            return GameBoard[0].CheckTower().Count == 0
                && GameBoard[1].CheckTower().Count == 0;
        }
```

In the `Main` method:



```
            while (!PlayGame.CheckWon())
            {
                PlayGame.GetMove();
                PlayGame.ShowBoard();
            }

            Console.WriteLine(" # # # # # # # # # # #   You have won!! Well d
```

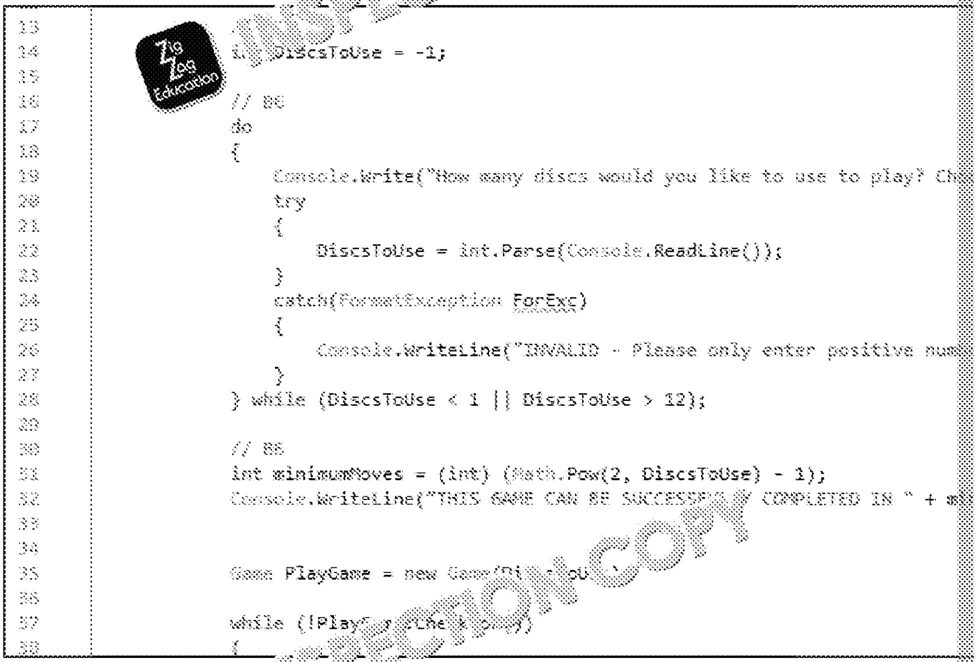
▩ B6

6 marks available for reading in the number of discs and validating it, then proceedi

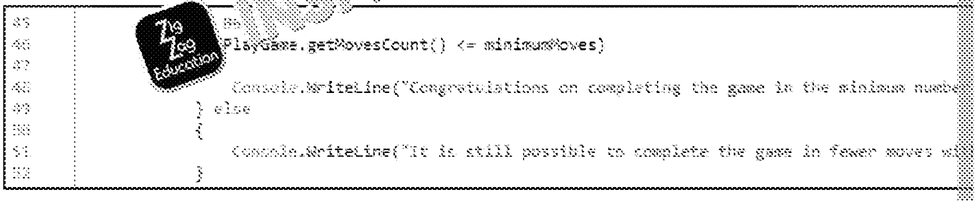
Marks could be awarded EARLY in `Program.cs` for validation [2] and outputs [2]

Marks can be awarded LATER in `Program.cs` for responding to how many moves

In `Program.cs`:



```

            int DiscsToUse = -1;

            // B6
            do
            {
                Console.Write("How many discs would you like to use to play? Ch
                try
                {
                    DiscsToUse = int.Parse(Console.ReadLine());
                }
                catch(FormatException ForExc)
                {
                    Console.WriteLine("INVALID - Please only enter positive num
                }
            } while (DiscsToUse < 1 || DiscsToUse > 12);

            // B6
            int minimumMoves = (int) (Math.Pow(2, DiscsToUse) - 1);
            Console.WriteLine("THIS GAME CAN BE SUCCESSFULLY COMPLETED IN " + m


            Game PlayGame = new Game(DiscsToUse);

            while (!PlayGame.CheckWon())
            {
```



```
            // B6
            if (PlayGame.getMovesCount() <= minimumMoves)
            {
                Console.WriteLine("Congratulations on completing the game in the minimum numbe
            } else
            {
                Console.WriteLine("It is still possible to complete the game in fewer moves wi
            }
```

Exercise 4 – Sorting Queues

Section A

■ A1

1 mark for saying three members:
3 members = 2 attributes + 1 method

■ A2

1 mark for anywhere that uses parameter passing:
Line 15/42/47/53

■ A3

2 marks (1 mark for explaining that a queue is FIFO (First-In, First-Out); 1 mark for e (First-In, Last-Out)):

The first element placed into a queue is the first element to be removed from the q placed into a stack is the last element to be removed from the stack.

■ A4

1 mark for explaining the cause of the error:

Data values can join a queue only at the tail of the queue. This method must be a item in the queue and enqueue subsequent data values there.

■ A5

1 mark for explaining that it outputs a useful/meaningful number of hyphens:

Earlier in the program a series of hyphens was used to denote a heading.
The FOR loop is used to produce a sequence of hyphens such that the sequence h heading text (which contained hyphens as well as the heading itself).

■ A6

1 mark for explaining wh
A newly created node has no successor (node that comes after it) in the queue, so object and therefore takes the value *Null*.

■ A7

3 marks (1 mark for explaining that an array cannot change its size at run-time; 1 m change its size while running to match the number of elements needed; 1 mark for e advantage over fixed-length arrays):

A fixed-length array has to declare the number of memory locations it will use, and t
A list has a dynamic size, so it doesn't take up more memory than it needs. This m be more efficient than if they used arrays instead.
Array immutability is thus a barrier here.

■ A8

1 mark for explaining how pointers need to be updated:

The current node at the tail of the queue needs to be found by traversing all poin
The new node must have been instantiated.
The tail node's pointer needs to point to the newly added node.

■ A9

3 marks (1 mark for explaining that a circular queue has a fixed size; 1 mark for exp start and end pointers; 1 mark for explaining that elements are placed at the front o room remaining at the rear of the queue):

A circular queue is a queue of a fixed length that uses start and end pointers to po last elements. If there is no space at the back of a circular queue, but there is still s new elements are added to the front of the queue, and the end pointer is moved t

▩ A10

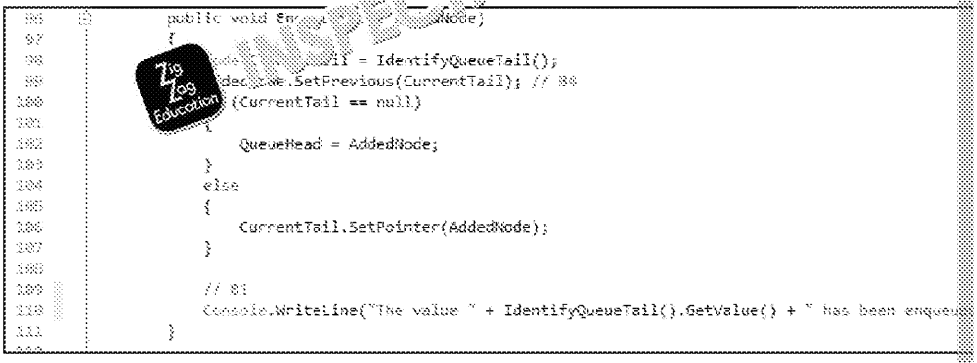
2 marks (1 mark for identifying that the queue is a dynamic queue (no fixed length) detect whether it is circular or dynamic):

The queue in the program is dynamic (not circular) because it has no end pointer a

Section B

▩ B1

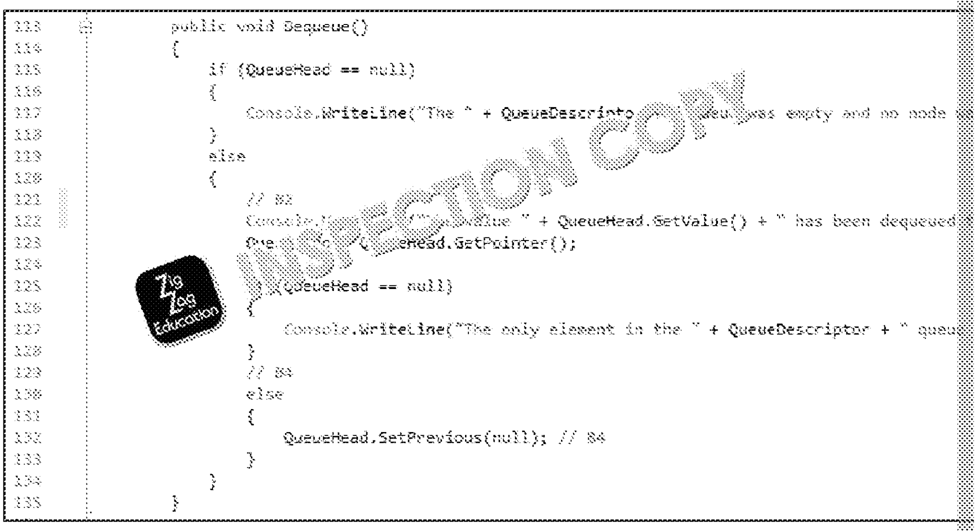
1 mark available for modifying the code as shown (or equivalent code):



```
        public void Enqueue(Node AddedNode)
        {
            Node CurrentTail = IdentifyQueueTail();
            AddedNode.SetPrevious(CurrentTail); // B4
            if (CurrentTail == null)
            {
                QueueHead = AddedNode;
            }
            else
            {
                CurrentTail.SetPointer(AddedNode);
            }

            // B1
            Console.WriteLine("The value " + IdentifyQueueTail().GetValue() + " has been enqueu
        }
```

▩ B2

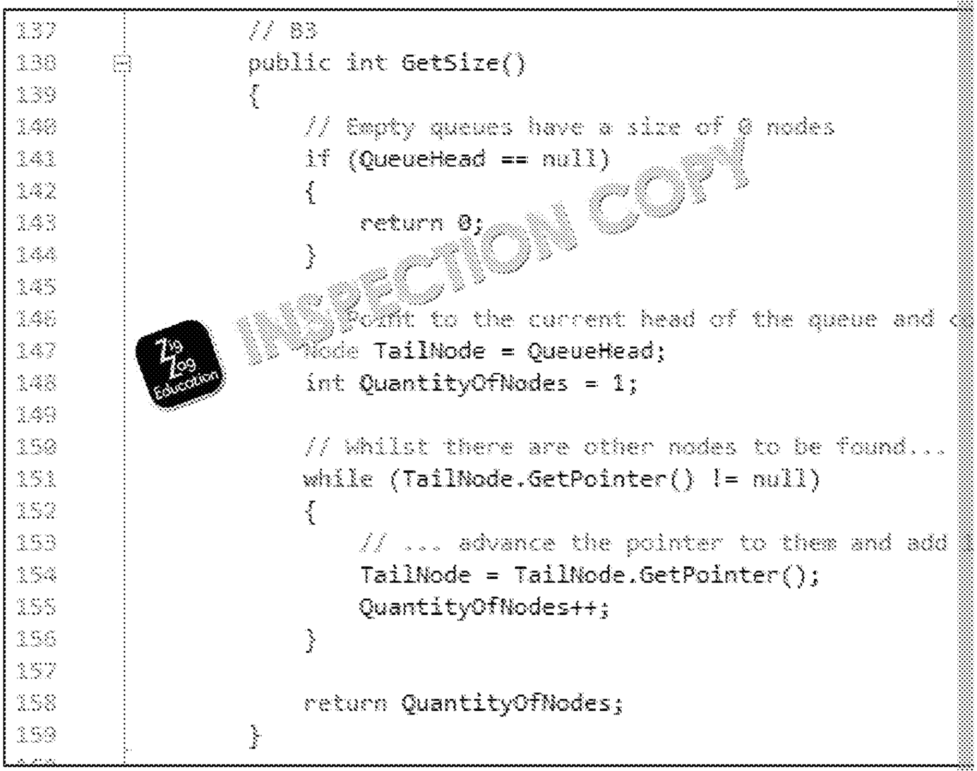
1 mark available for modifying the code as shown (or equivalent code):



```
        public void Dequeue()
        {
            if (QueueHead == null)
            {
                Console.WriteLine("The " + QueueDescriptor + " queue was empty and no node 
            }
            else
            {
                // B2
                Console.WriteLine("The value " + QueueHead.GetValue() + " has been dequeued
                QueueHead = QueueHead.GetPointer();

                if (QueueHead == null)
                {
                    Console.WriteLine("The only element in the " + QueueDescriptor + " queu
                }
                // B4
                else
                {
                    QueueHead.SetPrevious(null); // B4
                }
            }
        }
```

B3

5 marks available for modifying the code as shown (or equivalent code):



```
        // B3
        public int GetSize()
        {
            // Empty queues have a size of 0 nodes
            if (QueueHead == null)
            {
                return 0;
            }

            // Point to the current head of the queue and c
            Node TailNode = QueueHead;
            int QuantityOfNodes = 1;

            // Whilst there are other nodes to be found...
            while (TailNode.GetPointer() != null)
            {
                // ... advance the pointer to them and add
                TailNode = TailNode.GetPointer();
                QuantityOfNodes++;
            }

            return QuantityOfNodes;
        }
```

Marks can be awarded for:

- handling null pointers
- setting the head of the queue to be the tail which deals with lists of lengt
- advancing the tail pointer while there are more nodes in the list (which wo length of 1)
- keeping a running total of how many nodes were found
- returning an answer accurately

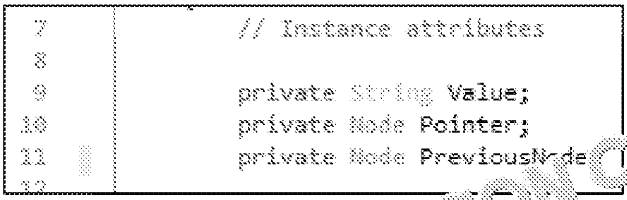
B4

6 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

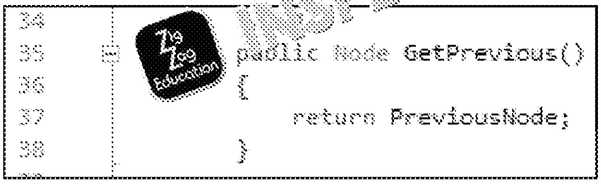
- adding a `previousNode` attribute to the `Node` class
- setting the `previousNode` attribute according to a parameter passed into t
- modifying the `addValue` procedure to correctly set the `previousNode` att

In `Node.cs`: [0 marks] for this:



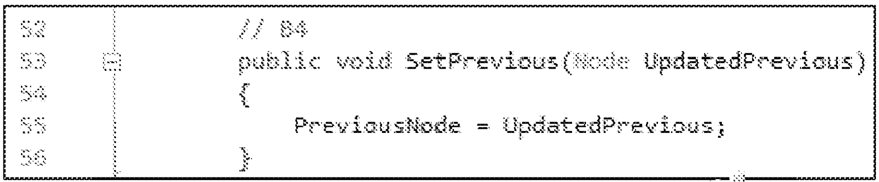
```
        // Instance attributes

        private String Value;
        private Node Pointer;
        private Node PreviousNode
```

[1 mark] for this:



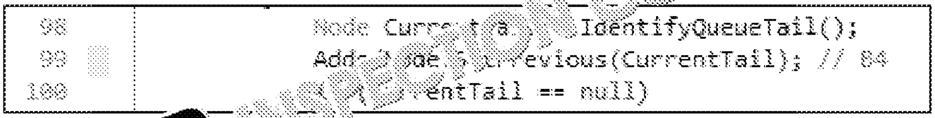
```
        public Node GetPrevious()
        {
            return PreviousNode;
        }
```

[1 mark] for this:



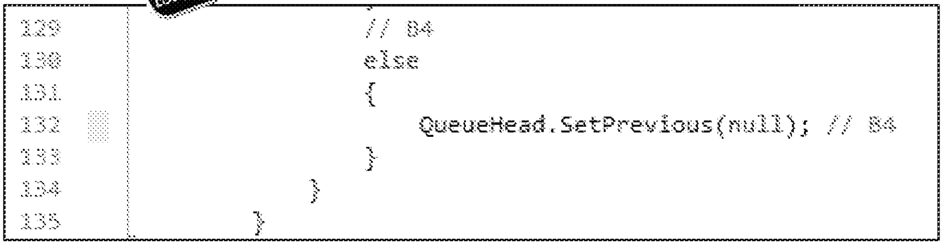
```
        // B4
        public void SetPrevious(Node UpdatedPrevious)
        {
            PreviousNode = UpdatedPrevious;
        }
```

In `Queue.cs`: [1 mark] for this:



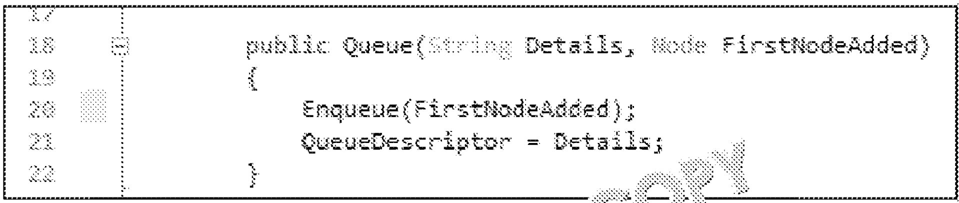
```
                Node CurrentTail = IdentifyQueueTail();
                AddedNode.SetPrevious(CurrentTail); // B4
                if (CurrentTail == null)
```

[1 mark] for this:



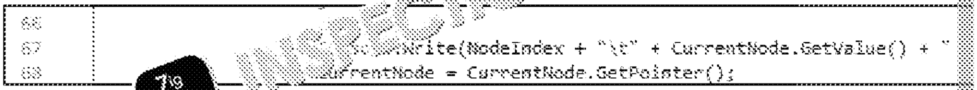
```
                    // B4
                    else
                    {
                        QueueHead.SetPrevious(null); // B4
                    }
                }
            }
```

[1 mark] for this:



```
        public Queue(String Details, Node FirstNodeAdded)
        {
            Enqueue(FirstNodeAdded);
            QueueDescriptor = Details;
        }
```

Suitable output from `Queue.PrintQueue`: [1 mark] for this:

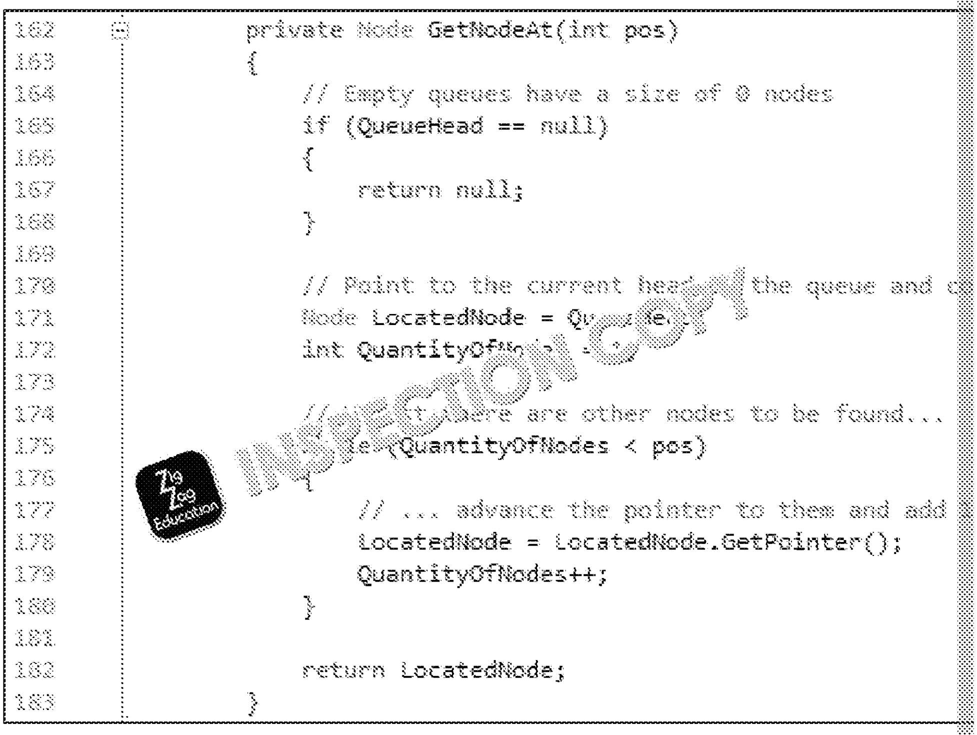


```

                Console.Write(NodeIndex + "\t" + CurrentNode.GetValue() + "
                CurrentNode = CurrentNode.GetPointer();
```

▩ B5

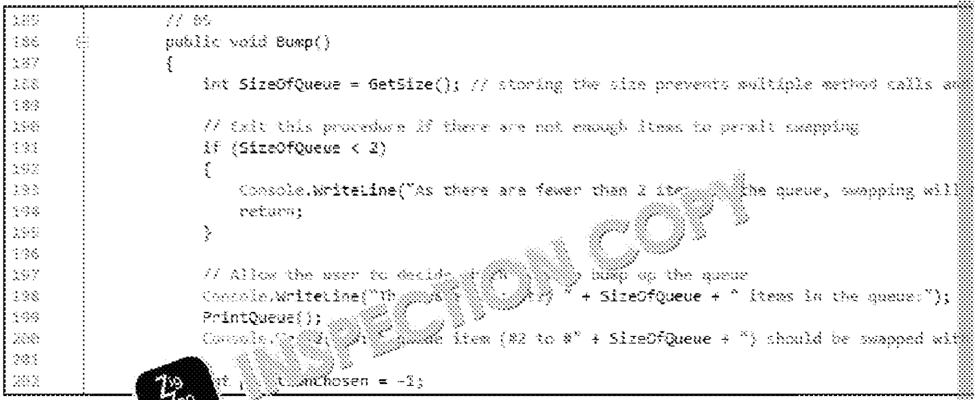
6 marks available for modifying the code as shown (or equivalent code):

[2 marks] for successfully producing the `GetNodeAt(n)` function:

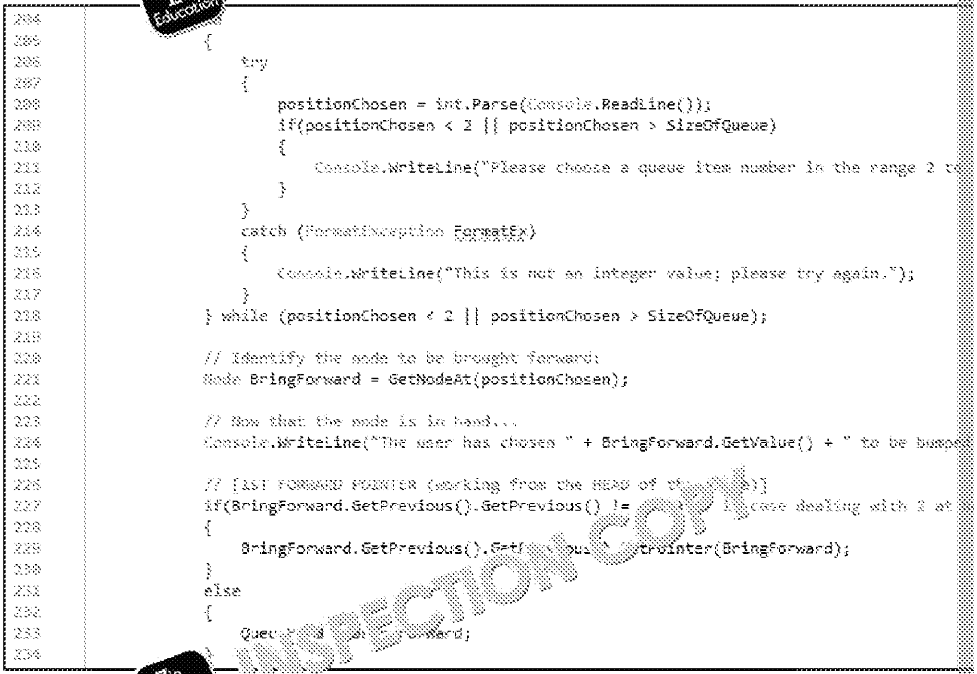


```
        private Node GetNodeAt(int pos)
        {
            // Empty queues have a size of 0 nodes
            if (QueueHead == null)
            {
                return null;
            }

            // Point to the current head of the queue and c
            Node LocatedNode = QueueHead;
            int QuantityOfNodes = 0;

            // While there are other nodes to be found...
            while (QuantityOfNodes < pos)
            {
                // ... advance the pointer to them and add
                LocatedNode = LocatedNode.GetPointer();
                QuantityOfNodes++;
            }

            return LocatedNode;
        }
```

[3 marks] for the `Bump()` procedure handling short queues effectively AND robus

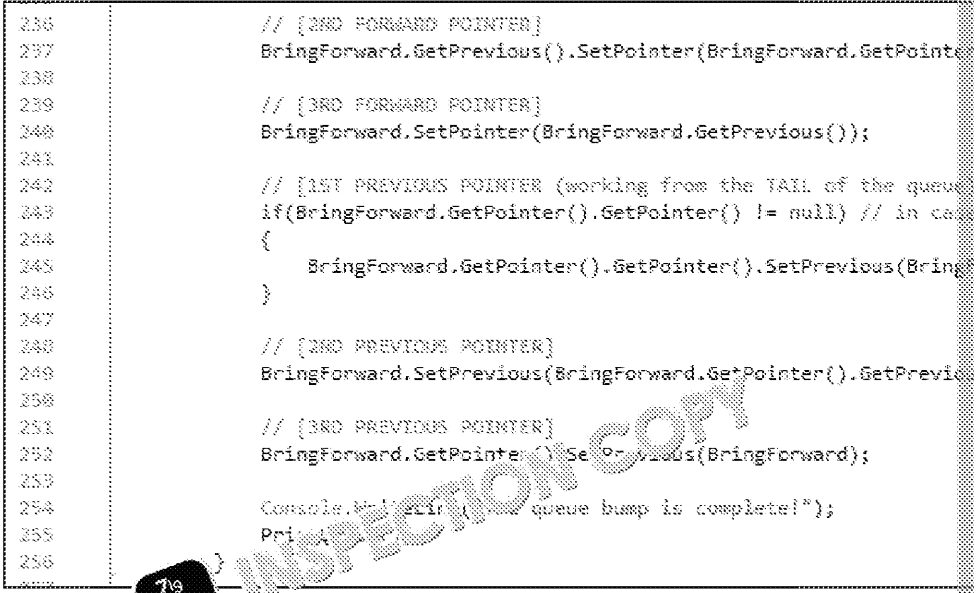


```
        // 8%
        public void Bump()
        {
            int SizeOfQueue = GetSize(); // storing the size prevents multiple method calls a

            // Exit this procedure if there are not enough items to permit swapping
            if (SizeOfQueue < 3)
            {
                Console.WriteLine("As there are fewer than 2 ite[illegible]the queue, swapping will
                return;
            }

            // Allow the user to decide [illegible] bump up the queue
            Console.WriteLine("Th[illegible]" + SizeOfQueue + " items in the queue:");
            PrintQueue();
            Console.[illegible] queue item (#2 to #" + SizeOfQueue + ") should be swapped wit

            [illegible]nChosen = -1;
```



```
            [illegible]
            {
                try
                {
                    positionChosen = int.Parse(Console.ReadLine());
                    if(positionChosen < 2 || positionChosen > SizeOfQueue)
                    {
                        Console.WriteLine("Please choose a queue item number in the range 2 t
                    }
                }
                catch (FormatException FormatEx)
                {
                    Console.WriteLine("This is not an integer value; please try again.");
                }
            } while (positionChosen < 2 || positionChosen > SizeOfQueue);

            // Identify the node to be brought forward:
            Node BringForward = GetNodeAt(positionChosen);

            // Now that the node is in hand...
            Console.WriteLine("The user has chosen " + BringForward.GetValue() + " to be bumpe

            // [1ST FORWARD POINTER (working from the HEAD of th[illegible])]
            if(BringForward.GetPrevious().GetPrevious() != [illegible] case dealing with 2 at
            {
                BringForward.GetPrevious().Get[illegible]Pointer(BringForward);
            }
            else
            {
                Queu[illegible]orward;
            }
```

[4 marks] fo[illegible]mplex job of handling pointers:

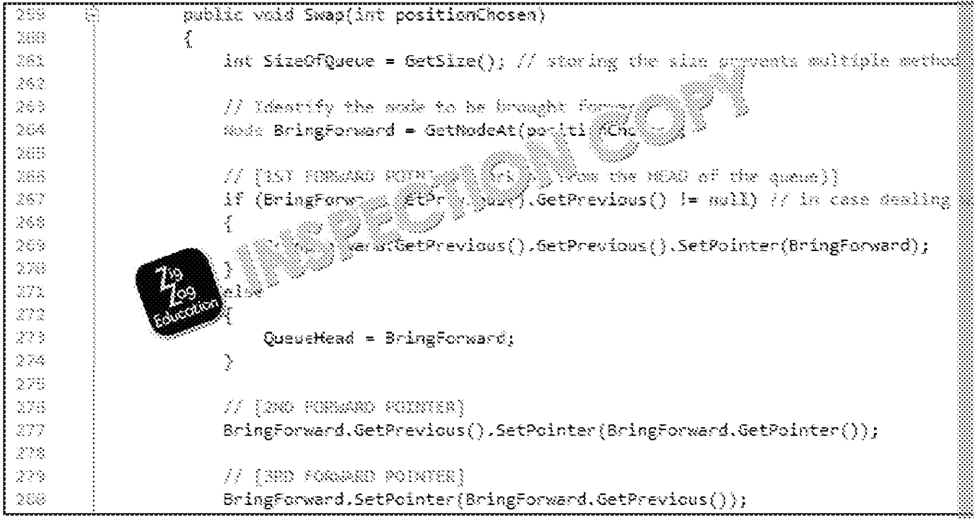


```
            // [2ND FORWARD POINTER]
            BringForward.GetPrevious().SetPointer(BringForward.GetPointe

            // [3RD FORWARD POINTER]
            BringForward.SetPointer(BringForward.GetPrevious());

            // [1ST PREVIOUS POINTER (working from the TAIL of the queu
            if(BringForward.GetPointer().GetPointer() != null) // in ca
            {
                BringForward.GetPointer().GetPointer().SetPrevious(Bring
            }

            // [2ND PREVIOUS POINTER]
            BringForward.SetPrevious(BringForward.GetPointer().GetPrevi

            // [3RD PREVIOUS POINTER]
            BringForward.GetPointe[illegible]Se[illegible]vious(BringForward);

            Console.Wri[illegible] queue bump is complete!");
            Pri[illegible]
        }
```

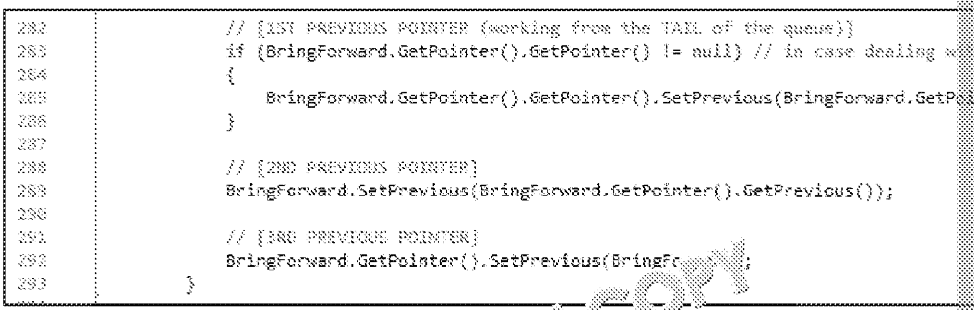
B6

9 marks available for modifying the code as shown (or equivalent code):

[3 marks] for setting up `Swap()` to take in a parameter and proceed as `Bump()` d



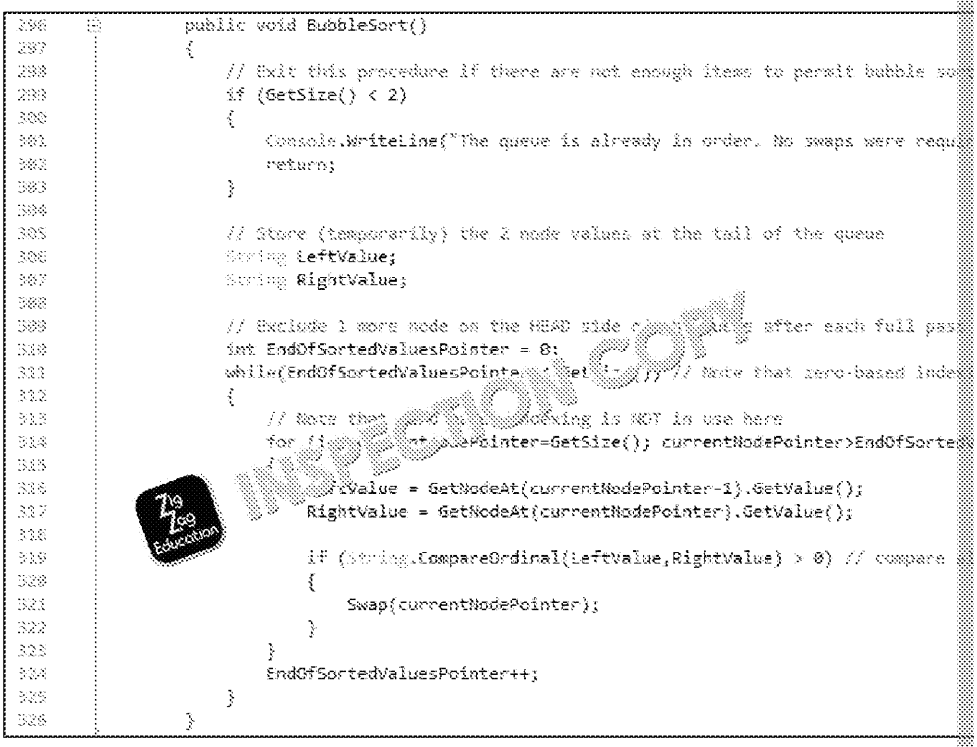
```
        public void Swap(int positionChosen)
        {
            int SizeOfQueue = GetSize(); // storing the size prevents multiple method

            // Identify the node to be brought forwar
            Node BringForward = GetNodeAt(positi[illegible]Cho[illegible]

            // [1ST FORWARD POINT[illegible] from the HEAD of the queue)]
            if (BringForw[illegible]etPr[illegible].GetPrevious() != null) // in case dealing
            {
                [illegible]ward.GetPrevious().GetPrevious().SetPointer(BringForward);
            }
            else
            {
                QueueHead = BringForward;
            }

            // [2ND FORWARD POINTER]
            BringForward.GetPrevious().SetPointer(BringForward.GetPointer());

            // [3RD FORWARD POINTER]
            BringForward.SetPointer(BringForward.GetPrevious());
```



```
            // [1ST PREVIOUS POINTER (working from the TAIL of the queue)]
            if (BringForward.GetPointer().GetPointer() != null) // in case dealing w
            {
                BringForward.GetPointer().GetPointer().SetPrevious(BringForward.GetP
            }

            // [2ND PREVIOUS POINTER]
            BringForward.SetPrevious(BringForward.GetPointer().GetPrevious());

            // [3RD PREVIOUS POINTER]
            BringForward.GetPointer().SetPrevious(BringFo[illegible];
        }
```

[6 marks] for the logic of the `Bubb[illegible]` method:

- [1] for handling li[illegible]than three items
- [1] f[illegible]nt[illegible]e left and right values
- [1] [illegible]cessfully determining their alphabetical order
- [1] [illegible]erating correctly (OUTER loop)
- [1] for iterating correctly (INNER loop)
- [1] for swapping effectively



```
        public void BubbleSort()
        {
            // Exit this procedure if there are not enough items to permit bubble so
            if (GetSize() < 2)
            {
                Console.WriteLine("The queue is already in order. No swaps were requ
                return;
            }

            // Store (temporarily) the 2 node values at the tail of the queue
            String LeftValue;
            String RightValue;

            // Exclude 1 more node on the HEAD side o[illegible]e after each full pas
            int EndOfSortedValuesPointer = 0;
            while(EndOfSortedValuesPointe[illegible]Get[illegible]) // Note that zero-based inde
            {
                // Note that [illegible]ndexing is NOT in use here
                for (i[illegible]nt[illegible]ePointer=GetSize(); currentNodePointer>EndOfSorte
                {
                    [illegible]tValue = GetNodeAt(currentNodePointer-1).GetValue();
                    RightValue = GetNodeAt(currentNodePointer).GetValue();

                    if (String.CompareOrdinal(LeftValue,RightValue) > 0) // compare
                    {
                        Swap(currentNodePointer);
                    }
                }
                EndOfSortedValuesPointer++;
            }
        }
```

Exercise 5 – Draughts

Section A

▩ A1

1 mark for:

Line 13 of `Board.cs`

▩ A2

1 mark for 1 example in each line below:

Declared: `Board.cs` Line [illegible], `Piece.cs` Line 7/8
Initialised: `Board.cs` Line 13, `Piece.cs` Line 14/15
Read: `Board.cs` Line 21, `Piece.cs` Line 22/27

▩ A3

1 mark for giving a valid reason to use private methods; for example:

A method may be made public so that it can be accessed from other parts of the p
which the attribute is declared).
Setting `PlacePieces` to private is done as it is needed only from within this class
... so to protect programmers from accidentally misusing the method in the wrong

▩ A4

1 mark for each point:

It needs one so that its colour can be determined at the point of need [1], ... but it c
because all pieces will be initialised without being kinged [1] and the king=false sett

▩ A5

1 mark per point; must include first point:

Option 2 is correct [1 – essential]
The board is a 2D array of Piece objects, many of them null pointers [1]
If we used the pieces storing their position, we would have to be able to iterate th
image of the board, and this would require them to be in a data structure anyway

▩ A6

1 mark per point:

- The main method constructs a new `Board` object; only one for one game
- The `Display()` method of this one board is called from within the main m
- Heading rows get output. [*NB As an extension task, this feature could make BoardSize variable in Section B to make it always output the correct heading*
- The various rows of the board are iterated through using the `GetLength` rows there are...
- ... and within each row the columns are iteratively visited by using the `Ge` higher parameter to work out the number of columns
- Each square gets output as a visualisation including the character code R o for white squares. Black squares have no visible characters. All rows end w determined by calling the accessor method named `GetColour()` of the
- The board ends with a marker ruler.

▩ A7

2 marks (1 mark for explaining why it is bad practice; 1 mark for suggesting what sh

It is bad practice to hard-code in a value that is used throughout the program, as i
at a later point, every instance of the value in the program needs to be changed. I
constant) that contains this value should be used so that if the value needs to be c
in only one place in the program. The name of the constant being visible througho
readability/maintainability.

▩ A8

2 marks (1 mark for stating what it does; 1 mark for explaining how it works); for ex

- MOD checks that a square is black.
- It does this before setting a piece on the board. If a square is white, it dro
- It works by adding the row and column numbers. All white squares have a using the result of Sum MOD 2. If it yields a 0, the square is white.

▩ A9

2 marks (1 mark for explaining what inheritance is; 1 mark for explaining why inher relating it to draughts); for example:

Inheritance is when one class takes on the functionality of a different class. It is us code can be reused if multiple classes share the same data or methods. Here, it ca specialised version of a playing piece with some unique attributes and methods w existing `Piece` class.

▩ A10

3 marks (1 mark for explaining what a function is, 1 mark for explaining what a pro what a method is):

A function is a subroutine that returns a value, whereas a procedure is a subrouti method is a subroutine that is part of a certain class (a method can be either a fu

▩ A11

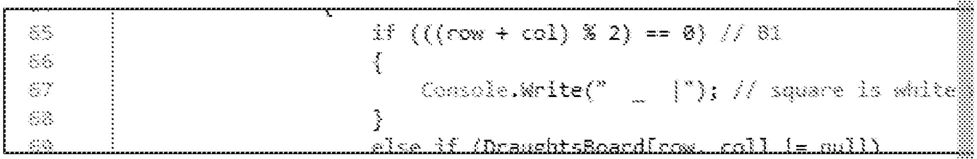
2 marks (1 mark for explaining that a class is a template used to define objects; 1 m an existing instance of a class):

A class is a template of what attribute and methods are needed for objects of that that class that has its own concrete attributes. Here, each board created could ha boards are all objects but their key characteristics are defined in their class from w

Section B

▩ B1

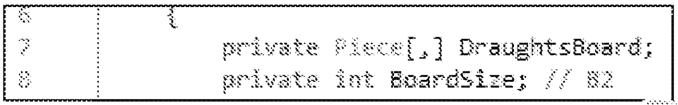
1 mark available for modifying the code as shown (or equivalent code):



```
                    if (((row + col) % 2) == 0) // B1
                    {
                        Console.Write("  _  |"); // square is white
                    }
                    else if (DraughtsBoard[row, col] != null)
```

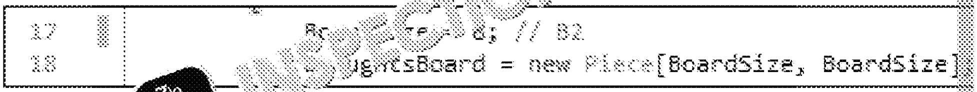
▩ B2

1 mark available for modifying the code as shown (or equivalent code):



```
    {
        private Piece[,] DraughtsBoard;
        private int BoardSize; // B2
```

1 mark for the constructor:

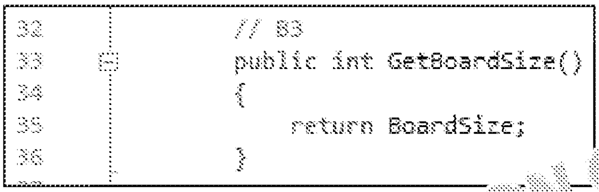


```
            BoardSize = 8; // B2
            DraughtsBoard = new Piece[BoardSize, BoardSize];
```

▩ B3

2 marks available for modifying the code as shown (or equivalent code):

1 mark per meaningful line:



```
        // B3
        public int GetBoardSize()
        {
            return BoardSize;
        }
```

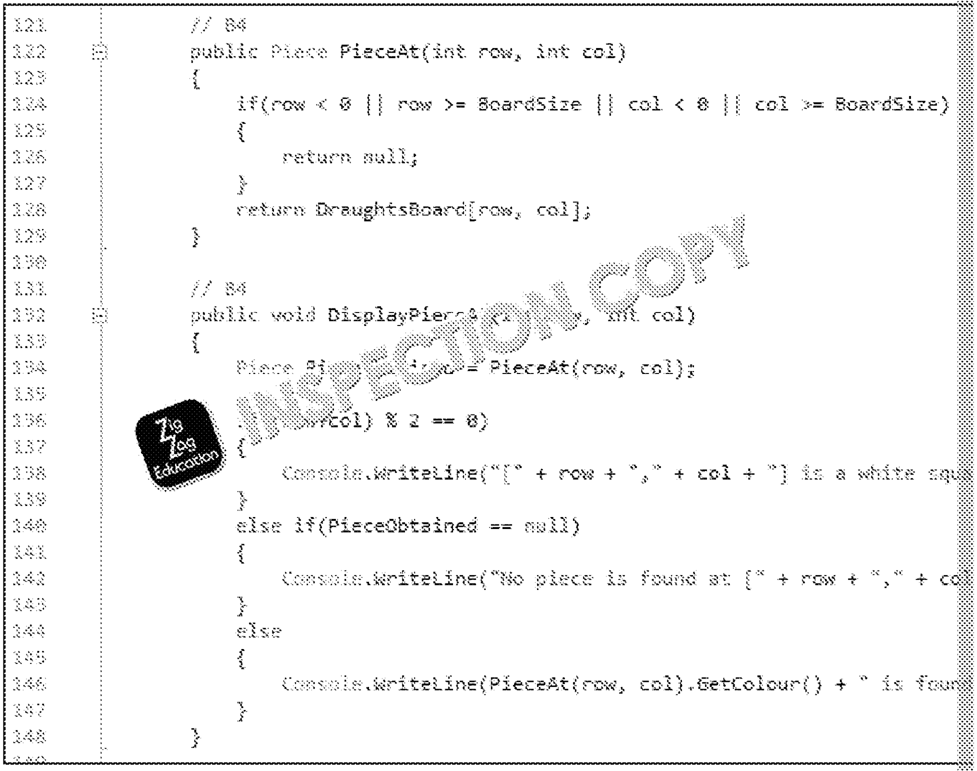
▩ B4

4 marks available for developing the methods (as shown below, or equivalent code)

Marks could be awarded for:

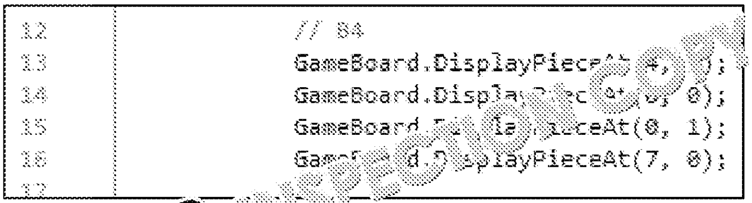
- [1] creating a `PieceAt` function in the `Board` class that takes a row and a
- [1] returning the piece at the position given by the input list, applying su
- [1] checking for nulls and white squares in the `DisplayPieceAt` metho
- [1] displaying the contents of all squares with pieces on them appropriat

In `Board.cs`:



```
        // B4
        public Piece PieceAt(int row, int col)
        {
            if(row < 0 || row >= BoardSize || col < 0 || col >= BoardSize)
            {
                return null;
            }
            return DraughtsBoard[row, col];
        }

        // B4
        public void DisplayPieceAt(int row, int col)
        {
            Piece PieceObtained = PieceAt(row, col);

            if((row+col) % 2 == 0)
            {
                Console.WriteLine("[" + row + "," + col + "] is a white squ
            }
            else if(PieceObtained == null)
            {
                Console.WriteLine("No piece is found at [" + row + "," + co
            }
            else
            {
                Console.WriteLine(PieceAt(row, col).GetColour() + " is foun
            }
        }
```

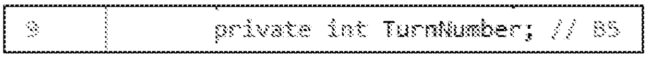
In `Program.cs`:



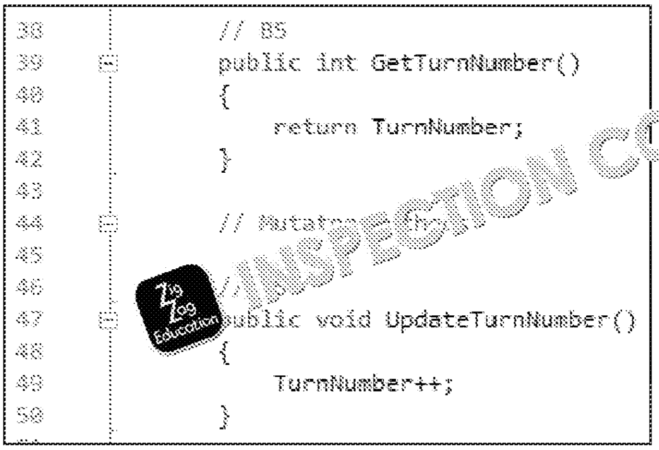
```
            // B4
            GameBoard.DisplayPieceAt(4, [illegible]);
            GameBoard.DisplayPieceAt([illegible], 0);
            GameBoard.DisplayPieceAt(0, 1);
            GameBoard.DisplayPieceAt(7, 0);
```

B5

2 marks for the TurnNumber work:



```
        private int TurnNumber; // B5
```

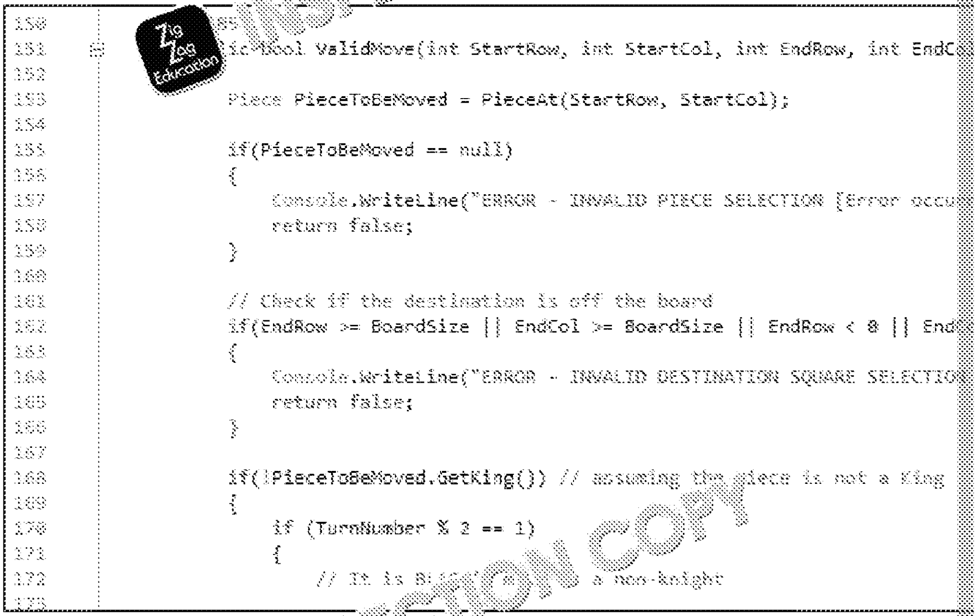


```
        // B5
        public int GetTurnNumber()
        {
            return TurnNumber;
        }

        // Mutat[illegible]

        //
        public void UpdateTurnNumber()
        {
            TurnNumber++;
        }
```

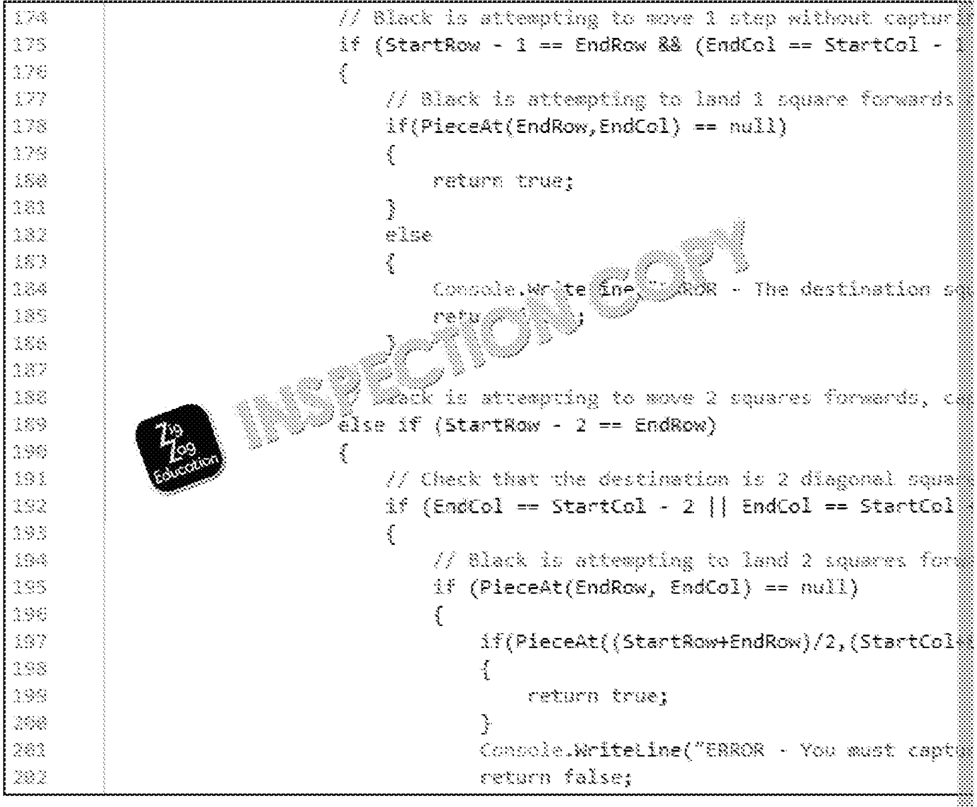
10 marks for validating the move attempted:

Marks could be awarded for:

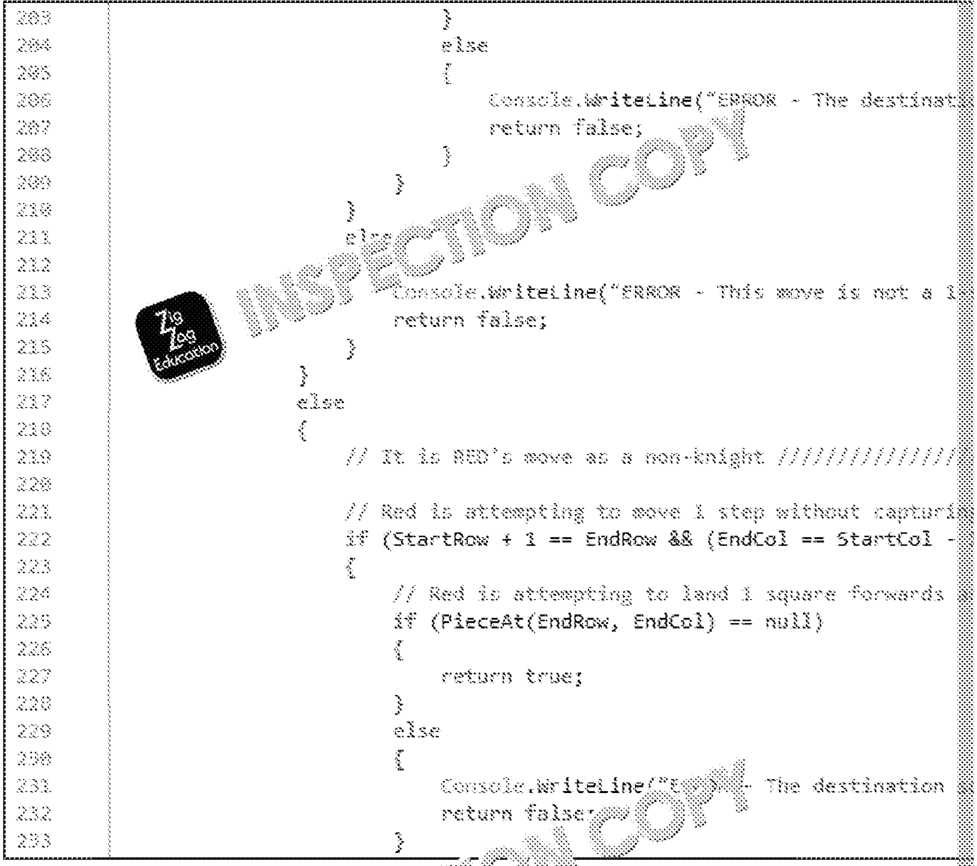
- creating a `validMove` function that returns *True* if a given move by a given
- returning *False* if there is no piece at the start position to move
- returning *False* if the end position is not on the board
- returning *False* if the player tries to move a token non-diagonally or more t
- returning *False* if the end position is not empty
- returning *False* if the player tries to move a token two spaces without taking
- returning *False* if a player tries to move a non-king piece backwards
- handling non-integer input
- checking for the existence of a piece at the square BEFORE proceeding to ask
- **overall** readability/commenting of code to make this complex algorithm re



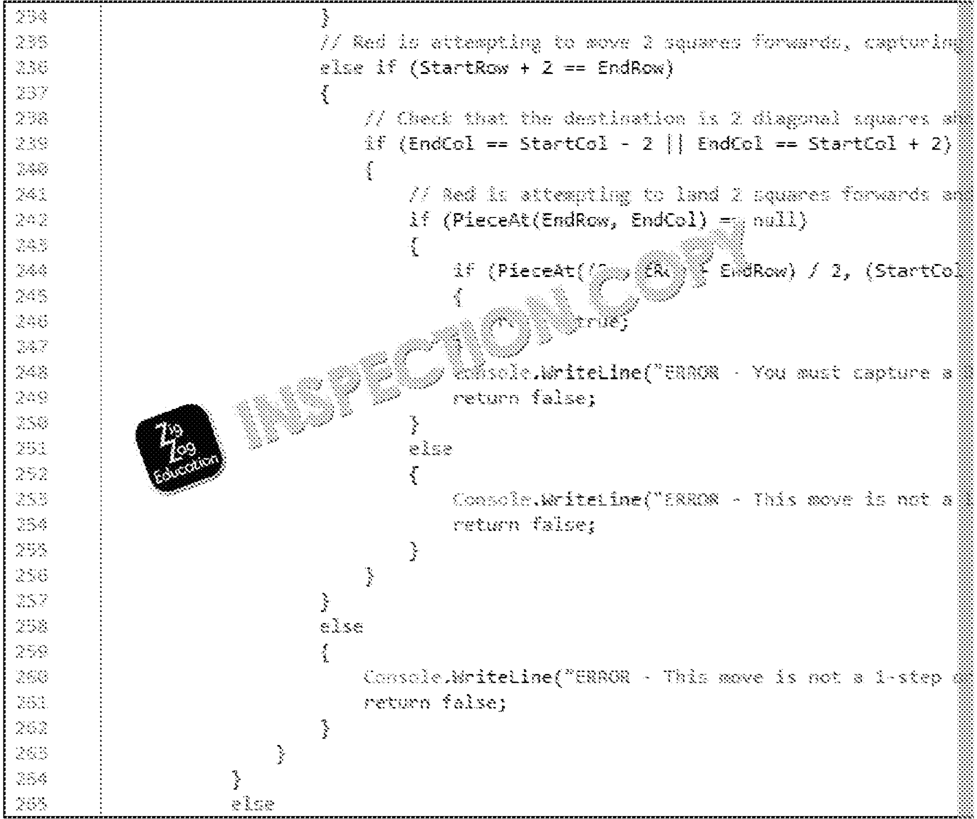
```
        // B5
        public bool ValidMove(int StartRow, int StartCol, int EndRow, int EndC

            Piece PieceToBeMoved = PieceAt(StartRow, StartCol);

            if(PieceToBeMoved == null)
            {
                Console.WriteLine("ERROR - INVALID PIECE SELECTION [Error occu
                return false;
            }

            // Check if the destination is off the board
            if(EndRow >= BoardSize || EndCol >= BoardSize || EndRow < 0 || End
            {
                Console.WriteLine("ERROR - INVALID DESTINATION SQUARE SELECTIO
                return false;
            }

            if(!PieceToBeMoved.GetKing()) // assuming the piece is not a King
            {
                if (TurnNumber % 2 == 1)
                {
                    // It is [illegible] a non-knight
```



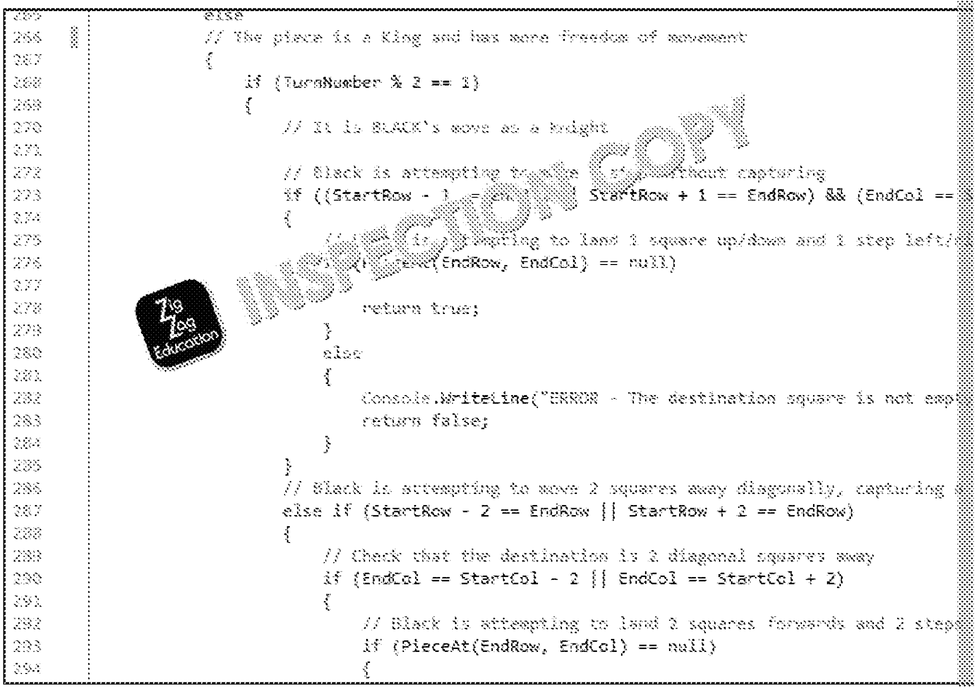
```
    // Black is attempting to move 1 step without captur
    if (StartRow - 1 == EndRow && (EndCol == StartCol - 
    {
        // Black is attempting to land 1 square forwards 
        if(PieceAt(EndRow,EndCol) == null)
        {
            return true;
        }
        else
        {
            Console.WriteLine("ERROR - The destination sq
            return false;
        }

    // Black is attempting to move 2 squares forwards, c
    else if (StartRow - 2 == EndRow)
    {
        // Check that the destination is 2 diagonal squa
        if (EndCol == StartCol - 2 || EndCol == StartCol 
        {
            // Black is attempting to land 2 squares for
            if (PieceAt(EndRow, EndCol) == null)
            {
                if(PieceAt((StartRow+EndRow)/2,(StartCol
                {
                    return true;
                }
                Console.WriteLine("ERROR - You must capt
                return false;
```



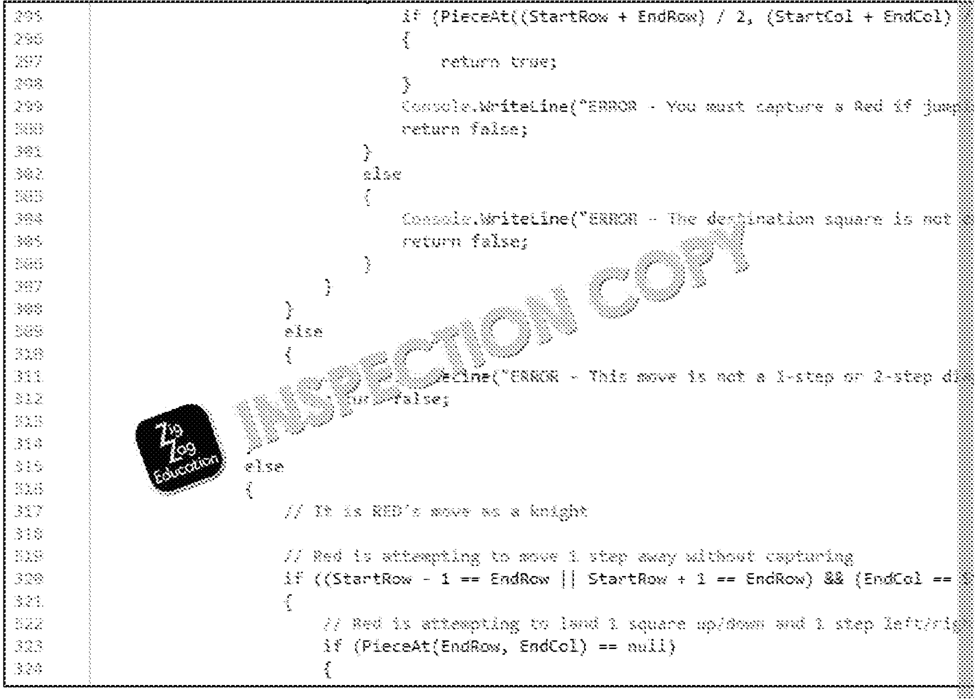
```
            }
            else
            {
                Console.WriteLine("ERROR - The destinat
                return false;
            }
        }
    }
    else
    {
        Console.WriteLine("ERROR - This move is not a 1
        return false;
    }
}
else
{
    // It is RED's move as a non-knight //////////////

    // Red is attempting to move 1 step without capturi
    if (StartRow + 1 == EndRow && (EndCol == StartCol -
    {
        // Red is attempting to land 1 square forwards 
        if (PieceAt(EndRow, EndCol) == null)
        {
            return true;
        }
        else
        {
            Console.WriteLine("ERROR - The destination 
            return false;
        }
```



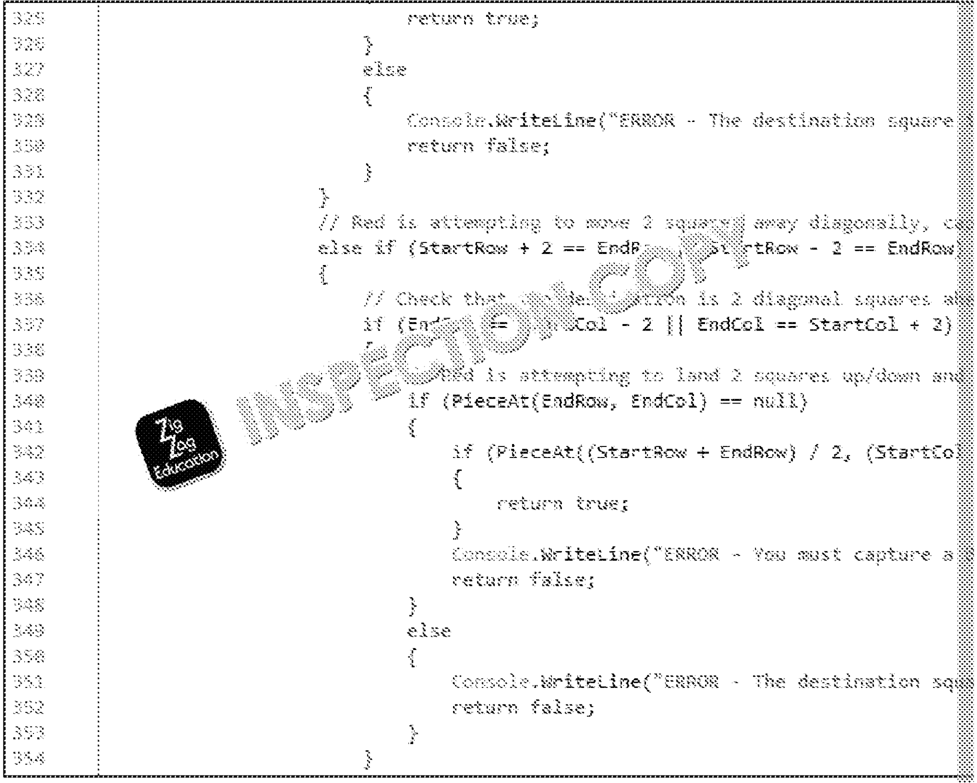
```
                }
                // Red is attempting to move 2 squares forwards, capturing
                else if (StartRow + 2 == EndRow)
                {
                    // Check that the destination is 2 diagonal squares aw
                    if (EndCol == StartCol - 2 || EndCol == StartCol + 2)
                    {
                        // Red is attempting to land 2 squares forwards an
                        if (PieceAt(EndRow, EndCol) == null)
                        {
                            if (PieceAt((StartRow + EndRow) / 2, (StartCol
                            {
                                return true;

                            Console.WriteLine("ERROR - You must capture a
                            return false;
                        }
                        else
                        {
                            Console.WriteLine("ERROR - This move is not a
                            return false;
                        }
                    }
                }
                else
                {
                    Console.WriteLine("ERROR - This move is not a 1-step d
                    return false;
                }
            }
        }
        else
```



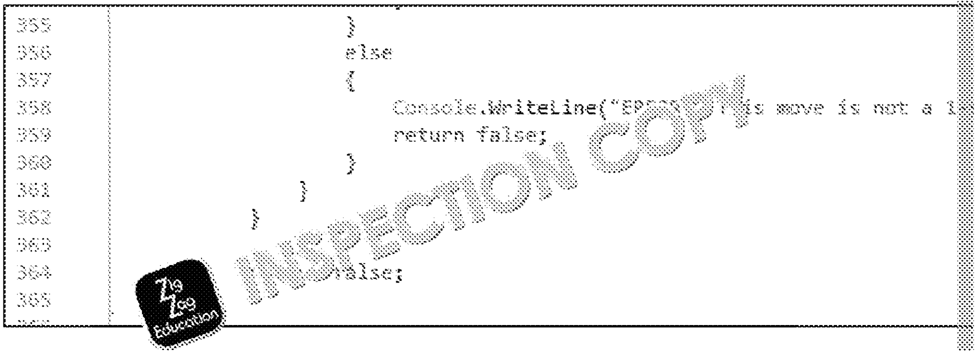
```
        else
        // The piece is a King and has more freedom of movement
        {
            if (TurnNumber % 2 == 1)
            {
                // It is BLACK's move as a knight

                // Black is attempting to move 1 step without capturing
                if ((StartRow - 1 == EndRow || StartRow + 1 == EndRow) && (EndCol ==
                {
                    // Black is attempting to land 1 square up/down and 1 step left/
                    if (PieceAt(EndRow, EndCol) == null)
                    {
                        return true;
                    }
                    else
                    {
                        Console.WriteLine("ERROR - The destination square is not emp
                        return false;
                    }
                }
                // Black is attempting to move 2 squares away diagonally, capturing a
                else if (StartRow - 2 == EndRow || StartRow + 2 == EndRow)
                {
                    // Check that the destination is 2 diagonal squares away
                    if (EndCol == StartCol - 2 || EndCol == StartCol + 2)
                    {
                        // Black is attempting to land 2 squares forwards and 2 step
                        if (PieceAt(EndRow, EndCol) == null)
                        {
```



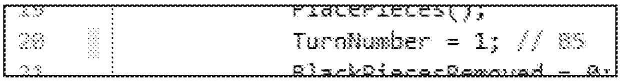
```
                if (PieceAt((StartRow + EndRow) / 2, (StartCol + EndCol)
                {
                    return true;
                }
                Console.WriteLine("ERROR - You must capture a Red if jump
                return false;
            }
            else
            {
                Console.WriteLine("ERROR - The destination square is not
                return false;
            }
        }
    }
    else
    {
        Console.WriteLine("ERROR - This move is not a 1-step or 2-step di
        return false;


}
else
{
    // It is RED's move as a knight

    // Red is attempting to move 1 step away without capturing
    if ((StartRow - 1 == EndRow || StartRow + 1 == EndRow) && (EndCol ==
    {
        // Red is attempting to land 1 square up/down and 1 step left/rig
        if (PieceAt(EndRow, EndCol) == null)
        {
```



```
            return true;
        }
        else
        {
            Console.WriteLine("ERROR - The destination square
            return false;
        }
    }
    // Red is attempting to move 2 squares away diagonally, ca
    else if (StartRow + 2 == EndRow || StartRow - 2 == EndRow)
    {
        // Check that the destination is 2 diagonal squares a
        if (EndCol == StartCol - 2 || EndCol == StartCol + 2)
        {
            // Red is attempting to land 2 squares up/down and
            if (PieceAt(EndRow, EndCol) == null)
            {
                if (PieceAt((StartRow + EndRow) / 2, (StartCol
                {
                    return true;
                }
                Console.WriteLine("ERROR - You must capture a
                return false;
            }
            else
            {
                Console.WriteLine("ERROR - The destination squ
                return false;
            }
        }
```



```
        }
        else
        {
            Console.WriteLine("ERROR - This move is not a 1
            return false;
        }
    }
}

return false;

```



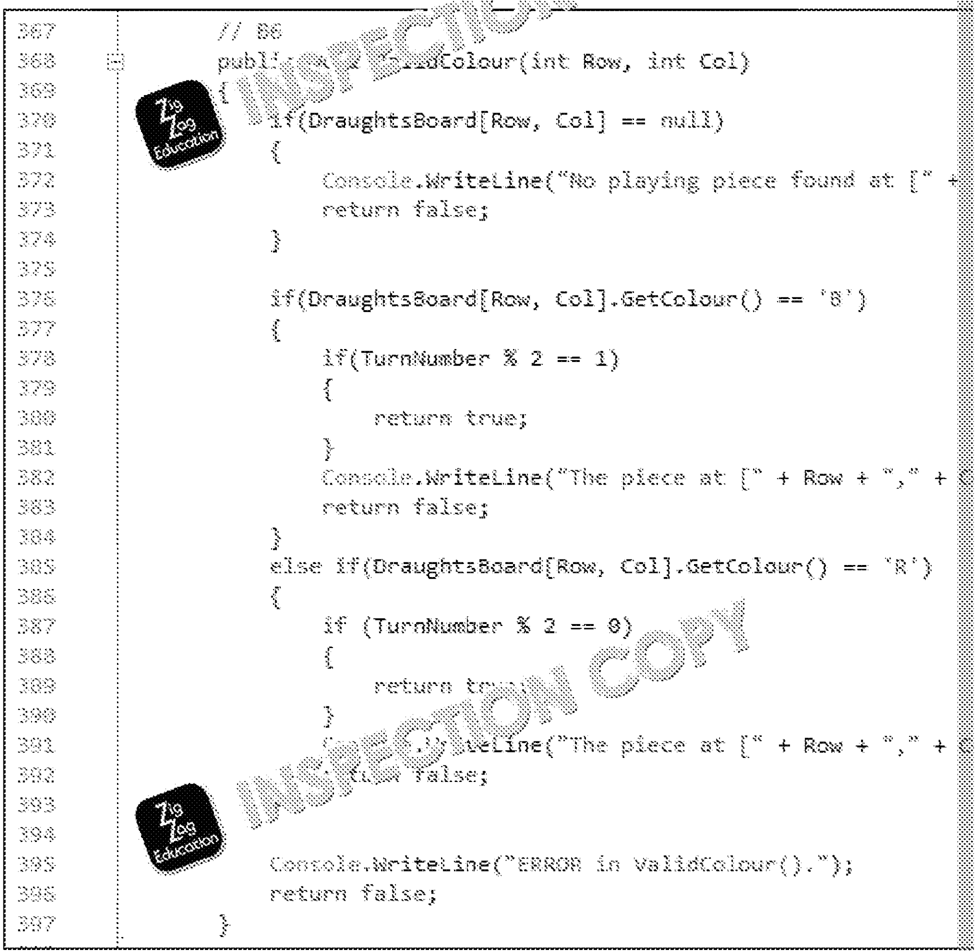
```
            PlacePieces();
            TurnNumber = 1; // B5
            BlackPiecesRemoved = 0;
```

B6

5 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- handling null pointers at vacant squares
- returning *True* when Black is found on a black square on Black's turn
- returning *True* when Red is found on a black square on Red's turn
- returning *False* in other cases
- suitable console output



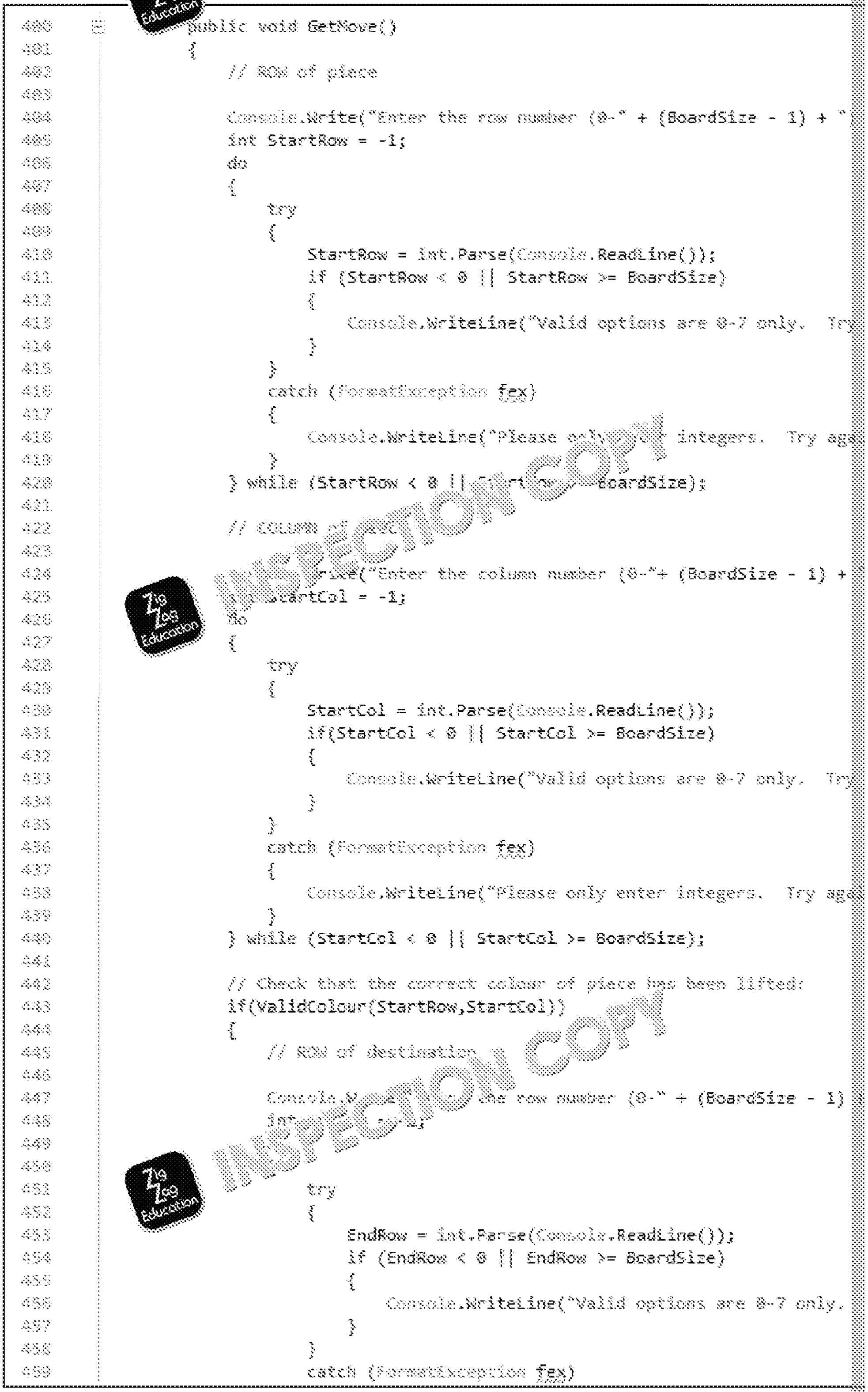
```
        // B6
        public bool ValidColour(int Row, int Col)
        {
            if(DraughtsBoard[Row, Col] == null)
            {
                Console.WriteLine("No playing piece found at [" +
                return false;
            }

            if(DraughtsBoard[Row, Col].GetColour() == 'B')
            {
                if(TurnNumber % 2 == 1)
                {
                    return true;
                }
                Console.WriteLine("The piece at [" + Row + "," + C
                return false;
            }
            else if(DraughtsBoard[Row, Col].GetColour() == 'R')
            {
                if (TurnNumber % 2 == 0)
                {
                    return true;
                }
                Console.WriteLine("The piece at [" + Row + "," + C
                return false;
            }

            Console.WriteLine("ERROR in ValidColour().");
            return false;
        }
```

▩ B7

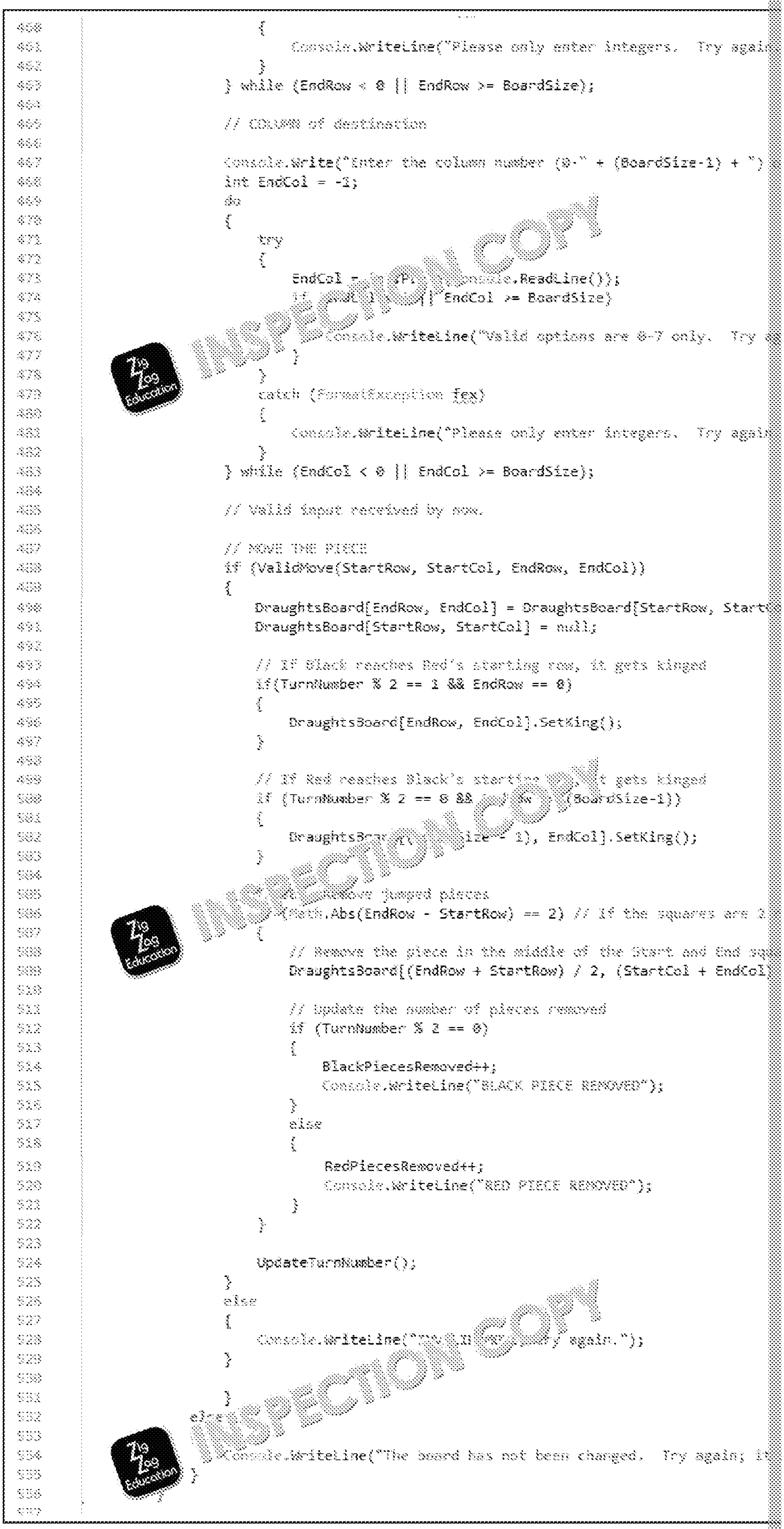
8 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- creating a `GetMove` function that returns start and end positions as a pa
  by player input is valid
- getting a start position and an end position from the user
- checking whether or not the move is valid
- asking the user for new input if the move is invalid
- displaying appropriate messages depending on whether or not the move
- amending `TurnNumber`
- sensible outputs
- leaving the board unaffected by an invalid move

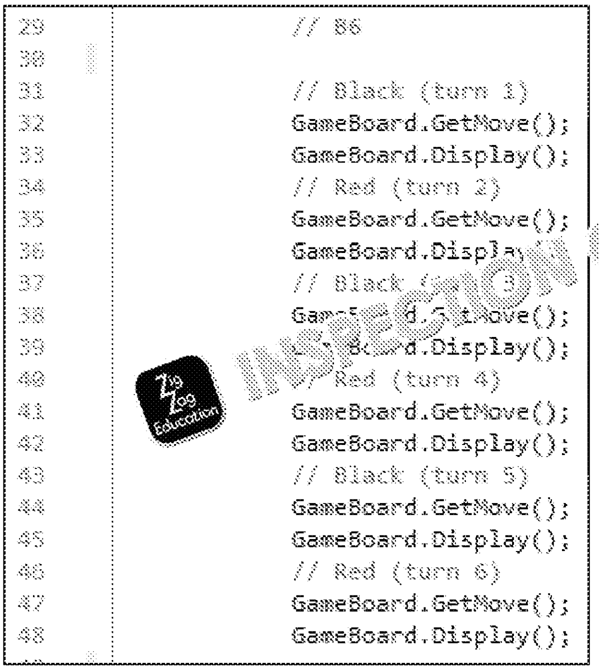


```
public void GetMove()
{
    // ROW of piece

    Console.Write("Enter the row number (0-" + (BoardSize - 1) + "
    int StartRow = -1;
    do
    {
        try
        {
            StartRow = int.Parse(Console.ReadLine());
            if (StartRow < 0 || StartRow >= BoardSize)
            {
                Console.WriteLine("Valid options are 0-7 only.  Try
            }
        }
        catch (FormatException fex)
        {
            Console.WriteLine("Please only enter integers.  Try ag
        }
    } while (StartRow < 0 || StartRow >= BoardSize);

    // COLUMN of piece

    Console.Write("Enter the column number (0-"+ (BoardSize - 1) +
    int StartCol = -1;
    do
    {
        try
        {
            StartCol = int.Parse(Console.ReadLine());
            if(StartCol < 0 || StartCol >= BoardSize)
            {
                Console.WriteLine("Valid options are 0-7 only.  Try
            }
        }
        catch (FormatException fex)
        {
            Console.WriteLine("Please only enter integers.  Try ag
        }
    } while (StartCol < 0 || StartCol >= BoardSize);

    // Check that the correct colour of piece has been lifted:
    if(ValidColour(StartRow,StartCol))
    {
        // ROW of destination

        Console.Write("Enter the row number (0-" + (BoardSize - 1)
        int EndRow = -1;
        [illegible]
        [illegible]
            try
            {
                EndRow = int.Parse(Console.ReadLine());
                if (EndRow < 0 || EndRow >= BoardSize)
                {
                    Console.WriteLine("Valid options are 0-7 only.
                }
            }
            catch (FormatException fex)
```



```
            {
                Console.WriteLine("Please only enter integers.  Try agai
            }
        } while (EndRow < 0 || EndRow >= BoardSize);

        // COLUMN of destination

        Console.Write("Enter the column number (0-" + (BoardSize-1) + ")
        int EndCol = -1;
        do
        {
            try
            {
                EndCol = int.Parse(Console.ReadLine());
                if (EndCol < 0 || EndCol >= BoardSize)
                {
                    Console.WriteLine("Valid options are 0-7 only.  Try a
                }
            }
            catch (FormatException fex)
            {
                Console.WriteLine("Please only enter integers.  Try agai
            }
        } while (EndCol < 0 || EndCol >= BoardSize);

        // Valid input received by now.

        // MOVE THE PIECE
        if (ValidMove(StartRow, StartCol, EndRow, EndCol))
        {
            DraughtsBoard[EndRow, EndCol] = DraughtsBoard[StartRow, StartC
            DraughtsBoard[StartRow, StartCol] = null;

            // If Black reaches Red's starting row, it gets kinged
            if(TurnNumber % 2 == 1 && EndRow == 0)
            {
                DraughtsBoard[EndRow, EndCol].SetKing();
            }

            // If Red reaches Black's starting row, it gets kinged
            if (TurnNumber % 2 == 0 && EndRow == (BoardSize-1))
            {
                DraughtsBoard[(BoardSize - 1), EndCol].SetKing();
            }

            // Remove jumped pieces
            if (Math.Abs(EndRow - StartRow) == 2) // If the squares are 2
            {
                // Remove the piece in the middle of the Start and End squ
                DraughtsBoard[(EndRow + StartRow) / 2, (StartCol + EndCol)

                // Update the number of pieces removed
                if (TurnNumber % 2 == 0)
                {
                    BlackPiecesRemoved++;
                    Console.WriteLine("BLACK PIECE REMOVED");
                }
                else
                {
                    RedPiecesRemoved++;
                    Console.WriteLine("RED PIECE REMOVED");
                }
            }

            UpdateTurnNumber();
        }
        else
        {
            Console.WriteLine("INVALID MOVE.  Try again.");
        }

        }
    else
    {
        Console.WriteLine("The board has not been changed.  Try again; it
    }
}
```

In Program.cs: Test code:



```
            // B6

            // Black (turn 1)
            GameBoard.GetMove();
            GameBoard.Display();
            // Red (turn 2)
            GameBoard.GetMove();
            GameBoard.Display();
            // Black (turn 3)
            GameBoard.GetMove();
            GameBoard.Display();
            // Red (turn 4)
            GameBoard.GetMove();
            GameBoard.Display();
            // Black (turn 5)
            GameBoard.GetMove();
            GameBoard.Display();
            // Red (turn 6)
            GameBoard.GetMove();
            GameBoard.Display();
```

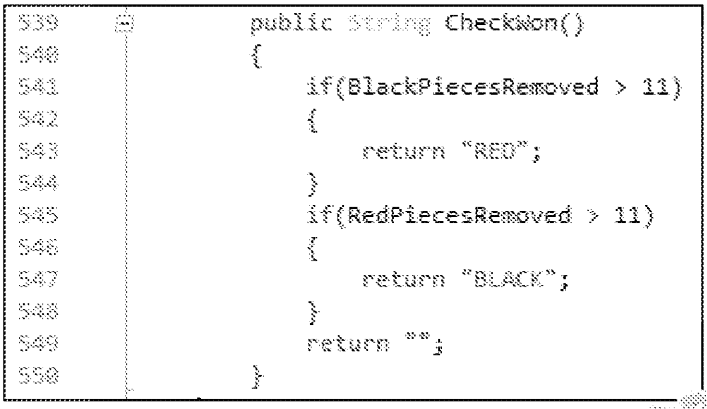
▩ B8

6 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

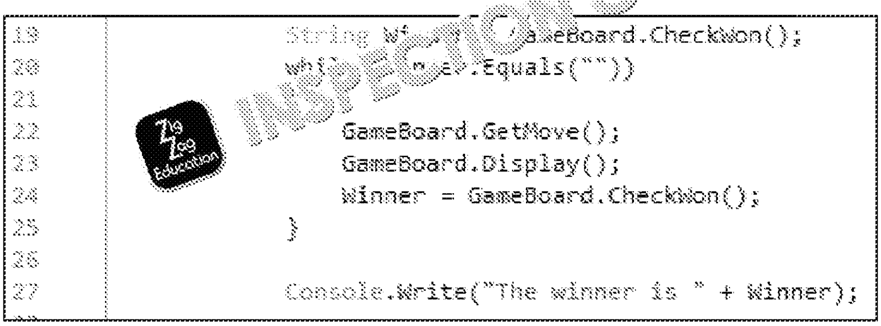
- creating a CheckWon function that returns either the playing piece colour empty string if no player has won
- calling CheckWon after each move
- creating a game loop that exits once a player has won
- alternating turns between each player
- displaying a message at the end of the game to say which player has won
- successfully modifying GetMove to remove a piece

In Board.cs:



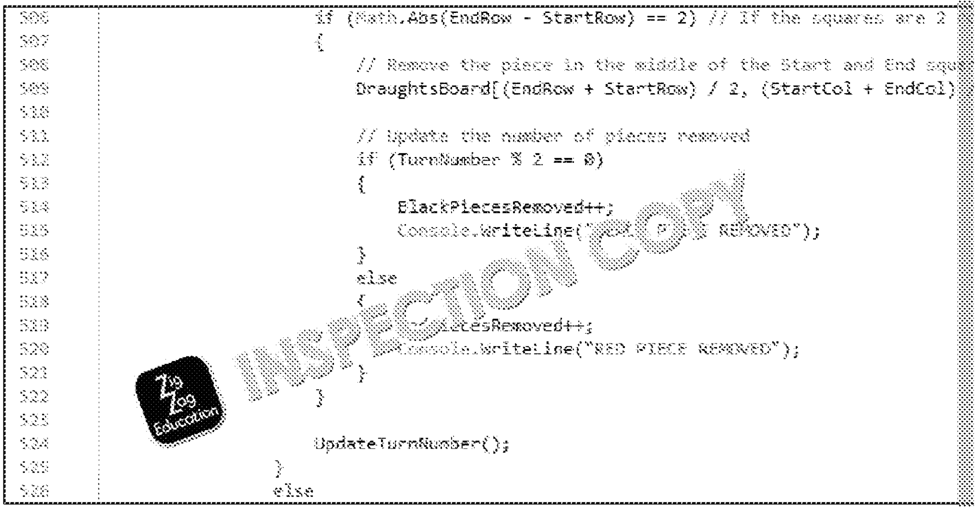
```
        public String CheckWon()
        {
            if(BlackPiecesRemoved > 11)
            {
                return "RED";
            }
            if(RedPiecesRemoved > 11)
            {
                return "BLACK";
            }
            return "";
        }
```

In Program.cs:



```
            String Winner = GameBoard.CheckWon();
            while(Winner.Equals(""))
            {
                GameBoard.GetMove();
                GameBoard.Display();
                Winner = GameBoard.CheckWon();
            }

            Console.Write("The winner is " + Winner);
```

Removing a piece (in `Board.cs`):



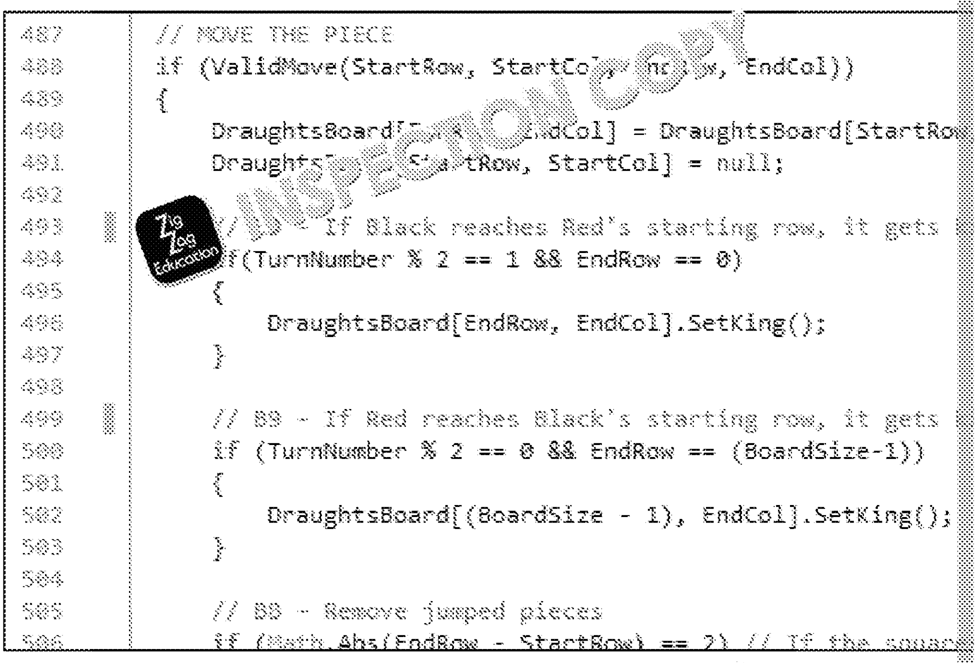
```
                if (Math.Abs(EndRow - StartRow) == 2) // If the squares are 2
                {
                    // Remove the piece in the middle of the Start and End squ
                    DraughtsBoard[(EndRow + StartRow) / 2, (StartCol + EndCol)

                    // Update the number of pieces removed
                    if (TurnNumber % 2 == 0)
                    {
                        BlackPiecesRemoved++;
                        Console.WriteLine("BLACK PIECE REMOVED");
                    }
                    else
                    {
                        RedPiecesRemoved++;
                        Console.WriteLine("RED PIECE REMOVED");
                    }
                }

                UpdateTurnNumber();
            }
            else
```

B9

4 marks available for modifying the code as shown (or equivalent code):

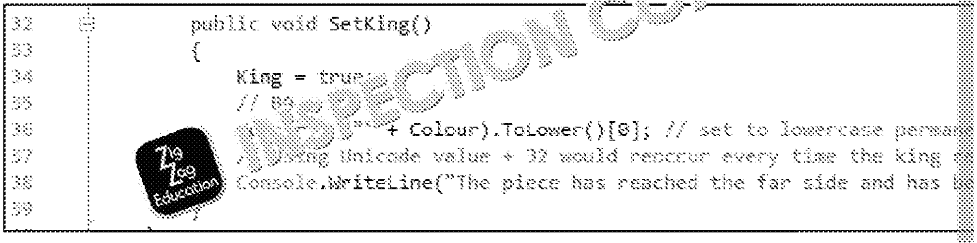
Marks could be awarded for:

- correctly placing this after the move has taken place
- checking the turn and the end row criteria simultaneously
- treating Red differently in case of a different-sized board
- calling the amended `SetKing()` method



```
// MOVE THE PIECE
if (ValidMove(StartRow, StartCol, EndRow, EndCol))
{
    DraughtsBoard[EndRow, EndCol] = DraughtsBoard[StartRo
    DraughtsBoard[StartRow, StartCol] = null;

    // B9 - If Black reaches Red's starting row, it gets
    if(TurnNumber % 2 == 1 && EndRow == 0)
    {
        DraughtsBoard[EndRow, EndCol].SetKing();
    }

    // B9 - If Red reaches Black's starting row, it gets
    if (TurnNumber % 2 == 0 && EndRow == (BoardSize-1))
    {
        DraughtsBoard[(BoardSize - 1), EndCol].SetKing();
    }

    // B8 - Remove jumped pieces
    if (Math.Abs(EndRow - StartRow) == 2) // If the squar
```

In `Piece.cs`:



```
        public void SetKing()
        {
            King = true;
            // B9
            Symbol = ("" + Colour).ToLower()[0]; // set to lowercase perma
            // Using Unicode value + 32 would reoccur every time the king 
            Console.WriteLine("The piece has reached the far side and has b
        }
```

Exercise 6 – Tree Traversal

Section A

A1

1 mark each:

Call to a constructor: Line
Private attribute declaration: Line
Use of a null pointer: Line

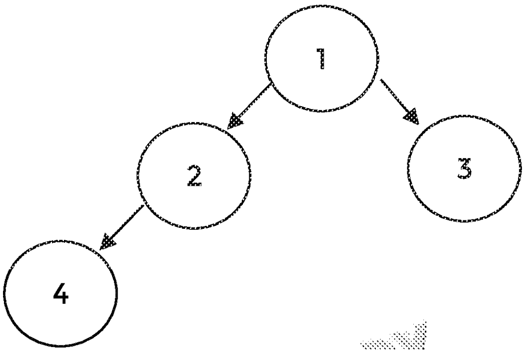
A2

1 mark for:

Line

A3

2 marks (1 mark for drawing a binary tree; 1 mark for arranging the values correctly



A4

2 marks (1 mark for explaining that binary tree nodes can have a maximum of two
that nodes in a multi-branch tree can have any number of child nodes):

The nodes in a binary tree can have a maximum of two child nodes, whereas any
have any number of child nodes.

A5

1 mark for explaining encapsulation and 1 mark for relating it to Node:

The `Node` class contains private attributes with public accessor/mutator methods.
in that they cannot be directly referenced for read/write access, instead forcing the
methods.

A6

1 mark for each:

Tree, Node, int/integer, String/string.

A7

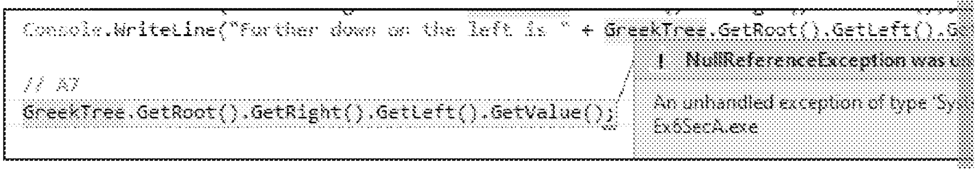
1 mark for:

```
GreekTree.GetRoot().GetRight().GetLeft().GetValue();
```

A8

2 marks (1 mark for explaining it would throw an exception, 1 mark for saying null

An exception would be thrown. There is no node, so it would be a null reference



```
Console.WriteLine("Further down on the left is " + GreekTree.GetRoot().GetLeft().G

// A7
GreekTree.GetRoot().GetRight().GetLeft().GetValue();
```

NullReferenceException was u
An unhandled exception of type 'Sy
Ex6SecA.exe

■ A9

1 mark for giving the correct depth-first (post-order) tree traversal:

4, 2, 3, 1

■ A10

1 mark for giving the correct depth-first (pre-order) tree traversal:

1, 2, 4, 3

■ A11

1 mark for giving the correct depth-first (in-order) tree traversal:

4, 2, 1, 3

■ A12

1 mark for giving the correct depth-first (post-order) tree traversal:

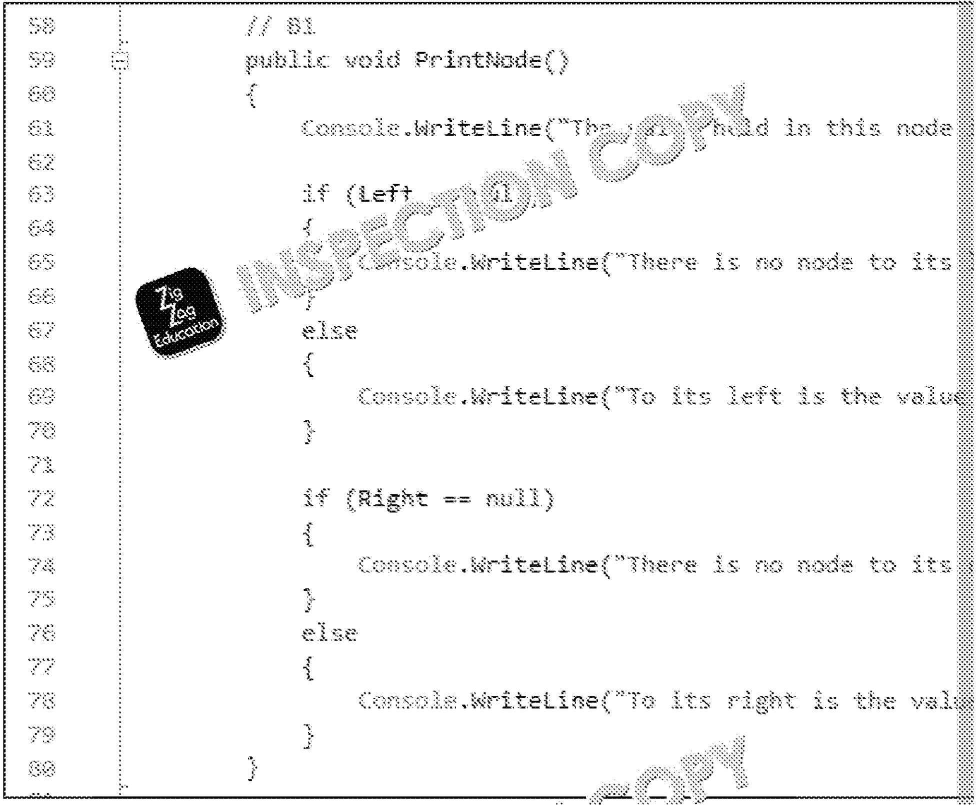
1, 2, 3, 4

Section B

■ B1

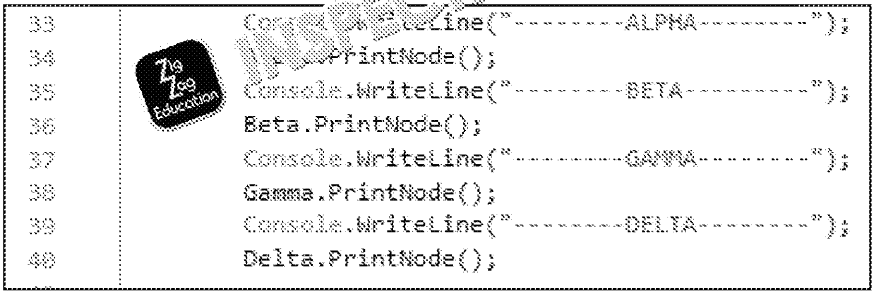
3 marks for the method; 1 mark for the test code:

In `Node.cs`:



```
        // B1
        public void PrintNode()
        {
            Console.WriteLine("The value held in this node
            
            if (Left == null)
            {
                Console.WriteLine("There is no node to its
            }
            else
            {
                Console.WriteLine("To its left is the value
            }
            
            if (Right == null)
            {
                Console.WriteLine("There is no node to its
            }
            else
            {
                Console.WriteLine("To its right is the val
            }
        }
```

In `Program.cs`:

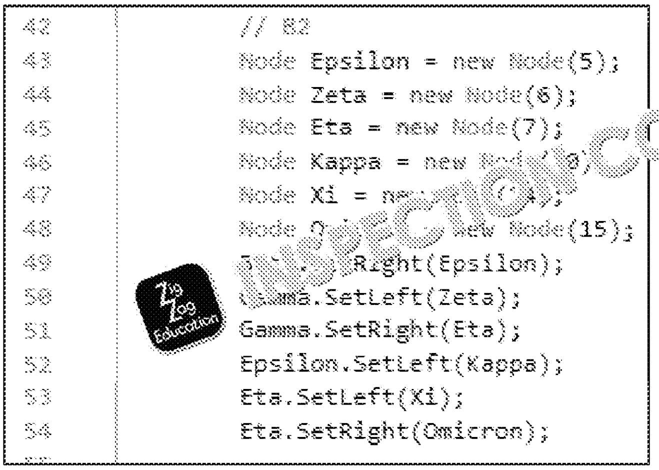


```
        Console.WriteLine("--------ALPHA--------");
        Alpha.PrintNode();
        Console.WriteLine("--------BETA---------");
        Beta.PrintNode();
        Console.WriteLine("---------GAMMA--------");
        Gamma.PrintNode();
        Console.WriteLine("--------DELTA--------");
        Delta.PrintNode();
```

▩ B2

1 mark for instantiating the new nodes; 3 marks for assembling the tree:

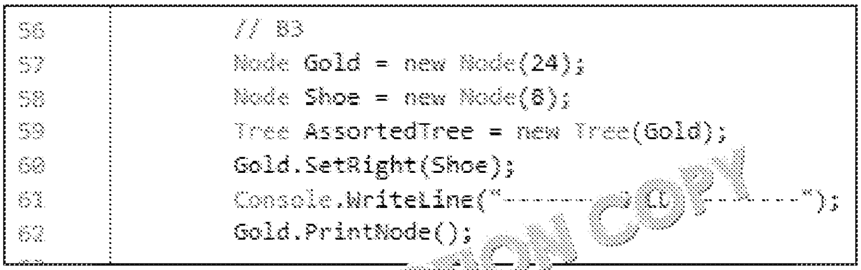
In Program.cs:



```
        // B2
        Node Epsilon = new Node(5);
        Node Zeta = new Node(6);
        Node Eta = new Node(7);
        Node Kappa = new Node(8);
        Node Xi = new Node(14);
        Node Omicron = new Node(15);
        Beta.SetRight(Epsilon);
        Gamma.SetLeft(Zeta);
        Gamma.SetRight(Eta);
        Epsilon.SetLeft(Kappa);
        Eta.SetLeft(Xi);
        Eta.SetRight(Omicron);
```

▩ B3

1 mark for instantiating a new Tree, 1 mark for instantiating both Gold and Shoe, 1 Shoe into the new Tree correctly, 1 mark for correct output:



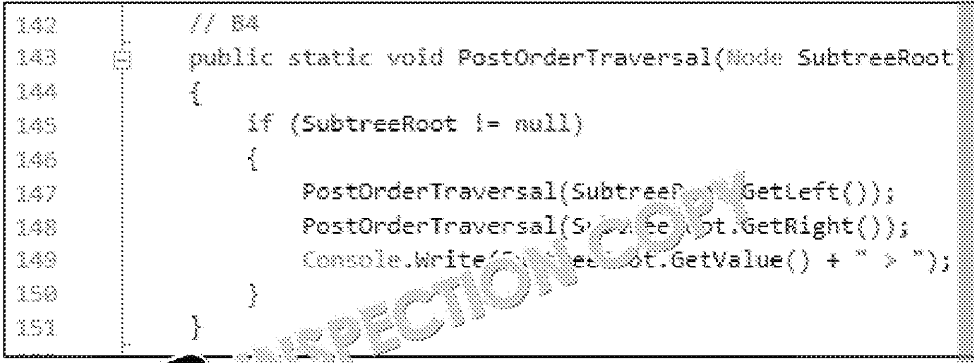
```
        // B3
        Node Gold = new Node(24);
        Node Shoe = new Node(8);
        Tree AssortedTree = new Tree(Gold);
        Gold.SetRight(Shoe);
        Console.WriteLine("---------[illegible]---------");
        Gold.PrintNode();
```

▩ B4

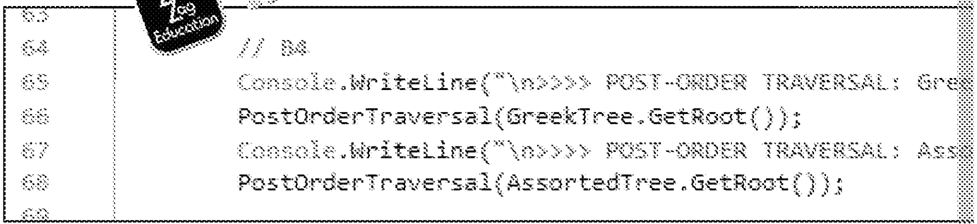
5 marks available for modifying the code as shown (or equivalent code):

Marks are awarded for:

- creating a PostOrderTraversal procedure that takes a Node object as
- implementing the IF structure correctly to handle null values (leaf nodes)
- performing a recursive call by passing the left/right node
- displaying all values in the tree in the correct order for depth-first, post-o
- appropriately modifying the main program procedure



```
    // B4
    public static void PostOrderTraversal(Node SubtreeRoot
    {
        if (SubtreeRoot != null)
        {
            PostOrderTraversal(SubtreeRoot.GetLeft());
            PostOrderTraversal(SubtreeRoot.GetRight());
            Console.Write(SubtreeRoot.GetValue() + " > ");
        }
    }
```



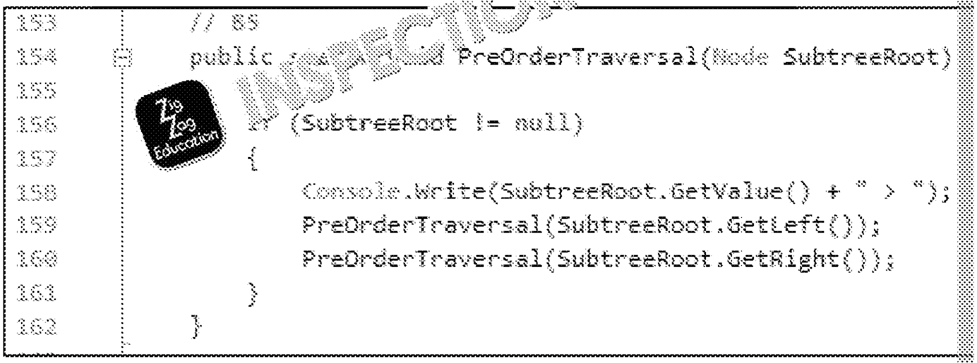
```
        // B4
        Console.WriteLine("\n>>>> POST-ORDER TRAVERSAL: Gre
        PostOrderTraversal(GreekTree.GetRoot());
        Console.WriteLine("\n>>>> POST-ORDER TRAVERSAL: Ass
        PostOrderTraversal(AssortedTree.GetRoot());
```

▩ B5

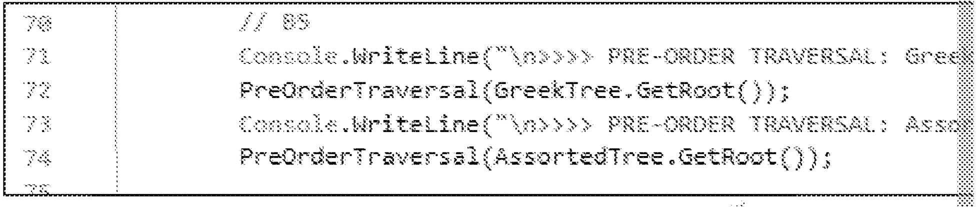
5 marks available for modifying the code as shown (or equivalent code):

Marks are awarded for:

- creating a `PreOrderTraversal` procedure that takes a `Node` object as i
- implementing the IF structure correctly to handle null values (leaf nodes)
- performing a recursive call by passing the left/right node
- displaying all values in the tree in the correct order for depth-first, pre-or
- appropriately modifying the main program procedure



```
    // B5
    public static void PreOrderTraversal(Node SubtreeRoot)
    {
        if (SubtreeRoot != null)
        {
            Console.Write(SubtreeRoot.GetValue() + " > ");
            PreOrderTraversal(SubtreeRoot.GetLeft());
            PreOrderTraversal(SubtreeRoot.GetRight());
        }
    }
```



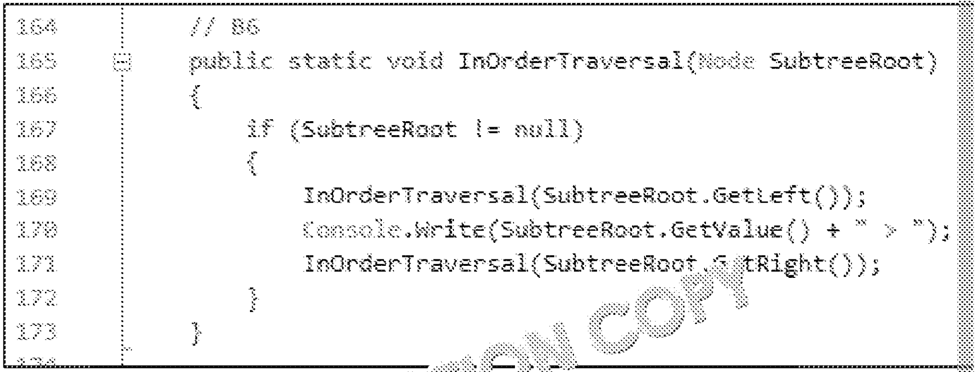
```
            // B5
            Console.WriteLine("\n>>>> PRE-ORDER TRAVERSAL: Gree
            PreOrderTraversal(GreekTree.GetRoot());
            Console.WriteLine("\n>>>> PRE-ORDER TRAVERSAL: Asso
            PreOrderTraversal(AssortedTree.GetRoot());
```

▩ B6

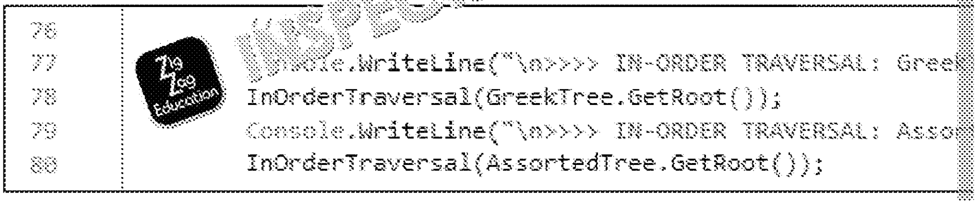
5 marks available for modifying the code as shown (or equivalent code):

Marks are awarded for:

- creating a `InOrderTraversal` procedure that takes a `Node` object as in
- implementing the IF structure correctly to handle null values (leaf nodes)
- performing a recursive call by passing the left/right node
- displaying all values in the tree in the correct order for depth-first, post-o
- appropriately modifying the main program procedure



```
    // B6
    public static void InOrderTraversal(Node SubtreeRoot)
    {
        if (SubtreeRoot != null)
        {
            InOrderTraversal(SubtreeRoot.GetLeft());
            Console.Write(SubtreeRoot.GetValue() + " > ");
            InOrderTraversal(SubtreeRoot.GetRight());
        }
    }
```



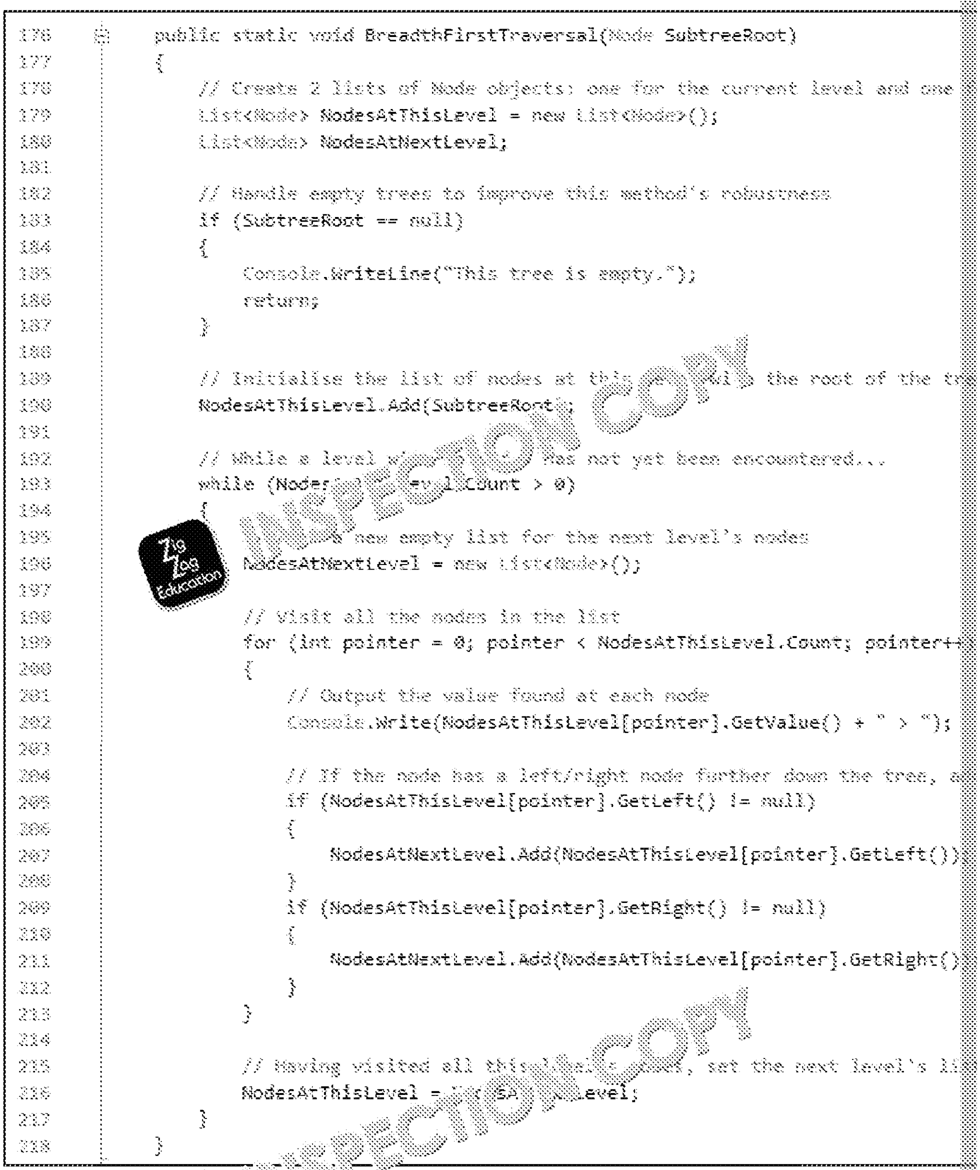
```
            // B6
            Console.WriteLine("\n>>>> IN-ORDER TRAVERSAL: Greek
            InOrderTraversal(GreekTree.GetRoot());
            Console.WriteLine("\n>>>> IN-ORDER TRAVERSAL: Asso
            InOrderTraversal(AssortedTree.GetRoot());
```

B7

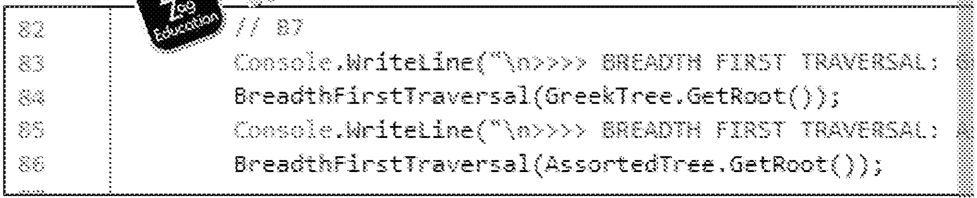
7 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for the bullets shown:

- [1] Create 2 lists of Node objects: one for the current level and one for th
- [1] Handle empty trees to improve this method's robustness
  - Output a message
  - Return/Exit
- [1] Initialise the list of nodes at this level with the root of the tree passed i
- [1] While a level with no nodes has not yet been encountered...
  - [1] Form a new empty list for the next level's nodes
  - [1] Visit all the nodes in the list
    - Output the value found at each node
    - If the node has a left/right node further down the tree, ad
  - [1] Having visited all this level's nodes, set the next level's list to b



```
public static void BreadthFirstTraversal(Node SubtreeRoot)
{
    // Create 2 lists of Node objects: one for the current level and one
    List<Node> NodesAtThisLevel = new List<Node>();
    List<Node> NodesAtNextLevel;

    // Handle empty trees to improve this method's robustness
    if (SubtreeRoot == null)
    {
        Console.WriteLine("This tree is empty.");
        return;
    }

    // Initialise the list of nodes at this level with the root of the tr
    NodesAtThisLevel.Add(SubtreeRoot);

    // While a level with no nodes has not yet been encountered...
    while (NodesAtThisLevel.Count > 0)
    {
        // Form a new empty list for the next level's nodes
        NodesAtNextLevel = new List<Node>();

        // Visit all the nodes in the list
        for (int pointer = 0; pointer < NodesAtThisLevel.Count; pointer++
        {
            // Output the value found at each node
            Console.Write(NodesAtThisLevel[pointer].GetValue() + " > ");

            // If the node has a left/right node further down the tree, a
            if (NodesAtThisLevel[pointer].GetLeft() != null)
            {
                NodesAtNextLevel.Add(NodesAtThisLevel[pointer].GetLeft())
            }
            if (NodesAtThisLevel[pointer].GetRight() != null)
            {
                NodesAtNextLevel.Add(NodesAtThisLevel[pointer].GetRight()
            }
        }

        // Having visited all this level's nodes, set the next level's li
        NodesAtThisLevel = NodesAtNextLevel;
    }
}
```

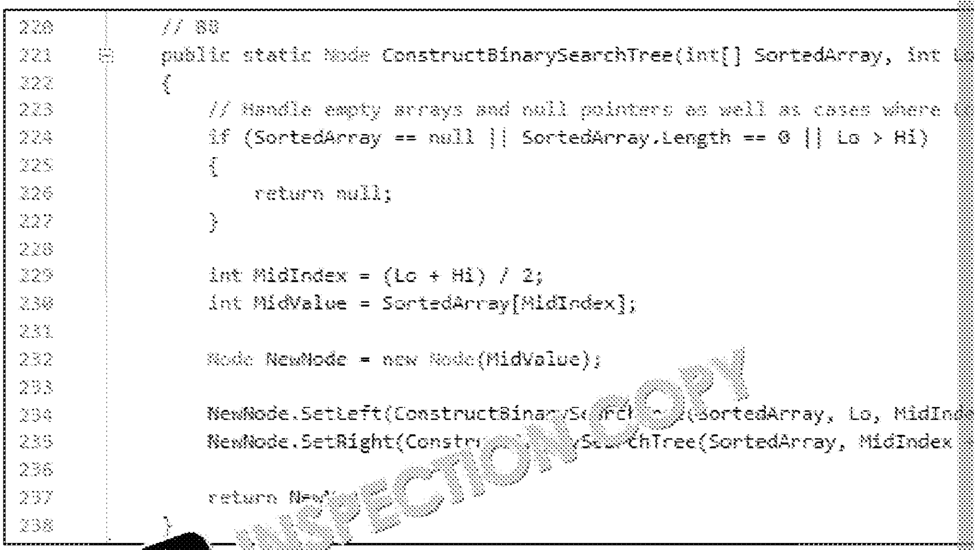


```
    // B7
    Console.WriteLine("\n>>>> BREADTH FIRST TRAVERSAL:
    BreadthFirstTraversal(GreekTree.GetRoot());
    Console.WriteLine("\n>>>> BREADTH FIRST TRAVERSAL:
    BreadthFirstTraversal(AssortedTree.GetRoot());
```

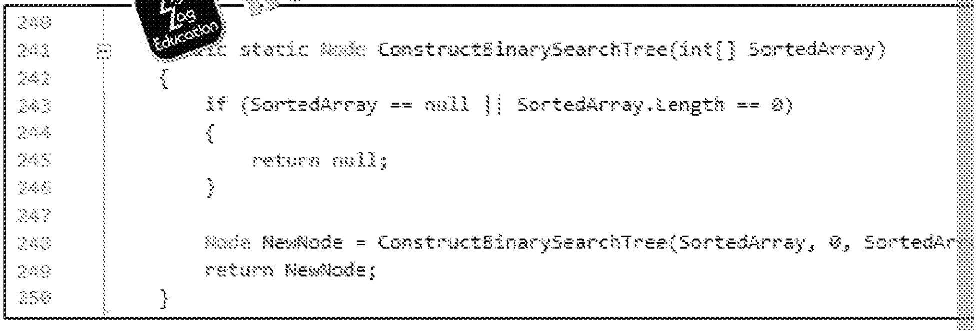
B8

14 marks available for modifying the code as shown (or equivalent code):

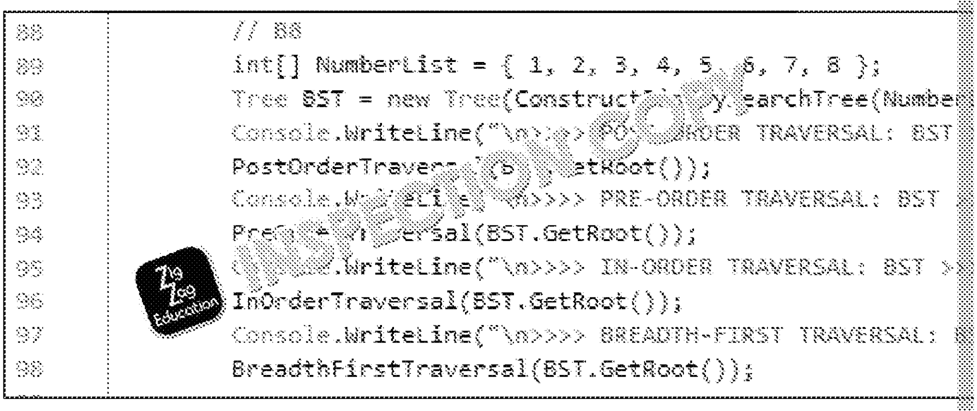
- [1] taking in three parameters: int[], int, int
- [1] handling null, length=0 and Lo>Hi
- [1] calculating middle index
- [1] deriving mid-value
- [1] initialising a new node with the mid-value
- [1] setting the left subtree's node
- [1] setting the right subtree's node
- [1] return statement
- [1] overloading [illegible] correctly
- [2] [illegible] one-parameter version of the method with one OR only
- [1] setting Lo and Hi automatically
- [1] Main calls the method
- [1] Main performs all four traversals



```
    // B8
    public static Node ConstructBinarySearchTree(int[] SortedArray, int L
    {
        // Handle empty arrays and null pointers as well as cases where L
        if (SortedArray == null || SortedArray.Length == 0 || Lo > Hi)
        {
            return null;
        }

        int MidIndex = (Lo + Hi) / 2;
        int MidValue = SortedArray[MidIndex];

        Node NewNode = new Node(MidValue);

        NewNode.SetLeft(ConstructBinarySearchTree(SortedArray, Lo, MidInd
        NewNode.SetRight(ConstructBinarySearchTree(SortedArray, MidIndex 

        return NewNode;
    }
```



```

    public static Node ConstructBinarySearchTree(int[] SortedArray)
    {
        if (SortedArray == null || SortedArray.Length == 0)
        {
            return null;
        }

        Node NewNode = ConstructBinarySearchTree(SortedArray, 0, SortedAr
        return NewNode;
    }
```



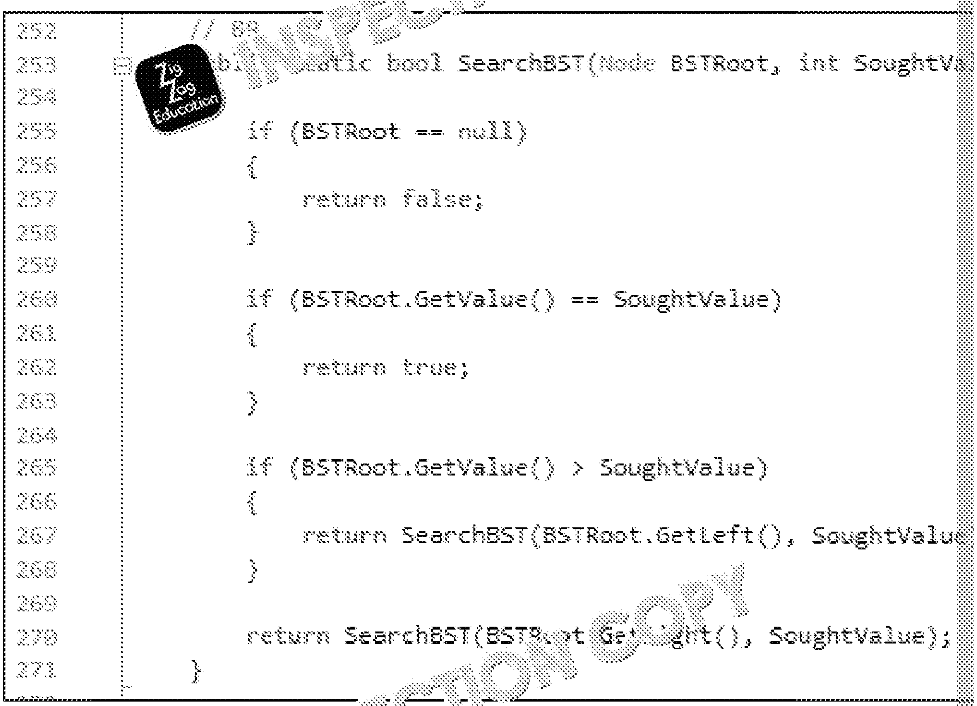
```
            // B8
            int[] NumberList = { 1, 2, 3, 4, 5, 6, 7, 8 };
            Tree BST = new Tree(ConstructBinarySearchTree(Number
            Console.WriteLine("\n>>>> POST-ORDER TRAVERSAL: BST 
            PostOrderTraversal(BST.GetRoot());
            Console.WriteLine("\n>>>> PRE-ORDER TRAVERSAL: BST 
            PreOrderTraversal(BST.GetRoot());
            Console.WriteLine("\n>>>> IN-ORDER TRAVERSAL: BST >
            InOrderTraversal(BST.GetRoot());
            Console.WriteLine("\n>>>> BREADTH-FIRST TRAVERSAL: 
            BreadthFirstTraversal(BST.GetRoot());
```

▩ B9

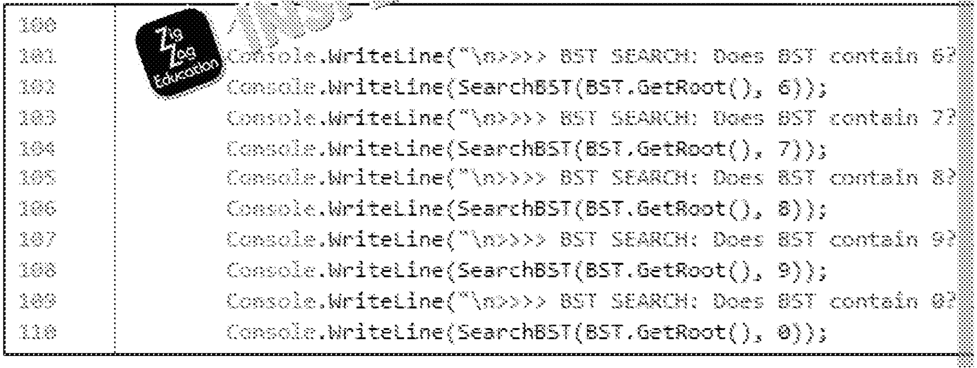
6 marks available for modifying the code as shown (or equivalent code):

Marks are awarded for:

- creating a `SearchBST` function that takes a search value and node as inp
- handling nulls
- returning *True* if the value is found
- recursively checking the left subtree
- recursively checking the right subtree
- providing intelligible output from main program



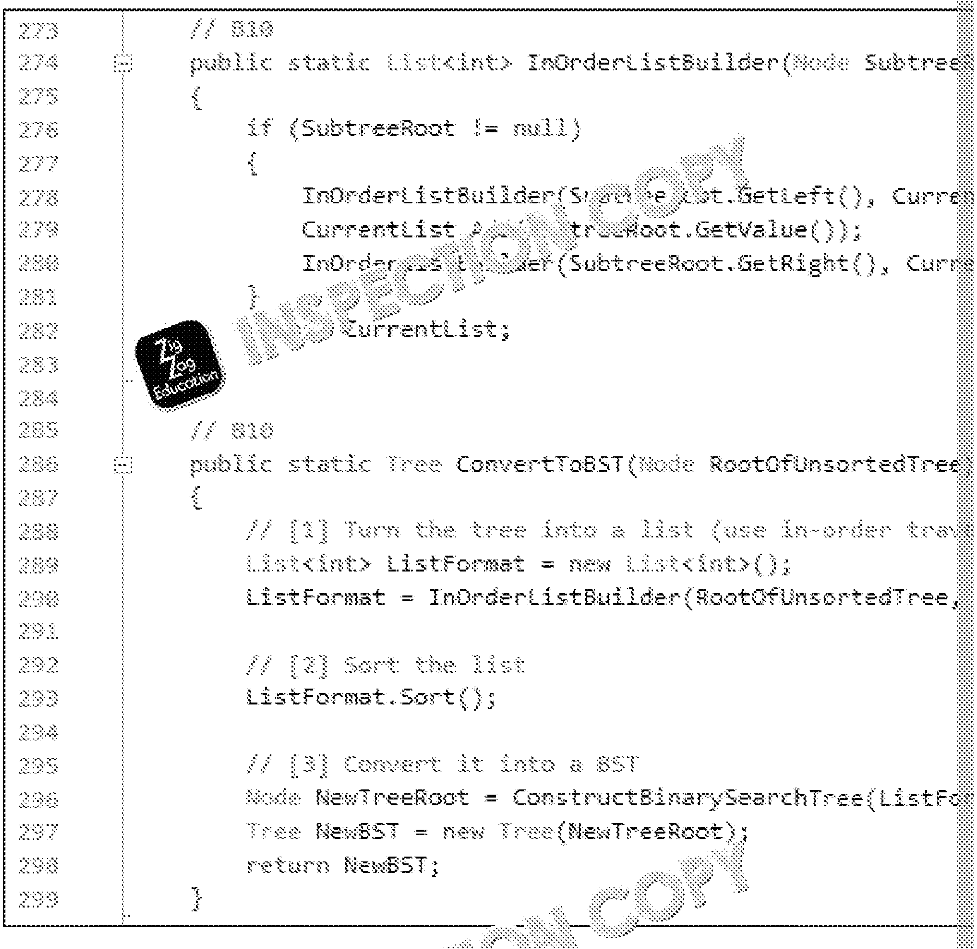
```
        // B9
        public static bool SearchBST(Node BSTRoot, int SoughtVa

            if (BSTRoot == null)
            {
                return false;
            }

            if (BSTRoot.GetValue() == SoughtValue)
            {
                return true;
            }

            if (BSTRoot.GetValue() > SoughtValue)
            {
                return SearchBST(BSTRoot.GetLeft(), SoughtValue
            }

            return SearchBST(BSTRoot.GetRight(), SoughtValue);
        }
```



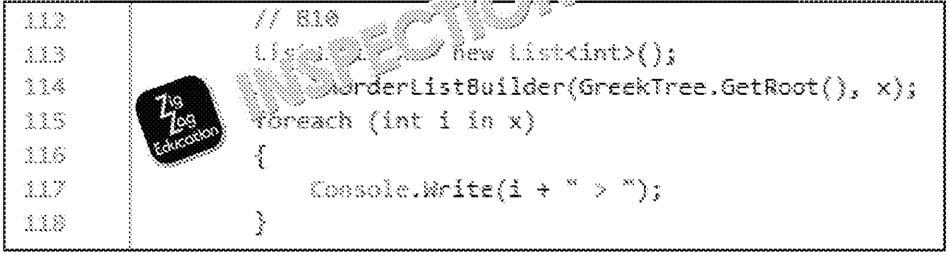
```
            //
            Console.WriteLine("\n>>>> BST SEARCH: Does BST contain 6?
            Console.WriteLine(SearchBST(BST.GetRoot(), 6));
            Console.WriteLine("\n>>>> BST SEARCH: Does BST contain 7?
            Console.WriteLine(SearchBST(BST.GetRoot(), 7));
            Console.WriteLine("\n>>>> BST SEARCH: Does BST contain 8?
            Console.WriteLine(SearchBST(BST.GetRoot(), 8));
            Console.WriteLine("\n>>>> BST SEARCH: Does BST contain 9?
            Console.WriteLine(SearchBST(BST.GetRoot(), 9));
            Console.WriteLine("\n>>>> BST SEARCH: Does BST contain 0?
            Console.WriteLine(SearchBST(BST.GetRoot(), 0));
```

B10

3 marks per correctly sorted program:



```
        // B10
        public static List<int> InOrderListBuilder(Node Subtree
        {
            if (SubtreeRoot != null)
            {
                InOrderListBuilder(SubtreeRoot.GetLeft(), Curre
                CurrentList.Add(SubtreeRoot.GetValue());
                InOrderListBuilder(SubtreeRoot.GetRight(), Curr
            }
            return CurrentList;
        }

        // B10
        public static Tree ConvertToBST(Node RootOfUnsortedTree
        {
            // [1] Turn the tree into a list (use in-order trav
            List<int> ListFormat = new List<int>();
            ListFormat = InOrderListBuilder(RootOfUnsortedTree,

            // [2] Sort the list
            ListFormat.Sort();

            // [3] Convert it into a BST
            Node NewTreeRoot = ConstructBinarySearchTree(ListFo
            Tree NewBST = new Tree(NewTreeRoot);
            return NewBST;
        }
```



```
            // B10
            List<int> x = new List<int>();
            InOrderListBuilder(GreekTree.GetRoot(), x);
            foreach (int i in x)
            {
                Console.Write(i + " > ");
            }
```

B11

6 marks for the AddNode method developed from the breadth-first traversal method

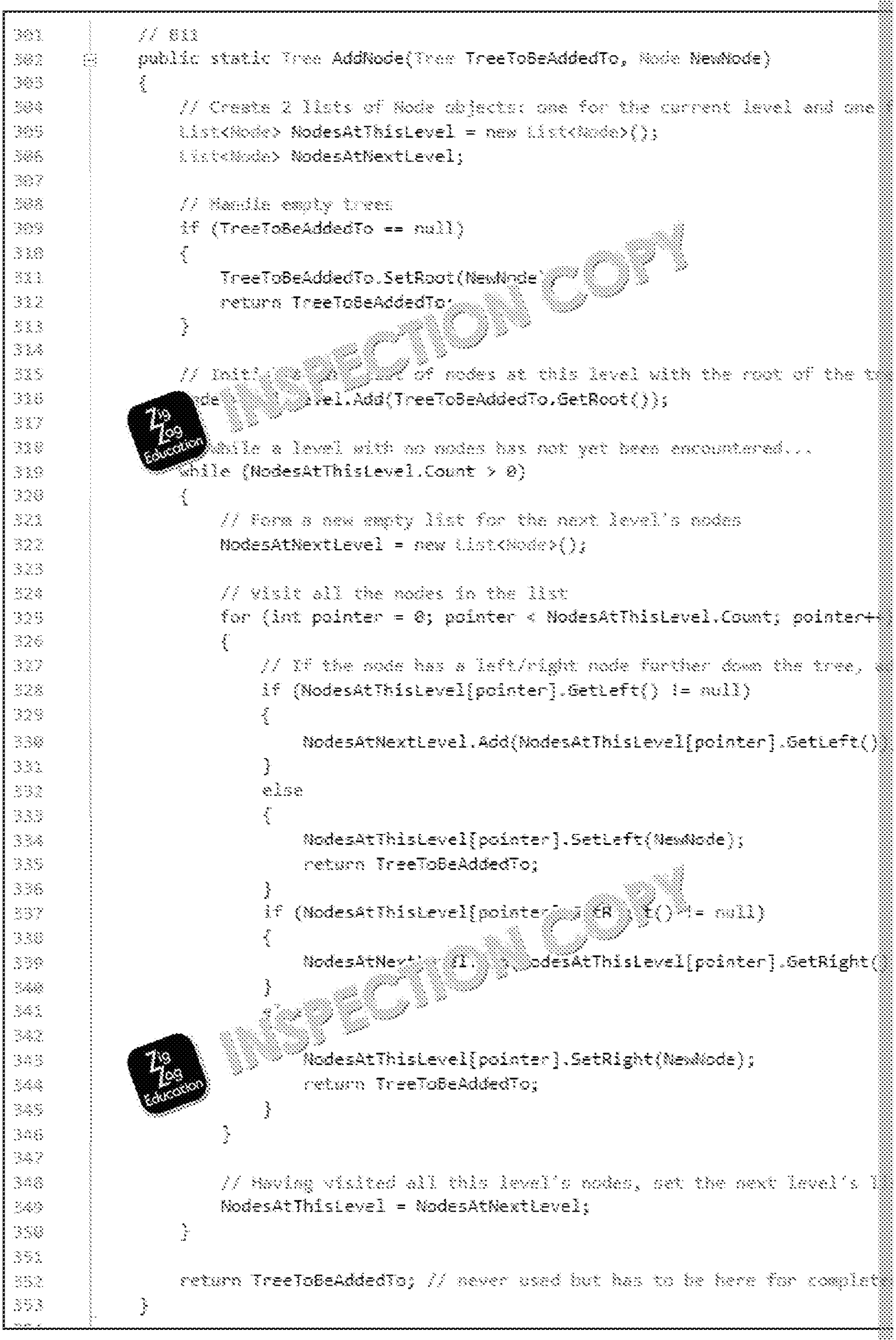
Marks are awarded for:

- [1] adding to the root when the Tree is a null pointer
- [1] using return and the Tree data type
- [1] retaining the WHILE loop
- [1] retaining the FOR loop
- [2] adding the two ELSE blocks which include return statements

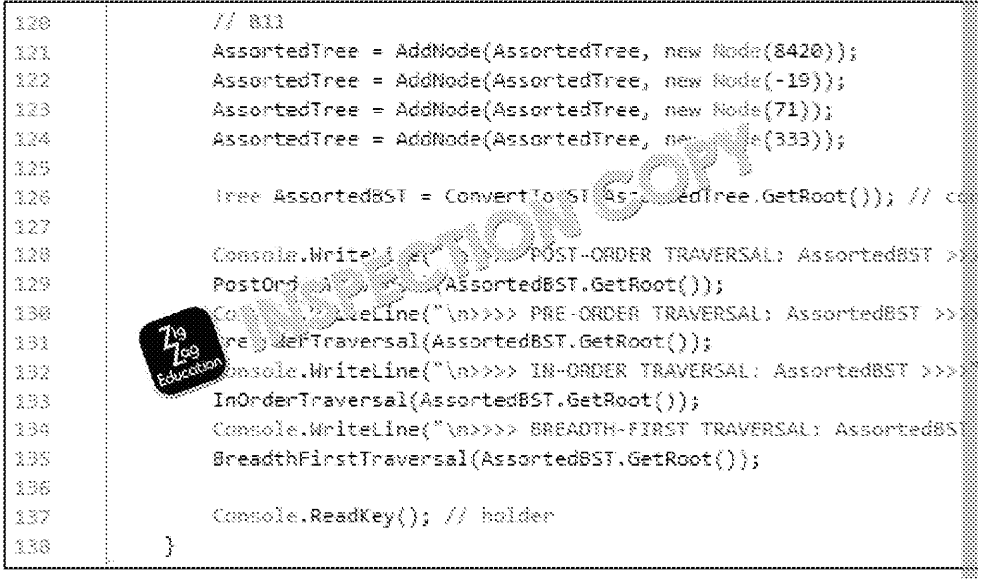
5 marks for the correct code in the Main method:

Marks are awarded for:

- [1] adding three or more new nodes to `AssortedTree`
- [1] declaring and initialising a new BST
- [1] deriving the BST values from `AssortedTree`
- [1] outputting all four traversals
- [1] correctness – the in-order traversal should be SORTED



```
// 811
public static Tree AddNode(Tree TreeToBeAddedTo, Node NewNode)
{
    // Create 2 lists of Node objects: one for the current level and one
    List<Node> NodesAtThisLevel = new List<Node>();
    List<Node> NodesAtNextLevel;

    // Handle empty trees
    if (TreeToBeAddedTo == null)
    {
        TreeToBeAddedTo.SetRoot(NewNode);
        return TreeToBeAddedTo;
    }

    // Initialise the list of nodes at this level with the root of the t
    NodesAtThisLevel.Add(TreeToBeAddedTo.GetRoot());

    // While a level with no nodes has not yet been encountered...
    while (NodesAtThisLevel.Count > 0)
    {
        // Form a new empty list for the next level's nodes
        NodesAtNextLevel = new List<Node>();

        // Visit all the nodes in the list
        for (int pointer = 0; pointer < NodesAtThisLevel.Count; pointer++
        {
            // If the node has a left/right node further down the tree,
            if (NodesAtThisLevel[pointer].GetLeft() != null)
            {
                NodesAtNextLevel.Add(NodesAtThisLevel[pointer].GetLeft()
            }
            else
            {
                NodesAtThisLevel[pointer].SetLeft(NewNode);
                return TreeToBeAddedTo;
            }
            if (NodesAtThisLevel[pointer].GetRight() != null)
            {
                NodesAtNextLevel.Add(NodesAtThisLevel[pointer].GetRight(
            }
            else
            {
                NodesAtThisLevel[pointer].SetRight(NewNode);
                return TreeToBeAddedTo;
            }
        }

        // Having visited all this level's nodes, set the next level's l
        NodesAtThisLevel = NodesAtNextLevel;
    }

    return TreeToBeAddedTo; // never used but has to be here for complet
}
```



```
    // 811
    AssortedTree = AddNode(AssortedTree, new Node(8420));
    AssortedTree = AddNode(AssortedTree, new Node(-19));
    AssortedTree = AddNode(AssortedTree, new Node(71));
    AssortedTree = AddNode(AssortedTree, new Node(333));

    Tree AssortedBST = ConvertToBST(AssortedTree.GetRoot()); // c

    Console.WriteLine("\n>>>> POST-ORDER TRAVERSAL: AssortedBST >
    PostOrderTraversal(AssortedBST.GetRoot());
    Console.WriteLine("\n>>>> PRE-ORDER TRAVERSAL: AssortedBST >>
    PreOrderTraversal(AssortedBST.GetRoot());
    Console.WriteLine("\n>>>> IN-ORDER TRAVERSAL: AssortedBST >>>
    InOrderTraversal(AssortedBST.GetRoot());
    Console.WriteLine("\n>>>> BREADTH-FIRST TRAVERSAL: AssortedBS
    BreadthFirstTraversal(AssortedBST.GetRoot());

    Console.ReadKey(); // holder
}
```

Exercise 7 – Dijkstra's Shortest Path

Section A

▩ A1

1 mark for correctly counting uses of the 'new' keyword:
25

▩ A2

1 mark for:
Line 100

▩ A3

2 marks (1 mark for a suitable definition; 1 mark for giving a valid reason to use pri

A public attribute is a (class or instance) variable that can be accessed from outsid made private to prevent it from being accidentally changed by other parts of the p
Alternative reason: it enables encapsulation of the entire class, meaning that you not worry about the details of how the class has been implemented (as an externa

▩ A4

1 mark for each appropriate comment given:

`Program.cs` Line 30:
A new edge [1] is built
... to connect node G to node H [1]
... one-way only [1]
... and it is assigned a weight of 10 [1].

▩ A5

1 mark for each appropriate comment given:

`Graph.cs` Lines 16–19:
A new graph is made [1] (as this is a constructor)
... but no parameters are passed in [1]
... so only an empty list of edges is created [1].

▩ A6

10 marks (1 mark per useful point):

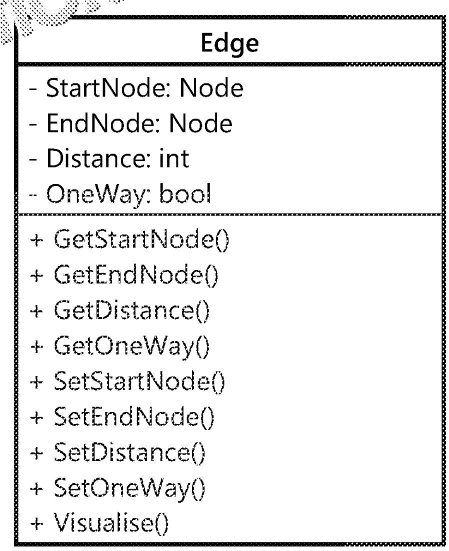
1. The method takes in one node and returns its closest node. [1]
2. If the node passed in as a parameter is null, return *Null* immediately. [1]
3. If the node sits on its own and there are no edges in the digraph, then ret
4. Next, the program iterates through all of the edges in the digraph and bu are connected directly to the node being investigated (`NodeToCheck`). It called EmanatingEdges and checks two possibilities:
    a. that the start node along that edge is the one being investigated
    b. that the end node along that edge is the one being investigated, one-way edge, in which case it would be relevant to include it [1
5. In the event that none of the edges in the temporary list (EmanatingEdge investigated, a *Null* is returned as no node exists as a meaningful answer.
6. Otherwise, the program will firstly output the temporary list of all of the e
7. It will then iterate through the list of edges and continuously overwrite th with every edge it finds to be shorter than previous ones. [1] In order temporary Edge variable with a high integer value so that any further con than this. [1]
8. To determine the node to be returned from this function, it checks if the n (`NodeToCheck`) is at the beginning of the edge, in which case it returns t Otherwise, it returns whichever node is at the beginning of the edge. [1]

A7

1 mark for each of the following:

1. Rectangular shape with three parts for: class name, attributes, methods [1
2. Class name = Edge [1]
3. Attributes = StartNode, EndNode, Distance, OneWay [1]. Data types show
4. Methods = 4× Get, 4× Set, Visualise [1]

Constructors can be omitted. Visibility settings can be omitted but all attributes ar public, indicated by – and + respectively below. Parameters can be omitted.



Edge

- StartNode: Node
- EndNode: Node
- Distance: int
- OneWay: bool

+ GetStartNode()
+ GetEndNode()
+ GetDistance()
+ GetOneWay()
+ SetStartNode()
+ SetEndNode()
+ SetDistance()
+ SetOneWay()
+ Visualise()

A8

2 marks (1 mark for explaining what a graph is; 1 mark for explaining the features o

A graph is a data type that consists of a set of nodes and a set of edges that conn A tree graph contains no loops and no two nodes are connected by more than on

A9

1 mark for explaining the purpose of Dijkstra's shortest path algorithm:

Dijkstra's shortest path algorithm is used to find the shortest path between two nod

A10

4 marks (1 mark for describing how all edges from the start node are checked; 1 mar to visit is selected; 1 mark for describing how nodes are not revisited; 1 mark for desc once the end node is visited):

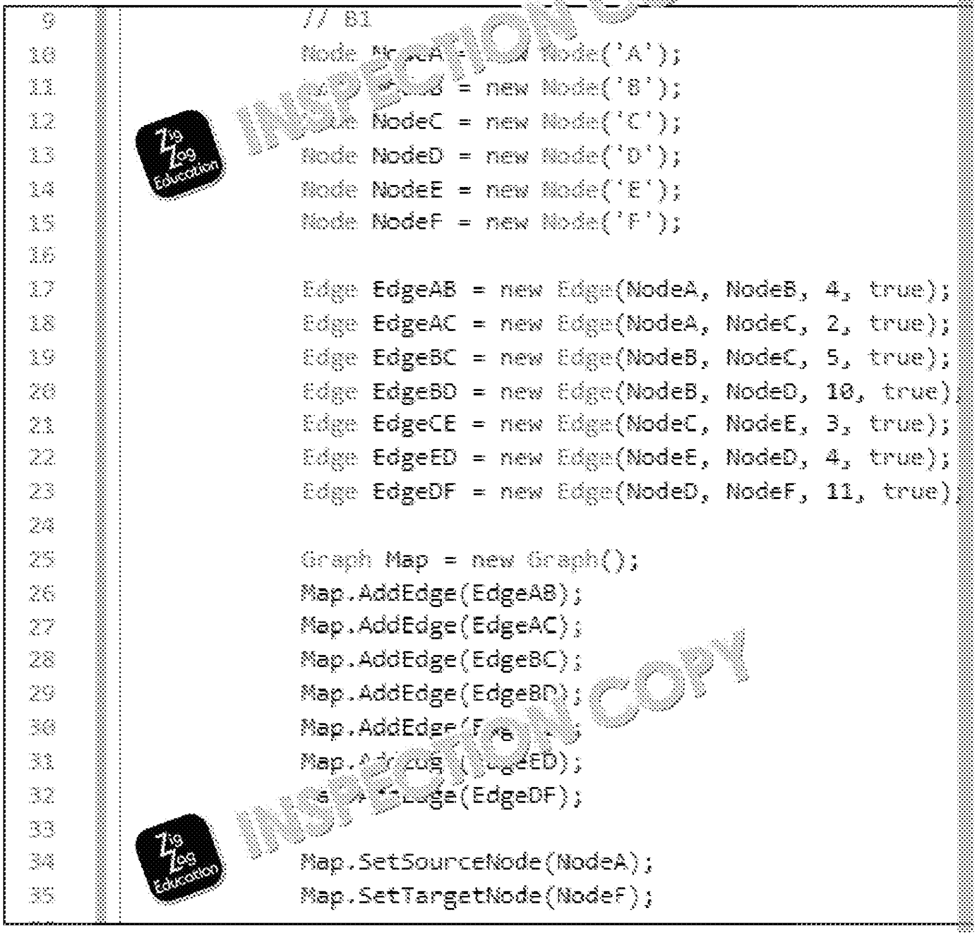
Dijkstra's algorithm starts at the given start node. All edges connected to this nod visits the closest connected node. The edges connected to the new node are chec note of the path that was taken to reach this node and visits whichever node is cl and hasn't already been visited. This process repeats until the given end node is vi

Section B

B1

1 mark available for each part of the modification (or equivalent code):

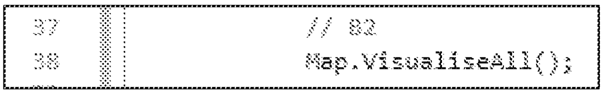
- creating six node objects [1]
- creating seven edge objects [1]
- creating and populating a graph object with all edges [1]
- setting the start and end nodes as A and F respectively [1]



```
            // B1
            Node NodeA = new Node('A');
            Node NodeB = new Node('B');
            Node NodeC = new Node('C');
            Node NodeD = new Node('D');
            Node NodeE = new Node('E');
            Node NodeF = new Node('F');

            Edge EdgeAB = new Edge(NodeA, NodeB, 4, true);
            Edge EdgeAC = new Edge(NodeA, NodeC, 2, true);
            Edge EdgeBC = new Edge(NodeB, NodeC, 5, true);
            Edge EdgeBD = new Edge(NodeB, NodeD, 10, true);
            Edge EdgeCE = new Edge(NodeC, NodeE, 3, true);
            Edge EdgeED = new Edge(NodeE, NodeD, 4, true);
            Edge EdgeDF = new Edge(NodeD, NodeF, 11, true);

            Graph Map = new Graph();
            Map.AddEdge(EdgeAB);
            Map.AddEdge(EdgeAC);
            Map.AddEdge(EdgeBC);
            Map.AddEdge(EdgeBD);
            Map.AddEdge(EdgeCE);
            Map.AddEdge(EdgeED);
            Map.AddEdge(EdgeDF);

            Map.SetSourceNode(NodeA);
            Map.SetTargetNode(NodeF);
```

B2

1 mark available for modifying Program.cs as shown without rewriting any code blo



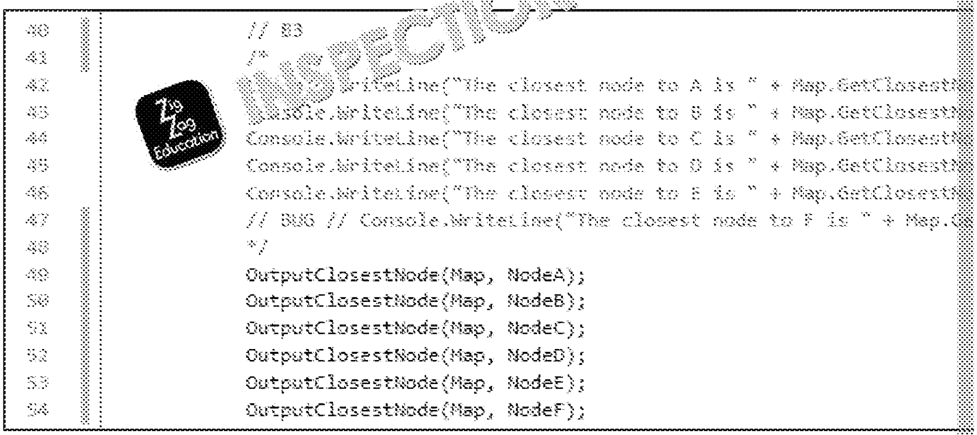
```
            // B2
            Map.VisualiseAll();
```

▩ B3

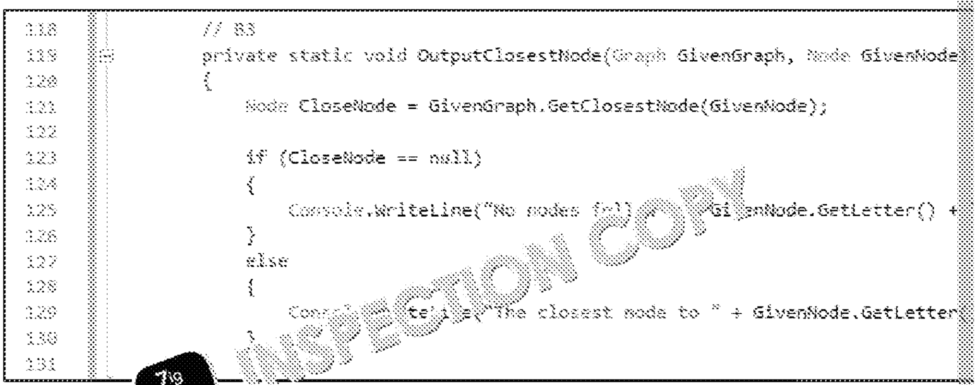
5 marks available for modifying the code as shown (or equivalent code):

Marks should be awarded for:

- calling the `GetClosestNode` method
- handling the case where `GetClosestNode` returns a null value
- handling the case where a node is returned
- correctly calling the new method from `Main()`
- correct use of private, static and parameters



```
            // B3
            /*
            Console.WriteLine("The closest node to A is " + Map.GetClosest
            Console.WriteLine("The closest node to B is " + Map.GetClosest
            Console.WriteLine("The closest node to C is " + Map.GetClosest
            Console.WriteLine("The closest node to D is " + Map.GetClosest
            Console.WriteLine("The closest node to E is " + Map.GetClosest
            // BUG // Console.WriteLine("The closest node to F is " + Map.G
            */
            OutputClosestNode(Map, NodeA);
            OutputClosestNode(Map, NodeB);
            OutputClosestNode(Map, NodeC);
            OutputClosestNode(Map, NodeD);
            OutputClosestNode(Map, NodeE);
            OutputClosestNode(Map, NodeF);
```



```
        // B3
        private static void OutputClosestNode(Graph GivenGraph, Node GivenNode
        {
            Node CloseNode = GivenGraph.GetClosestNode(GivenNode);

            if (CloseNode == null)
            {
                Console.WriteLine("No nodes fo[illegible] GivenNode.GetLetter() +
            }
            else
            {
                Console.WriteLine("The closest node to " + GivenNode.GetLetter
            }

```

▩ B4

3 marks per correctly completed table; Section A may proceed to visit row H (albeit

1 mark per correct path stated at the end

Section A

Node	Visited?	Shortest Distance to Start Node	P
A	V	0	-
B	V	∞ → 2	A
C	V	∞ → 10	B
D	V	∞ → 4	A
E	V	∞ → 11	D
F	V	∞ → 5	B
G	V	∞ → 11	D
H		∞ → 21	G
I		∞ → 24 → 20 MINIMUM VALUE FOUND	G →

Section B

Node	Visited?	Shortest Distance to Start Node	P
A	V	0	-
B	V	∞ → 4	A
C	V	∞ → 2	A
D	V	∞ → 14 → 9	B →
E	V	∞ → 5	C
F		∞ → 20 MINIMUM VALUE FOUND	D

- Se[illegible]: Shortest path from A to I = A, B, F, I [1]
- Sec[illegible] Shortest path from A to F = A, C, E, D, F [1]

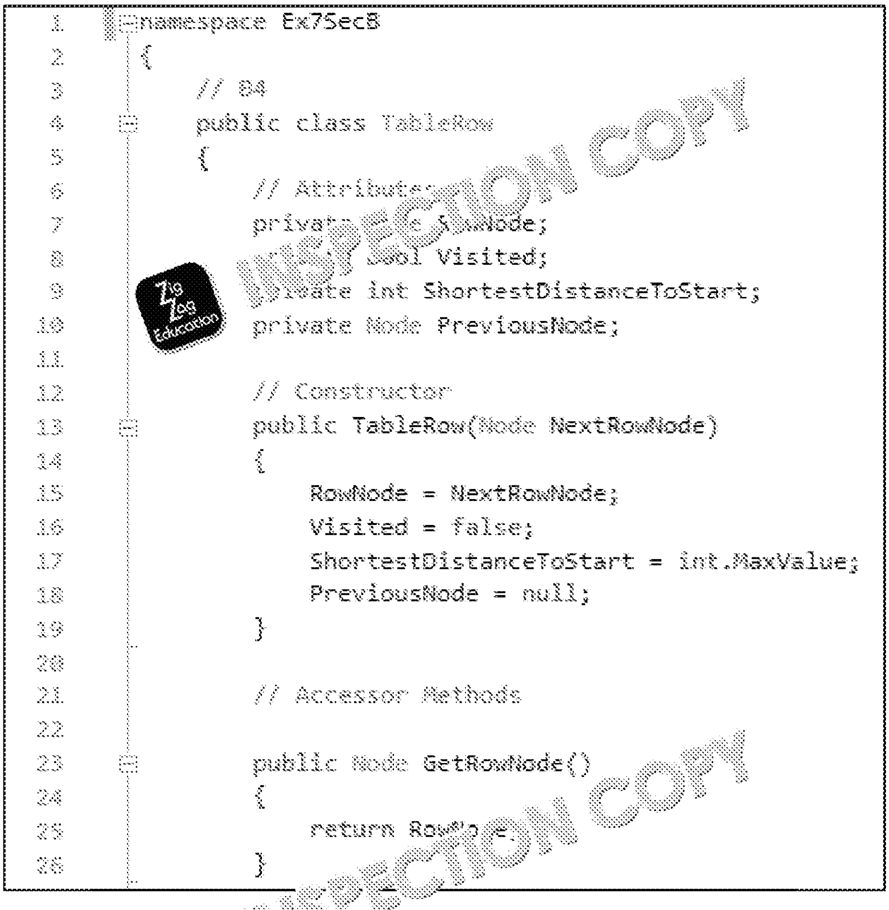
Program modifications:

Steps (1) and (2)

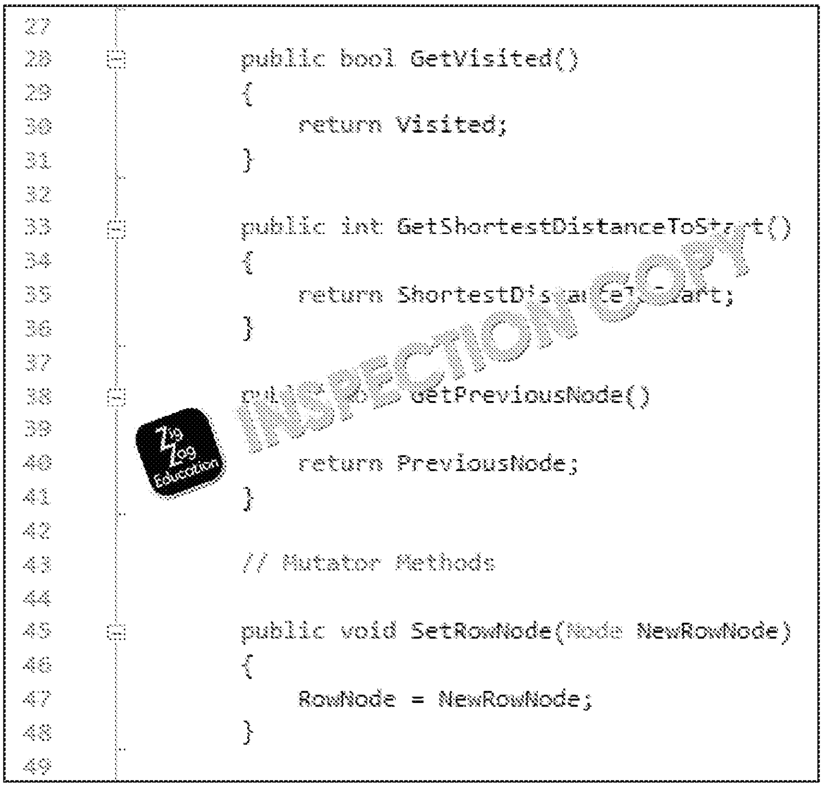
1 mark for a correct list of attributes

1 mark for a complete set of accessors and mutators

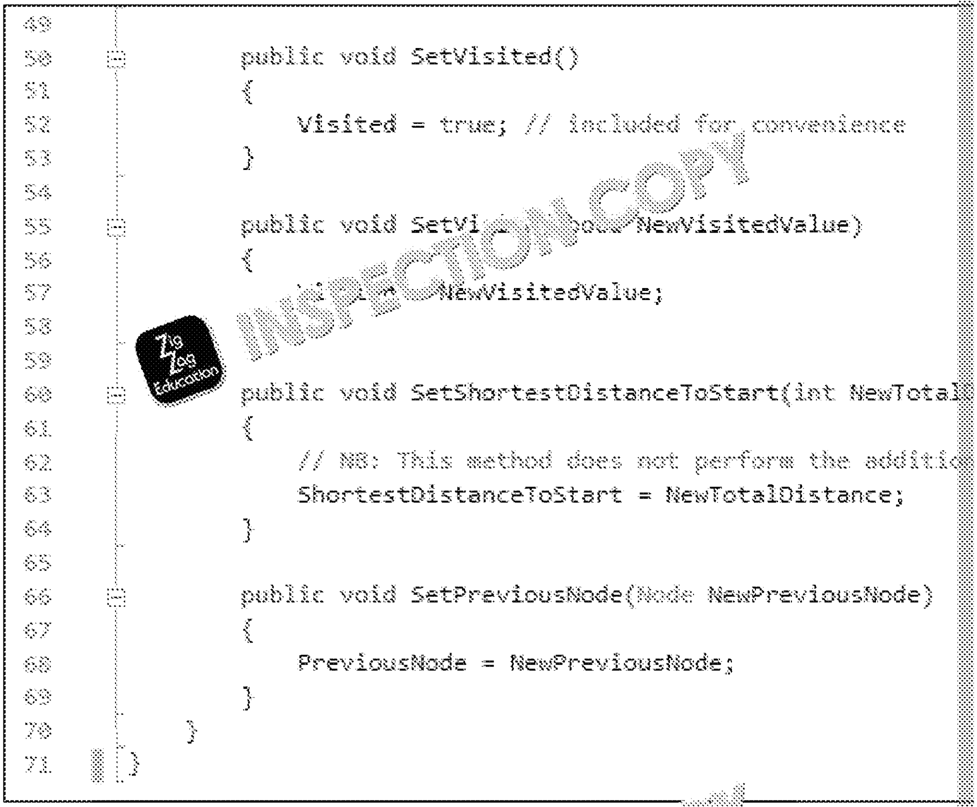
TableRow.cs



```
namespace Ex75ecB
{
    // B4
    public class TableRow
    {
        // Attributes
        private Node RowNode;
        private bool Visited;
        private int ShortestDistanceToStart;
        private Node PreviousNode;

        // Constructor
        public TableRow(Node NextRowNode)
        {
            RowNode = NextRowNode;
            Visited = false;
            ShortestDistanceToStart = int.MaxValue;
            PreviousNode = null;
        }

        // Accessor Methods

        public Node GetRowNode()
        {
            return RowNode;
        }
```



```
        public bool GetVisited()
        {
            return Visited;
        }

        public int GetShortestDistanceToStart()
        {
            return ShortestDistanceToStart;
        }

        public Node GetPreviousNode()
        {
            return PreviousNode;
        }

        // Mutator Methods

        public void SetRowNode(Node NewRowNode)
        {
            RowNode = NewRowNode;
        }

```



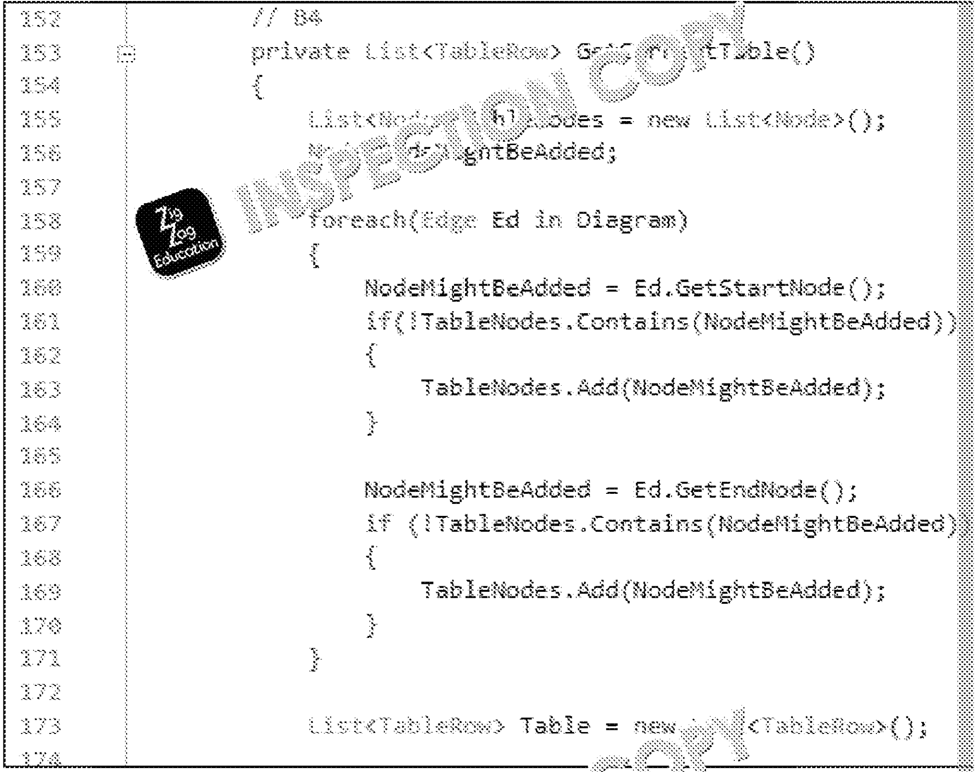
```

        public void SetVisited()
        {
            Visited = true; // included for convenience
        }

        public void SetVisited(bool NewVisitedValue)
        {
            Visited = NewVisitedValue;
        }

        public void SetShortestDistanceToStart(int NewTotal
        {
            // NB: This method does not perform the additio
            ShortestDistanceToStart = NewTotalDistance;
        }

        public void SetPreviousNode(Node NewPreviousNode)
        {
            PreviousNode = NewPreviousNode;
        }
    }
}
```

Graph.cs

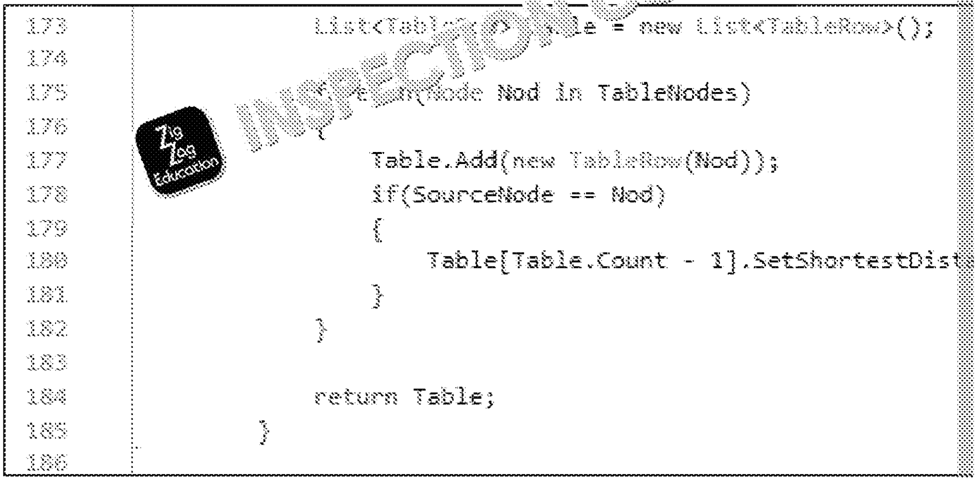
Step (3)

4 marks for successfully implementing all parts of the algorithm in C#

1 mark lost per area of difficulty encountered, i.e. not achieved



```
        // B4
        private List<TableRow> GetStartTable()
        {
            List<Node> TableNodes = new List<Node>();
            Node NodeMightBeAdded;

            foreach(Edge Ed in Diagram)
            {
                NodeMightBeAdded = Ed.GetStartNode();
                if(!TableNodes.Contains(NodeMightBeAdded))
                {
                    TableNodes.Add(NodeMightBeAdded);
                }

                NodeMightBeAdded = Ed.GetEndNode();
                if (!TableNodes.Contains(NodeMightBeAdded))
                {
                    TableNodes.Add(NodeMightBeAdded);
                }
            }

            List<TableRow> Table = new List<TableRow>();
```

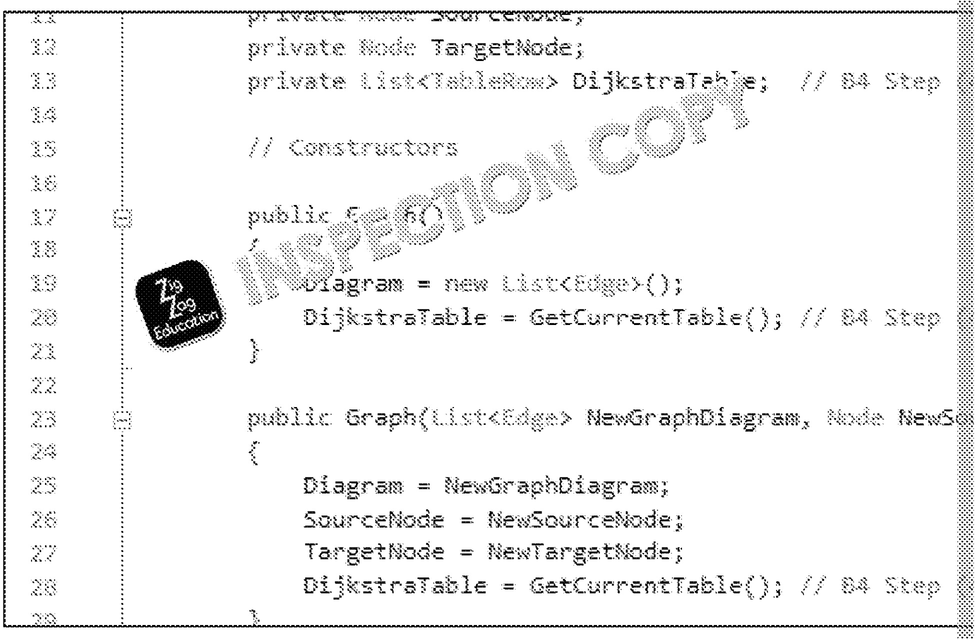


```
            List<TableRow> Table = new List<TableRow>();

            foreach(Node Nod in TableNodes)
            {
                Table.Add(new TableRow(Nod));
                if(SourceNode == Nod)
                {
                    Table[Table.Count - 1].SetShortestDist
                }
            }

            return Table;
        }
```

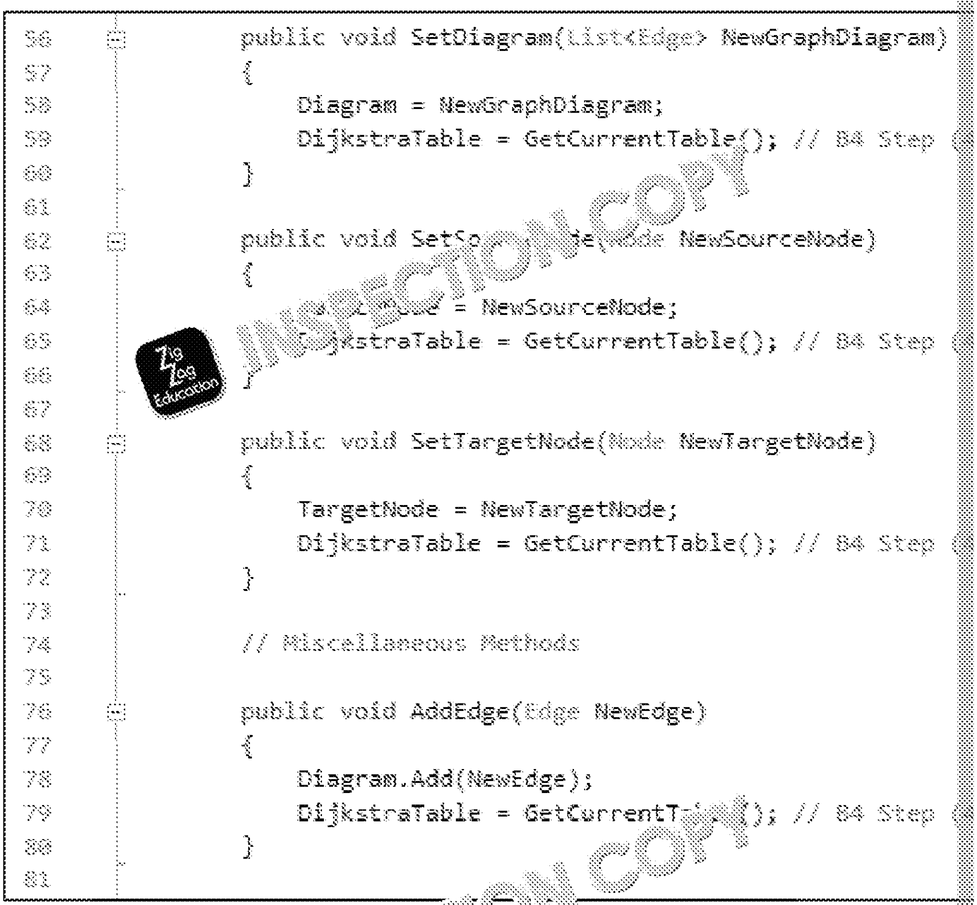
Step (4)

6 marks for successfully implementing all parts of the algorithm in C#

1 mark lost per area of difficulty encountered, i.e. not achieved



```
        private Node SourceNode;
        private Node TargetNode;
        private List<TableRow> DijkstraTable;  // B4 Step (

        // Constructors

        public Graph()
        {
            Diagram = new List<Edge>();
            DijkstraTable = GetCurrentTable(); // B4 Step (
        }

        public Graph(List<Edge> NewGraphDiagram, Node NewSo
        {
            Diagram = NewGraphDiagram;
            SourceNode = NewSourceNode;
            TargetNode = NewTargetNode;
            DijkstraTable = GetCurrentTable(); // B4 Step (
        }
```

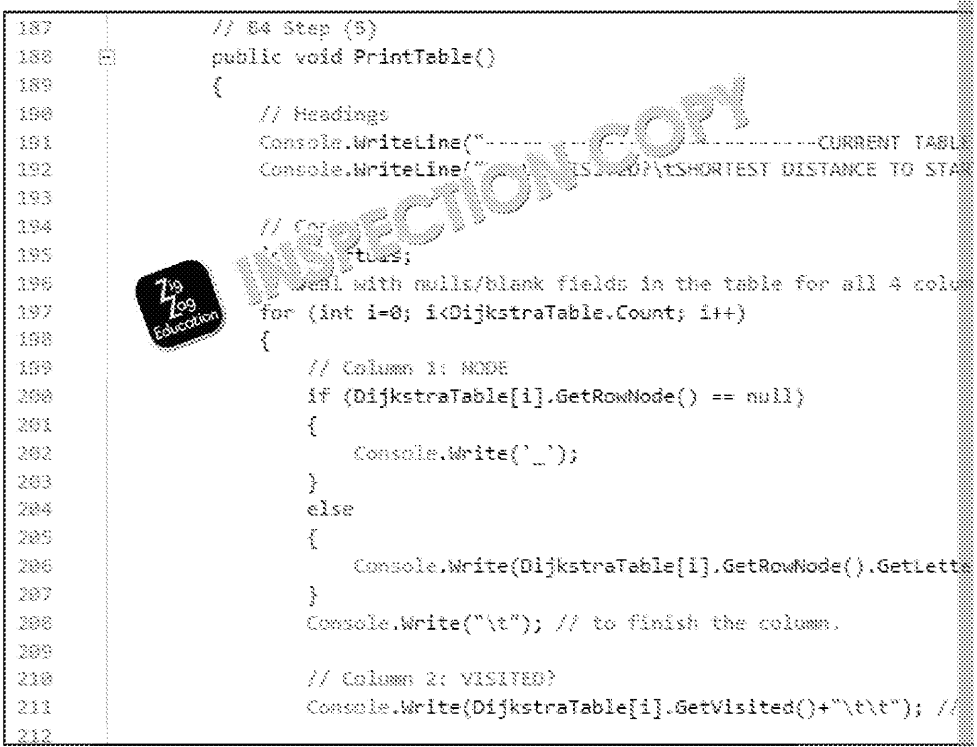


```
        public void SetDiagram(List<Edge> NewGraphDiagram)
        {
            Diagram = NewGraphDiagram;
            DijkstraTable = GetCurrentTable(); // B4 Step (
        }

        public void SetSourceNode(Node NewSourceNode)
        {
            SourceNode = NewSourceNode;
            DijkstraTable = GetCurrentTable(); // B4 Step (
        }

        public void SetTargetNode(Node NewTargetNode)
        {
            TargetNode = NewTargetNode;
            DijkstraTable = GetCurrentTable(); // B4 Step (
        }

        // Miscellaneous Methods

        public void AddEdge(Edge NewEdge)
        {
            Diagram.Add(NewEdge);
            DijkstraTable = GetCurrentTable(); // B4 Step (
        }
```

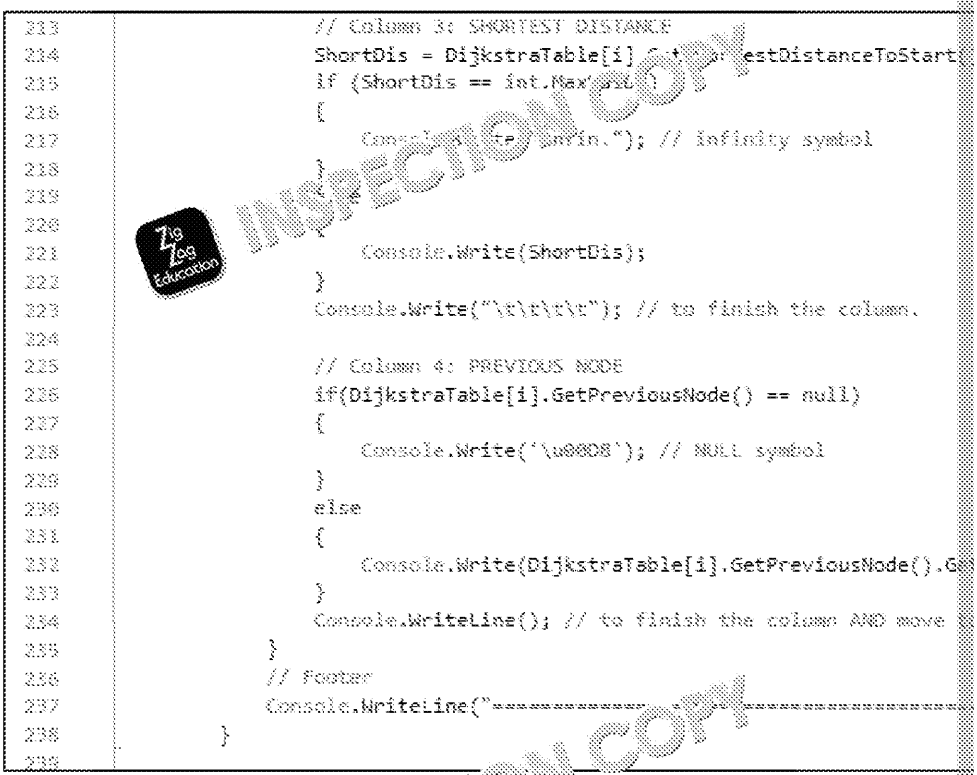
Step (5)

6 marks for successfully implementing all parts of the algorithm in C#

1 mark lost per area of difficulty encountered, i.e. not achieved



```
        // B4 Step (5)
        public void PrintTable()
        {
            // Headings
            Console.WriteLine("------ -- -- -- ----------CURRENT TABL
            Console.WriteLine("[illegible] VISITED?\tSHORTEST DISTANCE TO STA

            // Co[illegible]
            [illegible]tuas;
            // Deal with nulls/blank fields in the table for all 4 colu
            for (int i=0; i<DijkstraTable.Count; i++)
            {
                // Column 1: NODE
                if (DijkstraTable[i].GetRowNode() == null)
                {
                    Console.Write('_');
                }
                else
                {
                    Console.Write(DijkstraTable[i].GetRowNode().GetLett
                }
                Console.Write("\t"); // to finish the column.

                // Column 2: VISITED?
                Console.Write(DijkstraTable[i].GetVisited()+"\t\t"); //

```



```
                // Column 3: SHORTEST DISTANCE
                ShortDis = DijkstraTable[i].Get[illegible]estDistanceToStart(
                if (ShortDis == int.Max[illegible])
                {
                    Con[illegible]te[illegible]in."); // infinity symbol
                }
                [illegible]
                {
                    Console.Write(ShortDis);
                }
                Console.Write("\t\t\t\t"); // to finish the column.

                // Column 4: PREVIOUS NODE
                if(DijkstraTable[i].GetPreviousNode() == null)
                {
                    Console.Write('\u00D8'); // NULL symbol
                }
                else
                {
                    Console.Write(DijkstraTable[i].GetPreviousNode().G
                }
                Console.WriteLine(); // to finish the column AND move
            }
            // Footer
            Console.WriteLine("============ ===================
        }

```

▩ B5

1 mark avai[illegible]or [illegible] the main bullet points in the question being fully imple given

Marks could be further:

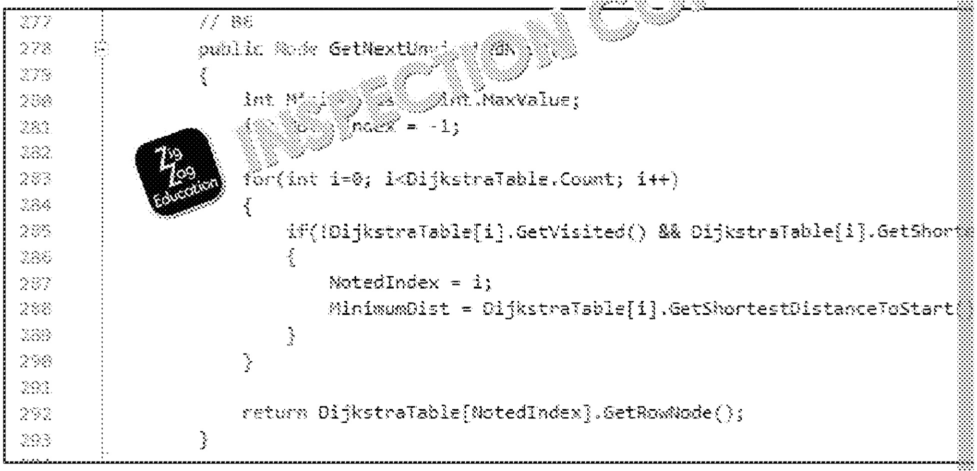
- awarded as bonuses for coming up with original extensions/improvemen
- awarded for excellent code style even if some areas required assistance
- deducted for very poor coding style

B6

3 marks available for a complete, working implementation of the requirements, eve
example shown

Marks are awarded for:

- a correct method signature and return statement
- the correct deployment of IF and FOR
- accurate use of list indexing and method calls throughout



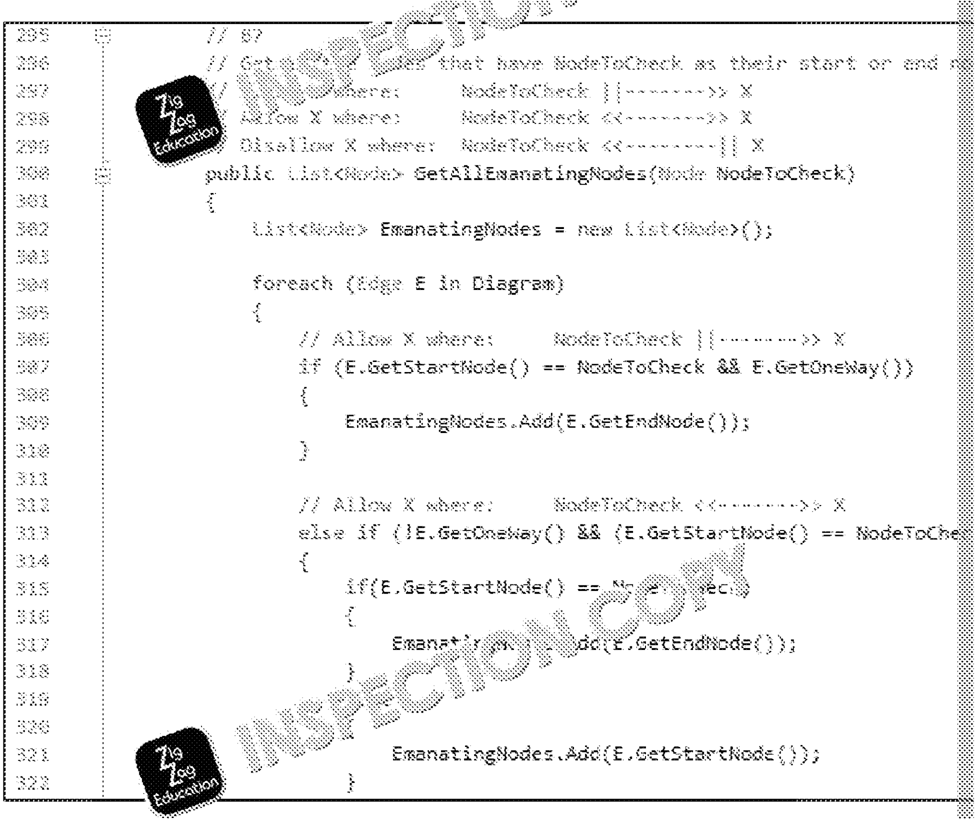
```
        // B6
        public Node GetNextUnvisitedNode()
        {
            int MinimumDist = int.MaxValue;
            int NotedIndex = -1;

            for(int i=0; i<DijkstraTable.Count; i++)
            {
                if(!DijkstraTable[i].GetVisited() && DijkstraTable[i].GetShort
                {
                    NotedIndex = i;
                    MinimumDist = DijkstraTable[i].GetShortestDistanceToStart
                }
            }

            return DijkstraTable[NotedIndex].GetRowNode();
        }
```

B7

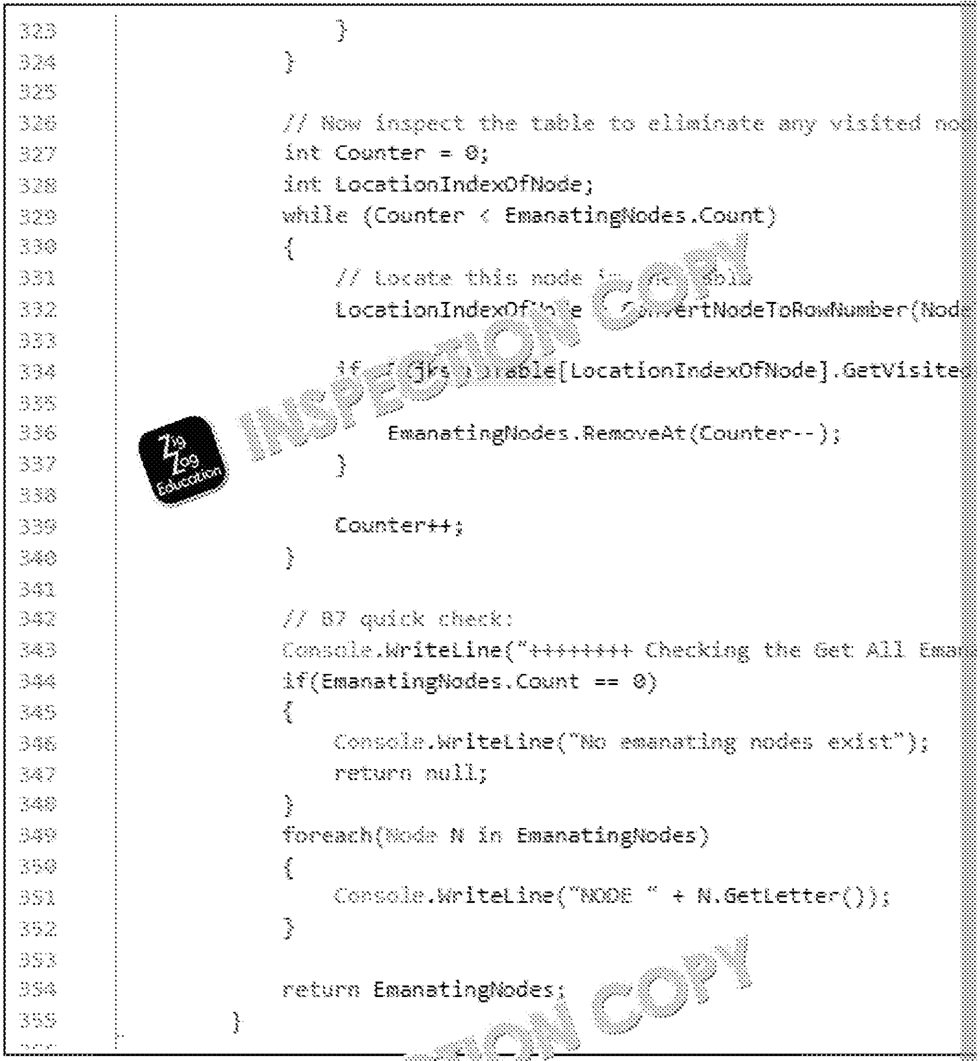
5 marks available for implementing the code as shown (or equivalent code):

Marks could be awarded for:

- [2] finding the relevant nodes which emanate from the given node in the di
- [2] isolating only those that have not yet been visited
- [1] implementing the quick check and returning a successful outcome



```
        // B7
        // Get all the nodes that have NodeToCheck as their start or end n
        // Allow X where:     NodeToCheck ||-------->> X
        // Allow X where:     NodeToCheck <<-------->> X
        // Disallow X where:  NodeToCheck <<--------|| X
        public List<Node> GetAllEmanatingNodes(Node NodeToCheck)
        {
            List<Node> EmanatingNodes = new List<Node>();

            foreach (Edge E in Diagram)
            {
                // Allow X where:     NodeToCheck ||-------->> X
                if (E.GetStartNode() == NodeToCheck && E.GetOneWay())
                {
                    EmanatingNodes.Add(E.GetEndNode());
                }

                // Allow X where:     NodeToCheck <<-------->> X
                else if (!E.GetOneWay() && (E.GetStartNode() == NodeToChe
                {
                    if(E.GetStartNode() == NodeToCheck)
                    {
                        EmanatingNodes.Add(E.GetEndNode());
                    }


                        EmanatingNodes.Add(E.GetStartNode());
                    }
```



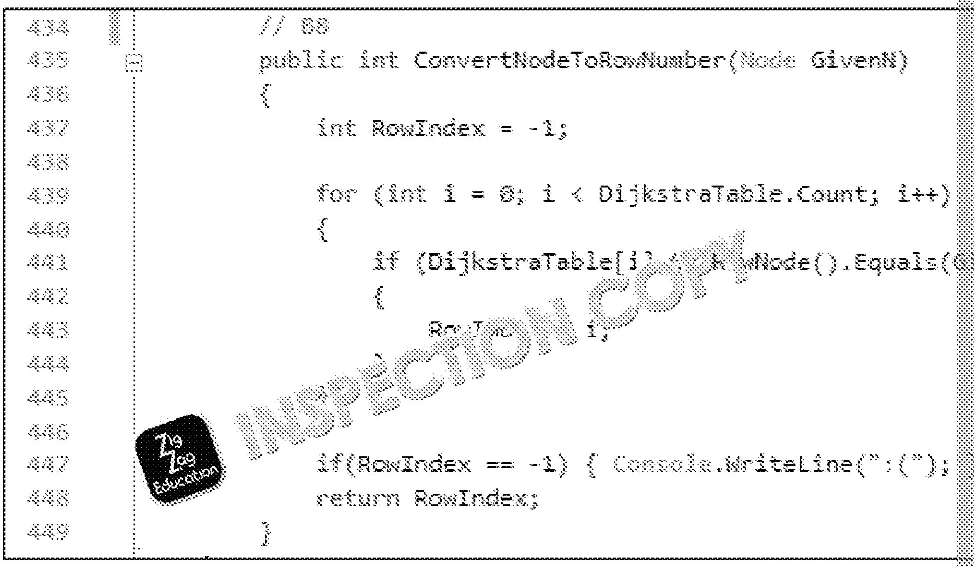
```
                }
            }

            // Now inspect the table to eliminate any visited no
            int Counter = 0;
            int LocationIndexOfNode;
            while (Counter < EmanatingNodes.Count)
            {
                // Locate this node in the table
                LocationIndexOfNode = ConvertNodeToRowNumber(Nod

                if (DijkstraTable[LocationIndexOfNode].GetVisite
                {
                    EmanatingNodes.RemoveAt(Counter--);
                }

                Counter++;
            }

            // B7 quick check:
            Console.WriteLine("++++++++ Checking the Get All Eman
            if(EmanatingNodes.Count == 0)
            {
                Console.WriteLine("No emanating nodes exist");
                return null;
            }
            foreach(Node N in EmanatingNodes)
            {
                Console.WriteLine("NODE " + N.GetLetter());
            }

            return EmanatingNodes;
        }
```

■ B8

3 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- iterating through all rows
- appropriately checking for the equality of two objects
- handling the return statement, including consideration of possible errors



```
        // B8
        public int ConvertNodeToRowNumber(Node GivenN)
        {
            int RowIndex = -1;

            for (int i = 0; i < DijkstraTable.Count; i++)
            {
                if (DijkstraTable[i].GetRowNode().Equals(G
                {
                    RowIndex = i;
                }
            }

            if(RowIndex == -1) { Console.WriteLine(":(");
            return RowIndex;
        }
```

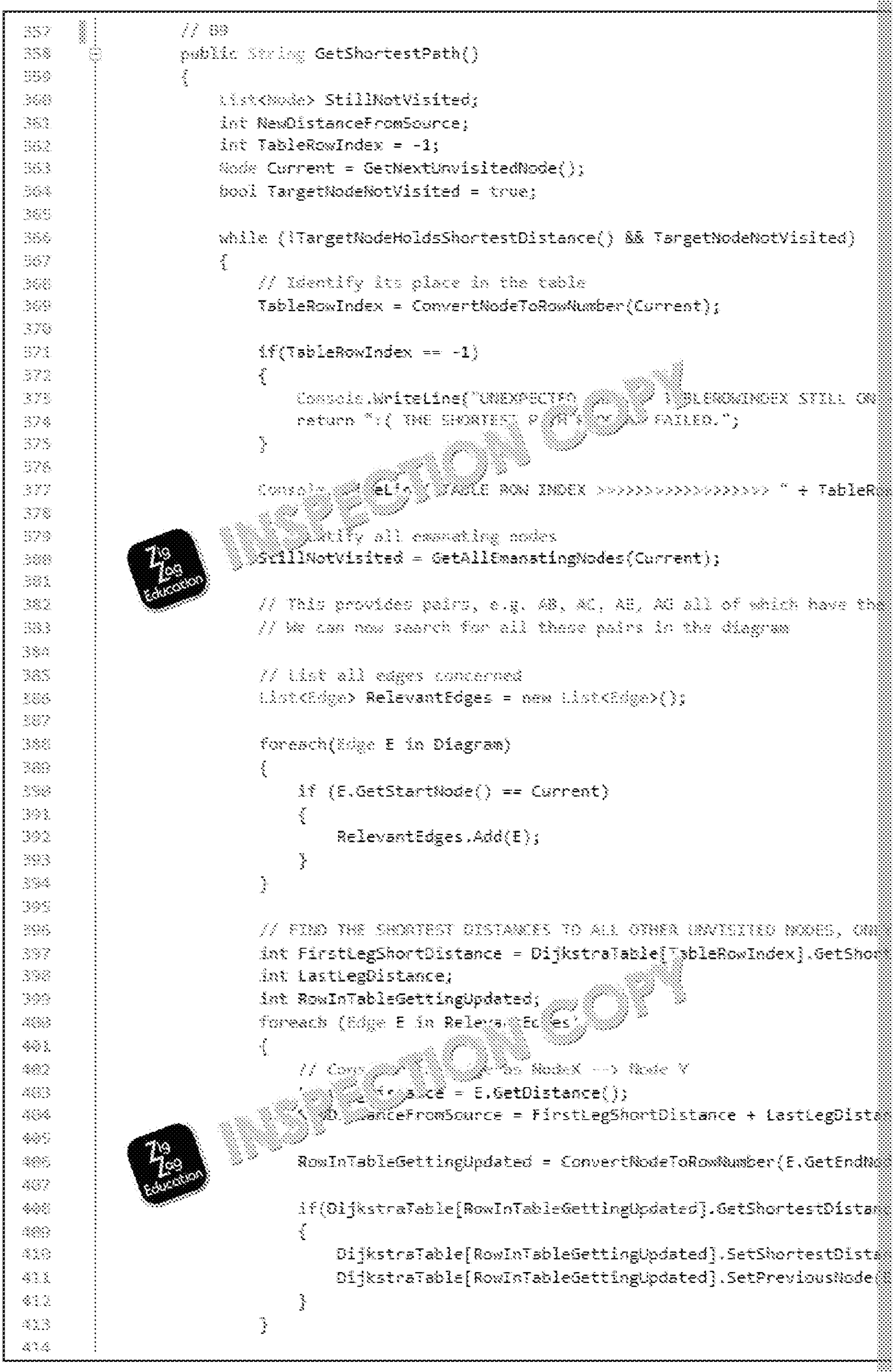
B9

9 marks available for modifying the code as shown (or equivalent code):

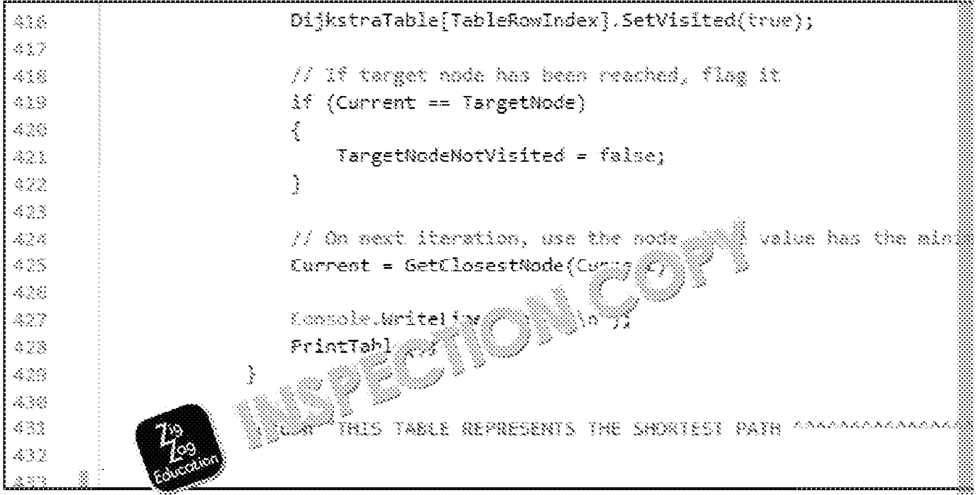
This task represents the 'coming together' of several earlier sections and will be in managed to complete earlier sections or who have not been furnished with workin completion of this task.

Marks could be awarded for:

- [1] implementing variables as described
- [1] establishing the WHILE loop
- [1] using `TableRowIndex` correctly (NB Line 377 is for testing only)
- [1] deploying `GetAllEmanatingNodes` correctly
- [1] [illegible] of relevant edges
- [3] [illegible] the shortest distances and updating the table with ONLY the s
- [1] terminating the iteration correctly and returning a string but outputtin



```
357          // B9
358          public String GetShortestPath()
359          {
360              List<Node> StillNotVisited;
361              int NewDistanceFromSource;
362              int TableRowIndex = -1;
363              Node Current = GetNextUnvisitedNode();
364              bool TargetNodeNotVisited = true;
365
366              while (!TargetNodeHoldsShortestDistance() && TargetNodeNotVisited)
367              {
368                  // Identify its place in the table
369                  TableRowIndex = ConvertNodeToRowNumber(Current);
370
371                  if(TableRowIndex == -1)
372                  {
373                      Console.WriteLine("UNEXPECTED [illegible] TABLEROWINDEX STILL ON
374                      return ":( THE SHORTEST PATH [illegible] FAILED.";
375                  }
376
377                  Console.WriteLine("TABLE ROW INDEX >>>>>>>>>>>>>>>>>> " + TableR
378
379                  // Identify all emanating nodes
380                  StillNotVisited = GetAllEmanatingNodes(Current);
381
382                  // This provides pairs, e.g. AB, AC, AE, AG all of which have the
383                  // We can now search for all these pairs in the diagram
384
385                  // List all edges concerned
386                  List<Edge> RelevantEdges = new List<Edge>();
387
388                  foreach(Edge E in Diagram)
389                  {
390                      if (E.GetStartNode() == Current)
391                      {
392                          RelevantEdges.Add(E);
393                      }
394                  }
395
396                  // FIND THE SHORTEST DISTANCES TO ALL OTHER UNVISITED NODES, ON
397                  int FirstLegShortDistance = DijkstraTable[TableRowIndex].GetSho
398                  int LastLegDistance;
399                  int RowInTableGettingUpdated;
400                  foreach (Edge E in RelevantEdges)
401                  {
402                      // Con[illegible] as NodeX --> Node Y
403                      LastLegDistance = E.GetDistance();
404                      NewDistanceFromSource = FirstLegShortDistance + LastLegDista
405
406                      RowInTableGettingUpdated = ConvertNodeToRowNumber(E.GetEndNo
407
408                      if(DijkstraTable[RowInTableGettingUpdated].GetShortestDista
409                      {
410                          DijkstraTable[RowInTableGettingUpdated].SetShortestDista
411                          DijkstraTable[RowInTableGettingUpdated].SetPreviousNode(
412                      }
413                  }
414
```



```
                DijkstraTable[TableRowIndex].SetVisited(true);

                // If target node has been reached, flag it
                if (Current == TargetNode)
                {
                    TargetNodeNotVisited = false;
                }

                // On next iteration, use the node whose value has the min
                Current = GetClosestNode(Current);

                Console.WriteLine("\n");
                PrintTable();
            }

            // ^^^ THIS TABLE REPRESENTS THE SHORTEST PATH ^^^^^^^^^^^^^^^
```

Exercise 8 – Bomb Search

Section A

▩ A1

1 mark for:

`GetMove()`

▩ A2

1 mark for:

Class name: Program
Line number: 1[illegible]

▩ A3

3 marks for:

The number of bombs that will be present/placed on the board... [1]
... will be one fifth of the total number of tiles...[1]
... but this is a DIV operation, so it will truncate any decimals [1] (it won't be exactl

▩ A4

3 marks for:

All new tiles are, by default, set to be hidden (not revealed) [1]
... and the number of adjacent bombs cannot yet be known, so it defaults to 0 [1]
... but whether it is a bomb or not can be passed in as a parameter [1].

▩ A5

1 mark for explaining one reason why the redundant code is useful, not how it works

During the game, tiles will be regularly revealed, so [illegible] of the method is int
require the programmer to pass in any parameter. [illegible]ing a tile is not a feature of
even if there is no `SetReveal()` [illegible] method.

▩ A6

1 mark for [illegible] it is an overflow exception, or more precisely:

`System.OverflowException`

▩ A7

*2 marks (1 mark for explaining how try–catch statements work, and 1 mark for explai
be useful in this case):*

[1] How it works: Try–catch statements are used to handle errors by attempting to
running the code in the catch block instead if an error is thrown within the try bloc
[1] In this case: If a System.OverflowException arises because negative numbers ar
program can exit cleanly or request input from the user to prevent it from crashin

▩ A8

2 marks for:

A list can contain any number of elements, [1]
... whereas an array has a fixed size (it is immutable). [1]
Also: The size of an array is stored as an attribute, but when working with lists a m
its size at run-time.

▩ A9

Any 2 marks from the following:

While it would still work, [1]
... the grid structure of the game board is suited to an immutable array [1]
... as it will never need to grow/shrink during the game [1]
... and arrays offer all of the structural features required [1].
Lists could perhaps be useful if adding advanced features later such as hidden zone

A10

1 mark for writing the correct code:

```
Arena.GetLength(1);
```

Section B

B1

5 marks available for the code as shown (or equivalent code):

- [1] for outputting revealed state
- [1] for outputting bomb status
- [1] for outputting the number of adjacent bombs
- [1] for style and clarity of output, e.g. meaningful sentences, not raw num
- [1] for suitable test code being added to `Program.cs`

B2

8 marks available for modifying the code as shown (or equivalent code):

- [1] per correctly functioning accessor method with suitable return type an
- [1] per correctly functioning mutator method with void return type and on

B3

16 marks available for modifying the code as shown (or equivalent code):

Marks are awarded for:

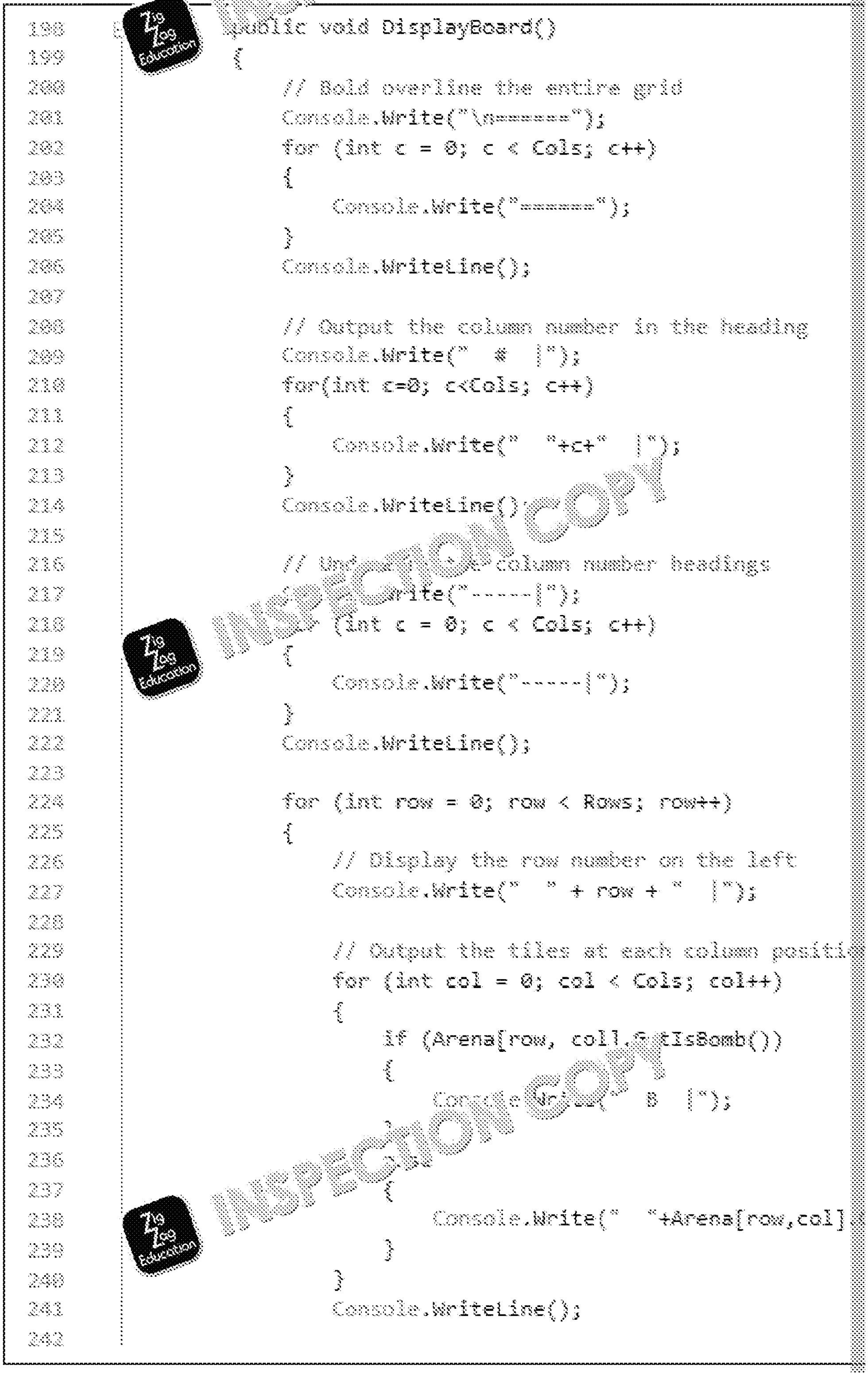
- [1] using the variable Bombs to make a list of Bomb tiles of the correct le
- [1] using Rows*Cols-Bombs to make a list of Safe tiles
- [1] creating an empty shuffled list
- [1] establishing a WHILE loop with two criteria joined using AND
- [1] implementing random selection (thus achieving shuffling)
- [1] adding the correct tile to the shuffled list each time
- [1] removing the correct tile from the correct list each time
- [1] adding any leftover elements from the correct list after one list is e
- [1] placing the tiles into the 2D array Arena
- [7] for building and applying a method for updating the adjacent bomb v the algorithm (described in the task) being achieved

■ B4

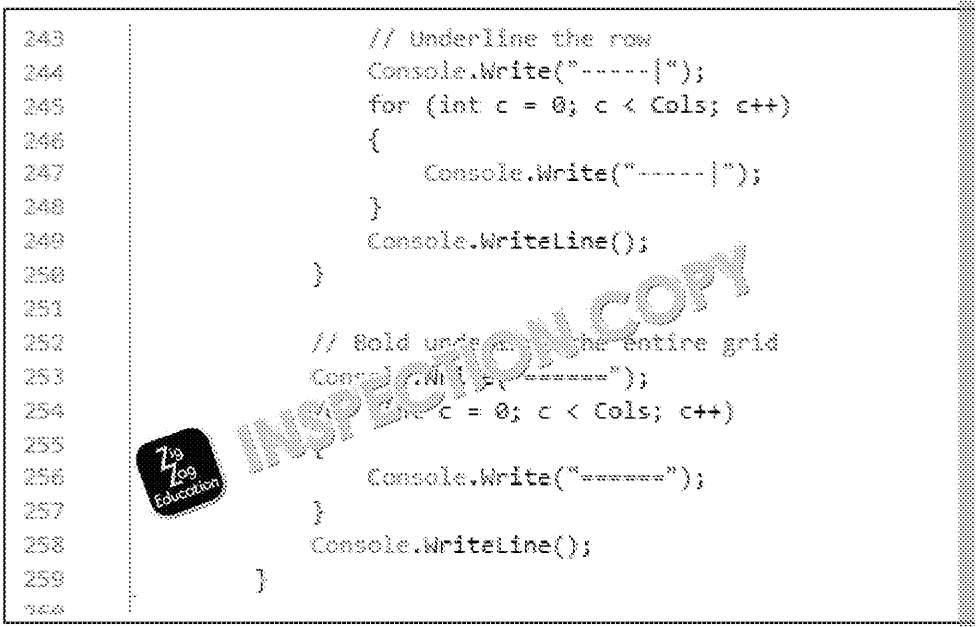
8 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- [1] displaying B for a bomb
- [1] displaying a single digit for how many bombs surround a tile
- [1] for partitioning the rows
- [1] for partitioning the columns
- [1] for displaying row numbers
- [1] for displaying column numbers
- [2] for the `DisplayGameBoard` method showing ? symbols and spaces

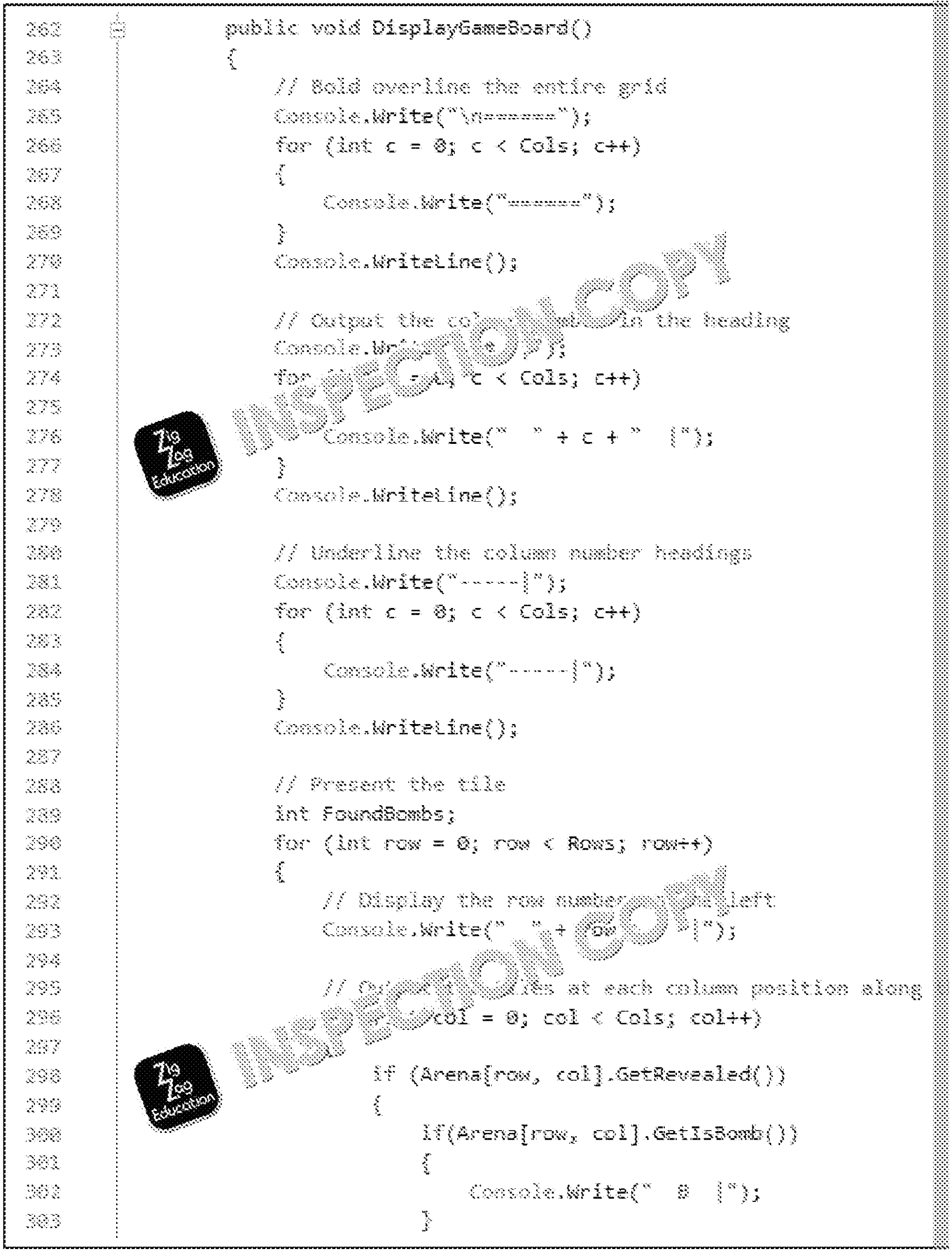


```
public void DisplayBoard()
{
    // Bold overline the entire grid
    Console.Write("\n======");
    for (int c = 0; c < Cols; c++)
    {
        Console.Write("======");
    }
    Console.WriteLine();

    // Output the column number in the heading
    Console.Write("  #  |");
    for(int c=0; c<Cols; c++)
    {
        Console.Write("  "+c+"  |");
    }
    Console.WriteLine();

    // Underline the column number headings
    Console.Write("-----|");
    for (int c = 0; c < Cols; c++)
    {
        Console.Write("-----|");
    }
    Console.WriteLine();

    for (int row = 0; row < Rows; row++)
    {
        // Display the row number on the left
        Console.Write("  " + row + "  |");

        // Output the tiles at each column positi
        for (int col = 0; col < Cols; col++)
        {
            if (Arena[row, col].GetIsBomb())
            {
                Console.Write("  B  |");
            }
            else
            {
                Console.Write("  "+Arena[row,col]
            }
        }
        Console.WriteLine();

```

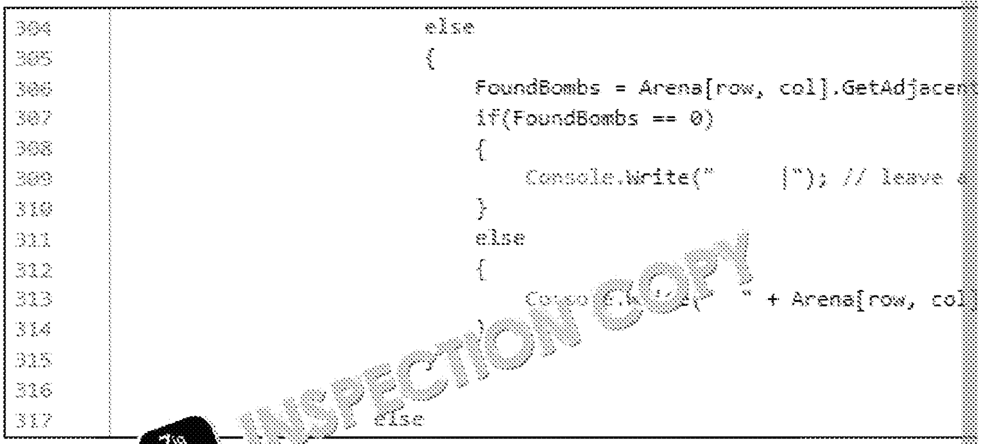


```
                // Underline the row
                Console.Write("-----|");
                for (int c = 0; c < Cols; c++)
                {
                    Console.Write("-----|");
                }
                Console.WriteLine();
            }

            // Bold underline the entire grid
            Console.Write("======");
            for (int c = 0; c < Cols; c++)
            {
                Console.Write("======");
            }
            Console.WriteLine();
        }
```

For the DisplayGameBoard method:

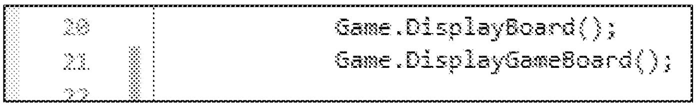


```
        public void DisplayGameBoard()
        {
            // Bold overline the entire grid
            Console.Write("\n======");
            for (int c = 0; c < Cols; c++)
            {
                Console.Write("======");
            }
            Console.WriteLine();

            // Output the column numbers in the heading
            Console.Write("     |");
            for (int c = 0; c < Cols; c++)
            {
                Console.Write("  " + c + "  |");
            }
            Console.WriteLine();

            // Underline the column number headings
            Console.Write("-----|");
            for (int c = 0; c < Cols; c++)
            {
                Console.Write("-----|");
            }
            Console.WriteLine();

            // Present the tile
            int FoundBombs;
            for (int row = 0; row < Rows; row++)
            {
                // Display the row number on the left
                Console.Write("  " + row + "  |");

                // Output the tiles at each column position along
                for (int col = 0; col < Cols; col++)
                {
                    if (Arena[row, col].GetRevealed())
                    {
                        if(Arena[row, col].GetIsBomb())
                        {
                            Console.Write("  8  |");
                        }
```



```
                        else
                        {
                            FoundBombs = Arena[row, col].GetAdjacen
                            if(FoundBombs == 0)
                            {
                                Console.Write("     |"); // leave 
                            }
                            else
                            {
                                Console.Write("  " + Arena[row, col
                            }


                    else
```

From Program.cs:



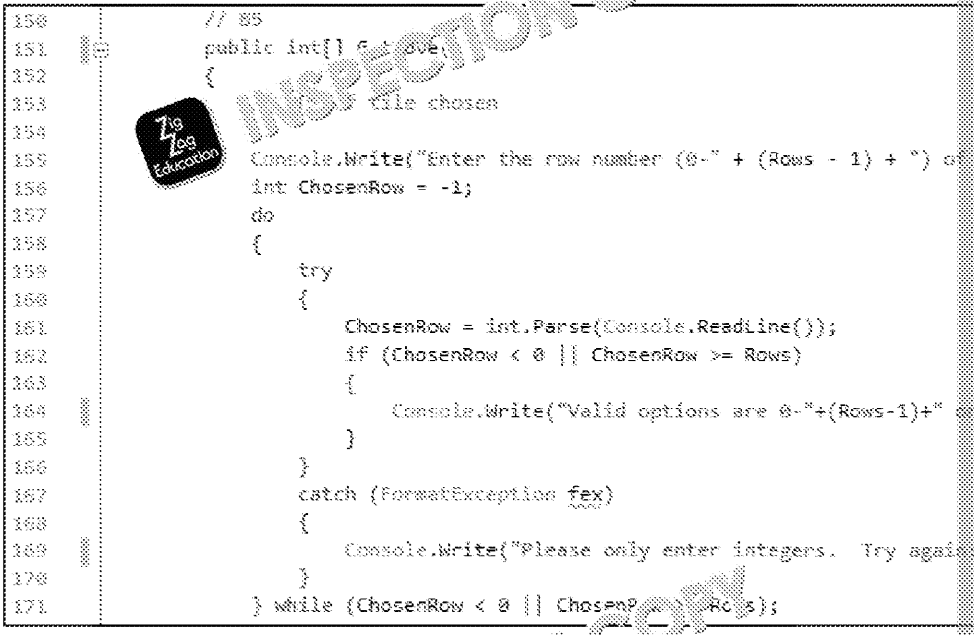
```
            Game.DisplayBoard();
            Game.DisplayGameBoard();
```

▩ B5

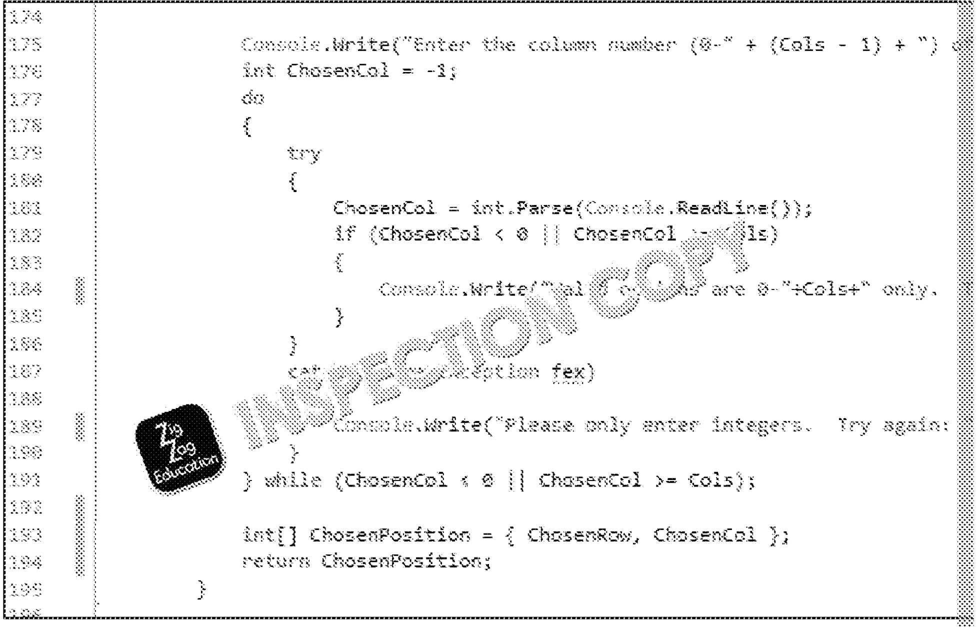
6 marks available for programming the GetMove() function as shown below (or equi

Marks could be awarded for:

- [1] per DO-WHILE (or alternative) to disallow progress until a valid value i
- [1] per use of TRY-CATCH to intercept invalid data types, max of [2] mark
- [1] for clear outputs to prompt the user throughout
- [1] for returning an array of two values



```
        // B5
        public int[] GetMove()
        {
            // [illegible] tile chosen

            Console.Write("Enter the row number (0-" + (Rows - 1) + ") o
            int ChosenRow = -1;
            do
            {
                try
                {
                    ChosenRow = int.Parse(Console.ReadLine());
                    if (ChosenRow < 0 || ChosenRow >= Rows)
                    {
                        Console.Write("Valid options are 0-"+(Rows-1)+" 
                    }
                }
                catch (FormatException fex)
                {
                    Console.Write("Please only enter integers.  Try agai
                }
            } while (ChosenRow < 0 || ChosenRow >= Rows);
```



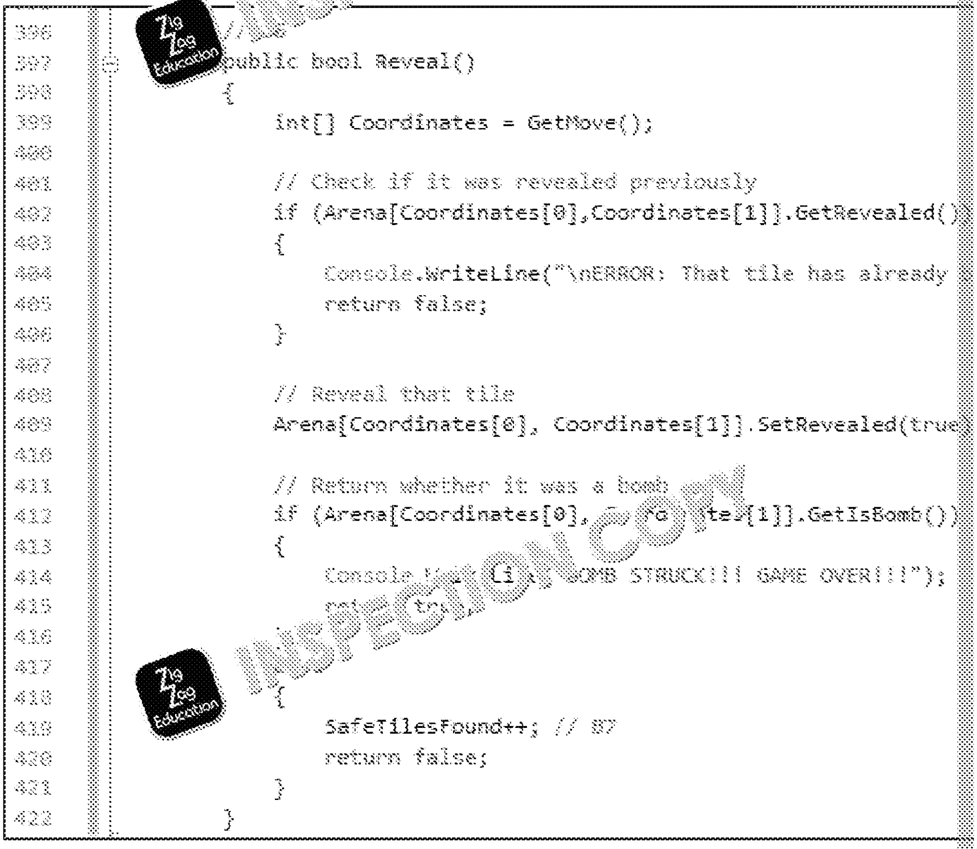
```
            Console.Write("Enter the column number (0-" + (Cols - 1) + ") a
            int ChosenCol = -1;
            do
            {
                try
                {
                    ChosenCol = int.Parse(Console.ReadLine());
                    if (ChosenCol < 0 || ChosenCol >= Cols)
                    {
                        Console.Write("Valid columns are 0-"+Cols+" only.
                    }
                }
                catch (FormatException fex)
                {
                    Console.Write("Please only enter integers.  Try again:
                }
            } while (ChosenCol < 0 || ChosenCol >= Cols);

            int[] ChosenPosition = { ChosenRow, ChosenCol };
            return ChosenPosition;
        }
```

▩ B6

8 marks available for modifying the code as shown (or equivalent code):

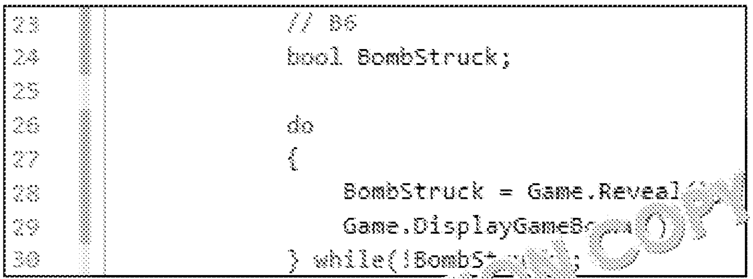
Marks could be awarded for:

- [1] calling `GetMove()`
- [1] displaying an appropriate message if the user selects a tile that has al returning *False* in this case
- [1] revealing a tile chosen by the user if and only if it has not already been re
- [1] returning *True* if a bomb tile is revealed
- [1] returning *False* if a safe tile has been revealed
- [3] modifying the main program procedure to continually display the sta to reveal tiles



```
        // B6
        public bool Reveal()
        {
            int[] Coordinates = GetMove();

            // Check if it was revealed previously
            if (Arena[Coordinates[0],Coordinates[1]].GetRevealed()
            {
                Console.WriteLine("\nERROR: That tile has already 
                return false;
            }

            // Reveal that tile
            Arena[Coordinates[0], Coordinates[1]].SetRevealed(true

            // Return whether it was a bomb
            if (Arena[Coordinates[0], Coordinates[1]].GetIsBomb())
            {
                Console.WriteLine("\nBOMB STRUCK!!! GAME OVER!!!");
                return true;
            }
            else
            {
                SafeTilesFound++; // B7
                return false;
            }
        }
```

From `Program.cs`:



```
            // B6
            bool BombStruck;

            do
            {
                BombStruck = Game.Reveal();
                Game.DisplayGameBoard();
            } while(!BombStruck);
```

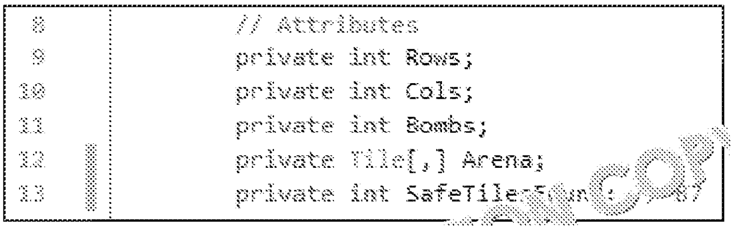
■ B7

6 marks available for modifying the code as shown (or equivalent code):

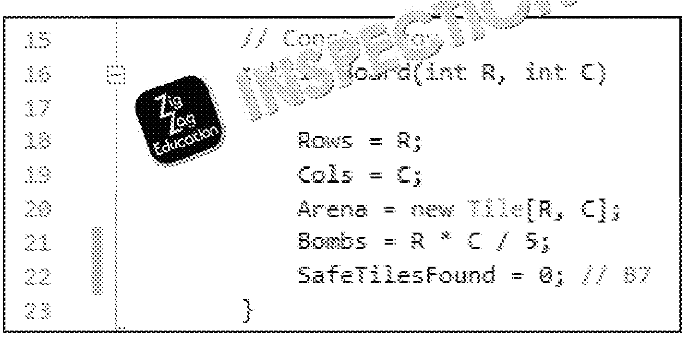
Marks could be awarded for:

- [1] adding a new counter variable as an instance attribute to `Board.cs`
- [1] adding a getter and setter method for the new attribute
- [1] adding an assignment statement to the constructor to initialise it to 0
- [1] incrementing the counter variable as part of the `Reveal()` method
- [2] amending the `Main()` method to use the new variable appropriately

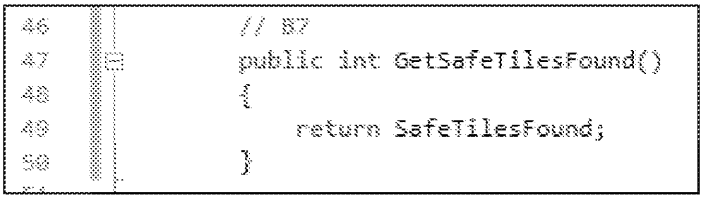
NOTE: The program must NOT output success and failure messages simultaneousl
make selections once all the safe tiles have been found.



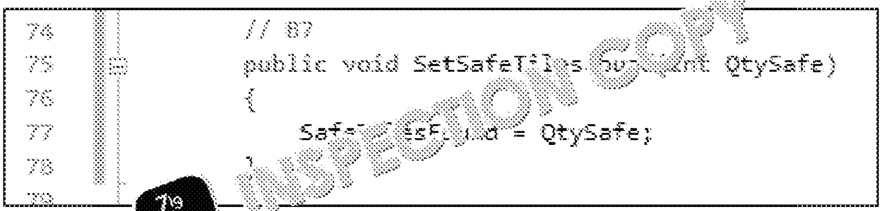
```
        // Attributes
        private int Rows;
        private int Cols;
        private int Bombs;
        private Tile[,] Arena;
        private int SafeTilesFound; // B7
```



```
        // Constructor
        public Board(int R, int C)
        {
            Rows = R;
            Cols = C;
            Arena = new Tile[R, C];
            Bombs = R * C / 5;
            SafeTilesFound = 0; // B7
        }
```

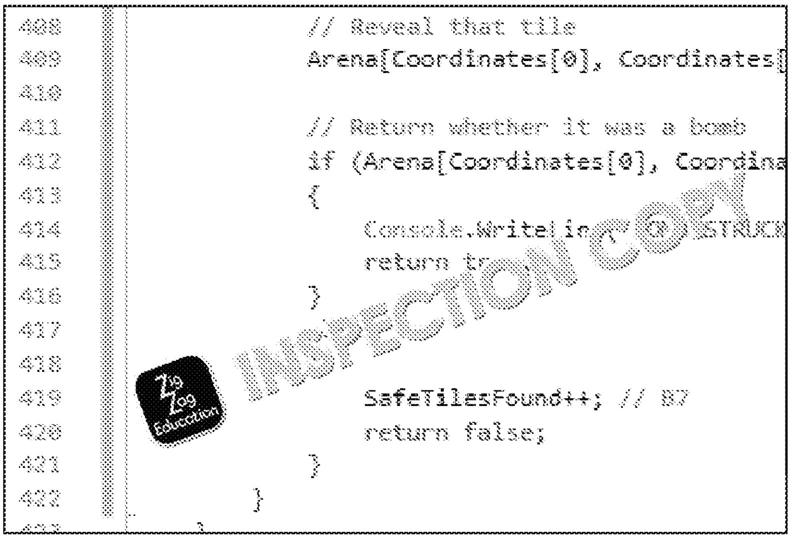


```
        // B7
        public int GetSafeTilesFound()
        {
            return SafeTilesFound;
        }
```



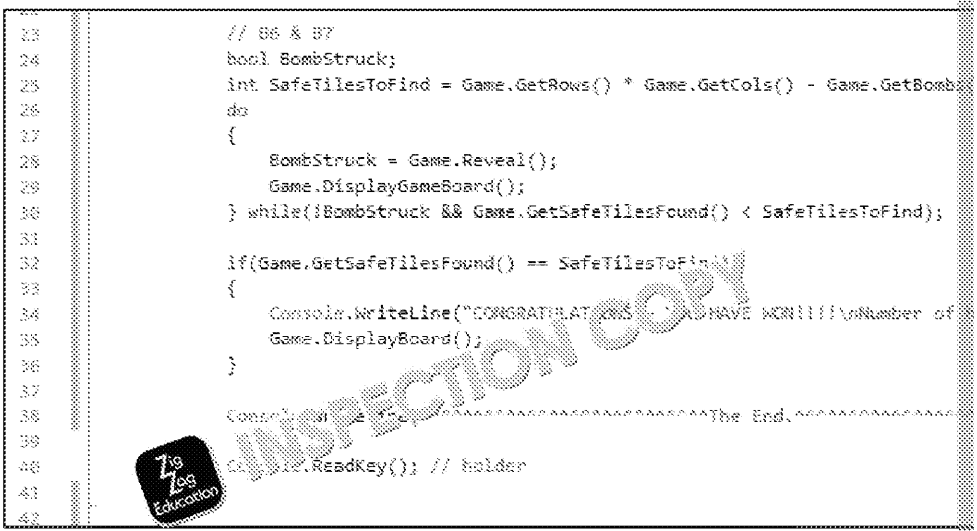
```
        // B7
        public void SetSafeTilesFound(int QtySafe)
        {
            SafeTilesFound = QtySafe;
        }
```

Within the function Board.Reveal():



```
        // Reveal that tile
        Arena[Coordinates[0], Coordinates[

        // Return whether it was a bomb
        if (Arena[Coordinates[0], Coordina
        {
            Console.WriteLine("OOPS! STRUCK
            return true;
        }


            SafeTilesFound++; // B7
            return false;
        }
    }
```

From Program.cs:



```
// B6 & B7
bool BombStruck;
int SafeTilesToFind = Game.GetRows() * Game.GetCols() - Game.GetBomb
do
{
    BombStruck = Game.Reveal();
    Game.DisplayGameBoard();
} while(!BombStruck && Game.GetSafeTilesFound() < SafeTilesToFind);

if(Game.GetSafeTilesFound() == SafeTilesToFind)
{
    Console.WriteLine("CONGRATULATIONS - YOU HAVE WON!!!!\nNumber of
    Game.DisplayBoard();
}

Console.WriteLine("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^The End.^^^^^^^^^^^^^^^

Console.ReadKey(); // holder
```

Exercise 9 – File Handling and Hash Tables

Section A

A1

1 mark for:

Line 52

A2

1 mark for giving a suitable line number:

Line 23 / Line 12

A3

2 marks (1 mark for any appropriate point made); for example:

In the `GenerateNewProductCode` method, the result of multiplying the barcod
by 10,000 using MOD, so the highest value it could be is 9,999, which means that
result [1].
The category number is a single digit, and when it is multiplied by 10,000 it will pr
in 0000 [1].
When these two numbers are added, the original category number thus becomes
digit. [1]

A4

2 marks (1 mark for each distinct relevant point):

A tuple is an immutable data structure where the values can be of various data ty
cannot be modified.

A5

1 mark for explaining why a tuple may not be used; for example:

The `Quant[illegible]` would have to change, which would mean constantly creating n
across, which is inefficient in terms of time and space.
The data types are all the same (long), so there is no need for the flexibility of oth

A6

1 mark for:

File

A7

*2 marks (1 mark for explaining that a hash function is used to place the data; 1 mar
entries is created when multiple data entries are placed in the same location):*

A hash function is used on the data to be stored to produce a number that correspo
where the data will be stored. If there is already data at that position in the table (i.e
data is appended to a list of data entries in that position in the table.

A8

a) *2 marks (1 mark for explaining that a hash function produces a value based on the giv
explaining that the function works only one way, i.e. that the output value can be c
the input cannot be calculated from the output value):*

A hash function is a function that produces a value within a certain limited ran
given. The original input cannot be calculated from the output value. The has
positions in hash files.

b) *2 marks (1 mark for explaining that a hash function generates collisions as the alg possible combinations to reduce storage space / table size; 1 mark for explaining th would be very inefficient as all of the unused locations would still need to be availa lot of memory/room):*

This phenomenon is called a collision. [1]
A hash function is a function that produces a value [illegible] depends on the key fo range of values typically similar to the number of [illegible] items that need to be s number of combinations for the [illegible] the algorithm generates too wide will be sparsely populated, which would be inefficient as that would require ad

A9

2 marks for [illegible]ing each term; 2 marks for comparative/contrasting remarks:

Serial files are appended with new data, so it is effectively organised into chronolo Sequential files have their data inserted at a position determined by the relative po some part of the data record can be compared with the other data records to dete sequence.

Serial files require less complex insertion operations but can take a long time to se N)) are not possible, but linear searches (O(N)) are.

Sequential files require more complex insert/delete operations, but they can be us performances.

A10

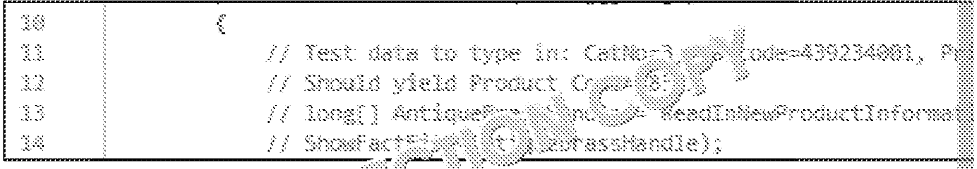
8 marks (-1 mark for each data entry not placed correctly):

Hash Table Location	First Entry	Ot
Table[0]	56551,395032849,1299,5,3	38399,43
Table[1]		
Table[2]		
Table[3]		
Table[4]	84097,373042803,2299,4,3	
Table[5]		
Table[6]		
Table[7]	58306,449598094,599,9,3	45857,94
Table[8]		
Table[9]	51565,534359435,1499,2,1	
Table[10]		
Table[11]		
Table[12]	[illegible]94[illegible]5849,399,12,7	30000,28
Table[13]		
Table[14]	77325,129819233,525,2,0	
Table[15]		
Table[16]	79250,976895865,4450,7,2	
Table[17]		
Table[18]		

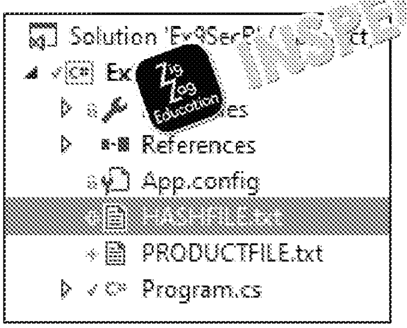
Section B

B1

1 mark available for commenting out code (using // or /   */) and 1 mark for creat precise name given:*



```
        {
            // Test data to type in: CatNo=3 [illegible]ode=439234001, P
            // Should yield Product C[illegible]8:
            // long[] AntiquePr[illegible] = ReadInNewProductInformat
            // ShowFact[illegible]BrassHandle);
```



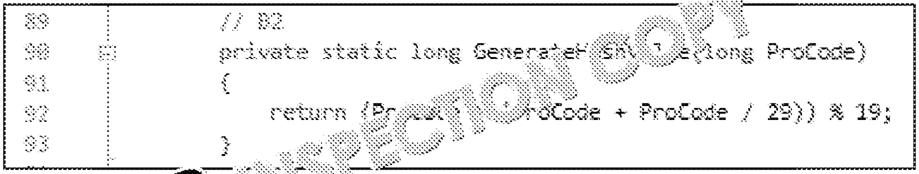
Solution 'Ex9SerP' ([illegible] ct
Ex
[illegible]es
References
App.config
[illegible]
PRODUCTFILE.txt
Program.cs

B2

2 marks available for modifying the code as shown (or equivalent code):

Marks awarded for:

- correct method signature
- correct implementation of the hashing algorithm



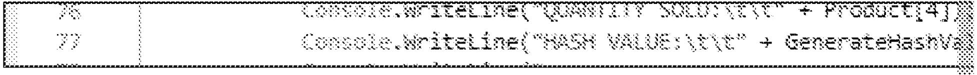
```
        // B2
        private static long GenerateH[illegible]e(long ProCode)
        {
            return (Pr[illegible]roCode + ProCode / 29)) % 19;
        }
```

B3

2 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- correctly inserting it in the same style as other lines
- calling the existing `GenerateHashValue` function



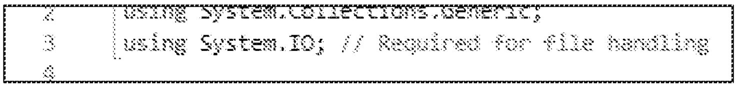
```
            Console.WriteLine("QUANTITY SOLD:\t\t" + Product[4]
            Console.WriteLine("HASH VALUE:\t\t" + GenerateHashVa
```

B4

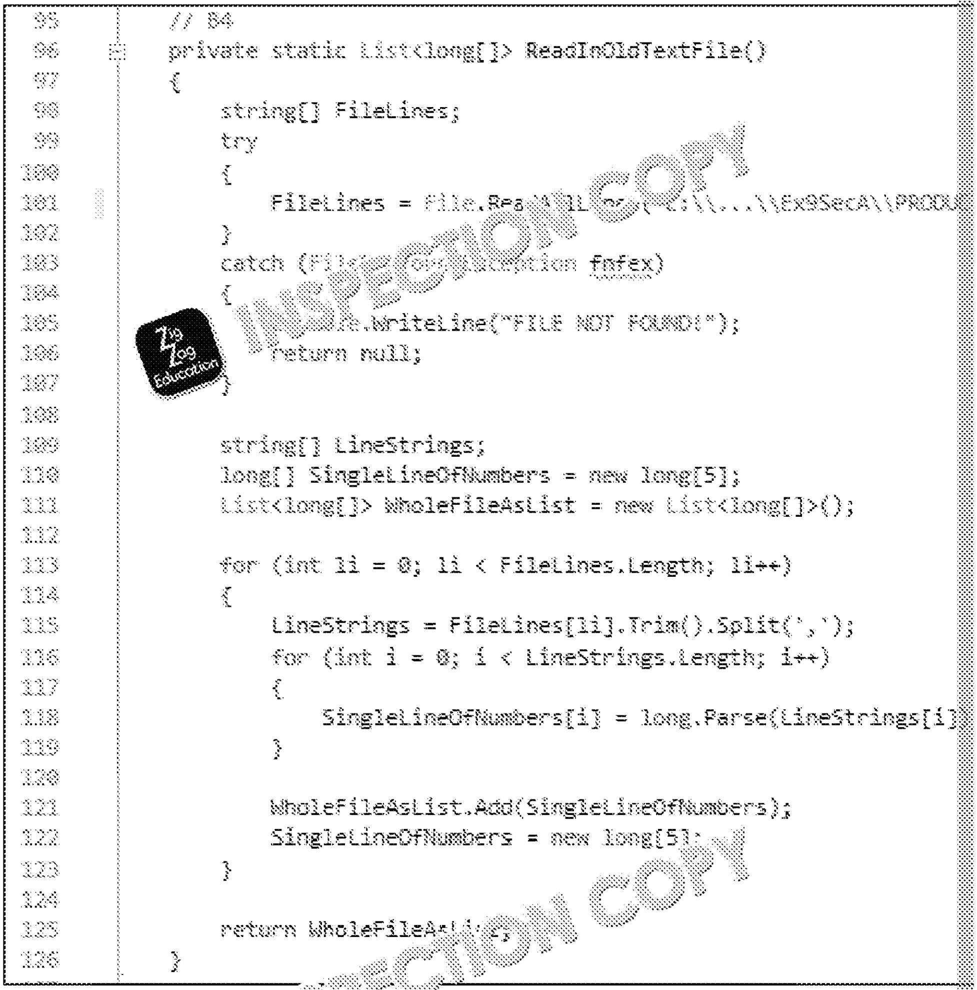
8 marks available for modifying the code as shown (or equivalent code):

Award mark for:

- using TRY-CATCH to intercept exceptions
- using a suitable read technique successfully
- handling the file as an array of strings, one per line
- trimming off the new line characters
- splitting where there are commas
- parsing the individual number strings to turn them into longs
- adding the newly read array to a list of arrays
- returning a list of arrays of long integers



```
using System.Collections.Generic;
using System.IO; // Required for file handling
```



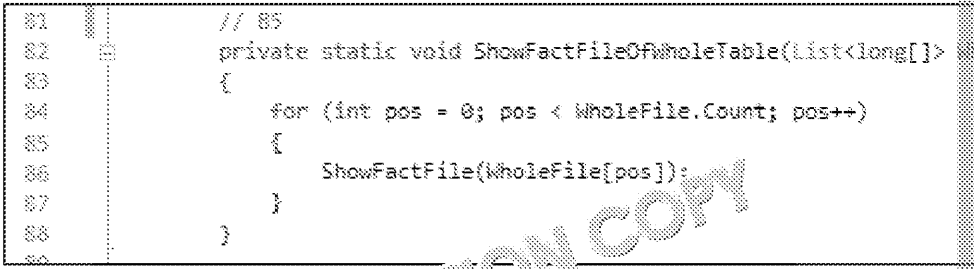
```
        // B4
        private static List<long[]> ReadInOldTextFile()
        {
            string[] FileLines;
            try
            {
                FileLines = File.ReadAllLines("C:\\...\\Ex9SecA\\PRODU
            }
            catch (FileNotFoundException fnfex)
            {
                Console.WriteLine("FILE NOT FOUND!");
                return null;
            }

            string[] LineStrings;
            long[] SingleLineOfNumbers = new long[5];
            List<long[]> WholeFileAsList = new List<long[]>();

            for (int li = 0; li < FileLines.Length; li++)
            {
                LineStrings = FileLines[li].Trim().Split(',');
                for (int i = 0; i < LineStrings.Length; i++)
                {
                    SingleLineOfNumbers[i] = long.Parse(LineStrings[i])
                }

                WholeFileAsList.Add(SingleLineOfNumbers);
                SingleLineOfNumbers = new long[5];
            }

            return WholeFileAsList;
        }
```

■ B5

3 marks available for modifying the code as shown (or equivalent code):

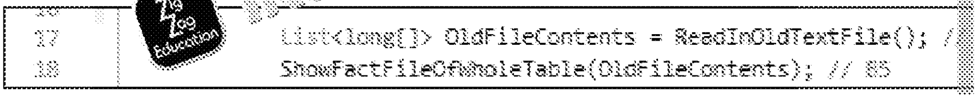
Marks could be awarded for:

- correct method signature
- iterating through the whole list
- calling the `ShowFactFile` method, passing each array in the list one at



```
        // B5
        private static void ShowFactFileOfWholeTable(List<long[]>
        {
            for (int pos = 0; pos < WholeFile.Count; pos++)
            {
                ShowFactFile(WholeFile[pos]);
            }
        }
```

In `Main()`:



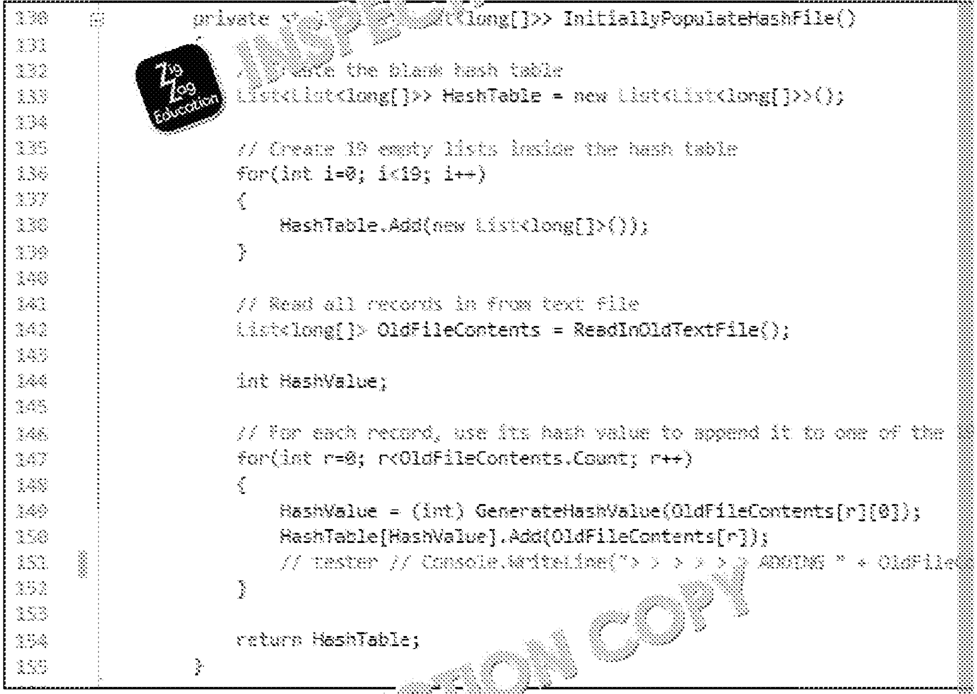
```
            List<long[]> OldFileContents = ReadInOldTextFile(); /
            ShowFactFileOfWholeTable(OldFileContents); // B5
```

▩ B6

6 marks available for modifying the code as shown (or equivalent code):

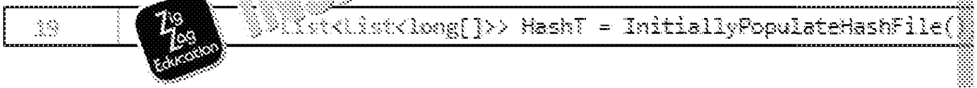
Marks could be awarded for:

- declaring and creating a HashTable variable with the correct data type
- pre-populating the HashTable with 19 empty lists
- calling the existing `ReadInOldTextFile()` method to gather data from
- finding the hash value for that row
- inserting the row into the correct list in the HashTable
- returning the whole HashTable



```
        private static List<List<long[]>> InitiallyPopulateHashFile()
        {
            // Create the blank hash table
            List<List<long[]>> HashTable = new List<List<long[]>>();

            // Create 19 empty lists inside the hash table
            for(int i=0; i<19; i++)
            {
                HashTable.Add(new List<long[]>());
            }

            // Read all records in from text file
            List<long[]> OldFileContents = ReadInOldTextFile();

            int HashValue;

            // For each record, use its hash value to append it to one of the
            for(int r=0; r<OldFileContents.Count; r++)
            {
                HashValue = (int) GenerateHashValue(OldFileContents[r][0]);
                HashTable[HashValue].Add(OldFileContents[r]);
                // tester // Console.WriteLine("> > > > > > ADDING " + OldFile
            }

            return HashTable;
        }
```

In `Main()`:



```
            List<List<long[]>> HashT = InitiallyPopulateHashFile(
```

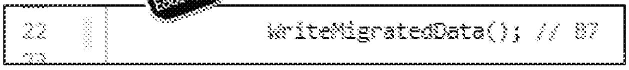
▩ B7

14 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded:

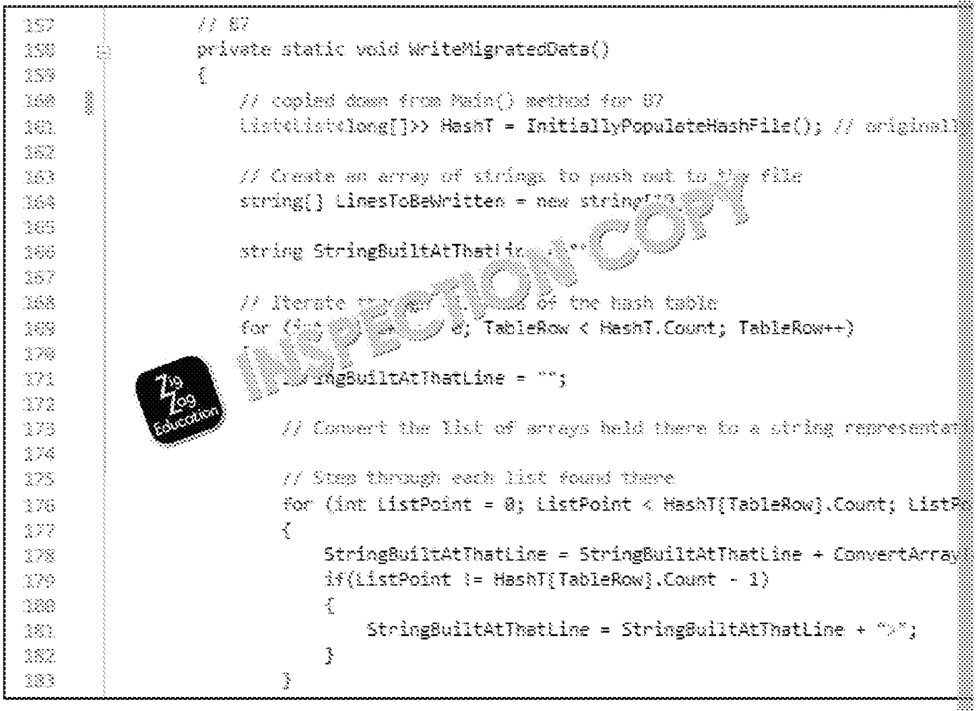
- for calling `InitiallyPopulateHashFile` to build a `HashTable`
- for creating an empty array of 19 strings to push out to the file
- for iterating through all rows of the hash table
- for iterating through all elements in the list on each row
- [4] for successfully converting arrays to strings
- for delimiting the different products using the '>' character...
- for ... except at the end of a row
- for adding the line to be written to the array of strings (that will be pushe
- for opening the file to write all the data
- for using TRY-CATCH to intercept and handling exceptions
- for a method call from `Main()`

In `Main()`:

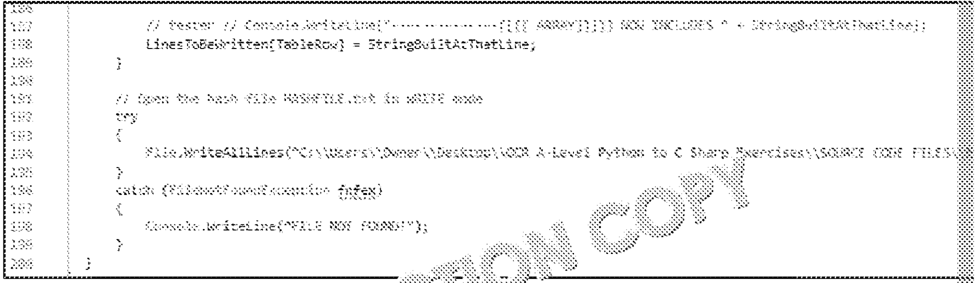


```
            WriteMigratedData(); // B7
```

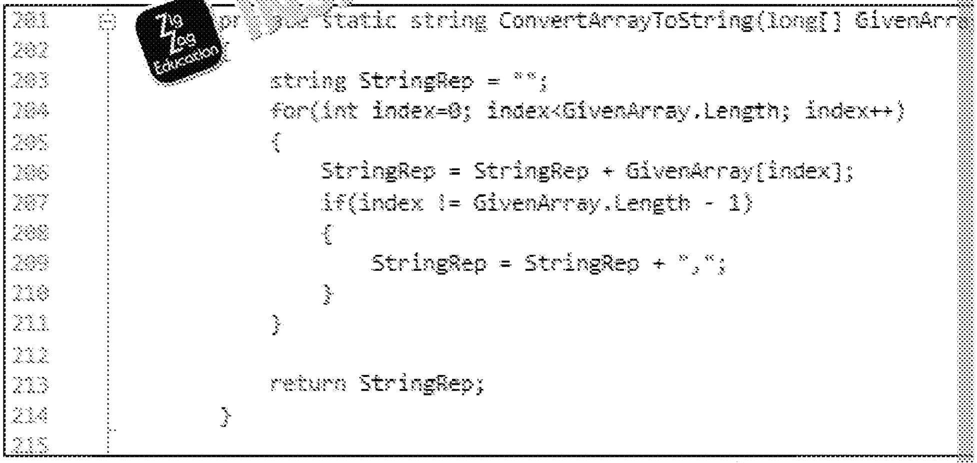
Outside `Main()`:



```
        // 87
        private static void WriteMigratedData()
        {
            // copied down from Main() method for 87
            List<List<long[]>> HashT = InitiallyPopulateHashFile(); // originall
            
            // Create an array of strings to push out to the file
            string[] LinesToBeWritten = new string[[illegible]
            
            string StringBuiltAtThatLine = "";
            
            // Iterate through each row of the hash table
            for (int TableRow = 0; TableRow < HashT.Count; TableRow++)
            {
                StringBuiltAtThatLine = "";
                
                // Convert the list of arrays held there to a string representat
                
                // Step through each list found there
                for (int ListPoint = 0; ListPoint < HashT[TableRow].Count; ListP
                {
                    StringBuiltAtThatLine = StringBuiltAtThatLine + ConvertArray
                    if(ListPoint != HashT[TableRow].Count - 1)
                    {
                        StringBuiltAtThatLine = StringBuiltAtThatLine + ">";
                    }
                }
```



```
                // tester // Console.WriteLine("--------------[[[[ ARRAY]]]] NOW INCLUDES " + StringBuiltAtThatLine);
                LinesToBeWritten[TableRow] = StringBuiltAtThatLine;
            }
            
            // Open the hash file HASHFILE.txt in WRITE mode
            try
            {
                File.WriteAllLines("C:\\Users\\Owner\\Desktop\\OCR A-Level Python to C Sharp Exercises\\SOURCE CODE FILES\\
            }
            catch (FileNotFoundException fnfex)
            {
                Console.WriteLine("FILE NOT FOUND!");
            }
        }
```



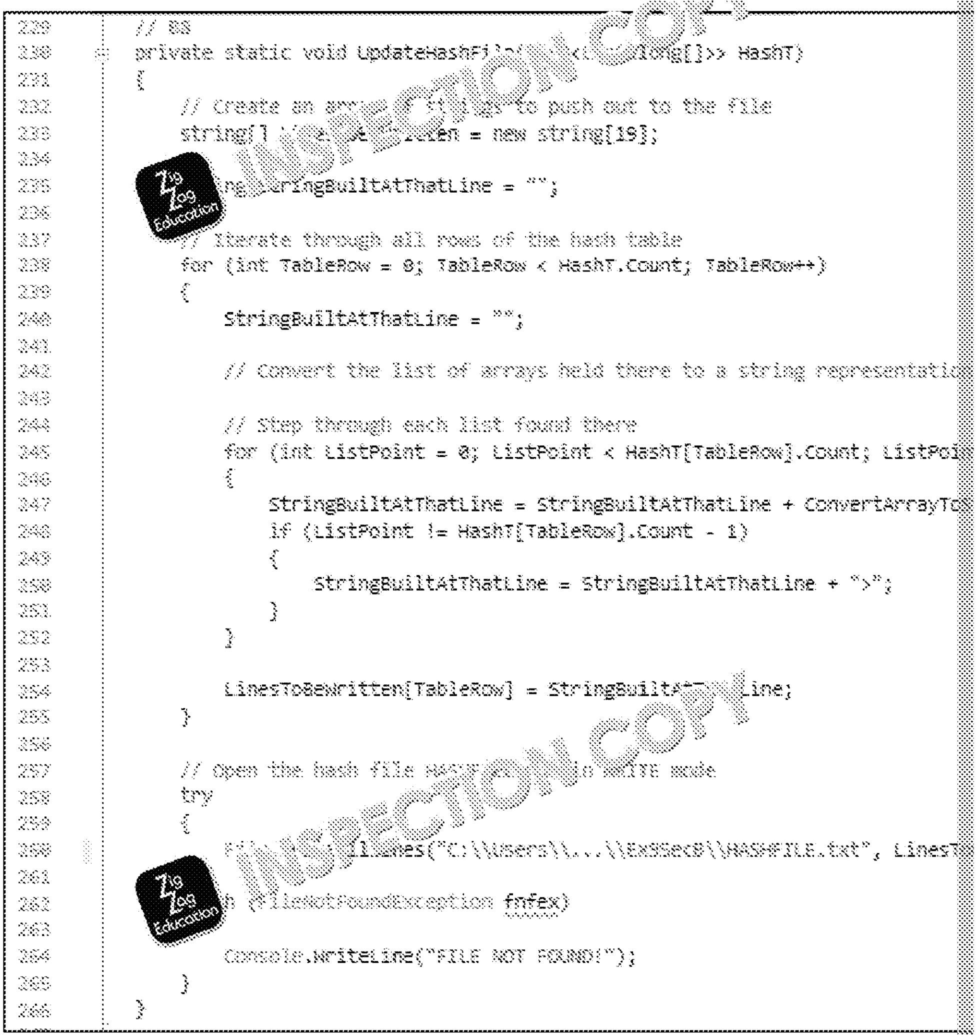
```
        private static string ConvertArrayToString(long[] GivenArra
        {
            string StringRep = "";
            for(int index=0; index<GivenArray.Length; index++)
            {
                StringRep = StringRep + GivenArray[index];
                if(index != GivenArray.Length - 1)
                {
                    StringRep = StringRep + ",";
                }
            }
            
            return StringRep;
        }
```

▩ B8

3 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- [1] passing HashT in as a parameter
- [1] no longer declaring HashT inside the method
- [1] removing the call to other method (`InitiallyPopulateHashFile`



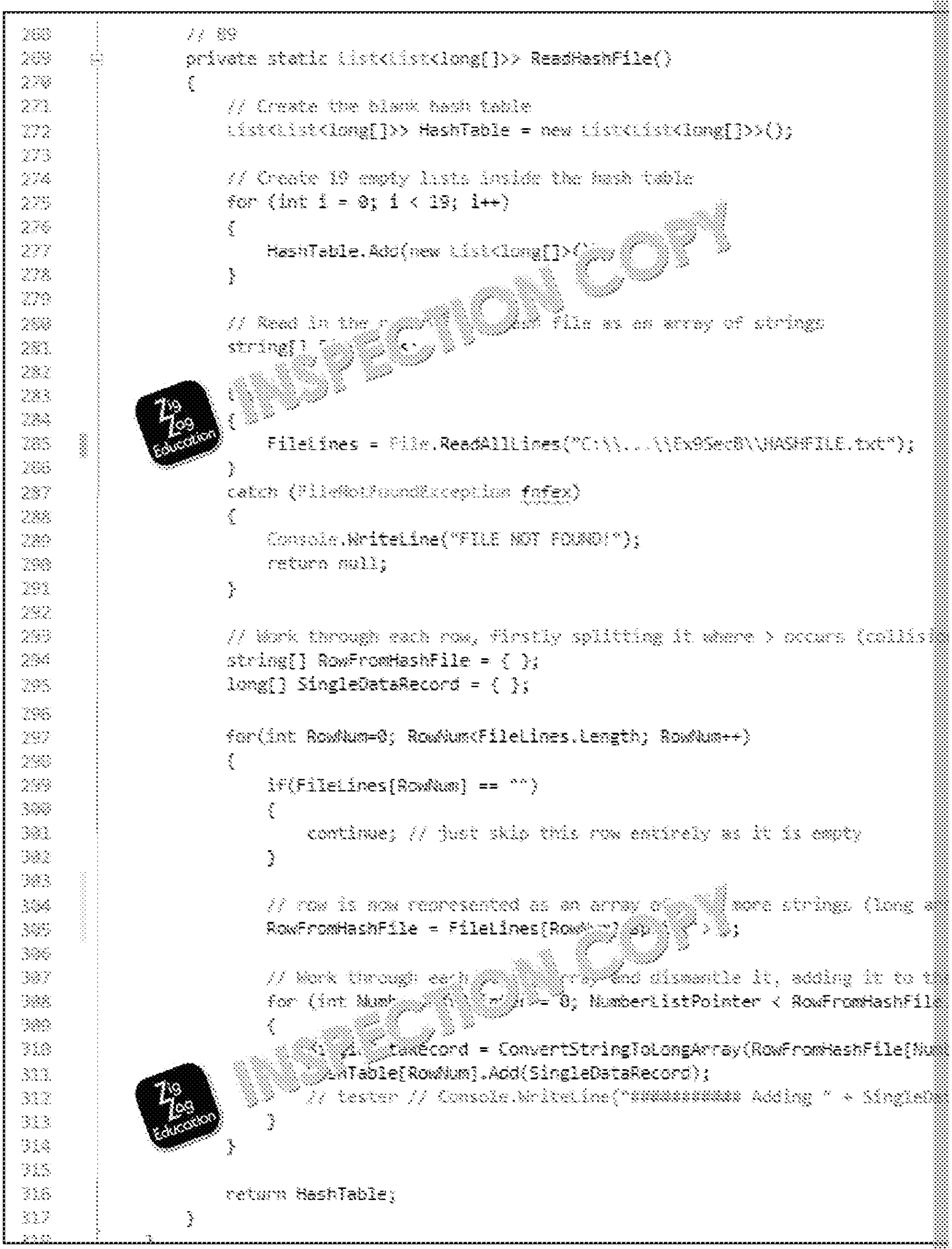
```
// B8
private static void UpdateHashFile(List<List<long[]>> HashT)
{
    // Create an array of strings to push out to the file
    string[] LinesToBeWritten = new string[19];

    string StringBuiltAtThatLine = "";

    // Iterate through all rows of the hash table
    for (int TableRow = 0; TableRow < HashT.Count; TableRow++)
    {
        StringBuiltAtThatLine = "";

        // Convert the list of arrays held there to a string representatio

        // Step through each list found there
        for (int ListPoint = 0; ListPoint < HashT[TableRow].Count; ListPoi
        {
            StringBuiltAtThatLine = StringBuiltAtThatLine + ConvertArrayTo
            if (ListPoint != HashT[TableRow].Count - 1)
            {
                StringBuiltAtThatLine = StringBuiltAtThatLine + ">";
            }
        }

        LinesToBeWritten[TableRow] = StringBuiltAtThatLine;
    }

    // Open the hash file HASHFILE.txt in WRITE mode
    try
    {
        File.WriteAllLines("C:\\Users\\...\\ExSSecB\\HASHFILE.txt", LinesT
    }
    catch (FileNotFoundException fnfex)
    {
        Console.WriteLine("FILE NOT FOUND!");
    }
}
```

▩ B9

8 marks available for modifying the code as shown (or equivalent code):

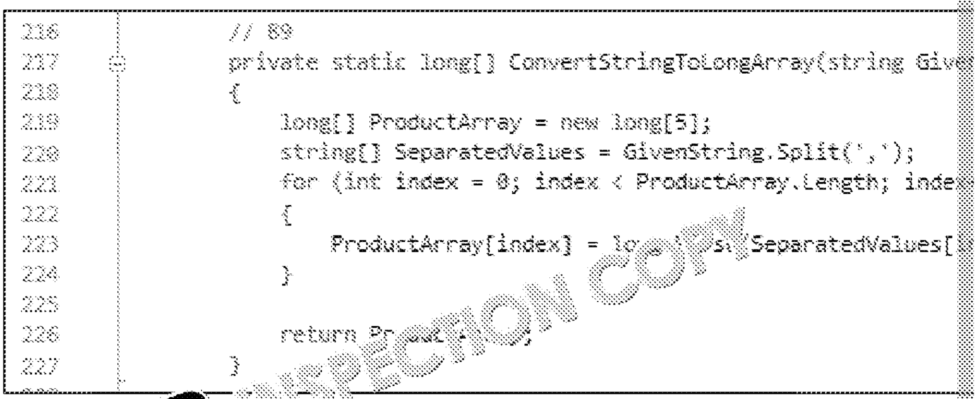
Marks could be awarded for:

- [1] for initialising a blank hash table structure as a list of lists of long integ
- [1] for accessing the file and pulling in all data (e.g. using `ReadAllLine`
- [1] for applying TRY-CATCH to the file read operation
- [1] for iterating through all file rows and skipping blanks
- [1] for splitting at the '>' character
- [1] for managing the addition of an entire row of arrays to the hash table
- [2] for converting a string to an array of longs (within or as a separate me



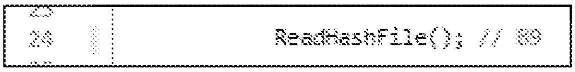
```
        // 89
        private static List<List<long[]>> ReadHashFile()
        {
            // Create the blank hash table
            List<List<long[]>> HashTable = new List<List<long[]>>();

            // Create 19 empty lists inside the hash table
            for (int i = 0; i < 19; i++)
            {
                HashTable.Add(new List<long[]>[illegible]
            }

            // Read in the [illegible] file as an array of strings
            string[] [illegible]

            [illegible]
            {
                FileLines = File.ReadAllLines("C:\\...\\Ex9SecB\\HASHFILE.txt");
            }
            catch (FileNotFoundException fnfex)
            {
                Console.WriteLine("FILE NOT FOUND!");
                return null;
            }

            // Work through each row, firstly splitting it where > occurs (collisi
            string[] RowFromHashFile = { };
            long[] SingleDataRecord = { };

            for(int RowNum=0; RowNum<FileLines.Length; RowNum++)
            {
                if(FileLines[RowNum] == "")
                {
                    continue; // just skip this row entirely as it is empty
                }

                // row is now represented as an array o[illegible] more strings (long a
                RowFromHashFile = FileLines[Row[illegible]] [illegible]'>'[illegible];

                // Work through each [illegible]rray and dismantle it, adding it to t
                for (int Num[illegible] = 0; NumberListPointer < RowFromHashFil
                {
                    [illegible]taRecord = ConvertStringToLongArray(RowFromHashFile[Nu
                    [illegible]hTable[RowNum].Add(SingleDataRecord);
                    // tester // Console.WriteLine("############ Adding " + SingleD
                }
            }

            return HashTable;
        }
```

Optional supporting method:



```
        // 89
        private static long[] ConvertStringToLongArray(string Give
        {
            long[] ProductArray = new long[5];
            string[] SeparatedValues = GivenString.Split(',');
            for (int index = 0; index < ProductArray.Length; inde
            {
                ProductArray[index] = lo[illegible]s[illegible]SeparatedValues[
            }

            return Pr[illegible];
        }
```

In `Main()`:



```
            ReadHashFile(); // 89
```

Exercise 10 – Reverse Polish

Section A

▩ A1

1 mark for:

It performs EXCLUSIVE OR on the bits of both operands.

▩ A2

1 mark for:

Line 9

▩ A3

2 marks (1 mark for explaining that if the given value can be converted into an integ mark for explaining that the function returns False if it fails to convert the value):

The `IsInt` function tries to convert the value into an integer. If the value can be c integer and the function returns *True*. Otherwise, if the value cannot be converted, CATCH block executes and the function returns *False*.

▩ A4

2 marks for giving the correct expression (1 mark if the given expression is only part

3 2 + 4 1 - * 4 /

▩ A5

2 marks for giving the correct expression (1 mark if the given expression is only part ignore redundant brackets:

(4 + 5) * (3 - 2 / 1)

▩ A6

1 mark for each point up to a maximum of 4

[1] It is a FILO (First-In, Last-Out) or LIFO (Last-In, First-Out) data structure.
[1] Items are pushed, i.e. added, on to the top of a non-full stack.
[1] Items are popped, i.e. removed, from the top of a non-empty stack.
[1] Other items in the stack are inaccessible until they are at the top of the stack.
[1] A stack pointer is a separate integer variable that notes where the top of the stac

▩ A7

2 marks for giving the correct expression (1 mark if the given expression is only part

6 3 - 2 *

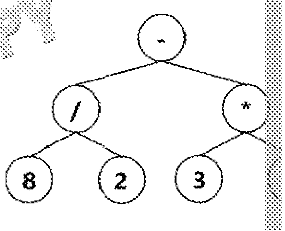
▩ A8

2 marks for giving the correct expression (1 mark if the given expression is only part ignore redundant brackets:

(6 - 3) * 2

▩ A9

6 marks (1 mark for each operator that has been given the correct child nodes; 1 mark for each operator that has its child nodes in the correct order, i.e. the left child node being the first operand given and the right child node being the second operand given)



▩ A10

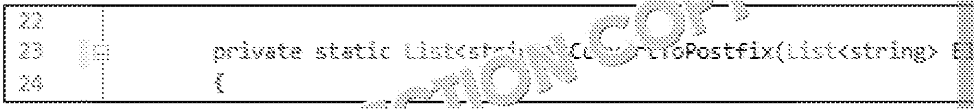
6 marks (1 mark for each operator that has been given the correct child nodes; 1 ma for each operator that has its child nodes in the correct order, i.e. the left child node being the first operand given and the right child node being the second operand giv

Section B

B1

2 marks available for modifying the code as shown (or closely equivalent code):

- [1] for return type
- [1] for parameter part

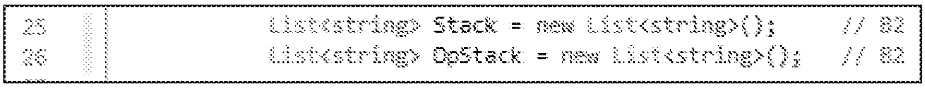


```
        private static List<string> ConvertToPostfix(List<string> E
        {
```

B2

2 marks available for modifying the code (as shown below or closely equivalent code

[1] per correctly written line

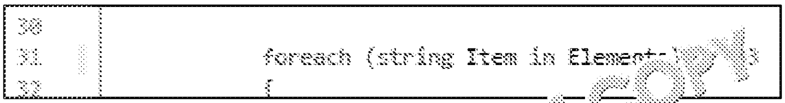


```
            List<string> Stack = new List<string>();      // B2
            List<string> OpStack = new List<string>();    // B2
```

B3

2 marks available for modifying the code (as shown below):

- [1] for foreach stepping through Elements
- [1] for calling the variable Item

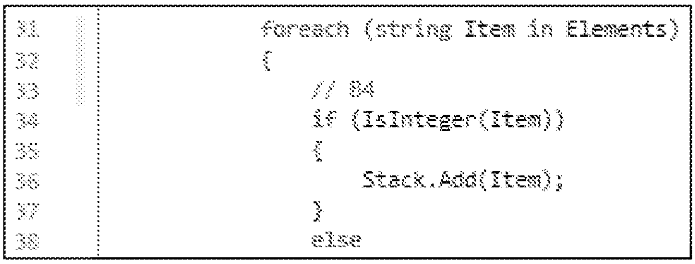


```
            foreach (string Item in Elements)    // B3
            {
```

B4

2 marks available for modifying the code as shown (or closely equivalent code):

- [1] for the if part calling the `IsInteger` method
- [1] for using the `Add` method of the `List` class

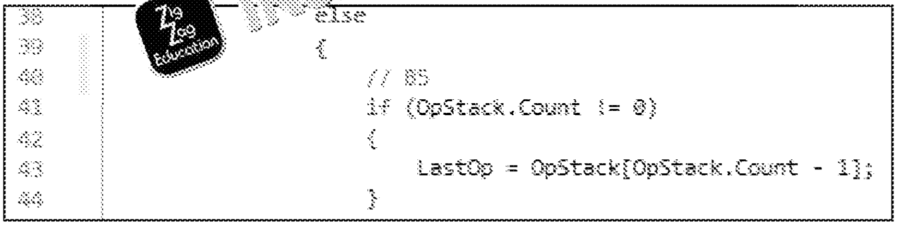


```
            foreach (string Item in Elements)
            {
                // B4
                if (IsInteger(Item))
                {
                    Stack.Add(Item);
                }
                else
```

B5

2 marks available for modifying the code as shown (or equivalent code):

- [1] for correctly checking the length of the list
- [1] for reading without removing the value at the top of OpStack



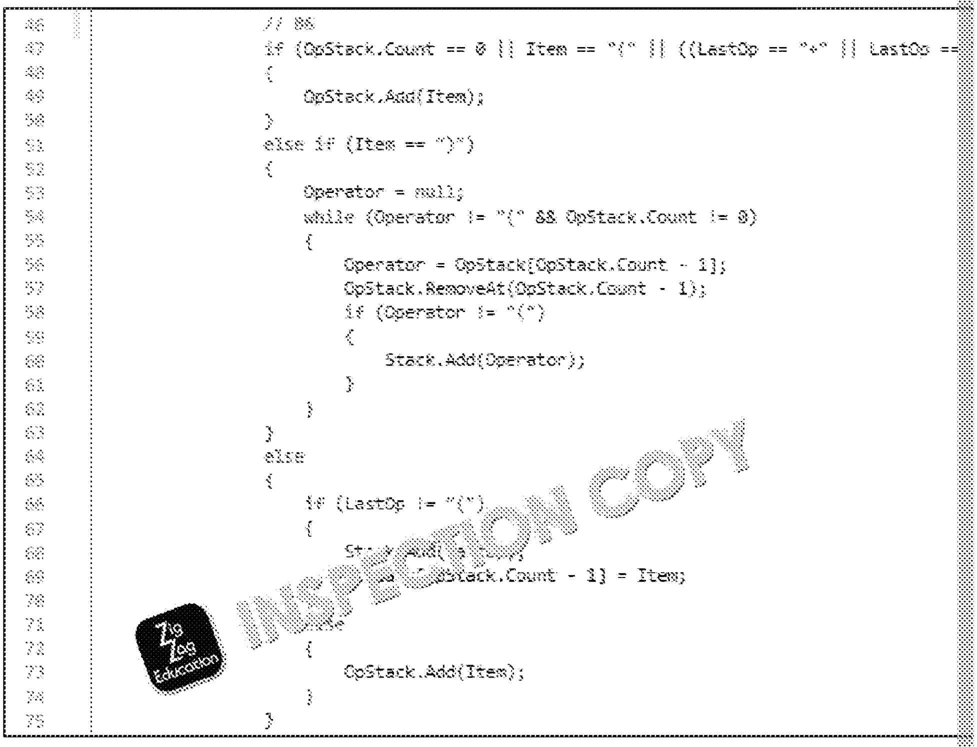
```
                else
                {
                    // B5
                    if (OpStack.Count != 0)
                    {
                        LastOp = OpStack[OpStack.Count - 1];
                    }
```

▩ B6

11 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- [2] for the correctly implemented IF statement, [1] of which is for the corr
- [1] for using the Add method to push to the OpStack stack/list
- [1] for correctly phrased ELSE IF and its condition
- [1] for setting up the WHILE loop correctly, with Operator pre-set to null
- [1] for implementing a pop operation correctly in two lines of code
- [1] for IF statement dealing with Operator != "(" by pushing Operator to S
- [1] for ELSE part containing a nested IF-ELSE
- [1] for IF block correctly adding LastOp to Stack
- [1] for IF block replacing Item at the END of OpStack, not popping it but ov
- [1] for ELSE block pushing Item to OpStack



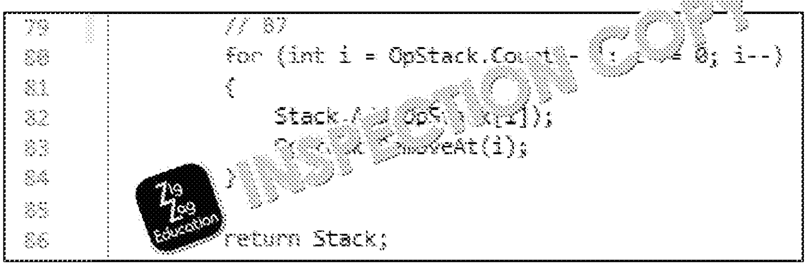
```
// B6
if (OpStack.Count == 0 || Item == "(" || ((LastOp == "+" || LastOp ==
{
    OpStack.Add(Item);
}
else if (Item == ")")
{
    Operator = null;
    while (Operator != "(" && OpStack.Count != 0)
    {
        Operator = OpStack[OpStack.Count - 1];
        OpStack.RemoveAt(OpStack.Count - 1);
        if (Operator != "(")
        {
            Stack.Add(Operator);
        }
    }
}
else
{
    if (LastOp != "(")
    {
        Stack.Add(LastOp);
        OpStack[OpStack.Count - 1] = Item;
    }
    else
    {
        OpStack.Add(Item);
    }
}
```

▩ B7

4 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- [1] descending counter implemented in FOR loop
- [1] starting at final element of OpStack
- [1] for pushing to Stack
- [1] for popping from OpStack



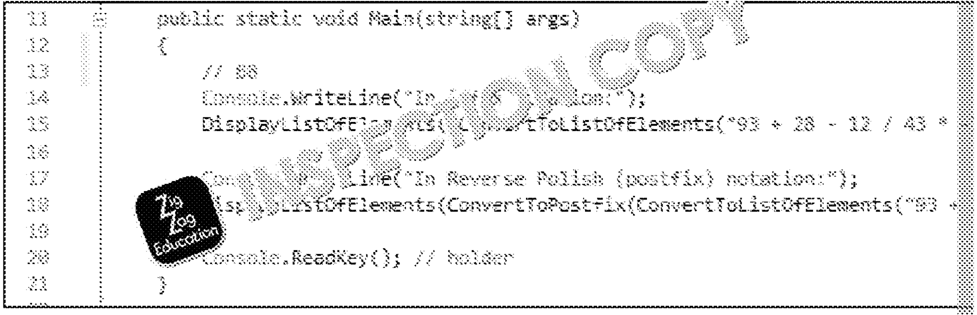
```
// B7
for (int i = OpStack.Count - 1; i >= 0; i--)
{
    Stack.Add(OpStack[i]);
    OpStack.RemoveAt(i);
}

return Stack;
```

■ B8

2 marks available for modifying the code as shown (or equivalent code):

Marks could be awarded for:

- [1] for calling `DisplayListOfElements` to show results
- [1] for calling `ConvertToPostifx` to show results



```
public static void Main(string[] args)
{
    // B8
    Console.WriteLine("In infix notation:");
    DisplayListOfElements(ConvertToListOfElements("93 + 28 - 12 / 43 *

    Console.WriteLine("In Reverse Polish (postfix) notation:");
    DisplayListOfElements(ConvertToPostfix(ConvertToListOfElements("93 +

    Console.ReadKey(); // holder
}
```