Question	Part	Marking guidance	Total marks
01	1	**Mark is for AO1 (understanding)** **A** (Line number 2) only; **If more than one lozenge shaded then mark is not awarded**	1
01	2	**Mark is for AO1 (understanding)** **C** (Line number 11) only; **If more than one lozenge shaded then mark is not awarded**	1
01	3	**Mark is for AO2 (apply)** **A** (1 subroutine call) only; **If more than one lozenge shaded then mark is not awarded**	1
01	4	**Mark is for AO2 (apply)** **B** (String) only; **If more than one lozenge shaded then mark is not awarded**;	1

Question	Part	Marking guidance	Total marks
01	5	**Mark is for AO2 (apply)** 2//twice//two; **I.** Minor spelling errors	1
01	6	**Mark is for AO2 (apply)** 2//two; **A.** true and false (or other possible indicators for true and false) **R.** Boolean	1
01	7	**Mark is for AO2 (apply)** 7; **A.** All of 3, 5 **and** 11 **A.** If instruction written out (a ← 2)	1
01	8	**Mark is for AO3 (program)** `q ← 2;` **A**. `a ← 1, a ← 4` **and** `FOR n ← 1 TO a` (only if all given)	1

Question	Part	Marking guidance	Total marks
02	1	**3 marks for AO2 (apply)** 1 mark for C written **once** and in column 1; 1 mark for A and B written **once** and both in column 2; 1 mark for A and B written **once** and in correct positions in column 2; Column 0 Column 1 Column 2 C A B	3
02	2	**3 marks for AO2 (apply)** 1 mark for A written **once** and in correct column (0); 1 mark for B written **once** and in correct column (2); 1 mark for C written **once** and in correct column (1); Column 0 Column 1 Column 2 A C B	3
02	3	**3 marks for AO2 (apply)** If any value is written more than once no marks for that value. 3 marks if A, B and C are all written **once**, in correct columns and in correct position (see diagram below). If not fully correct then a maximum of 2 from: 1 mark for A column 1 (even if not only value present); 2 marks for column 2 correct; 2 marks if B is above C in column 2 with A in column 2 as well in any position (assuming A, B and C are only written once); 1 mark if either one or both of B or C are present in column 2 (possibly with A as well and assuming B and C are only written once); 1 mark if A is in an incorrect column and B and C are in another incorrect column but are in the correct order and all are only written once; Column 0 Column 1 Column 2 A B C	3

Question	Part	Marking guidance	Total marks
02	4	**5 marks for AO3 (program)** **Note for mark C – DPT for same logical error in the Boolean condition** Maximum of 5 marks; Mark A for using a `WHILE` loop or similar to move from column 0 to column 2; Mark B for a Boolean condition that detects when the column 0 is empty; Mark C for using a second WHILE loop or similar to move the result from A and B into column 1 (both the loop and the associated Boolean condition need to be correct to gain this mark); **or** Mark A for using a `FOR` loop or similar to move from column 0 to column 2; Mark B for ascertaining the terminating value for the `FOR` loop; Mark C for using a second `FOR` loop or similar to move the result from A and B into column 1 (both the loop and the associated terminating value need to be correct to gain this mark); **and** Mark D for using the subroutines correctly throughout, i.e. called with appropriate parameters and return values handled correctly; Mark E if algorithm is completely correct; **A.** Minor spelling errors such as `HIEGHT` for `HEIGHT` **Example 1** `WHILE HEIGHT(0) > 0` (Part of A, B) `  MOVE(0, 2)` (Part of A) `ENDWHILE` `WHILE HEIGHT(2) > 0` (Part of C) `  MOVE(2, 1)` (Part of C) `ENDWHILE` (`MOVE` and `HEIGHT` are used correctly throughout so D and completely correct so also E.)	5

Example 2

```
DO                              (Part of A)
   MOVE(0, 2)                   (Part of A)
WHILE HEIGHT(0) > 0             (Part of A, B)
DO                              (Part of C)
   MOVE(2, 1)                   (Part of C)
WHILE HEIGHT(2) > 0             (Part of C)
```

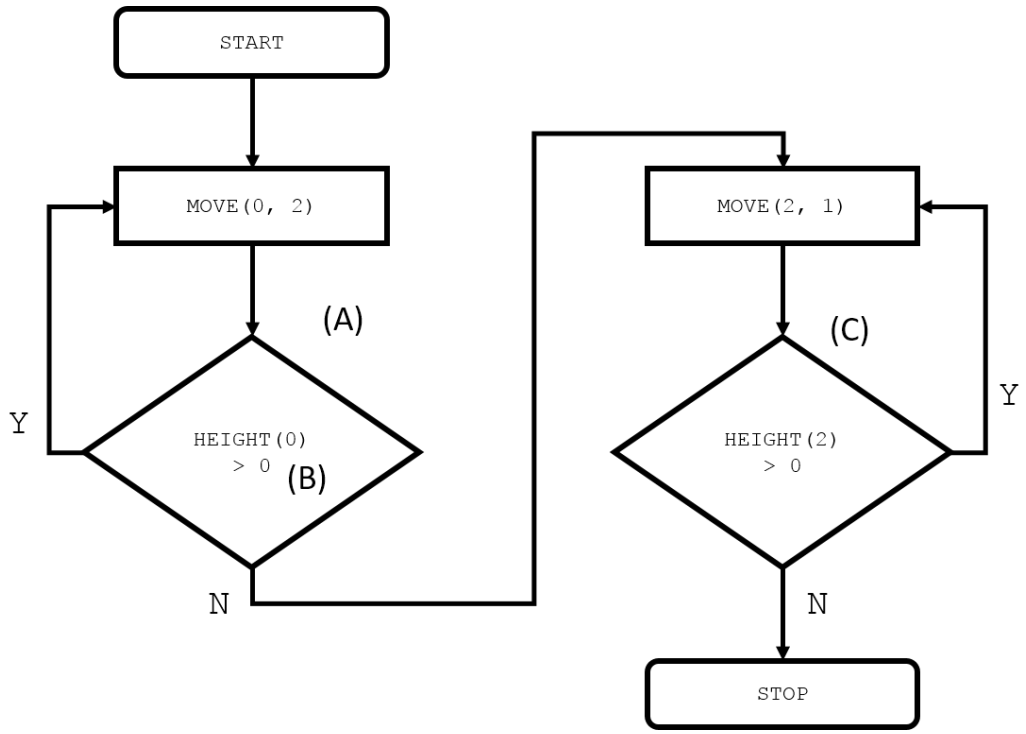
(`MOVE` and `HEIGHT` are used correctly throughout so D and completely correct so also E.)

Example 3

```
REPEAT                          (Part of A)
   MOVE(0, 2)                   (Part of A)
UNTIL HEIGHT(0) = 0             (Part of A, B)
REPEAT                          (Part of C)
   MOVE(2, 1)                   (Part of C)
WHILE HEIGHT(2) = 0             (Part of C)
```

(`MOVE` and `HEIGHT` are used correctly throughout so D and completely correct so also E.)

Example 4



(`MOVE` and `HEIGHT` are used correctly throughout so D and completely correct so also E.)

Example 5

```
number_of_blocks ← HEIGHT(0)                (Part of B)
FOR x ← 0 TO number_of_blocks               (Part of A, Part of B)
   MOVE(0, 2)                               (Part of A)
ENDFOR
FOR x ← 0 TO number_of_blocks               (Part of C)
   MOVE(2, 1)                               (Part of C)
ENDFOR                                      (Part of C)
```

(`MOVE` and `HEIGHT` are used correctly throughout so D and completely correct so also E.)

03	1	**3 marks for AO2 (apply)** 1 mark if column `z` increments by 1 and starts at 0; 1 mark if column `z` has the final value 3; 1 mark if `correct` column is correct;	3

z	correct
0	false
1	true
2	
3	

03	2	**Mark is for AO2 (apply)** false; **I.** Case	1

03	3	**Mark is for AO2 (apply)** Second row only;	1

New Line	Tick one box
`IF user = us[z] OR pass = ps[z] THEN`	
`IF user = us[z] AND pass = ps[z] THEN`	Tick
`IF NOT (user = us[z] AND pass = ps[z]) THEN`	

Question	Part	Marking guidance	Total marks
03	4	**Mark is for AO2 (apply)** Maximum 2 marks from: The program will return true as soon as a match (between username and password) is found; So there is no need to (always) iterate over the complete array(s)/list of usernames; (If a match is found and is not last in the list) the algorithm will complete in fewer steps/less time; **A**. the programmer has used fewer variables	2

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>04</td><td>1</td><td>2 marks for AO1 (understanding)

Correct table is:

<table>
<tr><th>Values</th><th>Data type</th></tr>
<tr><td><code>true</code>, <code>false</code></td><td>Boolean;</td></tr>
<tr><td><code>0</code>, <code>1</code>, <code>2</code></td><td>Integer;</td></tr>
</table>

A. Bool/bool/boolean instead of Boolean
A. Int/int instead of integer</td><td>2</td></tr>
<tr><td>04</td><td>2</td><td>Mark is for AO1 (recall)

Decomposition;

A. Top-down design;</td><td>1</td></tr>
<tr><td>04</td><td>3</td><td>2 marks for AO2 (apply)

1 mark for giving a new identifier that describes this purpose, eg <code>notHit</code> (alternatively award this mark if the explanation is incorrect but the identifier describes the purpose stated in the answer);
1 mark for explaining the purpose of the subroutine is to see if a hit has been made at the specified location;</td><td>2</td></tr>
<tr><td>04</td><td>4</td><td>2 marks for AO2

(A local variable) is only accessible//declarable//within scope (in the subroutine);
(A local variable) only exists while the subroutine/program block is executing;</td><td>2</td></tr>
</table>

Question	Part	Marking guidance	Total marks

04	5	**11 marks for AO3 (program)**
		
		Any fully correct answer should get 11 marks even if it does not map exactly to the following mark points.
		
		Max 10 marks if the answer includes any errors.
		
		Mark A: for creating a subroutine with an identifier that defines its purpose;
		Mark B: for passing the board as a parameter;
		Mark C: for using iteration to loop over all (15) locations in the board;
		Mark D: for using indices (or similar) to identify the value of each cell//a FOR-EACH loop used correctly;
		Mark E: for using selection to ascertain if a cell is a hit (value 2);
		Mark F: for incrementing a variable that stores how many hits have been made;
		Mark G: for ascertaining the number of cells yet to be hit (value 1), possibly by using the subroutine F;
		Mark H: for suitable variable initialisation;
		Mark I: for outputting 'Winner' if the number yet to be hit is zero;
		Mark J: for outputting 'Almost there' if the number yet to be hit is 1–3 inclusive;
		Mark K: for outputting the Mark F variable;
		
		A. For marks I,J and K accept returning the number of hits and messages in place of outputting to the screen on this occasion only.
		
		Example of complete correct answer:
		
		`SUBROUTINE howFarAwayFromEnding(board)  [A, B]`
		`  hits ← 0  [part H]`
		`  yetToBeHit ← 0  [part H]`
		`  FOR x ← 0 TO 14  [C]`
		`  IF board[x] = 2 THEN  [D, E]`
		`  hits ← hits + 1  [F]`
		`  ELSE`
		`  IF board[x] = 1 THEN  [part G]`
		`  yetToBeHit ← yetToBeHit + 1  [part G]`
		`  ENDIF`
		`  ENDIF`
		`  ENDFOR`
		`  OUTPUT hits  [K]`
		`  IF yetToBeHit = 0 THEN  [part I]`
		`  OUTPUT 'Winner'  [part I]`
		`  ELSE IF yetToBeHit < 4 THEN  [part J]`
		`  OUTPUT 'Almost there'  [part J]`
		`  ENDIF`
		`ENDSUBROUTINE`
			11

Example of complete correct answer that uses FOREACH:

```
SUBROUTINE howFarAwayFromEnding(board)   [A, B]
   hits ← 0                              [part H]
   yetToBeHit ← 0                        [part H]
   FOREACH cell IN board                 [C, D]
      IF cell = 2 THEN.                  [E]
         hits ← hits + 1                 [F]
      ELSE
         IF cell = 1 THEN                [part G]
            yetToBeHit ← yetToBeHit + 1  [part G]
         ENDIF
      ENDIF
   ENDFOREACH
   OUTPUT hits                           [K]
   IF yetToBeHit = 0 THEN                [part I]
      OUTPUT 'Winner'                    [part I]
   ELSE IF yetToBeHit < 4 THEN           [part J]
      OUTPUT 'Almost there'              [part J]
   ENDIF
ENDSUBROUTINE
```

Example of complete correct answer that doesn't use Mark G variable:

```
SUBROUTINE howFarAwayFromEnding(board)   [A, B]
   hits ← 0                              [part H]
   FOR x ← 0 TO 14                       [C]
      IF board[x] = 2 THEN               [D, E]
         hits ← hits + 1                 [F]
      ENDIF
   ENDFOR
   OUTPUT hits                           [K]
   IF (6 - hits) = 0 THEN          [part G, part I]
      OUTPUT 'Winner'                    [part I]
   ELSE IF (6 - hits) < 4 THEN     [part G, part J]
      OUTPUT 'Almost there'              [part J]
   ENDIF
ENDSUBROUTINE
```

Qu	Part	Marking guidance	Total marks
05	1	**4 marks for AO2 (apply)** first (calculated) value of 10; next calculated value of 5; next calculated value of 16; all values of 8, 4, 2 and 1 in that order; Stop marking at the first incorrect value. Max of 3 marks if additional outputs are given. **Output** 3 10 5 16 8 4 2 1	4
05	2	**2 marks for AO1 (understanding)** **Max 2** from: (The developer has) modularised their code // used subroutines; (The developer has) decomposed the problem // broken the problem down into sub-problems; (The developer has) created interfaces (to the subroutines); (The developer has) used parameters; (The developer has) used return values; (The developer has) used local variables;	2

Qu	Part	Marking guidance	Total marks
06	1	**3 marks for AO2 (apply)** 1 mark for index 0 set to off; 1 mark for index 2 set to on; 1 mark for index 3 set to off; **Max 2 marks** if one error anywhere in the array. **Max 1 mark** if two errors anywhere in the array. **0 marks** if more than two errors anywhere in the array. 0 1 2 3 4 5 6 off off on off off off on	3
06	2	**3 marks for AO2 (apply)** 1 mark for indices 0, 1 and 2 set to on, on and off respectively; 1 mark for index 4 set to off; 1 mark for index 5 set to off; **Max 2 marks** if one error anywhere in the array. **Max 1 mark** if two errors anywhere in the array. **0 marks** if more than two errors anywhere in the array. 0 1 2 3 4 5 6 on on off off off off on	3
06	3	**3 marks for AO2 (apply)** 1 mark for index 0 set to on and index 1 set to off; 1 mark for index 2 set to on; 1 mark for indices 5 and 6 set to off and on respectively; **Max 2 marks** if one error anywhere in the array. **Max 1 mark** if two errors anywhere in the array. **0 marks** if more than two errors anywhere in the array. 0 1 2 3 4 5 6 on off on on off off on	3

Qu	Part	Marking guidance	Total marks

06	4	**3 marks for AO3 (program)**
		
		3 marks if each of the subroutines is used correctly exactly once to produce the correct final array;;;
		
		2 marks if the subroutines are used correctly to produce the correct final array but three subroutines are not used or a subroutine is used more than once;;
		
		1 mark if at least two subroutines (possibly the same) are used correctly but the final array is incorrect;
		
		A. 1 mark for `RANGEOFF(-1, 7);`
		
		First full mark example answer:
		
		`RANGEOFF(0, 6)`
		`NEIGHBOUR(0)`
		`SWITCH(6)`
		
		Second full mark example answer:
		
		`RANGEOFF(0, 6)`
		`SWITCH(6)`
		`NEIGHBOUR(0)`
		
		An example 2 mark answer (not all subroutines are used):
		
		`RANGEOFF(0, 6)`
		`SWITCH(6)`
		`SWITCH(0)`
			3

Question	Part	Marking guidance	Total marks
07	1	**4 marks for AO2 (apply)** • 1 mark for the first value of column `a` **and** the first value of column `b` both correct; • 1 mark for column `a` correctly integer dividing the first value in column `a` by 2 down to 1 and no other values; • 1 mark for minimum of six values in the `b` column, incrementing by one. The number of values in the `b` column must match the number of values in the `a` column; • 1 mark for `OUTPUT` being the final value of `b` and no other values in the output column, and no other values in column `n`; **A.** follow through from column `b` **I.** Different rows used as long as the order within columns is clear **I.** Duplicate values on consecutive rows within a column	4

n	a	b	OUTPUT
50	50	0	
	25	1	
	12	2	
	6	3	
	3	4	
	1	5	
			5

Question	Part	Marking guidance	Total marks
07	2	**Mark is for AO2 (apply)** 1; **R.** the **word** one	1

Question	Part	Marking guidance	Total marks
07	3	**Mark is for AO2 (apply)** 1 mark for giving a new identifier that describes this purpose, eg `count` // `total` // `times` // `numberOfTimes` // `counter`	1

Question	Part	Marking guidance	Total marks
07	4	**2 marks for AO2 (apply)** **Maximum of 2 marks** from: • The `REPEAT...UNTIL` structure tests the condition at the end // the `WHILE...ENDWHILE` structure tests the condition at the beginning; • The `REPEAT...UNTIL` structure will always execute at least once // the `WHILE...ENDWHILE` loop may never execute; • If the value of `n` is 1 (or less) then the `REPEAT...UNTIL` structure will cause the value of `a` / `b` to change, but the `WHILE...ENDWHILE` structure will not; **R.** The `REPEAT...UNTIL` structure repeats lines of code until a condition is true **R.** The `WHILE...ENDWHILE` structure repeats lines of code until a condition is false	2

Question	Part	Marking guidance	Total marks

08	1	**6 marks for AO3 (program)**
		1 mark for each correct item in the correct location
		
		`SUBROUTINE getSize(sampRate, res, seconds)`
		`    size ← sampRate * res * seconds`
		`    size ← size / 8`
		`    RETURN size`
		`ENDSUBROUTINE`
		
		`OUTPUT getSize(100, 16, 60)`
		
		I. Case
		R. Incorrect order of parameters
			6

Question	Part	Marking guidance	Total marks
08	2	**Mark is for AO1 (understanding)** A variable that is only accessible / visible within the subroutine; // A variable that only exists while the subroutine is running;	1

Question	Part	Marking guidance	Total marks
08	3	**3 marks for AO1 (understanding)** **Max 3** marks from: • subroutines can be developed in isolation/independently/separately; • easier to discover errors // testing is more effective (than without a subroutine); • subroutines make program code easier to understand; **A.** 'easier to read' for this year only • subroutines make it easier for a team of programmers to work together on a large project; • subroutines make it easier to reuse code;	3

Question	Part	Marking guidance	Total marks

09	1	**4 marks for AO3 (refine)**
		1 mark for initialising `j` to 0 in correct place;
		1 mark for using `i` and `j` as indices in `ticket`;
		1 mark for incrementing `j` by 1 in correct place;
		1 mark for incrementing `i` by 1 in correct place;
		
		A. `i` and `j` in opposite indices in `ticket`
		I. Case
		
		C# Example 1 (fully correct)
		
		`int i = 0;`
		`while (i < 3) {`
		`  int j = 0;`
		`  while (j < 3) {`
		`  ticket[i, j] = generateKeyTerm();`
		`  j = j + 1;`
		`  }`
		`  i = i + 1;`
		`}`
		
		C# Example 2 (fully correct)
		
		`int i = 0;`
		`while (i < 3) {`
		`  int j = 0;`
		`  while (j < 3) {`
		`  ticket[i, j] = generateKeyTerm();`
		`  j++;`
		`  }`
		`  i++;`
		`}`
		
		Python Example 1 (fully correct)
		
		`i = 0`
		`while i < 3:`
		`  j = 0`
		`  while j < 3:`
		`  ticket[i][j] = generateKeyTerm()`
		`  j = j + 1`
		`  i = i + 1`
			4

Python Example 2 (fully correct)

```
i = 0
while i < 3:
    j = 0
    while j < 3:
        ticket[i][j] = generateKeyTerm()
        j += 1
    i += 1
```

VB.NET Example 1 (fully correct)

```
Dim i As Integer = 0
While (i < 3)
    Dim j As Integer = 0
    While (j < 3)
        ticket(i, j) = generateKeyTerm()
        j = j + 1
    End While
    i = i + 1
End While
```

VB.NET Example 2 (fully correct)

```
Dim i As Integer = 0
While (i < 3)
    Dim j As Integer = 0
    While (j < 3)
        ticket(i, j) = generateKeyTerm()
        j += 1
    End While
    i += 1
End While
```

Question	Part	Marking guidance	Total marks
09	2	**4 marks for AO3 (design), 4 marks for AO3 (program)** Any solution that does not map to the mark scheme refer to lead examiner **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for defining a subroutine called `checkWinner`; **A.** if syntax is incorrect **Mark B** for passing the entire array `ticket` as a parameter to the subroutine; **Mark C** for the use of iteration / selection to attempt to access each element in the `ticket` array; **Mark D** for the use of a selection construct for displaying the output(s); **Program Logic** **Mark E** for initialising a counter to 0 and incrementing the counter in the relevant place; **Mark F** for the correct use of indices which accesses each element in the array; **Mark G** for using a Boolean condition that tests for equality of the array elements with the correct value `"*"`; **Mark H** for outputting the word `Bingo` **and** the count of asterisks in the relevant place; **I.** Case **Maximum 7 marks** if any errors in code.	8

C# Example 1 (fully correct)

All design marks are achieved (**Marks A, B, C and D**)

```
static void checkWinner(string[,] ticket)
{
   int count = 0;                                  (Part of E)
   for (int i = 0; i < 3; i++) {                   (Part of F)
      for (int j = 0; j < 3; j++) {                (Part of F)
         if (ticket[i, j] == "*") {                (G)
            count = count + 1;                     (Part of E)
        }
      }
   }
   if (count == 9) {                               (Part of H)
      Console.WriteLine("Bingo");                  (Part of H)
   }
   else {
      Console.WriteLine(count);
   }
}
```

C# Example 2 (fully correct)

All design marks are achieved (**Marks A, B, C and D**)

```
static void checkWinner(string[,] ticket)
{
   int count = 0;                                  (Part of E)
   if (ticket[0, 0] == "*") {                      (F, G)
       count += 1; }                               (Part of E)
   if (ticket[0, 1] == "*") {
       count += 1; }
   if (ticket[0, 2] == "*") {
       count += 1; }
   if (ticket[1, 0] == "*") {
       count += 1; }
   if (ticket[1, 1] == "*") {
       count += 1; }
   if (ticket[1, 2] == "*") {
       count += 1; }
   if (ticket[2, 0] == "*") {
       count += 1; }
   if (ticket[2, 1] == "*") {
       count += 1; }
   if (ticket[2, 2] == "*") {
       count += 1; }
   if (count < 9) {
      Console.WriteLine(count);                    (Part of H)
   }
   else {
      Console.WriteLine("Bingo");                  (Part of H)
   }
```

```
}
```

C# Example 3 (fully correct)

All design marks are achieved (**Marks A, B, C and D**)

```
static void checkWinner(string[,] ticket){
   int count = 0;                                  (Part of E)

   int i = 0;                                      (Part of F)
   while (i < 3) {                                 (Part of F)

      if (ticket[0, i] == "*") {                   (Part of F, G)
         count += 1; }                             (Part of E)
      i++;                                         (Part of F)
   }
   i = 0;
   while (i < 3) {
      if (ticket[1, i] == "*") {
         count += 1; }
      i++;
   }

   i = 0;
   while (i < 3) {
      if (ticket[2, i] == "*") {
         count += 1; }
      i++;
   }
   if (count < 9) {                                (Part of H)
     Console.WriteLine(count);
   }
   else {
      Console.WriteLine("Bingo");                  (Part of H)
   }
}
```

I. Indentation in C#

I. Missing `static` in C#

Python Example 1 (fully correct)

All design marks are achieved (**Marks A, B, C and D**)

```
def checkWinner(ticket):
   count = 0                                       (Part of E)
   for i in range(3):                              (Part of F)
      for j in range(3):                           (Part of F)
         if ticket[i][j] == "*":                   (Part of F, G)
            count = count + 1                      (Part of E)
   if count == 9:
      print("Bingo")                               (Part of H)
   else:
      print(count)                                 (Part of H)
```

Python Example 2 (fully correct)

All design marks are achieved (**Marks A, B, C and D**)

```
def checkWinner(ticket):
    count = 0                                   (Part of E)
   if ticket[0][0] == "*":                      (F, G)
      count += 1                                (Part of E)
   if ticket[0][1] == "*":
      count += 1
   if ticket[0][2] == "*":
      count += 1
   if ticket[1][0] == "*":
      count += 1
   if ticket[1][1] == "*":
      count += 1
   if ticket[1][2] == "*":
      count += 1
   if ticket[2][0] == "*":
      count += 1
   if ticket[2][1] == "*":
      count += 1
   if ticket[2][2] == "*":
      count += 1
   if count < 9:
      print(count)                              (Part of H)
   else:
      print("Bingo")                            (Part of H)
```

Python Example 3 (fully correct)
All design marks are achieved (**Marks A, B, C and D**)

```
def checkWinner(ticket):
    count = 0                                   (Part of E)
    i = 0
    while i < 3:                                (Part of F)
       if ticket[0][i] == "*":                  (Part of F, G)
          count = count + 1                     (Part of E)
       i = i + 1

    i = 0
    while i < 3:
       if ticket[1][i] == "*":
          count = count + 1
       i = i + 1

    i = 0
    while i < 3:
       if ticket[2][i] == "*":
          count = count + 1
       i = i + 1

    if count == 9:
       print("Bingo")                           (Part of H)
    else:
       print(count)                             (Part of H)
```

VB.NET Example 1 (fully correct)
All design marks are achieved (**Marks A, B, C and D**)

```
Sub checkWinner(ticket)

    Dim count As Integer = 0                    (Part of E)
    For i = 0 To 2                              (Part of F)
       For j = 0 To 2                           (Part of F)
          If ticket(i, j) = "*" Then            (G)
             count = count + 1                  (Part of E)
          End If
       Next
    Next

    If count = 9 Then
       Console.WriteLine("Bingo")               (Part of H)
    Else
       Console.WriteLine(count)                 (Part of H)
    End If
End Sub
```

VB.NET Example 2 (fully correct)
All design marks are achieved (**Marks A, B, C and D**)

```
Sub checkWinner(ticket)

   Dim count As Integer = 0                    (Part of E)
   If ticket(0, 0) = "*" Then                  (F, G)
      count = count + 1                        (Part of E)
   End If
   If ticket(0, 1) = "*" Then
      count = count + 1
   End If
   If ticket(0, 2) = "*" Then
      count = count + 1
   End If
   If ticket(1, 0) = "*" Then
      count = count + 1
   End If
   If ticket(1, 1) = "*" Then
      count = count + 1
   End If
   If ticket(1, 2) = "*" Then
      count = count + 1
   End If
   If ticket(2, 0) = "*" Then
      count = count + 1
   End If
   If ticket(2, 1) = "*" Then
      count = count + 1
   End If
   If ticket(2, 2) = "*" Then
      count = count + 1
   End If
   If count < 9 Then
      Console.WriteLine(count)                 (Part of H)
   Else
      Console.WriteLine("Bingo")               (Part of H)
   End If
End Sub
```

VB.NET Example 3 (fully correct)

All design marks are achieved (**Marks A, B, C and D**)

```
Sub checkWinner(ticket)

   Dim count As Integer = 0                (Part of E)
   Dim i As Integer = 0                    (Part of F)
   While i < 3                             (Part of F)
      If ticket(0,i) = "*" Then            (Part of F, G)
         count = count + 1                 (Part of E)
      End If
      i = i + 1                            (Part of F)
   End While

   i = 0
   While i < 3
      If ticket(1,i) = "*" Then
         count = count + 1
      End If
      i = i + 1
   End While

   i = 0
   While i < 3
      If ticket(2,i) = "*" Then
         count = count + 1
      End If
      i = i + 1
   End While

   If count = 9 Then
      Console.WriteLine("Bingo")           (Part of H)
   Else
      Console.WriteLine(count)             (Part of H)
   End If
End Sub
```

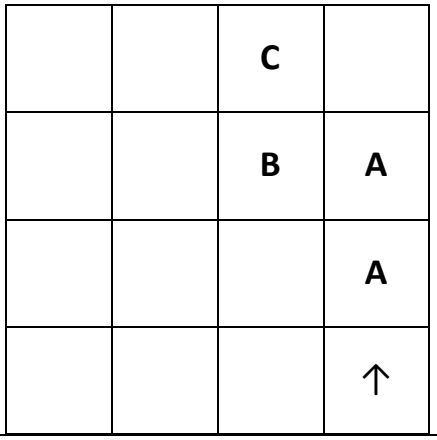
I. Indentation in VB.NET

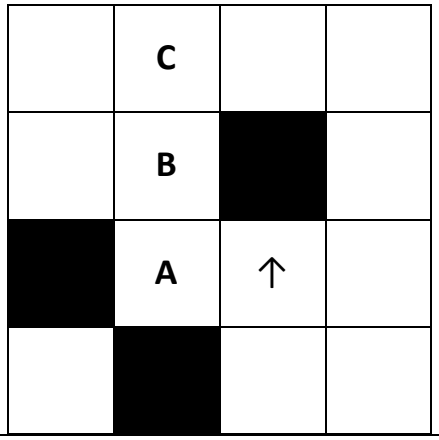
Question	Part	Marking guidance	Total marks
10		**2 marks for AO1 (understanding)** **1 mark** for the advantage, **1 mark** for the expansion point. **Maximum of 2 marks** from: ● Structured programs are easier to read / understand / modify; as they use logical structures // because the code is split into smaller subroutines / chunks / modules / sections; ● Easier to test / debug a program; that is divided into smaller subroutines / chunks / modules / sections; ● Subroutines can be reused; which reduces development time; ● Structured programs are easier to maintain; as they use clear well-documented interfaces // local variables / parameters / return values // as each subroutine is relatively independent of the other; ● Easier to divide projects up between teams; as code is split into subroutines / chunks / modules / sections; **Note to examiners:** both mark points can be taken from different bullets, so long as the explanation is relevant to the advantage eg: Structured programs are easier to understand. This makes it easier to divide projects between teams (2 marks). **Maximum 1 mark** if the explanation is not relevant to the stated advantage. **Maximum 1 mark** if there are two advantages but no explanation of either.	2

Question	Part	Marking guidance	Total marks
11		**4 marks for AO3 (design)** **1 mark** for each correct answer L1 `USERINPUT` L2 `username` L3 `''  R.""` L4 `User not found` **I.** case / spelling mistakes so long as it is clear which option from **Figure 6** has been selected. **Note to Examiners:** If the student has re-written the entire line and added in the correct missing item, award the mark.	4

Question	Part	Marking guidance	Total marks
12	1	**Mark is for AO2 (apply)** **A** Line number 2; **R.** If more than one lozenge shaded	1
12	2	**Mark is for AO2 (apply)** **C** Line number 11; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
12	3	**Mark is for AO2 (apply)** **A** 1 subroutine call; **R.** If more than one lozenge shaded	1
12	4	**Mark is for AO2 (apply)** **B** String; **R.** If more than one lozenge shaded	1
12	5	**Mark is for AO2 (apply)** 2//twice//two;	1

Question	Part	Marking guidance	Total marks
13	1	**3 marks for AO2 (apply)** **Mark as follows:** **1 mark** for the robot moving to **both** squares marked **A**; **1 mark** for the robot moving to the square marked **B**; **1 mark** for the robot moving to the square marked **C**; 	3

Question	Part	Marking guidance	Total marks
13	2	**3 marks for AO2 (apply)** **Mark as follows:** **1 mark** for the robot moving to the square marked **A**; **1 mark** for the robot moving to the square marked **B**; **1 mark** for the robot moving to the square marked **C**; 	3
14		**2 Marks for AO1 (understanding)** Max 2 marks from: Subroutines can be developed in isolation/independently/separately; Easier to discover errors/testing is more effective (than without a structure); Subroutines can be updated without affecting the overall program; **A.** Other valid reasons	2

Question	Part	Marking guidance	Total marks
15	1	**3 marks for AO2 (apply)** 1 mark for C written once and in column 1; 1 mark for A and B written once and both in column 2 (in any order); 1 mark for A and B written once and in correct positions in column 2; Column 0 Column 1 Column 2 ____ C A B	3
15	2	**3 marks for AO2 (apply)** 1 mark for A written once and in correct column (0); 1 mark for B written once and in correct column (2); 1 mark for C written once and in correct column (1); Column 0 Column 1 Column 2 A C B	3

Question	Part	Marking guidance	Total marks
15	3	**4 marks for AO3 (design)** **Mark A** for using a `WHILE` loop or similar to move from column 0 to column 2; **Mark B** for a Boolean condition that detects when column 0 is empty; **Mark C** for using a second WHILE loop or similar to move the result from A and B into column 1 (both the loop and the associated Boolean condition need to be correct to gain this mark); **or** **Mark A** for using a `FOR` loop or similar to move from column 0 to column 2; **Mark B** for ascertaining the terminating value for the `FOR` loop; **Mark C** for using a second `FOR` loop or similar to move the result from A and B into column 1 (both the loop and the associated terminating value need to be correct to gain this mark); **and** **Mark D** for using the subroutines correctly throughout, i.e. called with appropriate parameters and return values handled correctly; **A.** Minor spelling errors such as `HIEGHT` for `HEIGHT` **I.** Case **Example 1** `WHILE HEIGHT(0) > 0` (**Part of A, B**) `  MOVE(0, 2)` (**Part of A**) `ENDWHILE` `WHILE HEIGHT(2) > 0` (**Part of C**) `  MOVE(2, 1)` (**Part of C**) `ENDWHILE` (`MOVE` and `HEIGHT` are used correctly throughout so **D**.) **Example 2** `DO` (**Part of A**) `  MOVE(0, 2)` (**Part of A**) `WHILE HEIGHT(0) > 0` (**Part of A, B**) `DO` (**Part of C**) `  MOVE(2, 1)` (**Part of C**) `WHILE HEIGHT(2) > 0` (**Part of C**) (`MOVE` and `HEIGHT` are used correctly throughout so **D**.)	4

Example 3

```
REPEAT                          (Part of A)
   MOVE(0, 2)                   (Part of A)
UNTIL HEIGHT(0) = 0             (Part of A, B)
REPEAT                          (Part of C)
   MOVE(2, 1)                   (Part of C)
WHILE HEIGHT(2) = 0             (Part of C)
```

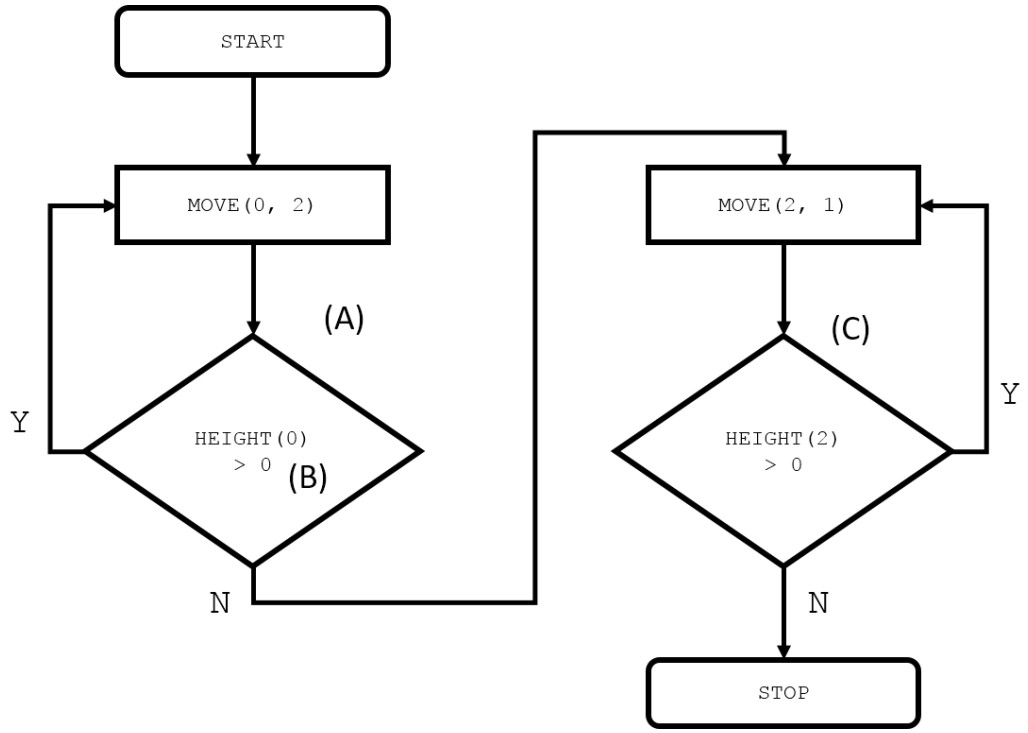
(MOVE and HEIGHT are used correctly throughout so **D**.)

Example 4

```
number_of_blocks ← HEIGHT(0)          (Part of B)
FOR x ← 0 TO number_of_blocks         (Part of A, Part of B)
   MOVE(0, 2)                         (Part of A)
ENDFOR
FOR x ← 0 TO number_of_blocks         (Part of C)
   MOVE(2, 1)                         (Part of C)
ENDFOR                                (Part of C)
```

(MOVE and HEIGHT are used correctly throughout so **D**.)

Example 5

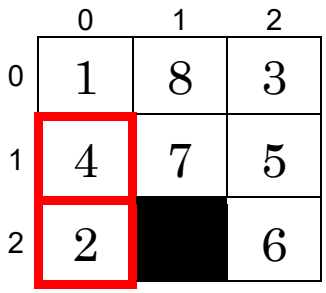


(MOVE and HEIGHT are used correctly throughout so **D**.)

Question	Part	Marking guidance	Total marks
16		**1 mark for AO3 (refine)** B; **R.** if more than 1 lozenge shaded	1

Question	Part	Marking guidance	Total marks
17	1	**Mark is for AO2 (apply)** 11;	1

Question	Part	Marking guidance	Total marks
17	2	**Mark is for AO2 (apply)** 17;	1

Question	Part	Marking guidance	Total marks
18	1	**2 marks for AO2 (apply)**  **1 mark** for 4 in the correct position; **1 mark** for 2 in the correct position; **Maximum 1 mark** if any errors. **A**. `0` instead of blank space or any other sensible indicator for the blank space. **A**. unaffected cell contents not shown as long as it is clear which is the blank space. **A**. answers written on **Figure 15** if board is left blank.	2

Question	Part	Marking guidance	Total marks
18	2	**2 marks for AO2 (apply)** **A** Nested iteration is used; **C** The number of comparisons made between `getTile(i, j)` and `0` will be nine; **R.** if more than two lozenges shaded	2

Question	Part	Marking guidance	Total marks
18	3	**Mark is for AO2 (apply)** (The first iteration structure) is used to iterate through the rows; **Note to examiners:** award both marks (Q12.3 and Q12.4) if the student answers are correct but the opposite way around, ie 'columns' for Q12.3 and 'rows' for Q12.4	1

18	4	**Mark is for AO2 (apply)** (The second iteration structure) is used to iterate through the columns; **Note to examiners:** award both marks (Q12.3 and Q12.4) if the student answers are correct but the opposite way around, ie 'columns' for Q12.3 and 'rows' for Q12.4	1

Question	Part	Marking guidance	Total marks
18	5	**Mark is for AO2 (apply)** To find/store the position/coordinates of the blank space // to find the tile/value of `getTile` that is blank/`0`;	1

Question	Part	Marking guidance	Total marks
18	6	**1 mark for AO3 (design), 3 marks for AO3 (program)** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for the use of a selection structure with multiple conditions // use of multiple selection structures // an iteration structure containing one selection structure; **Program Logic** **Mark B** for correctly checking three consecutive values in `getTile` (even if the wrong row/column); **Mark C** for fully correct indices used in `getTile` for the first row; **Mark D** for a structure that would output either `Yes` or `No` correctly in all circumstances, but never both; **A.** if conditions are not fully correct **I**. Case **I.** Messages or no messages with input statements **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect **Maximum 3 marks** if any errors in code. **Note to examiners** In C#/VB.NET examples, explicit variable declarations are not shown. Refer to the specific language type issues section of the appropriate Marking guidance document. Any correct variable declarations in student answers should be accepted.	4

C# Example 1 (fully correct)

Design mark is achieved (**Mark A**)

```
if (getTile(0, 0) + 1 == getTile(0, 1)) {
    if (getTile(0, 1) + 1 == getTile(0, 2)) {
        Console.WriteLine("Yes");
    }
    else {
        Console.WriteLine("No");
    }
}
else {
    Console.WriteLine("No");
}
```

(Part B, Part C)

(Part B, Part C)

(Part D)

(Part D)

(Part D)

I. Indentation in C#

A. `Write` in place of `WriteLine`

Note to examiners: in a nested `if` statement, all pathways must be present to award Mark D (including the part shaded yellow above).

C# Example 2 (fully correct)
Design mark is achieved (**Mark A**)

```
if (getTile(0, 0) + 1 == getTile(0, 1)) {
   if (getTile(0, 0) + 2 == getTile(0, 2)) {
      Console.WriteLine("Yes");
   }
   else {
      Console.WriteLine("No");
   }
}
else {
   Console.WriteLine("No");
}
```

(Part B, Part C)
(Part B, Part C)
(Part D)
(Part D)
(Part D)

I. Indentation in C#
A. `Write` in place of `WriteLine`

Note to examiners: in a nested `if` statement, all pathways must be present to award Mark D (including the part shaded yellow above).

C# Example 3 (fully correct)
Design mark is achieved (**Mark A**)

```
if ((getTile(0, 1) - getTile(0, 0) == 1) &&
(getTile(0, 2) - getTile(0, 1) == 1)) {
   Console.WriteLine("Yes");
}
else {
   Console.WriteLine("No");
}
```

(Part B, Part C)
(Part D)
(Part D)

I. Indentation in C#
A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)
Design mark is achieved (**Mark A**)

```
if getTile(0, 0) + 1 == getTile(0, 1):          (Part B, Part C)
    if getTile(0, 1) + 1 == getTile(0, 2):      (Part B, Part C)
        print("Yes")                            (Part D)
    else:
        print("No")                             (Part D)
else:
    print("No")                                 (Part D)
```

Note to examiners: in a nested `if` statement, all pathways must be present to award Mark D (including the part shaded yellow above).

Python Example 2 (fully correct)
Design mark is achieved (**Mark A**)

```
if getTile(0, 0) + 1 == getTile(0, 1):          (Part B, Part C)
    if getTile(0, 0) + 2 == getTile(0, 2):      (Part B, Part C)
        print("Yes")                            (Part D)
    else:
        print("No")                             (Part D)
else:
    print("No")                                 (Part D)
```

Note to examiners: in a nested `if` statement, all pathways must be present to award Mark D (including the part shaded yellow above).

Python Example 3 (fully correct)
Design mark is achieved (**Mark A**)

```
if getTile(0, 1) - getTile(0, 0) == 1 and
getTile(0, 2) - getTile(0, 1) == 1:
   print("Yes")
else:
   print("No")
```

(Part B, Part C)
(Part D)
(Part D)

VB.NET Example 1 (fully correct)
Design mark is achieved (**Mark A**)

```
If getTile(0, 0) + 1 = getTile(0, 1) Then
   If getTile(0, 1) + 1 = getTile(0, 2) Then
      Console.WriteLine("Yes")
   Else
      Console.WriteLine("No")
   End If
Else
   Console.WriteLine("No")
End If
```

(Part B, Part C)
(Part B, Part C)
(Part D)
(Part D)
(Part D)

I. Indentation in VB.NET
A. `Write` in place of `WriteLine`

Note to examiners: in a nested `if` statement, all pathways must be present to award Mark D (including the part shaded yellow above).

VB.NET Example 2 (fully correct)
Design mark is achieved (**Mark A**)

```
If getTile(0, 0) + 1 = getTile(0, 1) Then
   If getTile(0, 0) + 2 = getTile(0, 2) Then
      Console.WriteLine("Yes")
   Else
      Console.WriteLine("No")
   End If
Else
   Console.WriteLine("No")
End If
```

(Part B, Part C)
(Part B, Part C)
(Part D)
(Part D)
(Part D)

I. Indentation in VB.NET
A. `Write` in place of `WriteLine`

Note to examiners: in a nested `if` statement, all pathways must be present to award Mark D (including the part shaded yellow above).

VB.NET Example 3 (fully correct)
Design mark is achieved (**Mark A**)

```
If getTile(0, 1) - getTile(0, 0) = 1 And
getTile(0, 2) - getTile(0, 1) = 1 Then
   Console.WriteLine("Yes")
Else
   Console.WriteLine("No")
End If
```

(Part B, Part C)
(Part D)
(Part D)

I. Indentation in VB.NET
A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
18	7	**2 marks for AO3 (design), 4 marks for AO3 (program)** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for the use of an indefinite iteration structure that exists within their language; **Mark B** for the use of a selection structure or equivalent to check for a blank space; **Program Logic** **Mark C** for using user input and storing the result in two variables correctly for the row and column; **Mark D** for code that uses **both** the `solved` subroutine and the `checkSpace` subroutine in logically correct locations; **Mark E** for calling the `move` subroutine in a pathway following an `= True` condition (or equivalent) with the row and column from the user input as parameters; **Mark F** for outputting `Invalid move` when the tile does not get moved **and** asking the user to input row and column again in logically correct locations; **R**. if user is asked to re-input after the problem is solved. **I**. Case **I.** Messages or no messages with input statements **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect **Maximum 5 marks** if any errors in code. **Note to examiners** In C#/VB.NET examples, explicit variable declarations are not shown. Refer to the specific language type issues section of the appropriate Marking guidance document. Any correct variable declarations in student answers should be accepted.	6

C# Example 1 (fully correct)

All design marks are achieved (**Marks A and B**)

```
while (!solved()) {                                    (Part D)
   row =                                               (Part C)
Convert.ToInt32(Console.ReadLine());
   col =                                               (Part C)
Convert.ToInt32(Console.ReadLine());
   if (checkSpace(row, col)) {                         (Part D)
      move(row, col);                                  (E)
   }
   else {
      Console.WriteLine("Invalid move");               (F)
   }
}
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

C# Example 2 (fully correct)

All design marks are achieved (**Marks A and B**)

```
do {
   row =                                               (Part C)
Convert.ToInt32(Console.ReadLine());
   col =                                               (Part C)
Convert.ToInt32(Console.ReadLine());
   if (checkSpace(row, col)) {                         (Part D)
      move(row, col);                                  (E)
   }
   else {
      Console.WriteLine("Invalid move");               (F)
   }
} while (!solved);                                     (Part D)
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

All design marks are achieved (**Marks A and B**)

```
while not solved():                          (Part D)
   row = int(input())                        (Part C)
   col = int(input())                        (Part C)
   if checkSpace(row, col):                  (Part D)
      move(row, col)                         (E)
   else:
      print("Invalid move")                  (F)
```

Python Example 2 (fully correct)

All design marks are achieved (**Marks A and B**)

```
while solved() == False:                     (Part D)
   row = int(input())                        (Part C)
   col = int(input())                        (Part C)
   if checkSpace(row, col) == True:          (Part D)
      move(row, col)                         (E)
   else:
      print("Invalid move")                  (F)
```

VB.NET Example 1 (fully correct)
All design marks are achieved (**Marks A and B**)

```
While Not solved()                                  (Part D)
   row = Console.ReadLine()                         (Part C)
   col = Console.ReadLine()                         (Part C)
   If checkSpace(row, col) Then                     (Part D)
      move(row, col)                                (E)
   Else
      Console.WriteLine("Invalid move")             (F)
   End If
End While
```

I. Indentation in VB.NET
A. `Write` in place of `WriteLine`

VB.NET Example 2 (fully correct)
All design marks are achieved (**Marks A and B**)

```
Do                                                  (Part D)
   row = Console.ReadLine()                         (Part C)
   col = Console.ReadLine()                         (Part C)
   If checkSpace(row, col) Then                     (Part D)
      move(row, col)                                (E)
   Else
      Console.WriteLine("Invalid move")             (F)
   End If
Loop Until solved()                                 (Part D)
```

I. Indentation in VB.NET
A. `Write` in place of `WriteLine`

VB.NET Example 3 (fully correct)
All design marks are achieved (**Marks A and B**)

```
Do While Not solved()                               (Part D)
   row = Console.ReadLine()                         (Part C)
   col = Console.ReadLine()                         (Part C)
   If checkSpace(row, col) Then                     (Part D)
      move(row, col)                                (E)
   Else
      Console.WriteLine("Invalid move")             (F)
   End If
Loop
```

I. Indentation in VB.NET
A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
19	1	**Mark is for AO1 (recall)** **Maximum 1 mark** from: • They are only accessible within the subroutine; • They only exist/use memory while the subroutine is executing; • They have limited scope; **A.** Can have the same identifier as other variables outside of the subroutine	1

Question	Part	Marking guidance	Total marks
19	2	**2 marks for AO3 (design), 4 marks for AO3 (program)** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for the use of a definite iteration structure, or similar, that exists within their language to carry out the requirements of the task; **Mark B** for the use of a selection structure to check visitor numbers; **Program Logic** **Mark C** for correctly defining the subroutine and parameter; **Mark D** for accepting user input multiple times as per the parameter or equivalent, representing the number of days; **R.** if iteration syntax or boundaries are not fully correct **Mark E** for adding one to a counter variable inside a selection structure under the correct conditions, which has been appropriately initialised (to `0`); **Mark F** for returning the counter value calculated within the subroutine; **Maximum 5 marks** if any errors in code. **I.** Case **I.** Messages or no messages with input statements **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect **Note to examiners** In C#/VB.NET examples, explicit variable declarations are not shown. Refer to the specific language type issues section of the appropriate Marking guidance document. Any correct variable declarations in student answers should be accepted.	6

C# Example 1 (fully correct)

All design marks are achieved (**Marks A and B**)

```
static int countDays(days) {                          (C)
   count = 0;                                         (Part E)
   visitors = 0;
   for (i = 0; i < days; i++) {                       (Part D)
      visitors =
Convert.ToInt32(Console.ReadLine());                  (Part D)
      if (visitors > 200) {                           (Part E)
         count = count + 1;                           (Part E)
      }
   }
   return count;                                      (F)
}
```

A. with or without `static`
A. Alternative numerical data type for return value
I. Indentation in C#

Python Example 1 (fully correct)

All design marks are achieved (**Marks A and B**)

```
def countDays(days):                                  (C)
   count = 0                                          (Part E)
   for i in range(days):                              (Part D)
      visitors = int(input())                         (Part D)
      if visitors > 200:                              (Part E)
         count = count + 1                            (Part E)
   return count                                       (F)
```

Python Example 2 (fully correct)
All design marks are achieved (**Marks A and B**)

```
def countDays(days):                          (C)
    count = 0                                 (Part E)
    i = 0                                     (Part D)
    while (i < days):                         (Part D)
        if int(input()) > 200:                (Part D) (Part E)
            count += 1                        (Part E)
        i += 1                                (Part D)
    return count                              (F)
```

VB.NET Example 1 (fully correct)
All design marks are achieved (**Marks A and B**)

```
Function countDays(days) As Integer           (C)
    count = 0                                 (Part E)
    For i = 1 To days                         (Part D)
        visitors = Console.ReadLine()         (Part D)
        If visitors > 200 Then                (Part E)
            count = count + 1                 (Part E)
        End If
    Next
    Return count                              (F)
End Function
```

A. Alternative numerical data type for return value
I. Indentation in VB.NET

VB.NET Example 2 (fully correct)

All design marks are achieved (**Marks A and B**)

```
Function countDays(days) As Integer          (C)
   Dim count As Integer                      (Part E)
   For i = 1 To days                         (Part D)
      If Console.ReadLine() > 200 Then       (Part E)
         count += 1                          (Part E)
      End If
   Next
   Return count                              (F)
End Function
```

A. Alternative numerical data type for return value
I. Indentation in VB.NET