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		**7 marks for AO3 (program)**	7

If `CHAR_TO_CODE` is not used then a maximum of 6 marks.

Mark A for using user input;
Mark B for storing the result of user input in a variable or using the user input directly as a parameter to `CHAR_TO_CODE`;
Mark C for using selection to determine if character is lowercase or otherwise;
Mark D for using a Boolean expression that uses `CHAR_TO_CODE` with the input parameter being the user input (either directly or when stored in a variable);
Mark E for a Boolean expression that checks if the character code is between 97 and 122 (97+25) **inclusive**;
Mark F for outputting `LOWER` and `NOT LOWER` in logically separate places such as the `IF` and `ELSE` part of selection;

Mark G if the algorithm is completely correct;

A. `LOWER` and `NOT LOWER` stated in lower case for Mark F.
A. Any logically equivalent Boolean expression for Mark E.
A. Minor errors in spelling if the meaning is clear.

Example 1 (fully correct)

```
character ← USERINPUT                                (A, B)
character_code ← CHAR_TO_CODE(character)             (Part of D)
IF character_code ≥ 97 AND character_code ≤ 122 THEN(C, D, E)
   OUTPUT 'LOWER'                                    (Part of F)
ELSE
   OUTPUT 'NOT LOWER'                                (Part of F)
ENDIF
```

(G awarded as completely correct)

Example 2 (fully correct)

```
character_code ← CHAR_TO_CODE(USERINPUT)            (A, B, Part of
D)
IF character_code < 97 OR character_code > 122 THEN  (C, D, E)
   OUTPUT 'NOT LOWER'                               (Part of F)
ELSE
   OUTPUT 'LOWER'                                   (Part of F)
ENDIF
```

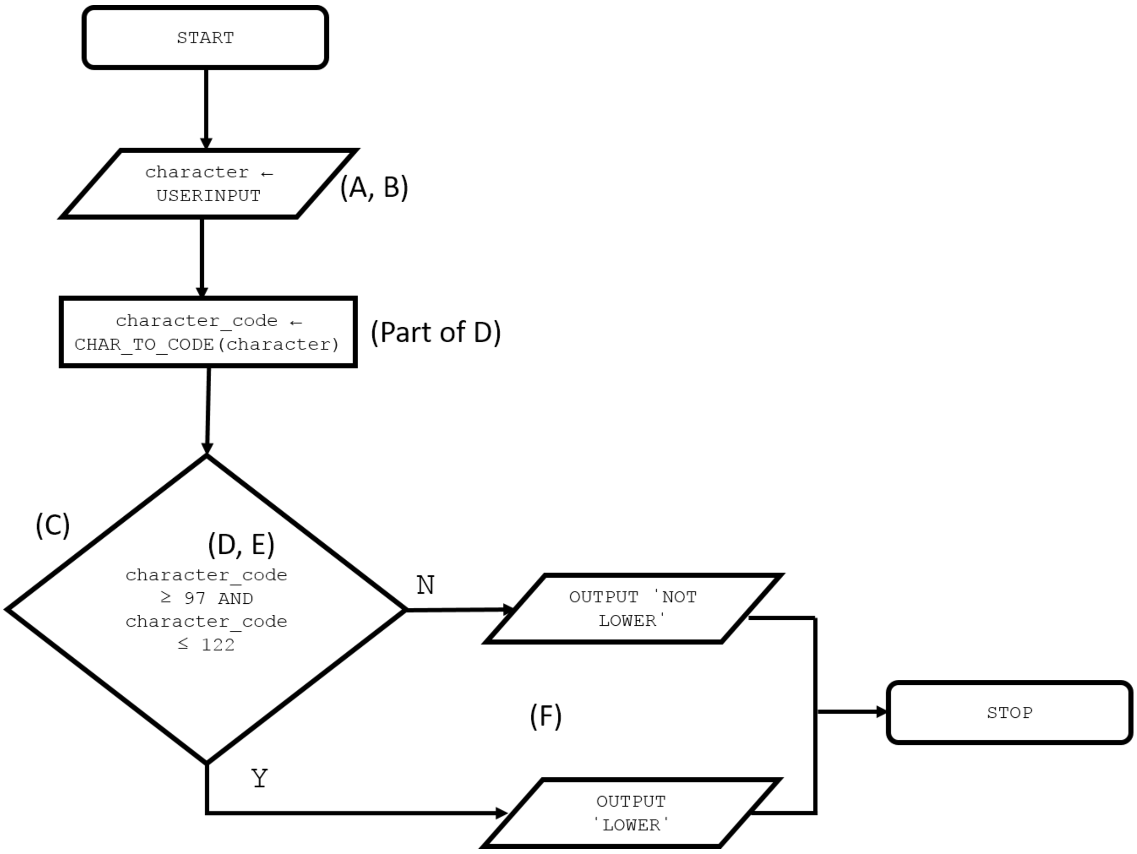
(G awarded as completely correct)

Example 3 (fully correct)

```
character ← USERINPUT                               (A, B)
character_code ← CHAR_TO_CODE(character)            (Part of D)
IF 97 ≤ character_code ≤ 122 THEN                   (C, D, E)
   OUTPUT 'LOWER'                                   (Part of F)
ELSE
   OUTPUT 'NOT LOWER'                               (Part of F)
ENDIF
```

(G awarded as completely correct)

Example 4 (fully correct)



(G awarded as completely correct)

Example 5 (6 marks)

```
IF CHAR_TO_CODE(USERINPUT) ≥ 97 AND
   CHAR_TO_CODE(USERINPUT) ≤ 122 THE               (A, B, C, D,
E)
   OUTPUT 'LOWER'                                  (Part of F)
ELSE
   OUTPUT 'NOT LOWER'                              (Part of F)
ENDIF
```

(G not awarded as USERINPUT used twice)

Example 6 (6 marks)

```
character_code ← CHAR_TO_CODE(USERINPUT)        (A, B, Part of
D)
IF character_code < 97 OR character_code > 122 THEN  (C, D, E)
   OUTPUT 'LOWER'                               (Part of F)
ELSE
   OUTPUT 'NOT LOWER'                           (Part of F)
ENDIF
```

(G not awarded as LOWER and NOT LOWER are in the wrong places)

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>03</td><td>1</td><td><b>Mark is for AO2 (apply)</b>

Boolean//bool;

<b>I.</b> Minor spelling mistakes</td><td>1</td></tr>
<tr><td>03</td><td>2</td><td><b>2 marks for AO2 (apply)</b>

(The identifier) <code>sorted</code> describes the purpose//role//meaning of the variable;
this makes the algorithm easier to understand//maintain//follow;

or

(The identifier) <code>s</code> does not describe the purpose//role//meaning of the variable;
this makes the algorithm harder to understand//maintain//follow;</td><td>2</td></tr>
<tr><td>03</td><td>3</td><td><b>Mark is for AO2 (apply)</b>

<b>A</b> (The algorithm uses a named constant.) only;
<b>If more than one lozenge shaded then mark is not awarded</b></td><td>1</td></tr>
<tr><td>03</td><td>4</td><td><b>6 marks for AO2 (apply)</b>

1 mark for column <code>arr[0]</code> correct;
1 mark for column <code>arr[1]</code> correct;
1 mark for column <code>arr[2]</code> correct <b>only if</b> <code>arr[0]</code> and <code>arr[1]</code> are correct;
1 mark for <code>sorted</code> column correct;
1 mark for <code>i</code> column correct;
1 mark for <code>t</code> column correct;

<table>
<tr><th colspan="3">Arr</th><th rowspan="2">sorted</th><th rowspan="2">i</th><th rowspan="2">t</th></tr>
<tr><th>0</th><th>1</th><th>2</th></tr>
<tr><td>4</td><td>1</td><td>6</td><td>false</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>true</td><td>0</td><td></td></tr>
<tr><td>1</td><td>4</td><td></td><td>false</td><td></td><td>4</td></tr>
<tr><td></td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td></td></tr>
<tr><td></td><td></td><td></td><td>true</td><td>0</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

<b>I.</b> different rows used as long as the order within columns is clear
<b>I.</b> duplicate values on consecutive rows within a column</td><td>6</td></tr>
</table>

Question	Part	Marking guidance	Total marks
03	5	**3 marks for AO2 (apply)** 1 mark if pairwise comparisons are made in the second row but allow for one pairwise comparison error; 1 mark if pairwise comparisons are made in the third row but allow for one pairwise comparison error (allow follow through from previous row); 1 mark if all correct; 7 3 4 1 2 8 5 6 **3 7 1 4 2 8 5 6** **1 3 4 7 2 5 6 8** 1 2 3 4 5 6 7 8	3
03	6	**Mark is for AO1 (understanding)** It is more (time) efficient// It will usually take fewer steps; **A.** quicker//it will take less time as long as the answer has been qualified.	1
03	7	**2 marks for AO2 (apply)** Maximum of 2 from: It allows the code to be (more easily) reused; It can be used to sort any array (not just the one on line 1); It would be easier to test; The code could be changed//updated without affecting the overall program; Makes the program easier to read//understand; **A.** Any other creditable answer as long as they are clearly distinct from the other responses.	2

Question	Part	Marking guidance	Total marks
04		**8 marks for AO3 (program)**	8

DPT. For repeated errors in user input and variable assignment.

Mark A for getting user input for the distance and storing in a variable;
Mark B for using a WHILE loop or similar to re-prompt for and re-assign the user input;
Mark C for using a correct Boolean condition with the validation structure;
Mark D for getting user input for the passengers;
Mark E for a fare that charges £2 per passenger;
Mark F for a fare that charges £1.50 for every kilometre;
Mark G for outputting the fare based on E and F (Even if E and/or F have been calculated incorrectly);

Mark H if the algorithm is completely correct;

Example 1 (fully correct)

```
distance ← USERINPUT                        (A)
WHILE distance ≤ 0                          (Part of B, C)
   distance ← USERINPUT                     (Part of B)
ENDWHILE
passengers ← USERINPUT                      (D)
fare ← 2 * passengers                       (E)
fare ← fare + (1.5 * distance)              (F)
OUTPUT fare                                 (G)
```

(Mark H as completely correct)

Example 2 (fully correct)

```
REPEAT                                      (Part of B)
   distance ← USERINPUT                     (A, Part of B)
UNTIL distance > 0                          (C)
fare ← (2 * USERINPUT) + (1.5 * distance)   (D, E, F)
OUTPUT fare                                 (G)
```

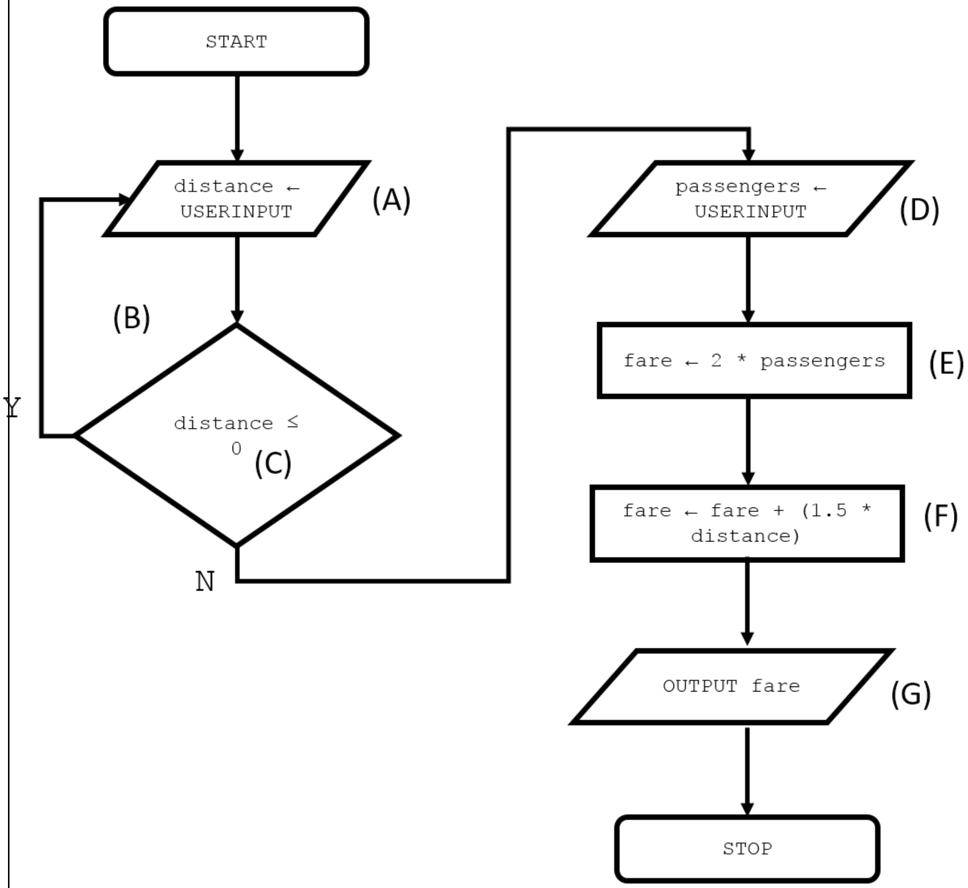
(Mark H as completely correct)

Example 3 (fully correct)

```
DO                                          (Part of B)
   distance ← USERINPUT                     (A, Part of B)
WHILE NOT (distance > 0)                    (C)
fare ← (2 * USERINPUT) + (1.5 * distance)   (D, E, F)
OUTPUT fare                                 (G)
```

(Mark H as completely correct)

Example 4 (fully correct)



(Mark H as completely correct)

Example 5 (7 marks)

```
distance ← USERINPUT                      (A)
WHILE distance ≤ 0                        (C)
   distance ← USERINPUT                   (Part of B)
ENDWHILE
passengers ← USERINPUT                    (D)
fare ← 2 * passengers                     (E)
fare ← 1.5 * distance                     (F)
OUTPUT fare                               (G)
```

(Mark H not awarded as the final fare does not include the cost of `2 * passengers`)

Example 6 (5 marks)

```
distance ← USERINPUT                         (A)
IF distance ≤ 0                              (C)
   distance ← USERINPUT
ENDIF
passengers ← USERINPUT                       (D)
fare ← 2 * passengers                        (E)
fare ← fare + (1.5 * distance)               (F)
OUTPUT fare                                  (G)
```

(Mark B not awarded as `IF` used instead of iteration and mark H not awarded as not completely correct)

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>05</td><td>1</td><td><b>3 marks for AO2 (apply)</b>

1 mark for C written <b>once</b> and in column 1;
1 mark for A and B written <b>once</b> and both in column 2;
1 mark for A and B written <b>once</b> and in correct positions in column 2;

<table><tr><td>Column 0</td><td>Column 1</td><td>Column 2</td></tr><tr><td></td><td></td><td>A</td></tr><tr><td></td><td>C</td><td>B</td></tr></table></td><td>3</td></tr>
<tr><td>05</td><td>2</td><td><b>3 marks for AO2 (apply)</b>

1 mark for A written <b>once</b> and in correct column (0);
1 mark for B written <b>once</b> and in correct column (2);
1 mark for C written <b>once</b> and in correct column (1);

<table><tr><td>Column 0</td><td>Column 1</td><td>Column 2</td></tr><tr><td>A</td><td>C</td><td>B</td></tr></table></td><td>3</td></tr>
<tr><td>05</td><td>3</td><td><b>3 marks for AO2 (apply)</b>

If any value is written more than once no marks for that value.

3 marks if A, B and C are all written <b>once</b>, 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;

<table><tr><td>Column 0</td><td>Column 1</td><td>Column 2</td></tr><tr><td></td><td></td><td>B</td></tr><tr><td></td><td>A</td><td>C</td></tr></table></td><td>3</td></tr>
</table>

Question	Part	Marking guidance	Total marks
05	4	**5 marks for AO3 (program)**	5

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.)

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)
```

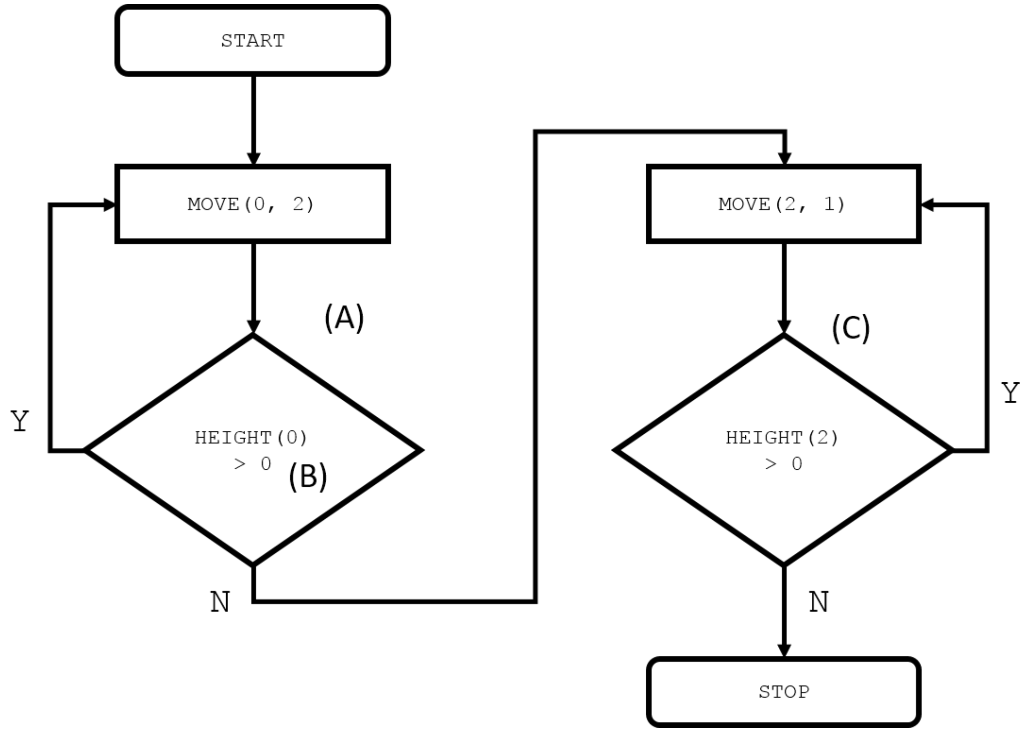
(MOVE and HEIGHT are used correctly throughout so D and completely correct so also E.)

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.)

Question	Part	Marking guidance	Total marks
06	1	**Mark is for AO2 (apply)** **C** `flourNeeded ← eggsUsed * 100`; **If more than one lozenge shaded then mark is not awarded**	1
06	2	**Mark is for AO2 (apply)** **A** Assignment; **If more than one lozenge shaded then mark is not awarded**	1
06	3	**4 marks for AO3 (program)** Max 3 marks if the answer contains any errors. **1 mark (A)** Indefinite iteration is used; **1 mark (B)** User input is used within the iteration/validation structure **and** the result is stored in the variable `eggsUsed`; **2 marks (C, D)** A Boolean condition checks the lower bound of `eggsUsed` is greater than zero/greater than or equal to one **and** the upper bound of `eggsUsed` is less than or equal to eight/less than nine (even if the **structure** is incorrect). This could possibly be one expression such as `0 < eggsUsed ≤ 8`;; If condition not completely correct then: **1 mark** The Boolean condition checks the lower bound of `eggsUsed` is greater than zero (even if the structure is incorrect) OR The Boolean condition checks the upper bound of `eggsUsed` is less than or equal to eight (even if the structure is incorrect) OR The Boolean conditions for the lower and upper bound are joined with the AND operator (even if the structure or the conditions themselves are incorrect); OR A method has been used that does not use a Boolean condition but is largely clear;	

Example 4 mark answer:

```
REPEAT                                      (A)
  eggsUsed ← USERINPUT                      (B)
UNTIL eggsUsed > 0 AND eggsUsed ≤ 8         (C, D)
```

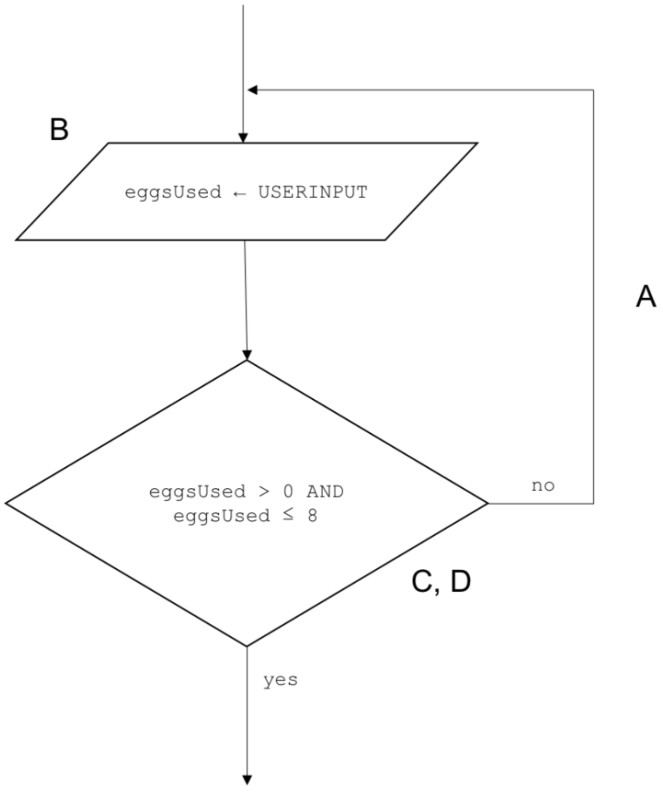
Example 4 mark answer:

```
DO                                          (A)
  eggsUsed ← USERINPUT                      (B)
WHILE eggsUsed < 1 OR eggsUsed > 8          (C, D)
```

Example 4 mark answer:

```
REPEAT                                      (A)
  eggsUsed ← USERINPUT                      (B)
UNTIL 0 < eggsUsed ≤ 8                      (C, D)
```

Example 4 mark answer:



<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>07</td><td>1</td><td>

4 marks for AO2 (apply)

Mark A for `totalSize` completely correct;
Mark B for `dataToBeSent` decrementing correctly by the value given for `totalSize` until it is ≤ 0 (award even if `totalSize` is incorrect);
Mark C for `numberOfPackets` starting at 0;
Mark D for minimum of three values in the `numberOfPackets` column, incrementing by one. The number of values in the `dataToBeSent` column must match the number of values in the `numberOfPackets` column;

Correct table is:

<table>
<tr><th>totalSize</th><th>dataToBeSent</th><th>numberOfPackets</th></tr>
<tr><td>300</td><td><i>750</i></td><td>0</td></tr>
<tr><td></td><td>450</td><td>1</td></tr>
<tr><td></td><td>150</td><td>2</td></tr>
<tr><td></td><td>-150</td><td>3</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

A. follow through for incorrect `totalSize`

</td><td>4</td></tr>
<tr><td>07</td><td>2</td><td>

Mark is for AO2 (apply)

(they are both) constants//their values do not change

</td><td>1</td></tr>
<tr><td>07</td><td>3</td><td>

Mark is for AO2 (apply)

A Input: `dataToBeSent`, output: `numberOfPackets`;
If more than one lozenge shaded then mark is not awarded

</td><td>1</td></tr>
<tr><td>07</td><td>4</td><td>

3 marks for AO3 (program)

A `dataToBeSent`;

B `totalSize`;

C `numberOfPackets + 1`;

A. `numberOfPackets++ for C;`

I. case and minor spelling mistakes

</td><td>3</td></tr>
</table>

08	1	**Mark is for AO2 (apply)** **C** Selection; **If more than one lozenge shaded then mark is not awarded**	1

08	2	**Mark is for AO2 (apply)** **D** String; **If more than one lozenge shaded then mark is not awarded**	1

08	3	**Mark is for AO2 (apply)** 3//three;	1

08	4	**2 marks for AO2 (apply)** `'no'` followed by `'yes'`; any value that isn't `'no'` followed by `'yes'` (allow by examples such as `'yes'` followed by `'yes'`); **R.** if a sequence does not contain two user inputs.	2

Question	Part	Marking guidance	Total marks
08	5	**3 marks for AO2 (apply)** Maximum three marks overall. **Maximum two marks from each section.** **Reason** • The output message is not descriptive enough/the user is not told what word/words they should use to answer (before user input); • The Boolean expression (at lines 3, 6 and 14) only matches exact values//the program is only written for the exact words `yes` and `no` // a **clear** indication that `y` is not recognised as `yes` or `n` is not recognised as `no`; • A clear explanation of how to fix the problem; **What would happen** Any clear descriptions of what would happen. Line numbers may or may not be included. If the logic and explanation is clear credit the answer. This can include but is not limited to: • Line 3 will only be true if they enter '`no`' // Line 3 will not be true if they enter anything other than '`no`'; • Line 6/14 will only be true if they enter '`yes`' // Line 6/14 will not be true if they enter anything other than '`yes`'; • if they enter '`n`' at line 2 the algorithm will execute an incorrect code block; • if they enter '`y`' at line 5 or line 13 an incorrect message will be output;	3

<table>
<tr><th>Qu</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>09</td><td>1</td><td>
<b>2 marks for AO2 (apply)</b>

The first value of result 16;
The last value of result 12;

Max 1 mark if more than two values are given for result.

The correct table is as follows:
<table>
<tr><th><code>result</code></th></tr>
<tr><td>16</td></tr>
<tr><td>12</td></tr>
<tr><td></td></tr>
</table>
</td><td>2</td></tr>
<tr><td>09</td><td>2</td><td>
<b>2 marks for AO2 (apply)</b>

The <code>x</code> column fully correct;
The <code>result</code> column fully correct;

If more values are given in any column then max 1 mark.

The correct table is as follows:
<table>
<tr><th><code>x</code></th><th><code>result</code></th></tr>
<tr><td></td><td>0</td></tr>
<tr><td>1</td><td>4</td></tr>
<tr><td>2</td><td>8</td></tr>
<tr><td>3</td><td>12</td></tr>
<tr><td></td><td></td></tr>
</table>
<b>I.</b> Horizontal alignment of values as long as the vertical order of values is correct.
</td><td>2</td></tr>
<tr><td>09</td><td>3</td><td>
<b>Mark is for AO2 (apply)</b>

(The purpose of the algorithms is) to multiply the value in <code>number</code> by 3;

<b>A.</b> The value 4 instead of <code>number</code>.
<b>NE.</b> Multiply two numbers.
</td><td>1</td></tr>
<tr><td>09</td><td>4</td><td>
<b>Mark is for AO2 (apply)</b>

The algorithm in <b>Figure 4</b> uses fewer steps/instructions;

<b>A.</b> The algorithm in <b>Figure 4</b> uses fewer variables;
<b>A.</b> The algorithm in <b>Figure 4</b> has fewer instructions so will take up less memory;
<b>A.</b> The algorithm in <b>Figure 4</b> will execute in less time;
<b>A.</b> Opposite statements for <b>Figure 5</b>.
<b>NE.</b> Reference to number of lines.
</td><td>1</td></tr>
</table>

Qu	Part	Marking guidance	Total marks
10		**6 marks for AO3 (program)**	6

Mark A for assigning user input to a variable (username);
Mark B for assigning user input to a variable (password, the identifier must be different to that used in mark A);
Mark C for using indefinite iteration and including user input within the iteration structure;
Mark D for using a Boolean condition that checks the username is `gower` and the password is `9Fdg3` / the username is `tuff` and the password is `888rG`;
Mark E for using the Boolean `OR` operator for both combinations of username and password, alternatively having sequential `IF` or `ELSE-IF` structures;
Mark F for outputting the string after the iteration structure;

Max 5 marks if the algorithm contains any errors.

I. use of quote marks for usernames or passwords.
I. minor spelling errors for username or passwords.

Example of fully correct answer:

```
REPEAT                                   [part C]
   username ← USERINPUT                  [A, part C]
   password ← USERINPUT                  [B, part C]
UNTIL (username = 'gower' AND            [D, E]
       password = '9Fdg3') OR
      (username = 'tuff' AND
       password = '888rG')
OUTPUT 'access granted'                  [F]
```

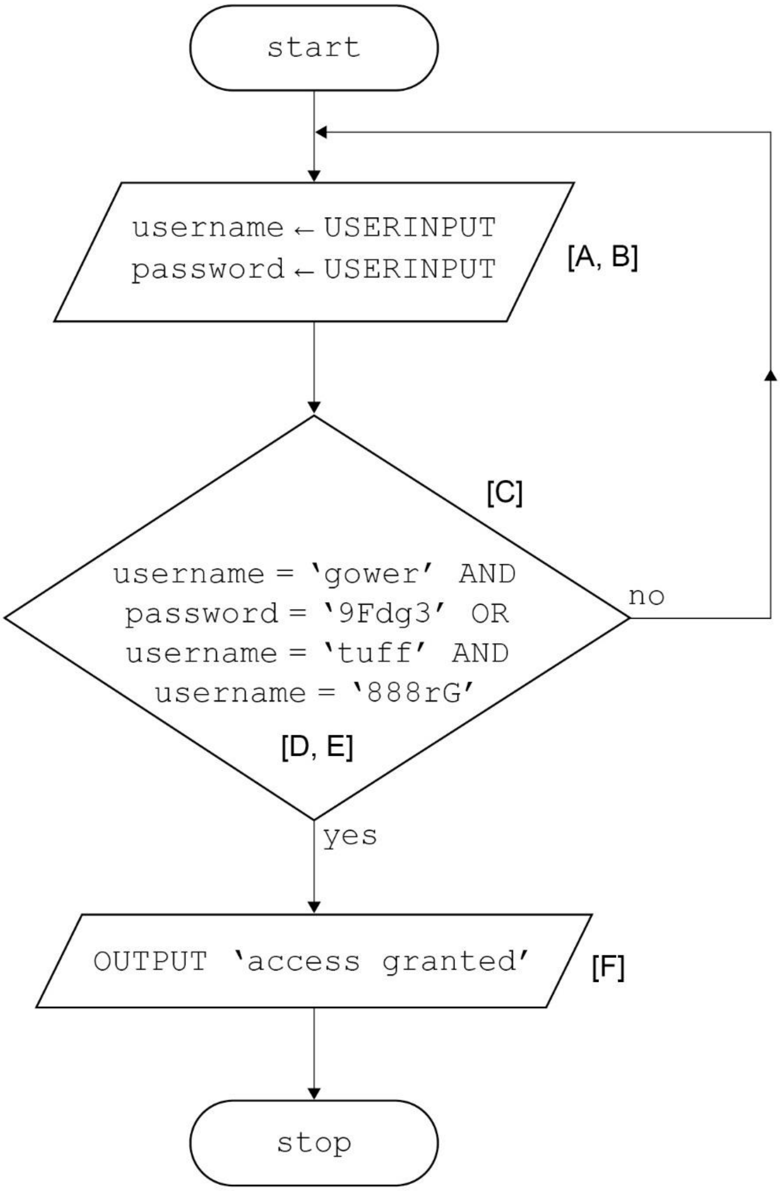
Another example of a fully correct answer:

```
username ← USERINPUT                     [A]
password ← USERINPUT                     [B]
WHILE NOT ((username = 'gower' AND       [D, E, part C]
            password = '9Fdg3') OR
           (username = 'tuff' AND
            password = '888rG'))
   username ← USERINPUT                  [part C]
   password ← USERINPUT                  [part C]
ENDWHILE
OUTPUT 'access granted'                  [F]
```

Another example of a fully correct answer:

```
username ← USERINPUT                      [A]
password ← USERINPUT                      [B]
valid ← false                            [part D]
WHILE NOT valid                         [part C, part D]
   IF (username = 'gower' AND           [part D, E]
       password = '9Fdg3') OR
      (username = 'tuff' AND
       password = '888rG')) THEN
      valid ← true
   ELSE
      username ← USERINPUT               [part C]
      password ← USERINPUT               [part C]
ENDWHILE
OUTPUT 'access granted'                   [F]
```

An example of a fully correct flowchart solution:



Qu	Part	Marking guidance	Total marks
11		**9 marks for AO3 (program)**	9

Mark A for assigning user input to a variable (weekend or weekday);
Mark B for assigning user input to a variable (temperature);
Mark C for using indefinite iteration to repeatedly input the temperature;
Mark D for a Boolean condition used to check the temperature between 20 and 45 inclusive;
Mark E for using selection to set ice creams to be 100 if the temp is between 20 and 30 inclusive;
Mark F for using selection to set ice creams to be 150 if the temp is between 31 and 38 inclusive;
Mark G for using selection to set ice creams to be 120 if the temp is higher than 38;
Mark H for doubling the quantity if it is a weekend (mark A is not required);
Mark I for **always** outputting the estimated number of ice creams;

Max 8 marks if solution contains any errors.

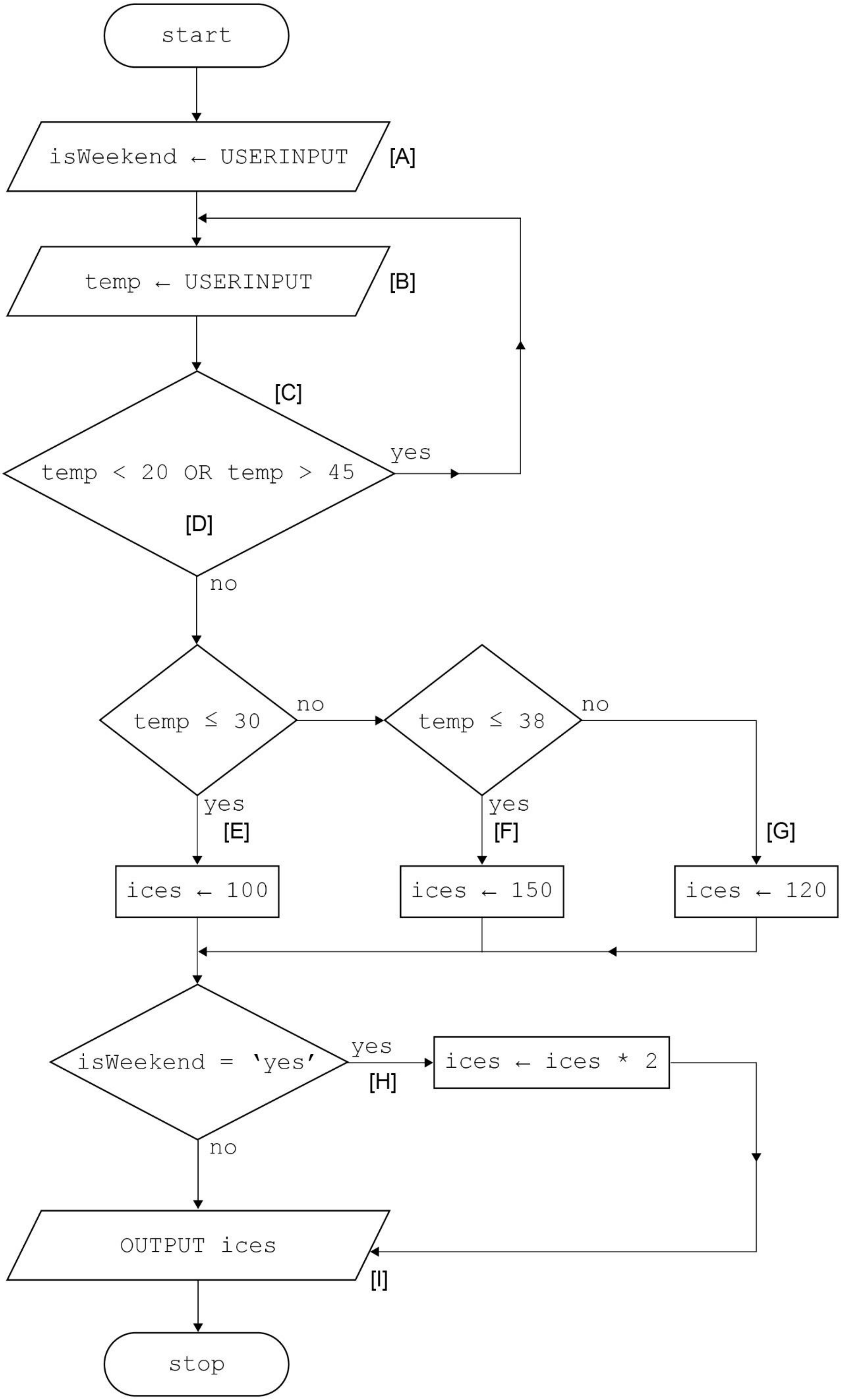
An example of a fully correct solution:

```
isWeekend ← USERINPUT                    [A]
temp ← USERINPUT                         [B]
WHILE temp < 20 OR temp > 45             [part C, D]
   temp ← USERINPUT                      [part C]
ENDWHILE
IF temp ≤ 30 THEN                        [part E]
   ices ← 100                            [part E]
ELSE IF temp ≤ 38 THEN                   [part F]
   ices ← 150                            [part F]
ELSE                                     [part G]
   ices ← 120                            [part G]
ENDIF
IF isWeekend = 'yes' THEN                [part H]
   ices ← ices * 2                        [part H]
ENDIF
OUTPUT ices                              [part I]
```

Another example of a fully correct solution:

```
isWeekend ← USERINPUT                    [A]
DO                                      [part C]
   temp ← USERINPUT                      [B]
WHILE temp < 20 OR temp > 45             [part C, D]
IF temp ≤ 30 THEN                        [part E]
   ices ← 100                            [part E]
ELSE IF temp ≤ 38 THEN                   [part F]
   ices ← 150                            [part F]
ELSE                                     [part G]
   ices ← 120                            [part G]
ENDIF
IF isWeekend = 'yes' THEN                [part H]
   ices ← ices * 2                        [part H]
ENDIF
OUTPUT ices                              [part I]
```

An example of a fully correct flowchart solution:


start
isWeekend ← USERINPUT
[A]
temp ← USERINPUT
[B]
[C]
temp < 20 OR temp > 45
yes
[D]
no
temp ≤ 30
no
temp ≤ 38
no
yes
[E]
yes
[F]
[G]
ices ← 100
ices ← 150
ices ← 120
isWeekend = 'yes'
yes
[H]
ices ← ices * 2
no
OUTPUT ices
[I]
stop

<table>
<tr><th>Qu</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>12</td><td>1</td><td><b>3 marks for AO2 (apply)</b>

1 mark for index 0 set to off;
1 mark for index 2 set to on;
1 mark for index 3 set to off;

<b>Max 2 marks</b> if one error anywhere in the array.
<b>Max 1 mark</b> if two errors anywhere in the array.
<b>0 marks</b> if more than two errors anywhere in the array.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><td>off</td><td>off</td><td>on</td><td>off</td><td>off</td><td>off</td><td>on</td></tr>
</table></td><td>3</td></tr>
<tr><td>12</td><td>2</td><td><b>3 marks for AO2 (apply)</b>

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;

<b>Max 2 marks</b> if one error anywhere in the array.
<b>Max 1 mark</b> if two errors anywhere in the array.
<b>0 marks</b> if more than two errors anywhere in the array.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><td>on</td><td>on</td><td>off</td><td>off</td><td>off</td><td>off</td><td>on</td></tr>
</table></td><td>3</td></tr>
<tr><td>12</td><td>3</td><td><b>3 marks for AO2 (apply)</b>

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;

<b>Max 2 marks</b> if one error anywhere in the array.
<b>Max 1 mark</b> if two errors anywhere in the array.
<b>0 marks</b> if more than two errors anywhere in the array.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><td>on</td><td>off</td><td>on</td><td>on</td><td>off</td><td>off</td><td>on</td></tr>
</table></td><td>3</td></tr>
</table>

Qu	Part	Marking guidance	Total marks

12	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
13	1	**Mark is for AO2 (apply)** **B** Line number 2; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
13	2	**Mark is for AO2 (apply)** **E** `16`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
13	3	**Mark is for AO2 (apply)** **A** Line number 1; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
13	4	**Mark is for AO2 (apply)** **B** Line number 2; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
13	5	**Mark is for AO2 (apply)** **D** This algorithm uses the multiplication operator; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
13	6	**Mark is for AO3 (refine)**	1

C#

A

```
    for (int x = 0; x < 5; x++) {
       Console.Write("Enter a number: ");
       int i = Convert.ToInt32(Console.ReadLine());
       if (i % 2 == 0) {
          Console.WriteLine(i * i);
       }
       else {
          Console.WriteLine(i);
       }
    }
```

Python

A

```
for x in range(0, 5):
    i = int(input("Enter a number: "))
    if i % 2 == 0:
       print(i * i)
    else:
       print(i)
```

VB.NET

C

```
      For x As Integer = 0 To 4
         Console.Write("Enter a number: ")
         Dim i As Integer = Console.ReadLine()
         If i Mod 2 = 0 Then
            Console.WriteLine(i * i)
         Else
            Console.WriteLine(i)
         End If
      Next
```

R. If more than one lozenge shaded

Question	Part	Marking guidance	Total marks
14		**2 marks for AO3 (design), 3 marks for AO3 (program)**	5

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 using meaningful variable names throughout and for using two variables to store the two email address inputs;
Mark B for the use of a selection construct // use of multiple selection constructs;

Program Logic
Mark C for using user input and storing the results in two variables correctly for the first email address and the second email address;
Mark D for a correct expression that checks if the first entered email address is equal to the second entered email address (or not equal to);
Mark E for outputting `Do not match` and `Match` in logically separate places such as the IF and ELSE part of selection, and for outputting the email address if both email addresses match;

A. Any suitable alternative messages.

I. Case
I. Messages or no messages with input statements

Maximum 4 marks if any errors in code.

C# Example 1 (fully correct)
All design marks are achieved (**Marks A and B**)

```
string email1 = Console.ReadLine();            (Part of C)
string email2 = Console.ReadLine();            (Part of C)

if (email1 != email2) {                        (D)
   Console.WriteLine("Do not match");          (Part of E)
}
else {
   Console.WriteLine("Match");                 (Part of E)
   Console.WriteLine(email1);                  (Part of E)
}
```

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

```
string em1 = Console.ReadLine();                    (Part of C)
string em2 = Console.ReadLine();                    (Part of C)

if (em1 == em2)     {                               (D)
   Console.WriteLine("Match");                      (Part of E)
   Console.WriteLine(em2);                          (Part of E)
}
else {
   Console.WriteLine("Do not match");               (Part of E)
}
```

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

```
email1 = input()                                    (Part of C)
email2 = input()                                    (Part of C)

if email1 != email2:                                (D)
   print("Do not match")                            (Part of E)
else:
   print("Match")                                   (Part of E)
   print(email1)                                    (Part of E)
```

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

```
em1 = input()                                       (Part of C)
em2 = input()                                       (Part of C)

if em1 == em2:                                      (D)
   print("Match")                                   (Part of E)
   print(em2)                                       (Part of E)
else:
   print("Do not match")                            (Part of E)
```

Python Example 3 (partially correct – 4 marks)
All design marks are achieved (**Marks A and B**)

```
email1 = input()                                    (Part of C)
email2 = input()                                    (Part of C)

if email1 == email2:                                (D)
   print("Match")
```

VB.NET Example 1 (fully correct)

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

```
Dim email1 As String = Console.ReadLine()     (Part of C)
Dim email2 As String = Console.ReadLine()     (Part of C)

If email1 <> email2 Then                       (D)
   Console.WriteLine("Do not match")          (Part of E)
Else
   Console.WriteLine("Match")                 (Part of E)
   Console.WriteLine(email1)                  (Part of E)
End If
```

VB.NET Example 2 (fully correct)

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

```
Dim em1 As String = Console.ReadLine()        (Part of C)
Dim em2 As String = Console.ReadLine()        (Part of C)

If em1 = em2 Then                             (D)
   Console.WriteLine("Match")                 (Part of E)
   Console.WriteLine(em2)                     (Part of E)
Else
   Console.WriteLine("Do not match")          (Part of E)
End If
```

Question	Part	Marking guidance	Total marks
15		**3 marks for AO3 (design) and 4 marks for AO3 (program)**	7

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 using meaningful variable names throughout;
Mark B for the use of a selection construct;
Mark C for the use of a nested selection construct or multiple conditions;

Program Logic
Mark D for using user input and storing the result in two variables correctly for the items sold **and** years of employment;
Mark E for correct expression that checks the years entered against the criteria for years employed;
Mark F for correct Boolean expressions throughout;
Mark G for outputting correct bonus depending on inputs entered in logically separate places such as IF, ELSE part of selection;

I. Case
I. Prompts

Maximum 6 marks if any errors in code

C# Example 1 (fully correct)

```
Console.Write("How many items?: ");
int items = Convert.ToInt32(Console.ReadLine()); (Part of A, D)
Console.Write("How many years employed?: ");
int years = Convert.ToInt32(Console.ReadLine()); (Part of A, D)
if (years <= 2) {                                (Part of B, E)
   if (items > 100) {                            (Part of C, F)
      Console.WriteLine(items * 2);              (Part of G)
   }
   else {                                        (Part of B, E)
      Console.WriteLine(0);                      (Part of G)
   }
}
else {                                           (Part of B, E)
   Console.WriteLine(items * 10);                (Part of G)
}
```

Python Example 1 (fully correct)

```
items = int(input("How many items?: "))                 (Part of A, D)
years = int(input("How many years employed?: "))        (Part of A, D)
if years <= 2:                                          (Part of B, E)
    if items > 100:                                     (Part of C, F)
        print(items * 2)                                (Part of G)
    else:                                               (Part of C, F)
        print(0)                                        (Part of G)
else:                                                   (Part of B, E)
    print(items * 10)                                   (Part of G)
```

Python Example 2 (fully correct)

```
items = int(input("Enter items: "))                  (Part of A, D)
years = int(input("Enter years employed: "))         (Part of A, D)
if years <= 2 and items > 100:                       (Part of B, C, E, F)
   print(items * 2)                                  (Part of G)
elif years > 2:                                      (Part of B, C, E, F)
   print(items * 10)                                 (Part of G)
else:                                                (Part of B, E)
   print(0)                                          (Part of G)
```

VB.NET Example 1 (fully correct)

```
Console.Write("Enter items: ")
Dim items As Integer = Console.ReadLine()     (Part of A, D)
Console.Write("Enter years: ")
Dim years As Integer = Console.ReadLine()     (Part of A, D)
If years <= 2 And items > 100 Then            (Part of B, C, E, F)
   Console.WriteLine(items * 2)               (Part of G)
ElseIf years > 2 Then                        (Part of B, C, E, F)
   Console.WriteLine(items * 10)              (Part of G)
Else                                          (Part of B, E)
   Console.WriteLine(0)                       (Part of G)
End If
```

Question	Part	Marking guidance	Total marks
16	1	**2 marks for AO3 (design), 2 marks for AO3 (program)**	4

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 idea of inputting a number within the iteration/validation structure;
Mark B for the use of indefinite iteration;

Program Logic
Mark C for using a Boolean condition that checks the lower or upper bound of `position`;
Mark D for using a Boolean condition that checks **BOTH** the lower and upper bounds of `position` correctly;
Marks C and **D** could be one expression eg `0 < position <= 100;`

I. Case
I. Missing prompts

Maximum 3 marks if any errors in code.

C# Example 1 (fully correct)
All design marks are achieved (**Marks A and B**)

```
while (position < 1 || position > 100) {                  (C,D)
   Console.Write("Enter card position: ");
   position = Convert.ToInt32(Console.ReadLine());
}
```

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

```
while (position <= 0 || position >= 101) {                (C,D)
   Console.Write("Enter card position: ");
   position = Convert.ToInt32(Console.ReadLine());
}
```

C# Example 3 (partially correct – 3 marks)
1 design mark achieved (**Mark A**)

```
if (position < 1 || position > 100) {                     (C,D)
   Console.Write("Enter card position: ");
   position = Convert.ToInt32(Console.ReadLine());
}
```

C# Example 4 (partially correct – 3 marks)

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

```
while (position < 1 || position >= 100) {        (Mark C)
   Console.Write("Enter card position: ");
   position = Convert.ToInt32(Console.ReadLine());
}
```

I. Indentation in C#

I. `WriteLine` instead of `Write`

Python Example 1 (fully correct)

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

```
while position < 1 or position > 100:            (C,D)
   position = int(input("Enter card position: "))
```

Python Example 2 (fully correct)

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

```
while position <= 0 or position >= 101:          (C,D)
   position = int(input("Enter card position: "))
```

Python Example 3 (partially correct – 3 marks)

1 design mark achieved (**Mark A**)

```
if position < 1 or position > 100:               (C,D)
   position = int(input("Enter card position: "))
```

Python Example 4 (partially correct – 3 marks)

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

```
while position < 1 or position >= 100:     (C)
   position = int(input("Enter card position: "))
```

VB.NET Example 1 (fully correct)

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

```
While position < 1 Or position > 100             (C,D)
   Console.Write("Enter card position: ")
   position = Console.ReadLine()
End While
```

VB.NET Example 2 (fully correct)

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

```
While position <= 0 Or position >= 101           (C,D)
   Console.Write("Enter card position: ")
   position = Console.ReadLine()
End While
```

VB.NET Example 3 (partially correct – 3 marks)

1 design mark achieved (**Mark A**)

```
If position < 1 Or position > 100 Then           (C,D)
   Console.Write("Enter card position: ")
   position = Console.ReadLine()
End If
```

		<u>VB.NET Example 4 (partially correct – 3 marks)</u>
		All design marks are achieved (**Marks A and B**)
		`Do While position < 1 Or position >= 100` (**Mark C**)
		`Console.Write("Enter card position: ")`
		`position = Convert.ToInt32(Console.ReadLine())`
		`Loop`
		
		I. Indentation in VB.NET
		I. `WriteLine` instead of `Write`	

Question	Part	Marking guidance	Total marks
16	2	**2 marks for AO3 (design), 4 marks for AO3 (program)**	6

Any solution that does not map to the mark scheme refer to lead examiner

<u>Program Design</u>

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 idea of using an iteration structure which attempts to access each element in the `cards` array; // attempts to repeat 100 times;

Mark B for the idea of using a selection structure which attempts to compare two cards;

<u>Program Logic</u>

Mark C for using a loop or similar to correctly iterate through the `cards` array using valid indices that do not go out of range;

Mark D for using correct Boolean conditions that compare values in the `cards` array;

Mark E for correctly checking if there are five values in the `cards` array that are in sequence;

Mark F for setting `gameWon` to `True` in the correct place;

I. Case

Maximum 5 marks if any errors in code.

<u>C# Example 1 (fully correct)</u>

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

```
int count = 1;                                    (Part of E)
for (int i = 0; i < 99; i++) {                    (C)
     if (cards[i] + 1 == cards[i+1]) {            (D, Part of E)
          count = count + 1;                      (Part of E)
          if (count == 5) {                       (Part F)
               gameWon = true;                    (Part F)
          }
     }
     else {
          count = 1;                              (Part of E)
     }
}
```

C# Example 2 (fully correct)

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

```
int count = 1;                          (Part of E)
int i = 0;                              (Part of C)
while (i < 99) {                        (Part of C)
     if (cards[i] + 1 == cards[i+1]) {  (D, Part of E)
          count = count + 1;            (Part of E)
          if (count == 5) {             (Part F)
               gameWon = true;          (Part F)
          }
     }
     else {
          count = 1;                    (Part of E)
     }
     i = i + 1;                         (Part of C)
}
```

I. Indentation in C#

Python Example 1 (fully correct)

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

```
count = 1                               (Part of E)
for i in range(99):                     (C)
  if cards[i] + 1 == cards[i + 1]:      (D, Part of E)
    count = count + 1                   (Part of E)
    if count == 5:                      (Part F)
      gameWon = True                    (Part F)
  else:
    count = 1                           (Part of E)
```

Python Example 2 (fully correct)

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

```
count = 0                               (Part of E)
i = 0                                   (Part of C)
while i < len(cards) - 1:               (Part of C)
  if cards[i] + 1 == cards[i + 1]:      (D, Part of E)
    count = count + 1                   (Part of E)
    if count == 4:                      (Part F)
      gameWon = True                    (Part F)
  else:
    count = 0                           (Part of E)
  i = i + 1                             (Part of C)
```

Python Example 3 (fully correct)

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

```
gameWon = False                                  (Part F)
for i in range(96):                              (C)
  count = 1                                      (Part of E)
  for j in range(1, 5):                          (Part of D)
    if cards[i + j] - 1 == cards[i + j - 1]:     (Part of D)
                                                 (Part of E)
      count += 1                                 (Part of E)
  if count == 5:                                 (Part F)
    gameWon = True                               (Part F)
```

VB.NET Example 1 (fully correct)

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

```
Dim count As Integer = 1                         (Part of E)
For i = 0 To 98                                  (C)
     If cards(i) + 1 = cards(i+1) Then           (D, Part of E)
          count = count + 1                      (Part of E)
          If count = 5 Then                      (Part F)
               gameWon = True                    (Part F)
          End If
     Else
          count = 1                              (Part of E)
     End If
Next
```

VB.NET Example 2 (fully correct)

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

```
Dim count As Integer = 0                         (Part of E)
Dim i As Integer = 0                             (Part of C)
While i < 99                                     (Part of C)
     If cards(i) + 1 = cards(i+1) Then           (D, Part of E)
          count = count + 1                      (Part of E)
          If count = 4 Then                      (Part F)
               gameWon = True                    (Part F)
          End If
     Else
          count = 0                              (Part of E)
     End If
     i = i + 1                                   (Part of C)
End While
```

I. Indentation in VB.NET

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>17</td><td>1</td><td>

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
```

</td><td>4</td></tr>
</table>

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
17	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 **<u>Program Design</u>** **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); **<u>Program Logic</u>** **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
```

<u>VB.NET Example 2 (fully correct)</u>
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
18	**1**	**Mark is for AO2 (apply)** **A** Line number 2; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
18	**2**	**Mark is for AO2 (apply)** **A** `0`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
18	**3**	**Mark is for AO2 (apply)** **C** 4; **R.** if more than one lozenge shaded	1

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td><b>19</b></td><td></td><td>
<b>3 marks for AO2 (apply)</b>
<table>
<tr><th>a</th><th>b</th><th>c</th></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>2</td></tr>
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>2</td><td>3</td><td>5</td></tr>
</table>
<b>1 mark</b> for correct first row;
<b>1 mark</b> for correct second row;
<b>1 mark</b> for correct third and fourth rows;

<b>Maximum 2 marks</b> if any errors

<b>I.</b> different rows used as long as the order within columns is clear
<b>I.</b> duplicate values on consecutive rows within a column

<b>Note to examiners:</b> Check vertically as well as horizontally for the effect of duplicate values.
</td><td>3</td></tr>
</table>

Question	Part	Marking guidance	Total marks
20		**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 inputting the number in the group and storing in a variable; **Mark B** for using selection; **Program Logic** **Mark C** for correctly multiplying the number in the group by 15; **Mark D** for using an appropriate correct Boolean condition(s) that covers all paths through the problem, eg >=6 // >5 or equivalent; **Mark E** for using an appropriate method to reduce the total charge by £5; **Mark F** for outputting the final total in a logical place; **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.	6

C# Example 1 (fully correct)

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

```
int group = Convert.ToInt32(Console.ReadLine());
int total = group * 15;                              (C)
if (group >= 6) {                                    (D)
   total = total - 5;                                (E)
}
Console.WriteLine(total);                            (F)
```

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

Python Example 1 (fully correct)

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

```
group = int(input())
total = group * 15                                   (C)
if group >= 6:                                       (D)
  total = total - 5                                  (E)
print(total)                                         (F)
```

VB.NET Example 1 (fully correct)

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

```
Dim group As Integer = Console.ReadLine()
Dim total As Integer = group * 15                          (C)
If (group >= 6) Then                                       (D)
   total = total - 5                                       (E)
End If
Console.WriteLine(total)                                   (F)
```

I. Indentation in VB.NET

A. `Write` in place of `WriteLine`

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>21</td><td>1</td><td>
<b>5 marks for AO2 (apply)</b>

<b>1 mark</b> for <code>count</code> column correct;
<b>1 mark</b> for column <code>i</code> correct;
<b>1 mark</b> for the first Natalie row, including <code>j</code> and <code>result</code> correct – not including <code>i</code> and <code>count</code>;
<b>1 mark</b> for the second Natalie row, including <code>j</code> and <code>result</code> correct – not including <code>i</code> and <code>count</code>;
<b>1 mark</b> for all of Alex and Roshana rows correct as for Natalie above;

<table>
<tr><th>count</th><th>i</th><th>person</th><th>j</th><th>result</th></tr>
<tr><td>0</td><td>0</td><td>Natalie</td><td>0</td><td>78</td></tr>
<tr><td>1</td><td></td><td></td><td>1</td><td>81</td></tr>
<tr><td>2</td><td>1</td><td>Alex</td><td>0</td><td>27</td></tr>
<tr><td>3</td><td></td><td></td><td>1</td><td>51</td></tr>
<tr><td>4</td><td>2</td><td>Roshana</td><td>0</td><td>52</td></tr>
<tr><td>5</td><td></td><td></td><td>1</td><td>55</td></tr>
<tr><td>6</td><td></td><td></td><td></td><td></td></tr>
</table>

<b>I.</b> different rows used as long as the order within columns is clear
<b>I.</b> duplicate values on consecutive rows within a column
<b>I.</b> quotes used around letters (person column)
<b>I.</b> minor spelling mistakes in the person column
</td><td>5</td></tr>
</table>

Question	Part	Marking guidance	Total marks
21	2	**Mark is for AO2 (apply)** **C** Change line number 7 to: `FOR j ← 0 TO 2` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
22		**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 using the variable `check` within their own code; **Mark B** for using selection or equivalent to check the grid references; **Program Logic** **Mark C** for correctly using an appropriate technique (slicing/indexing/`substring` function) with correct syntax to extract the left **and** right characters of input // for correctly comparing all nine possible valid grid references; **Mark D** for using **one** appropriate correct Boolean condition, eg =”A” // =”2” or equivalent; **Mark E** for having **all** the appropriate correct Boolean conditions to check the letters and numbers **AND** for `check` being set appropriately in all cases; **Mark F** for outputting an appropriate message in a logically appropriate location if their checks have failed; **Maximum 5 marks** if any errors in code. **I.** Case **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.	6

C# Example 1 (fully correct)

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

```
bool check = false;
while (check == false) {
   string square = "";
   while (square.Length != 2) {
      Console.Write("Enter grid reference: ");
      square = Console.ReadLine();
      square = square.ToUpper();
   }
   char letter = square[0];                              (Part of C,
   char number = square[1];                               Part of C)
   if ((letter == 'A' || letter == 'B' || letter               (D)
== 'C') && (number == '1' || number == '2' ||                  (E)
number == '3'))
   {
      check = true;
   }
   else
   {
      Console.WriteLine("Not valid, try again.");              (F)
   }
}
```

I. Indentation in C#

I. Duplicate } at the end of the program (as if student has missed the bracket in the writing lines)

A. use of double quotes for **Mark E**

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

```
check = False
while check == False:
   square = ""
   while len(square) != 2:
      square = input("Enter grid reference: ")
      square = square.upper()

   letter = square[0]                                    (Part of C,
   number = square[1]                                     Part of C)

   if letter in "ABC" and number in "123":                  (D)(E)
      check = True
   else:
      print("Not valid, try again. ")                          (F)
```

A. use of single quotes for **Mark E**

VB.NET Example 1 (fully correct)

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

```
Dim check As Boolean = False
While check = False
   Dim square As String = ""
   While square.Length <> 2
      Console.Write("Enter grid reference: ")
      square = Console.ReadLine()
      square = square.ToUpper()
   End While
   Dim letter As String = square(0)                     (Part of C,
   Dim number As String = square(1)                      Part of C)

   If (letter = "A" Or letter = "B" Or letter =                 (D)
"C") And (number = "1" Or number = "2" Or number                (E)
= "3") Then
      check = True
   Else
      Console.WriteLine("Not valid, try again. ")               (F)
   End If
End While
```

I. Indentation in VB.NET

I. Duplicate `End While` at the end of the program (as if student has missed the bracket in the writing lines)

A. `Write` in place of `WriteLine`

A. use of single quotes for **Mark E**

VB.NET Example 2 (fully correct)

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

```
Dim check As Boolean = False
While check = False
   Dim square As String = ""
   While square.Length <> 2
      Console.Write("Enter grid reference: ")
      square = Console.ReadLine()
      square = square.ToUpper()
   End While
   Dim letter As String = square.substring(0,1)      (Part of C,
   Dim number As String = square.substring(1,1)       Part of C)

   If (letter = "A" Or letter = "B" Or letter =            (D)
"C") And (number = "1" Or number = "2" Or number           (E)
= "3") Then
      check = True
   Else
      Console.WriteLine("Not valid, try again. ")          (F)
   End If
End While
```

I. Indentation in VB.NET

I. Duplicate `End While` at the end of the program (as if student has missed the bracket in the writing lines)

A. `Write` in place of `WriteLine`

A. use of single quotes for **Mark E**

Question	Part	Marking guidance	Total marks
23		**3 marks for AO2 (apply)** L1 `1;` L2 `i;` L3 `method;` **Note to Examiners:** If the student has re-written the entire line and added in the correct missing item, award the mark.	3

Question	Part	Marking guidance	Total marks
24		**3 marks for AO3 (design), 5 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 storing a user input in a variable with a meaningful name; **Mark B** for using an iteration structure which attempts to pay the bill; **Mark C** for using a selection structure with `ELSE` / `ELSEIF` // use of multiple selection constructs; **Program Logic** **Mark D** for getting the user input for the total amount of the bill (outside the loop) **AND** deducting a payment towards the bill (within the loop); **A.** if there is no loop and both elements are present in the right order. **Mark E** for a mechanism which will correctly terminate the iteration structure, **in all situations**, when the bill is fully paid; **Mark F** for two conditions. One which checks / handles if the amount left to pay is `0` (or less, ie bill is paid), **AND** one which checks if the amount left to pay is less than `0` (for tip); **Mark G** for outputting in an appropriate place `Tip is` and the tip as a number; **R.** if tip is outputted when the amount left to pay is not less than zero **Mark H** for outputting `Bill paid` **and** the amount left to pay in logically appropriate places; **Maximum 7 marks** if any errors in code. **I.** Case **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. **I.** Messages or no messages with input statements	8

C# Example 1 (fully correct)

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

```
bool billPaid = false;                                 (Part of E)
decimal total = Convert.ToDecimal                      (Part of D)
(Console.ReadLine());
while (billPaid == false)                              (Part of E)
{
   decimal partPayment = Convert.ToDecimal             (Part of D)
(Console.ReadLine());
   total = total - partPayment;                        (Part of D)
   Console.WriteLine(total);                           (Part of H)
   if (total == 0)                                     (Part of F)
   {
      Console.WriteLine("Bill paid");                  (Part of H)
      billPaid = true;                                 (Part of E)
   } else if (total < 0)                            (Part of F, G)
   {
      Console.WriteLine("Tip is " + -total);           (Part of G)
      billPaid = true;                                 (Part of E)
   }
}
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

```
total = float(input())                                 (Part of D)
billPaid = False                                       (Part of E)
while billPaid == False:                               (Part of E)
   partPayment = float(input())                        (Part of D)
   total = round(total – partPayment, 2)               (Part of D)
   print(total)                                        (Part of H)
   if total == 0:                                      (Part of F)
      print("Bill paid")                               (Part of H)
      billPaid = True                                  (Part of E)
   elif total < 0:                                  (Part of F, G)
      print(f"Tip is: {-total}")                       (Part of G)
      billPaid = True                                  (Part of E)
```

A. without rounding / `round()` statements

VB.NET Example 1 (fully correct)

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

```
Dim billPaid As Boolean = False                                  (Part of E)
Dim total As Decimal = Console.ReadLine()                        (Part of D)
While billPaid = False

   Dim partPayment As Decimal = Console.ReadLine()               (Part of D)
   total = total - partPayment
   Console.WriteLine(total)                                      (Part of D)
   If total = 0 Then                                             (Part of H)
      Console.WriteLine("Bill paid")                             (Part of F)
      billPaid = True                                            (Part of H)
   ElseIf total < 0                                              (Part of E)
      Console.WriteLine("Tip is " & -total)                   (Part of F, G)
      billPaid = True                                            (Part of G)
   End If                                                        (Part of E)
End While
```

I. Indentation in VB.NET

A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
25		**4 marks for AO3 (design), 7 marks for AO3 (program)** Any solution that does not map to the mark scheme refer to lead examiner **Note to Examiners:** For marks **E** and **J** be careful not to penalise the same error twice. For example, if they have used 6 instead of 7 in mark E and then 21 instead of 22 in mark J apply a **DPT** **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 attempting to randomly generate **two** numbers; **Mark B** for use of selection to check the current score against 21; **Mark C** for using iteration to keep rolling the dice; **Mark D** for outputting the dice rolls in appropriate places; **Program Logic** **Mark E** for generating **two** random numbers between 1 and 6 inclusive; **Mark F** for correctly adding the **two** dice values cumulatively to the previous score; **Mark G** for a loop that terminates if the current score is less than 21 **and** player chooses not to roll again; **Mark H** for a correct mechanism to end the game if the player has a score greater than or equal to 21; **Mark I** for a selection statement which correctly checks if the player has lost (final score is greater than 21) **OR** won (final score is 21); **Mark J** for generating a random number between 15 and 21 inclusive in a logically correct place **AND** checking if the result is greater than the final score; **Mark K** for **at least one** correct set of messages output in appropriate places to show whether the user has won or lost; **A.** yes/y, no/n or any other appropriate equivalents **Maximum 10 marks** if any errors in code. **I.** Case **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. **I.** Messages or no messages with input statements	11

C# Example 1 (fully correct)

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

```
Random r = new Random();
int score = 0;
string rollAgain = "yes";

while (rollAgain == "yes")                                  (C, Part of G,
                                                                Part of H)
{
   int dice1 = r.Next(1, 7);                               (Part of A,E)
   int dice2 = r.Next(1, 7);                               (Part of A,E)
   score = score + dice1 + dice2;                                    (F)
   Console.WriteLine("Roll 1: " + dice1);                   (Part of D)
   Console.WriteLine("Roll 2: " + dice2);                   (Part of D)
   Console.WriteLine("Current score: " + score);            (Part of D)
   if (score < 21)                                          (Part of G)
   {
      rollAgain = Console.ReadLine();                       (Part of G)
   } else
   {
      rollAgain = "no";                                     (Part of H)
   }
}
if (score > 21)                                              (Part of I)
{
   Console.WriteLine("You lost! ");                         (Part of K)
} else if (score == 21)                                      (Part of I)
{
   Console.WriteLine("You won! ");                          (Part of K)
} else                                                       (Part of I)
{
   if (r.Next(15, 22) > score)                                      (J)
   {
      Console.WriteLine("You lost! ");                      (Part of K)
   } else
   {
      Console.WriteLine("You won! ");                       (Part of K)
   }
}
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

```
import random
score = 0
rollAgain = "yes"

while rollAgain == "yes":                                (C, Part of G,
                                                              Part of H)
   dice1 = random.randrange(1, 7)                        (Part of A,E)
   dice2 = random.randrange(1, 7)                        (Part of A,E)
   score = score + dice1 + dice2                                   (F)
   print(f"Roll 1: {dice1}")                                       (D)
   print(f"Roll 2: {dice2}")
   print(f"Current score: {score}")
   if score < 21:                                          (Part of G)
      rollAgain = input()                                  (Part of G)
   else:
      rollAgain = "no"                                     (Part of H)
if score > 21:                                             (Part of I)
   print("You lost! ")                                     (Part of K)
elif score == 21:                                          (Part of I)
   print("You won! ")                                      (Part of K)
else:                                                      (Part of I)
   if random.randrange(15,22) > score:                             (J)
      print("You lost!")                                   (Part of K)
   else:
      print("You won! ")                                   (Part of K)
```

A. `random.randint(1, 6)`

A. `random.randint(15, 21)`

<u>VB.NET Example 1 (fully correct)</u>

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

```
Dim r As Random = New Random()
Dim score As Integer
Dim rollAgain As String = "yes"
Dim dice1, dice2 As Integer

While rollAgain = "yes"                                  (C, Part of G,
                                                              Part of H)
   dice1 = r.Next(1, 7)                                   (Part of A,E)
   dice2 = r.Next(1, 7)                                   (Part of A,E)
   score = score + dice1 + dice2                                   (F)
   Console.WriteLine("Roll 1: " & dice1)                   (Part of D)
   Console.WriteLine("Roll 2: " & dice2)                   (Part of D)
   Console.WriteLine("Current score: " & score)            (Part of D)
   If score < 21 Then                                      (Part of G)
      rollAgain = Console.ReadLine()                       (Part of G)
   Else
      rollAgain = "no"                                     (Part of H)
   End If
End While

If score > 21 Then                                         (Part of I)
   Console.WriteLine("You lost! ")                         (Part of K)
ElseIf score = 21 Then                                     (Part of I)
   Console.WriteLine("You won! ")                          (Part of K)
Else                                                       (Part of I)
   If r.Next(15, 22) > score Then                                  (J)
      Console.WriteLine("You lost! ")                      (Part of K)
   Else
      Console.WriteLine("You won! ")                       (Part of K)
   End If
End If
```

I. Indentation in VB.NET

A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
26	1	**Mark is for AO2 (apply)** **A** Line number 2; **R.** If more than one lozenge shaded	1
26	2	**Mark is for AO2 (apply)** **C** Line number 11; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
26	3	**Mark is for AO2 (apply)** **A** 1 subroutine call; **R.** If more than one lozenge shaded	1
26	4	**Mark is for AO2 (apply)** **B** String; **R.** If more than one lozenge shaded	1
26	5	**Mark is for AO2 (apply)** 2//twice//two;	1

27		**5 marks for AO3 (program)**
		1 mark for each correct item in the correct location.
		Python
		`num1 = int(input("Enter a number: "))`
		`num2 = int (input("Enter a second number: "))`
		`if num1 > num2:`
		`    print(" num1 is bigger.")`
		`elif num1 < num2:`
		`    print(" num2 is bigger.")`
		`else:`
		`    print("The numbers are equal.")`
		I. Case of response
		R. if any spelling mistakes
		C#
		`int num1;`
		`int num2;`
		`Console.WriteLine("Enter a number: ");`
		`num1 = int.Parse(Console.ReadLine());`
		`Console.WriteLine("Enter another number: ");`
		`num2 = int.Parse(Console.ReadLine());`
			5

```
if (num1 > num2)
{
    Console.WriteLine("   num1      is bigger.");
}
else
if (num1 <          num2)
{
    Console.WriteLine("   num2      is bigger.");
}
else
{
    Console.WriteLine("The numbers are equal.");
}
```

I. Case of response
R. if any spelling mistakes

VB.Net

```
Dim num1 As Integer
Dim num2 As Integer

Console.Write("Enter a number: ")

num1 = Console.ReadLine()

Console.Write("Enter another number: ")

num2 = Console.ReadLine()
If num1 > num2 Then
    Console.WriteLine(" num1      is bigger.")
ElseIf num1 <           num2 Then
    Console.WriteLine(" num2      is bigger.")
Else
    Console.WriteLine("The numbers are equal.")
End If
```

I. Case of response
R. if any spelling mistakes

Question	Part	Marking guidance	Total marks
28		**2 marks for AO3 (design) and 5 marks for AO3 (program)**	7

Program Design

Mark A for using meaningful variable names throughout (even if logic is incorrect);
Mark B for using suitable data types throughout (distance can be real or integer, passengers must be integer);

Program Logic

Mark C for getting user input for the distance in an appropriate place;
Mark D for getting user input for the number of passengers in an appropriate place;
Mark E for a fare that correctly charges £2 per passenger;
Mark F for a fare that correctly charges £1.50 for every kilometre;
Mark G for outputting the correct final fare;

I. Case of program code

Maximum 6 marks if any errors in code.

Python Example 1 (fully correct)
Mark A awarded.

```
distance = float(input())                   (Part of B, C)
passengers = int(input())                   (Part of B, D)
fare = 2 * passengers                       (E)
fare = fare + (1.5 * distance)              (F)
print(fare)                                 (G)
```

C# Example (fully correct)
Mark A awarded.

```
int passengers;                                  (Part of B)
double distance, fare;                           (Part of B)
distance = double.Parse(Console.ReadLine());     (C)
passengers = int.Parse(Console.ReadLine());      (D)
fare = 2 * passengers;                           (E)
fare = fare + (1.5 * distance);                  (F)
Console.WriteLine(fare);                         (G)
```

I. indentation in C#

VB Example (fully correct)
Marks A, **B** awarded.

```
Dim distance, fare As Double                     (Part of B)
Dim passengers As Integer                        (Part of B)
distance = Console.ReadLine()                    (C)
passengers = Console.ReadLine()                  (D)
```

```
fare = 2 * passengers                                  (E)
fare = fare + (1.5 * distance)                         (F)
Console.WriteLine(fare)                                (G)
```

I. indentation in VB.NET

Python Example 2 (partially correct – 6 marks)
Mark A awarded**. Mark B** not awarded because float conversion missing.

```
dist = input()              (C but NOT B)
pass = int(input())         (Part of B, D)
fare = 2 * pass             (E)
fare = 1.5 * dist           (F)
print fare                  (G – still awarded even though
                            parentheses missing in print command
                            as logic still clear)
```

Question	Part	Marking guidance	Total marks
29		**2 marks for AO3 (design), 3 marks for AO3 (program)**	5

Program Design

Mark A for the use of a selection construct (even if the logic is incorrect);
Mark B for the correct, consistent use of meaningful variable names throughout (even if the code would not work);

Program Logic

Mark C for using user input and storing the result in a variable correctly;
Mark D for a correct expression that checks if the entered password is `'secret'` (even if the syntax is incorrect);
Mark E for outputting `Welcome` and `Not welcome` correctly in logically separate places such as the `IF` and `ELSE` part of selection;

I. Case of output strings for **Mark E**, but spelling must be correct.
I. Case of program code

Maximum 4 marks if any errors in code.

Python Example 1 (fully correct)

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

```
password = input()                                        (C)
if password == 'secret':                                  (D)
    print('Welcome')                                      (Part of E)
else:
    print('Not welcome')                                  (Part of E)
```

C# Example (fully correct)

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

```
string password;
password = Console.ReadLine();                            (C)
if (password == "secret")                                 (D)
{
   Console.WriteLine("Welcome");                          (Part of E)
}
else
{
   Console.WriteLine("Not welcome");                      (Part of E)
}
```

I. indentation in C#

VB Example (fully correct)

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

```
Dim password As String
password = Console.ReadLine()                             (C)
```

```
If (password = "secret") Then                                  (D)
   Console.WriteLine("Welcome")                                (Part of E)
Else
   Console.WriteLine("Not welcome")                            (Part of E)
End If
```

I. indentation in VB.NET

<u>Python Example 2 (partially correct – 4 marks)</u>
Mark A is awarded. **Mark B** is not awarded.

```
p = input()                                                    (C)
if p == 'secret'                                               (D)
    print('Welcome')                                           (Part of E)
    else:
    print('Not welcome')                                       (Part of E)
```

Question	Part	Marking guidance	Total marks
30	1	**Mark is for AO2 (apply)** Boolean//bool; **I.** Case	1

Question	Part	Marking guidance	Total marks
30	2	**2 marks for AO2 (apply)** (The identifier) `swapsMade` describes the purpose//role//meaning of the variable; this makes the algorithm easier to understand//maintain//follow; or (The identifier) `s` does not describe the purpose//role//meaning of the variable; this makes the algorithm harder to understand//maintain//follow;	2

Question	Part	Marking guidance	Total marks
30	3	**Mark is for AO2 (apply)** **A** The algorithm uses a named constant; **R.** If more than one lozenge shaded	1

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>30</td><td>4</td><td>
<b>6 marks for AO2 (apply)</b>
1 mark for column <code>arr[0]</code> correct;
1 mark for column <code>arr[1]</code> correct;
1 mark for column <code>arr[2]</code> correct <b>only if</b> <code>arr[0]</code> and <code>arr[1]</code> are correct;
1 mark for <code>swapsMade</code> column correct;
1 mark for <code>i</code> column correct;
1 mark for <code>t</code> column correct;
<table>
<tr><th colspan="3">Arr</th><th rowspan="2">swapsMade</th><th rowspan="2">i</th><th rowspan="2">t</th></tr>
<tr><th>0</th><th>1</th><th>2</th></tr>
<tr><td>4</td><td>1</td><td>6</td><td>false</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>true</td><td>0</td><td></td></tr>
<tr><td>1</td><td>4</td><td></td><td>false</td><td></td><td>4</td></tr>
<tr><td></td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td></td></tr>
<tr><td></td><td></td><td></td><td>true</td><td>0</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td></td></tr>
</table>
<b>I.</b> different rows used as long as the order within columns is clear
<b>I.</b> duplicate values on consecutive rows within a column
</td><td>6</td></tr>
</table>

Question	Part	Marking guidance	Total marks
31		**3 marks for AO3 (design), 4 marks for AO3 (program)**	7

Program Design
Mark A for the idea of inputting a character and checking if it is lower case (even if the code would not work);
Mark B for the use of a selection construct (even if the logic is incorrect);
Mark C for the correct, consistent use of meaningful variable names throughout (even if the code would not work);

Program Logic
Mark D for using user input correctly;
Mark E for storing the result of user input in a variable correctly;
Mark F for a correct expression/method that checks if the character is lowercase;
Mark G for outputting `LOWER` and `NOT LOWER` correctly in logically separate places such as the `IF` and `ELSE` part of selection;

I. Case of output strings for **Mark G**, but spelling must be correct.
I. Case of program code

Maximum 6 marks if any errors in code.

Python Example 1 (fully correct)
All design marks are achieved (**Marks A, B and C**)

```
character = input()                                 (D,E)
if (character >= 'a') and (character <= 'z'):      (F)
   print('LOWER')                                   (Part of G)
else:
   print('NOT LOWER')                               (Part of G)
```

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

```
character = input()                                 (D,E)
if character.islower():                             (F)
    print('LOWER')                                  (Part of G)
else:
    print('NOT LOWER')                              (Part of G)
```

C# Example (fully correct)

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

```
char character = (char)Console.Read();                (D,E)
if (Char.IsLower(character))                          (F)
{
 Console.WriteLine("LOWER");                          (Part of G)
}
else
{
 Console.WriteLine("NOT LOWER");                      (Part of G)
}
```

I. indentation in C#

VB.Net Example (fully correct)

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

```
Dim character As Char
character = Console.ReadLine()                        (D,E)
If (Char.IsLower(character)) Then                     (F)
  Console.WriteLine("LOWER")                          (Part of G)
Else
  Console.WriteLine("NOT LOWER")                      (Part of G)
End If
```

I. indentation in VB.NET

Python Example 3 (partially correct – 5 marks)

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

```
character = input()                                   (D,E)
if (character > ′a′) or (character < ′z′):            (NOT F)
   print('NOT LOWER')                                 (NOT G)
else:
   print('LOWER')                                     (NOT G)
```

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>32</td><td>1</td><td>

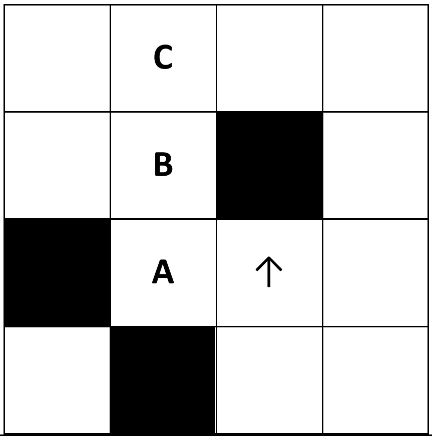
3 marks for AO2 (apply)

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**;

<table>
<tr><td></td><td></td><td>C</td><td></td></tr>
<tr><td></td><td></td><td>B</td><td>A</td></tr>
<tr><td></td><td></td><td></td><td>A</td></tr>
<tr><td></td><td></td><td></td><td>↑</td></tr>
</table>

</td><td>3</td></tr>
</table>

Question	Part	Marking guidance	Total marks
32	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

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>33</td><td>1</td><td><b>3 marks for AO2 (apply)</b>

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;

<table><tr><td>Column 0</td><td>Column 1</td><td>Column 2</td></tr><tr><td>_____</td><td>C</td><td>A
B</td></tr></table></td><td>3</td></tr>
<tr><td>33</td><td>2</td><td><b>3 marks for AO2 (apply)</b>

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);

<table><tr><td>Column 0</td><td>Column 1</td><td>Column 2</td></tr><tr><td>A</td><td>C</td><td>B</td></tr></table></td><td>3</td></tr>
</table>

Question	Part	Marking guidance	Total marks
33	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 **<u>Example 1</u>** `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**.) **<u>Example 2</u>** `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)
```

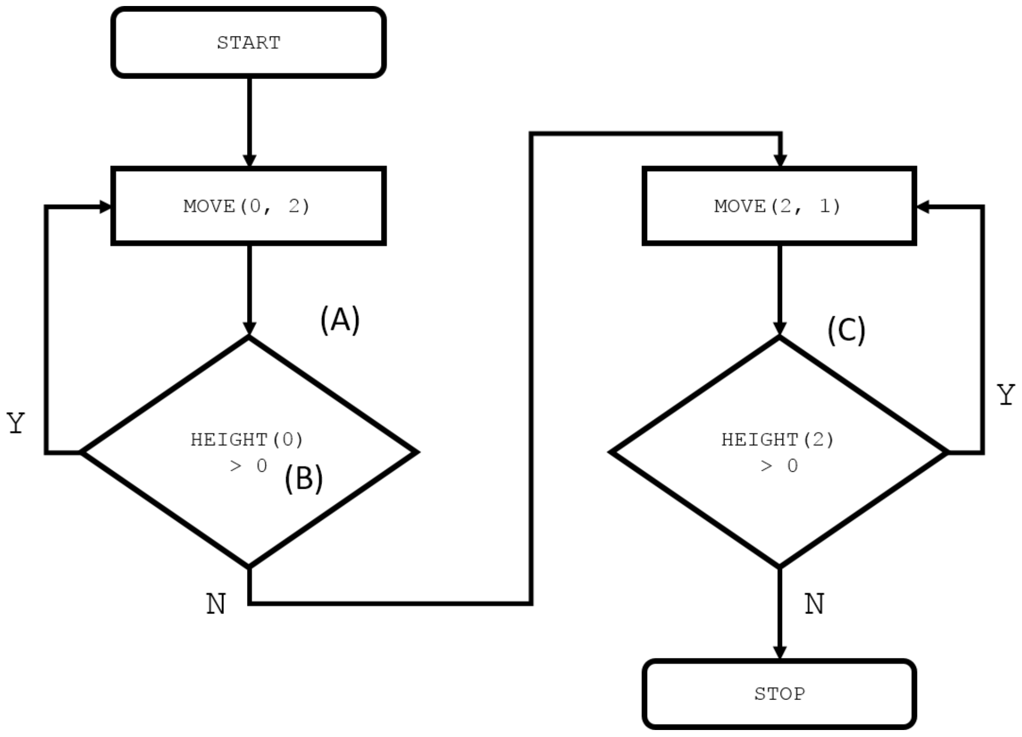
(MOVE and HEIGHT are used correctly throughout so **D**.)

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
34		**1 mark for AO3 (refine)** B; **R.** if more than 1 lozenge shaded	1

35		**4 marks for AO3 (refine)**
		<u>Program Logic</u>
		Mark A: for using a selection structure with else part **or** two selection structures (even if the syntax is incorrect)
		Mark B: for correct condition(s) in selection statement(s) (even if the syntax is incorrect)
		Mark C: for statement that subtracts two from `odd` under the correct conditions (even if the syntax is incorrect)
		Mark D: for `odd` being output and doing one of adding or subtracting two but not both each time loop repeats (even if the syntax is incorrect)
		I. while loop from question if included in answer
		I. case of program code
		Maximum 3 marks if any errors in code.
		<u>Python Example 1 (fully correct)</u>
		`print(odd)` (**Part of D**)
		`if number < 0` (**A, B**)
		`  odd = odd – 2` (**C, Part of D**)
		`else:`
		`  odd = odd + 2` (**Part of D**)
		<u>C# Example (fully correct)</u>
		`Console.WriteLine(odd);` (**Part of D**)
		`if (number < 0)` (**A, B**)
		`{`
		`  odd = odd - 2;` (**C, Part of D**)
		`}`
		`else`
		`{`
		`  odd = odd + 2;` (**Part of D**)
		`}`
		I. indentation in C#
			4

VB.Net Example (fully correct)

```
Console.WriteLine(odd)                (Part of D)
If number < 0 Then                    (A, B)
  odd = odd - 2                       (C, Part of D)
Else
  odd = odd + 2                       (Part of D)
End If
```

I. indentation in VB.Net

Python Example 2 (partially correct – 3 marks)

```
print(odd)                            (Part of D)
if number != 0                        (A, NOT B)
  odd = odd - 2                       (C, Part of D)
else:
  odd = odd + 2                       (Part of D)
```

Question	Part	Marking guidance	Total marks
36	**1**	**Mark is for AO2 (apply)** **D** `value ← LEN(film);` **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
36	**2**	**Mark is for AO2 (apply)** `POSITION(film, letter);` **I.** Case **R.** Quotes	1

Question	Part	Marking guidance	Total marks
36	**3**	**Mark is for AO2 (apply)** **C** integer; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
36	**4**	**Mark is for AO1 (understanding)** When a value is given to a variable; // When a variable is assigned a value;	1

Question	Part	Marking guidance	Total marks
36	5	**2 marks for AO3 (program)**	2

Program Logic

Mark A for using user input and storing the result in a variable;

Mark B for displaying `You entered` followed by the name of the film entered by the user in the appropriate place;

I. Case
I. Indentation
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 1 mark 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.

C# Example 1 (fully correct)

```
film = Console.ReadLine();                         (A)
Console.WriteLine("You entered " + film);          (B)
```

A. `Write` in place of `WriteLine`

C# Example 2 (fully correct)

```
film = Console.ReadLine();                    (A)
Console.Write("You entered ");                (Part B)
Console.WriteLine(film);                      (Part B)
```

Python Example 1 (fully correct)

```
film = input()                                (A)
print("You entered", film)                    (B)
```

Python Example 2 (fully correct)

```
film = input()                                (A)
print("You entered " + film)                  (B)
```

Python Example 3 (fully correct)

```
film = input()                                (A)
print(f"You entered {film}")                  (B)
```

VB.NET Example 1 (fully correct)

```
film = Console.ReadLine()                     (A)
Console.WriteLine("You entered " & film)      (B)
```

A. `Write` in place of `WriteLine`

VB.NET Example 2 (fully correct)

```
film = Console.ReadLine()                     (A)
Console.WriteLine("You entered " + film)      (B)
```

A. `Write` in place of `WriteLine`

VB.NET Example 3 (fully correct)

```
film = Console.ReadLine()                   (A)
Console.Write("You entered ")               (Part B)
Console.WriteLine(film)                     (Part B)
```

A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
37	1	**Mark is for AO2 (apply)** **B** Line number 2; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
37	2	**Mark is for AO2 (apply)** **A** `Almost`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
37	3	**Mark is for AO2 (apply)** **C** `20`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks

37	4	**Mark is for AO2 (apply)**
		1 mark for either of the following:
		`IF num ≤ 1 OR num > 20 THEN`
		//
		`IF num < 2 OR num > 20 THEN`
		I. Case
		A. answers that use an alternative style of pseudo-code
			1

Question	Part	Marking guidance	Total marks
37	5	**Mark is for AO2 (apply)** 16 / 17 / 18 / 19; **R.** If more than one value given and one of the values is not correct. **A.** If more than one value given and all are correct.	1

Question	Part	Marking guidance	Total marks
38		**2 marks for AO3 (design), 5 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 using meaningful variable names throughout; **Mark B** for the use of a selection structure to check the total mark is less than zero or equivalent; **Program Logic** **Mark C** for using user input and storing the result in a numeric variable for the number of late essays; **Mark D** for correctly summing the total marks using the contents of variables `e1`, `e2` and `e3` in all circumstances **and** either reducing the total by `10` **or** halving the total mark **Mark E** for **two** expressions / a **combined** expression that checks the number of late essays correctly; **Mark F** for a correct expression(s) that prevents the total mark being less than 0 (eg by resetting the total mark to 0 or preventing it going below 0); **Mark G** for outputting total mark in the correct place; **R.** if any required calculations are performed on total mark after the last time the variable is output. **Maximum 6 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.	7

C# Example 1 (fully correct)

```
lateCount = Convert.ToInt32(Console.ReadLine());   (C)
total = e1 + e2 + e3;                               (Part D)
if (lateCount == 1)                                 (Part E)
{
   total = total - 10;                              (Part D)
}
if (lateCount > 1)                                  (Part E)
{
   total = total / 2;                               (Part D)
}
if (total < 0)                                      (Part F)
{
   total = 0;                                       (Part F)
}
Console.WriteLine(total);                           (G)
```

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

Python Example 1 (fully correct)

```
lateCount = int(input())                 (C)
total = e1 + e2 + e3                     (Part D)
if lateCount == 1:                       (Part E)
   total = total - 10                    (Part D)
if lateCount > 1:                        (Part E)
   total = total / 2                     (Part D)
if total < 0:                            (Part F)
   total = 0                             (Part F)
print(total)                             (G)
```

Python Example 2 (fully correct)

```
lateCount = int(input())                 (C)
total = e1 + e2 + e3                     (Part D)
if lateCount == 1 and total >= 10:       (Part E,
                                         Part F)
   total = total - 10                    (Part D)
elif lateCount == 1 and total < 10:      (Part E,
                                         Part F)
   total = 0                             (Part F)
elif lateCount > 1:                      (Part E)
   total = total * 0.5                   (Part D)
print(total)                             (G)
```

VB.NET Example 1 (fully correct)

```
lateCount = Console.ReadLine()                          (C)
total = e1 + e2 + e3                                    (Part D)
If lateCount = 1 Then                                   (Part E)
   total = total - 10                                   (Part D)
End If
If lateCount > 1 Then                                   (Part E)
   total = total / 2                                    (Part D)
End If
If total < 0 Then                                       (Part F)
   total = 0                                            (Part F)
End If
Console.WriteLine(total)                                (G)
```

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

Question	Part	Marking guidance	Total marks
39		**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 idea of using concatenation to create the stock code; **Program Logic** **Mark B** for using user input correctly for the `sweetID`, `sweetName` and `brand`; **A.** similar distinct/meaningful variable names. **Mark C** for correctly creating each part of the stock code; **A.** if stock code is output instead of assigned to variable. **Mark D** for assigning the stock code / three string variables representing `sweetID`, `sweetName` and `brand` correctly to the variable `code` (even if the generated stock code is not correct); **R.** any other variable name for `code` **Maximum 3 marks** if any errors. **I.** `print` / `Console.WriteLine` statements **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 **R.** commas used to show concatenation	4

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.

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

```
sweetID = Console.ReadLine();                       (Part B)
sweetName = Console.ReadLine();                     (Part B)
brand = Console.ReadLine();                         (Part B)
code = sweetID + sweetName[0] + sweetName[1]        (C, D)
+ brand[0];
```

A. `sweetID.Substring(0, 2)`
I. Indentation

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

```
code = Console.ReadLine() +                         (B,C,D)
Console.ReadLine().Substring(0, 2) +
Console.ReadLine()[0];
```

I. Indentation

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

```
sweetID = input()                                   (Part B)
sweetName = input()                                 (Part B)
brand = input()                                     (Part B)
code = sweetID + sweetName[0] + sweetName[1]        (C, D)
+ brand[0]
```

A. `sweetID[0:2]`

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

```
code = input() + input()[0:2] + input()[0]        (B, C, D)
```

Python Example 3 (partially correct – 3 marks)
Design mark is achieved (**Mark A**)

```
code = input() + input() + input()                  (B, D)
```

VB.NET Example 1 (fully correct)

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

```
sweetID = Console.ReadLine()                                 (Part B)
sweetName = Console.ReadLine()                               (Part B)
brand = Console.ReadLine()                                   (Part B)
code = sweetID + sweetName(0) + sweetName(1)                 (C, D)
+ brand(0)
```

A. `sweetID.Substring(0, 2)`

I. Indentation

VB.NET Example 2 (fully correct)

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

```
code = Console.ReadLine() &                                  (B, C, D)
Console.ReadLine().Substring(0, 2) &
Console.ReadLine()(0)
```

I. Indentation

Question	Part	Marking guidance	Total marks
40	**1**	**Mark is for AO1 (understanding)** **D** An organised collection of values; **R.** If more than one lozenge shaded	1

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>40</td><td>2</td><td>

3 marks for AO2 (apply)

3 marks if all **four** are correct:
- **Book** on line 1
- **author** on line 3
- **Real** on line 4
- **Book** on line 7

2 marks if any **three** are correct
1 mark if any **two** are correct

<table>
<tr><td>1</td><td>RECORD <b>Book</b></td></tr>
<tr><td>2</td><td>   bookName : String</td></tr>
<tr><td>3</td><td>   <b>author</b> : String</td></tr>
<tr><td>4</td><td>   price : <b>Real</b></td></tr>
<tr><td>5</td><td>ENDRECORD</td></tr>
<tr><td>6</td><td>B1 ← Book("The Book Thief", "M Zusak", 9.99)</td></tr>
<tr><td>7</td><td>B2 ← <b>Book</b>("Divergent", "V Roth", 6.55)</td></tr>
</table>

I. Case

</td><td>3</td></tr>
</table>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>40</td><td>3</td><td>

3 marks for AO2 (apply)

```
IF B1.price > B2.price THEN
   OUTPUT B1.bookName
ELSEIF B1.price < B2.price THEN
   OUTPUT B2.bookName
ELSE
   OUTPUT "Neither"
ENDIF
```

1 mark for correctly using a selection structure with multiple conditions // use of multiple selection structures to compare B1 and B2 in some way (even if Boolean conditions incorrect);

1 mark for correct Boolean conditions throughout to compare the prices;

1 mark for displaying the correct output in each case;

Max 2 marks if any errors

I. Case
A. Pseudo-code statements written using different syntax as long as the logic is still correct.

</td><td>3</td></tr>
</table>

Question	Part	Marking guidance	Total marks
41	1	**Mark is for AO2 (apply)** `11;`	1

Question	Part	Marking guidance	Total marks
41	2	**Mark is for AO2 (apply)** `17;`	1

Question	Part	Marking guidance	Total marks
42		**2 marks for AO3 (design), 5 marks for AO3 (program)** **<u>Program Design</u>** **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 using meaningful variable names throughout; **Mark B** for the use of an indefinite iteration structure that exists within their language, for validation of the inputs; **<u>Program Logic</u>** **Mark C** for using user input and storing the result in two variables correctly for the username and password; **Mark D** for using correct Boolean expressions to check if the username and password entered matches at least **one** of the valid pairs; **A.** if the **only** error is missing quotes around string values **Mark E** for using correct Boolean expressions to check if the username and password entered matches **both of** the valid pairs; **R.** if **any** quotes missing around string values **Mark F** for allowing the user to enter the username and password again in an appropriate place (even if the Boolean expression is not correct); **DPT.** If mark C not awarded due to incorrect syntax. **Mark G** for displaying `Access granted` or `Access denied` in the appropriate places; **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 6 marks** if any errors in code.	7

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.

C# Example 1 (fully correct)
All design marks are achieved (**Marks A and B**)

```
username = Console.ReadLine();                          (Part C)
password = Console.ReadLine();                          (Part C)
while ((username != "Yusuf5" || password                (D, E)
!= "33kk") && (username != "Mary80" ||
password != "af5r"))
{
    Console.WriteLine("Access denied");                 (Part G)
    username = Console.ReadLine();                      (Part F)
    password = Console.ReadLine();                      (Part F)
}
Console.WriteLine ("Access granted");                   (Part G)
```

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

C# Example 2 (fully correct)

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

```
valid = false;
do {
   username = Console.ReadLine();          (Part C, Part F)
   password = Console.ReadLine();          (Part C, Part F)
   if (username == "Yusuf5" &&             (Part D, Part E)
password == "33kk") {
      valid = true;                        (Part D)
   }
   else if (username == "Mary80" &&        (Part D, Part E)
password == "af5r") {
       valid = true;                       (Part D)
   }
   if (!valid) {                           (Part G)
      Console.WriteLine("Access            (Part G)
denied");
   }
} while (!valid);
Console.WriteLine ("Access granted");      (Part G)
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

C# Example 3 (fully correct)

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

```
do {
   username = Console.ReadLine();          (Part C, Part F)
   password = Console.ReadLine();          (Part C, Part F)
   access = (username == "Yusuf5" &&       (D, E)
password == "33kk") || (username ==
"Mary80" && password == "af5r");
   if (access == false) {
      Console.WriteLine("Access            (Part G)
   }
} while (!access);
Console.WriteLine ("Access                 (Part G)
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

<u>Python Example 1 (fully correct)</u>
All design marks are achieved (**Marks A and B**)

```
username = input()                                    (Part C)
password = input()                                    (Part C)
while (username != "Yusuf5" or password !=            (D, E)
"33kk") and (username != "Mary80" or
password != "af5r"):
    print("Access denied")                            (Part G)
    username = input()                                (Part F)
    password = input()                                (Part F)
print("Access granted")                               (Part G)
```

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

```
access = False                                        (Part F)
while access == False:                                (Part F)
   username = input()                                 (Part C)
   password = input()                                 (Part C)
   if (username == "Yusuf5" and password              (D, E)
== "33kk") or (username == "Mary80" and
password == "af5r"):
      print("Access granted")                         (Part G)
      access = True
   else:
      print("Access denied")                          (Part G)
```

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

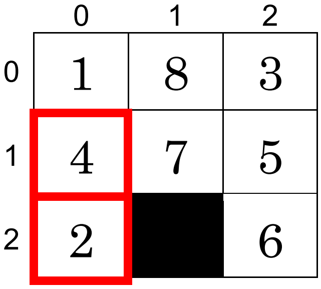
```
username = Console.ReadLine()                          (Part C)
password = Console.ReadLine()                          (Part C)
While (username <> "Yusuf5" Or password <>             (D, E)
"33kk") And (username <> "Mary80" Or
password <> "af5r")
    Console.WriteLine("Access denied")                 (Part G)
    username = Console.ReadLine()                      (Part F)
    password = Console.ReadLine()                      (Part F)
End While
Console.WriteLine ("Access granted")                   (Part G)
```

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**)

```
valid = False
Do
   username = Console.ReadLine()                      (Part C,
                                                       Part F)
   password = Console.ReadLine()                      (Part C,
                                                       Part F)
   If username = "Yusuf5" And password =              (Part D,
"33kk" Then                                            Part E)
      valid = True                                     (Part D)
   ElseIf username = "Mary80" And password             (Part D,
= "af5r" Then                                          Part E)
       valid = True                                    (Part D)
   End If
   If Not valid Then                                   (Part G)
      Console.WriteLine("Access denied")               (Part G)
   End If
Loop Until valid
Console.WriteLine ("Access granted")                   (Part G)
```

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

Question	Part	Marking guidance	Total marks
43	**1**	**2 marks for AO2 (apply)**  0 1 2 0 1 8 3 1 4 7 5 2 2 ■ 6 **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
43	**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
43	**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

43	**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
43	**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
43	6	**1 mark for AO3 (design), 3 marks for AO3 (program)** **<u>Program Design</u>** **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; **<u>Program Logic</u>** **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)) {            (Part B, Part C)

   if (getTile(0, 1) + 1 == getTile(0, 2)) {         (Part B, Part C)

      Console.WriteLine("Yes");                      (Part D)

   }

   else {

      Console.WriteLine("No");                       (Part D)

   }

}

else {

   Console.WriteLine("No");                          (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).

<u>C# Example 2 (fully correct)</u>
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)
      Console.WriteLine("Yes");                    (Part D)

   }

   else {

      Console.WriteLine("No");                     (Part D)

   }

}

else {

    Console.WriteLine("No");                       (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).

<u>C# Example 3 (fully correct)</u>
Design mark is achieved (**Mark A**)

```
if ((getTile(0, 1) - getTile(0, 0) == 1) &&        (Part B,
(getTile(0, 2) - getTile(0, 1) == 1)) {            Part C)

   Console.WriteLine("Yes");                       (Part D)

}

else {

   Console.WriteLine("No");                        (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     (Part B,
getTile(0, 2) - getTile(0, 1) == 1:           Part C)
   print("Yes")                               (Part D)
else:
   print("No")                                (Part D)
```

VB.NET Example 1 (fully correct)

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

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

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          (Part B,
                                                    Part C)
   If getTile(0, 0) + 2 = getTile(0, 2) Then       (Part B,
                                                    Part C)
      Console.WriteLine("Yes")                      (Part D)
   Else
      Console.WriteLine("No")                       (Part D)
   End If
Else
   Console.WriteLine("No")                          (Part D)
End If
```

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

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

Question	Part	Marking guidance	Total marks
43	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
44		**2 marks for AO3 (design), 6 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 which outputs `Bad move`; **Mark B** for the use of a nested selection structure // a selection structure with multiple conditions // use of multiple selection structures **Program Logic** **Mark C** for correctly inputting a move in an appropriate place within the `while` loop; **Mark D** for correctly checking the input for a move is either `1` or `2`; **I.** data validation attempts **Mark E** for adding the input value for a move to `pos` once per move; **Mark F** for resetting `pos` to `0` if the move takes a player beyond the end of the row; **A.** if the index used could go out of range. **Mark G** for a condition equivalent to `row() == "X"` that checks for the character `X` in `row` and resets `pos` to `0` if appropriate; **I.** missing or incorrect index number on `row`. **A.** if the index used could go out of range. **Mark H** for the correct use of indices to access the elements in the array `row` and the index does not go out of range; **Maximum 7 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.	8

C# Example 1 (fully correct)

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

```
   move =
Convert.ToInt32(Console.ReadLine());          (C)
   if (move == 1 || move == 2) {              (D)
      pos += move;                            (E)
   }
   if (pos > lastPos) {                       (Part F)
      pos = 0;                                (Part F)
      Console.WriteLine("Bad move");
   }
   else if (row[pos] == "X") {                (Part G, H)
      pos = 0;                                (Part G)
      Console.WriteLine("Bad move");
   }
```

I. Indentation

A. `Write` in place of `WriteLine`

C# Example 2 (7 marks)

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

No **Mark D** as program also adds numbers other than `1` or `2` to `pos`.

```
   move =
Convert.ToInt32(Console.ReadLine());          (C)
   if (pos + move > lastPos || row[pos +      (Part F,
move] == "X") {                                Part G, H)
      Console.WriteLine("Bad move");
      pos = 0;                                 (Part F,
                                               Part G)
   }
   else {
      pos = pos + move;                        (E)
   }
}
```

I. Indentation

A. `Write` in place of `WriteLine`

C# Example 3 (fully correct)

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

```
   move =                                              (C)
Convert.ToInt32(Console.ReadLine());
   if (move == 1) {                                    (Part D)
      if (row[pos + 1] == "X") {                       (Part G)
          pos = 0;                                     (Part G)
          Console.WriteLine("Bad move");
      }
      else {
          pos = pos + 1;                               (Part E)
      }
   }
   if (move == 2) {                                    (Part D)
      if (pos + move > lastPos || row[pos +            (Part F,
2] == "X") {                                           Part G,
                                                       H)
          pos = 0;                                     (Part F)
          Console.WriteLine("Bad move");
      }
      else {
          pos = pos + 2;                               (Part E)
      }
   }
```

I. Indentation

A. `Write` in place of `WriteLine`

<u>**Python Example 1 (fully correct)**</u>
All design marks are achieved (**Marks A and B**)

```
move = int(input())                          (C)
if move == 1 or move == 2:                   (D)
   pos += move                               (E)
if pos > lastPos:                            (Part F)
   pos = 0                                   (Part F)
   print("Bad move")
elif row[pos] == "X":                        (Part G, H)
   pos = 0                                   (Part G)
   print("Bad move")
```

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

```
move = int(input())                          (C)
if move == 1:                                (Part D)
   if row[pos + 1] == 'X':                   (Part G)
       print("Bad move")
       pos = 0                               (Part G)
   else:
       pos = pos + 1                         (Part E)
if move == 2:                                (Part D)
   if pos + 2 > lastPos or row[pos           (Part F, Part
+ 2] == 'X':                                 G, Part H)
       print("Bad move")
       pos = 0                               (Part F, Part
                                             G)
   else:
       pos = pos + 2                         (Part E)
```

<u>Python Example 3 (7 marks)</u>

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

No **Mark D** as program also adds numbers other than `1` or `2` to `pos`.

```
   move = int(input())                          (C)

   if pos + move > lastPos or row[pos +         (Part F,
move] == 'X':                                   Part G, H)

      print("Bad move")

      pos = 0                                   (Part F,
                                                Part G)
   else:

      pos = pos + move                          (E)
```

<u>VB.NET Example 1 (fully correct)</u>

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

```
   move =
Convert.ToInt32(Console.ReadLine())             (C)

   If move = 1 Or move = 2 Then                 (D)

      pos += move                               (E)
   End If
   If pos > lastPos Then                        (Part F)
      pos = 0                                   (Part F)
      Console.WriteLine("Bad move")
   ElseIf row(pos) = "X" Then                   (Part G, H)
      pos = 0                                   (Part G)
      Console.WriteLine("Bad move")
   End If
```

I. Indentation

A. `Write` in place of `WriteLine`

<u>VB.NET Example 2 (7 marks)</u>

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

```
    move = Convert.ToInt32(Console.ReadLine())              (C)
    If move = 1 Then                                        (Part D)
        If row(pos + 1) = "X" Then                          (Part G)
            Console.WriteLine("Bad move")
            pos = 0                                         (Part G)
        Else
            pos = pos + 1                                   (Part E)
        End If
    End If
    If move = 2 Then                                        (Part D)
        If pos + move > lastPos Or row(pos + 2) = "X" Then  (Part F, Part G)
            Console.WriteLine("Bad move")
            pos = 0                                         (Part F, Part G)
        Else
            pos = pos + 2                                   (Part E)
        End If
    End If
```

I. Indentation

A. `Write` in place of `WriteLine`

VB.NET Example 3 (6 marks)

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

No **Mark D** as program also adds numbers other than `1` or `2` to `pos`.

```
   move = Convert.ToInt32(Console.ReadLine())                         (C)

   If pos + move > lastPos Or row(pos + move) = "X" Then              (Part F, Part G)

      Console.WriteLine("Bad move")

      pos = 0                                                         (Part F, Part G)

   Else

      pos = pos + move                                                (E)

   End If
```

I. Indentation

A. `Write` in place of `WriteLine`