AQA

GCSE

COMPUTER SCIENCE

8525/1A, 8525/1B, 8525/1C

Paper 1 Computational thinking and programming skills

Mark scheme

June 2024

Version 1.0 Final

246G8525/1/MS

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

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

No student should be disadvantaged on the basis of their gender identity and/or how they refer to the gender identity of others in their exam responses.

A consistent use of 'they/them' as a singular and pronouns beyond 'she/her' or 'he/him' will be credited in exam responses in line with existing mark scheme criteria.

Further copies of this mark scheme are available from aqa.org.uk

The following annotation is used in the mark scheme:

; - means a single mark

// - means alternative response

/ - means an alternative word or sub-phrase

A - means acceptable creditworthy answer. Also used to denote a valid answer that goes beyond the expectations of the GCSE syllabus.

R - means reject answer as not creditworthy

NE - means not enough

I - means ignore

DPT - in some questions a specific error made by a candidate, if repeated, could result in the candidate failing to gain more than one mark. The DPT label indicates that this mistake should only result in a candidate losing one mark on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated.

Copyright information

AQA retains the copyright on all its publications. However, registered schools/colleges for AQA are permitted to copy material from this booklet for their own internal use, with the following important exception: AQA cannot give permission to schools/colleges to photocopy any material that is acknowledged to a third party even for internal use within the centre.

Copyright © 2024 AQA and its licensors. All rights reserved.

Note to Examiners

In the real-world minor syntax errors are often identified and flagged by the development environment. To reflect this, all responses in a high-level programming language will assess a candidate's ability to create an answer using precise programming commands/instructions but will avoid penalising them for minor errors in syntax.

When marking program code, examiners must take account of the different rules between the languages and only consider how the syntax affects the logic flow of the program. If the syntax is not perfect but the logic flow is unaffected then the response should not be penalised.

The case of all program code written by students is to be ignored for the purposes of marking. This is because it is not always clear which case has been used depending on the style and quality of handwriting used.

Examiners must ensure they follow the mark scheme instructions exactly. If an examiner is unsure as to whether a given response is worthy of the marks they must escalate the question to their team leader.

Level of response marking instructions

Level of response mark schemes are broken down into levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are marks in each level.

Before you apply the mark scheme to a student's answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student's answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity, you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level, you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level, ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The descriptors on how to allocate marks can help with this. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student's answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner's mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the Indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.

Question	Part	Marking guidance	Total marks
01	1	**Mark is for AO2 (apply)** **D** `value ← LEN(film);` **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	2	**Mark is for AO2 (apply)** `POSITION(film, letter);` **I.** Case **R.** Quotes	1

Question	Part	Marking guidance	Total marks
01	3	**Mark is for AO2 (apply)** **C** integer; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	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
01	5	**2 marks for AO3 (program)** **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)**	2

		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
02	**1**	**Mark is for AO2 (apply)** **B** Line number 2; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
02	**2**	**Mark is for AO2 (apply)** **A** `Almost`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
02	3	**Mark is for AO2 (apply)** **C** `20`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks

02	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
02	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

<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>
2 marks for AO3 (refine)

Mark A for using the correct variable name and assigning a numeric value to it;
Mark B for using correct code to generate a random number;

Maximum 1 mark 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

<u>C#</u>
<code>randomNumber = rGen.Next(1, 101) ;;</code>

<u>Python</u>
<code>randomNumber = random.randrange(1, 101) ;;</code>

<u>VB.NET</u>
<code>randomNumber = rGen.Next(1, 101) ;;</code>
</td><td>2</td></tr>
</table>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>03</td><td>2</td><td>
2 marks for AO2 (apply)

<table>
<tr><th>Test number</th><th>Test type</th><th>Test data</th><th>Expected result</th></tr>
<tr><td>1</td><td>Erroneous</td><td><code>150</code></td><td><code>Invalid number</code></td></tr>
<tr><td>2</td><td>Boundary</td><td><code>0</code>/<code>1</code>/<code>100</code>/<code>101</code></td><td><code>Invalid number</code> / <code>Valid number entered</code></td></tr>
<tr><td>3</td><td>Normal</td><td>Anything between <code>1</code> and <code>100</code> inclusive</td><td><code>Valid number entered</code></td></tr>
</table>
1 mark for correct Erroneous Expected result and Normal Test data;

1 mark for correct Boundary Test data and Boundary Expected result - if 0 or 101 given as Boundary test data then expected result must be <code>Invalid number</code>, if 1 or 100 given as Boundary test data then Expected result must be <code>Valid number entered</code>;
</td><td>2</td></tr>
</table>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>03</td><td>3</td><td>2 marks for AO2 (apply)

<table>
<tr><th>Error type</th><th>Description</th></tr>
<tr><td>Syntax error</td><td><code>whil</code>/<code>while</code> is spelt incorrectly;</td></tr>
<tr><td>Logic error</td><td><code>>=100</code> should be <code>>100</code>;</td></tr>
</table>

A. Accept answers that just include the incorrect / corrected code
R. If the same response is given for both errors</td><td>2</td></tr>
</table>

Question	Part	Marking guidance	Total marks
04	**1**	**Mark is for AO1 (recall)** Removing unnecessary detail/information/data from the problem/task; **R.** references to code/programs	1

Question	Part	Marking guidance	Total marks
04	**2**	**Mark is for AO1 (recall)** Decomposition; **I.** minor spelling errors	1

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

<table>
<tr><th>Input value for
<code>numberOfGuests</code></th><th>Input value for
<code>numberOfRooms</code></th><th>Output</th></tr>
<tr><td><code>50</code></td><td><code>30</code></td><td>3200;</td></tr>
<tr><td><code>20</code></td><td><code>10</code></td><td>1125;</td></tr>
<tr><td><code>500</code></td><td><code>5</code></td><td>1500;</td></tr>
</table>

DPT. Quotes around output values
I. commas in output values</td><td>3</td></tr>
</table>

Question	Part	Marking guidance	Total marks
06		**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
07		**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

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>08</td><td></td><td>
6 marks for AO2 (apply)

1 mark for the <code>i</code> column correct;

1 mark for the first value in the <code>daysTotal</code> column correct;
I. preceding zeroes

1 mark for the rest of <code>daysTotal</code> column correct;

1 mark for the second value of <code>weeks[0]</code> column correct;

1 mark for the rest of <code>weeks</code> columns correct;

1 mark for the correct total of <code>weeks[0]</code>, <code>weeks[1]</code> and <code>weeks[2]</code> in the final column and no other value;
I. preceding zeroes
A. follow through value as long as the total is correct for the three final values the student has written in the <code>weeks</code> columns.

Maximum of 5 marks if any errors.

<table>
<tr><th rowspan="2">i</th><th rowspan="2">daysTotal</th><th colspan="3">weeks</th><th>weeksTotal</th></tr>
<tr><th>[0]</th><th>[1]</th><th>[2]</th><th></th></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td>0</td><td></td></tr>
<tr><td>0</td><td>30</td><td>4</td><td>0</td><td>0</td><td></td></tr>
<tr><td>1</td><td>48</td><td>4</td><td>6</td><td>0</td><td></td></tr>
<tr><td>2</td><td>16</td><td>4</td><td>6</td><td>2</td><td></td></tr>
<tr><td></td><td></td><td colspan="3"></td><td>12</td></tr>
</table>

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

Question	Part	Marking guidance	Total marks
09	**1**	**Mark is for AO1 (understanding)** **D** An organised collection of values; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
09	2	**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 1 `RECORD Book` 2 `  bookName : String` 3 `  author : String` 4 `  price : Real` 5 `ENDRECORD` 6 `B1 ← Book("The Book Thief", "M Zusak", 9.99)` 7 `B2 ← Book("Divergent", "V Roth", 6.55)` **I.** Case	3

Question	Part	Marking guidance	Total marks

09	3	**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.
			3

Question	Part	Marking guidance	Total marks
10	1	**Mark is for AO2 (apply)** `11`;	1

Question	Part	Marking guidance	Total marks
10	2	**Mark is for AO2 (apply)** `17`;	1

Question	Part	Marking guidance	Total marks
11		**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 an indefinite iteration structure that exists within their language, for validation of the inputs; **Program Logic** **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`

Python Example 1 (fully correct)
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)
```

Python Example 2 (fully correct)
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**)

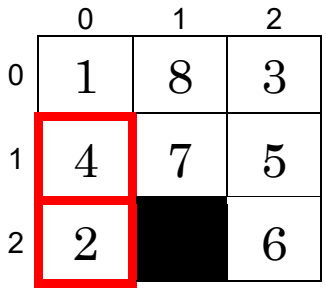
```
username = Console.ReadLine()                          (Part C)
password = Console.ReadLine()                          (Part C)
While (username <> "Yusuf5" Or password <>             (D, E)
"33kk") And (username <> "Mary80" Or
password <> "af5r")
    Console.WriteLine("Access denied")                 (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, Part E)
"33kk" Then
      valid = True                                     (Part D)
   ElseIf username = "Mary80" And password             (Part D, Part E)
= "af5r" Then
       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
12	1	**2 marks for AO2 (apply)**  **1 mark** for 4 in the correct position; **1 mark** for 2 in the correct position; **Maximum 1 mark** if any errors. **A**. `0` instead of blank space or any other sensible indicator for the blank space. **A**. unaffected cell contents not shown as long as it is clear which is the blank space. **A**. answers written on **Figure 15** if board is left blank.	2

Question	Part	Marking guidance	Total marks
12	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
12	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

12	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
12	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
12	6	**1 mark for AO3 (design), 3 marks for AO3 (program)** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for the use of a selection structure with multiple conditions // use of multiple selection structures // an iteration structure containing one selection structure; **Program Logic** **Mark B** for correctly checking three consecutive values in `getTile` (even if the wrong row/column); **Mark C** for fully correct indices used in `getTile` for the first row; **Mark D** for a structure that would output either `Yes` or `No` correctly in all circumstances, but never both; **A.** if conditions are not fully correct **I**. Case **I.** Messages or no messages with input statements **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect **Maximum 3 marks** if any errors in code. **Note to examiners** In C#/VB.NET examples, explicit variable declarations are not shown. Refer to the specific language type issues section of the appropriate Marking guidance document. Any correct variable declarations in student answers should be accepted.	4

C# Example 1 (fully correct)

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

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

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

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

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

(Part D) — `Console.WriteLine("No");`

(Part D) — `Console.WriteLine("No");`

I. Indentation in C#

A. `Write` in place of `WriteLine`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
if getTile(0, 1) - getTile(0, 0) == 1 and      (Part B, Part C)
getTile(0, 2) - getTile(0, 1) == 1:
   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, Part C)
getTile(0, 2) - getTile(0, 1) = 1 Then
   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
12	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
13		**3 marks for AO1 (understanding)** **Maximum 3 marks** from the following: • start at the beginning/end of a list/array; • iterates sequentially through the list; • compare the contents of **each** position with the data being searched; • if it matches, the item has been found (and the search ends); • if the end/beginning of the list/array is reached without finding the search term then item is not in the list/array; **Max 2 marks** if any errors such as referring to binary search or having the items in order	3

Question	Part	Marking guidance	Total marks
14	1	**Mark is for AO1 (recall)** **Maximum 1 mark** from: • They are only accessible within the subroutine; • They only exist/use memory while the subroutine is executing; • They have limited scope; **A.** Can have the same identifier as other variables outside of the subroutine	1

Question	Part	Marking guidance	Total marks
14	2	**2 marks for AO3 (design), 4 marks for AO3 (program)** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for the use of a definite iteration structure, or similar, that exists within their language to carry out the requirements of the task; **Mark B** for the use of a selection structure to check visitor numbers; **Program Logic** **Mark C** for correctly defining the subroutine and parameter; **Mark D** for accepting user input multiple times as per the parameter or equivalent, representing the number of days; **R.** if iteration syntax or boundaries are not fully correct **Mark E** for adding one to a counter variable inside a selection structure under the correct conditions, which has been appropriately initialised (to `0`); **Mark F** for returning the counter value calculated within the subroutine; **Maximum 5 marks** if any errors in code. **I.** Case **I.** Messages or no messages with input statements **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect **Note to examiners** In C#/VB.NET examples, explicit variable declarations are not shown. Refer to the specific language type issues section of the appropriate Marking guidance document. Any correct variable declarations in student answers should be accepted.	6

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

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

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

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

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

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

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

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

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

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

VB.NET Example 2 (fully correct)

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

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

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

Question	Part	Marking guidance	Total marks
15		**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 +
move] == "X") {                                         (Part F, 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 =
Convert.ToInt32(Console.ReadLine());                    (C)
    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 +
2] == "X") {                                            (Part F, 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`

Python Example 1 (fully correct)
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")
```

Python Example 2 (fully correct)
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
+ 2] == 'X':                                 (Part F, Part G, Part H)
       print("Bad move")
       pos = 0                               (Part F, Part G)
   else:
       pos = pos + 2                         (Part E)
```

Python Example 3 (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 = 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)
```

VB.NET Example 1 (fully correct)
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`

VB.NET Example 2 (7 marks)

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`