GCSE
COMPUTER SCIENCE
8525/1A, 8525/1B, 8525/1C

Paper 1 Computational thinking and programming skills

Mark scheme

June 2022

Version: 1.0 Final

2 2 6 G 8 5 2 5 / 1 / M S

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.

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.

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)** **B** Line number 2; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	2	**Mark is for AO2 (apply)** **E** `16`; **R.** If more than one lozenge shaded	1

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

Question	Part	Marking guidance	Total marks
01	4	**Mark is for AO2 (apply)** **B** Line number 2; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	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

01	6	**Mark is for AO3 (refine)**
		
		<u>C#</u>
		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);`
		`   }`
		`}`
		
		<u>Python</u>
		A `for x in range(0, 5):`
		`   i = int(input("Enter a number: "))`
		`   if i % 2 == 0:`
		`      print(i * i)`
		`   else:`
		`      print(i)`
		
		<u>VB.NET</u>
		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
			1

Question	Part	Marking guidance	Total marks
02	1	**2 marks for AO2 (apply)** **Input value of** `orderTotal` / **Input value of** `deliveryDistance` / **Output** `55.5` / `2` / **`1.5`;** `35.0` / `5` / **`7.0`;** **A.** `7`	2

Input value of `orderTotal`	**Input value of** `deliveryDistance`	**Output**
`55.5`	`2`	**`1.5`;**
`35.0`	`5`	**`7.0`;** **A.** `7`

Question	Part	Marking guidance	Total marks
02	2	**Mark is for AO2 (apply)** 2 // two;	1

Question	Part	Marking guidance	Total marks
02	3	**2 marks for AO2 (apply)** **I.** Case **A.** Programming language specific data types eg Single in VB.NET	2

Variable identifier	**Data type**
`deliveryCost`	**Float // Real // Decimal**
`messageOne`	**String // str**

Question	Part	Marking guidance	Total marks

02	4	**Mark is for AO1 (recall)**
		
		Boolean // Bool;
		Int // Integer;
		Date/Time;
		Character;
		
		R. Any answer that was given in **02.3**
		
		I. Case
		A. Any reasonable data type
			1

Question	Part	Marking guidance	Total marks

03	1	**Mark is for AO3 (refine)**
		
		C#
		`string displayMessage = carReg + " is not valid";`
		
		Python
		`displayMessage = carReg + " is not valid"`
		
		VB.NET
		`Dim displayMessage As String = carReg + " is not valid"` //
		`Dim displayMessage As String = carReg & " is not valid"`
		
		I. Case
		I. Space between variable outputs
		I. Order of strings
			1

Question	Part	Marking guidance	Total marks

03	2	**Mark is for AO3 (refine)**
		
		C#
		`charge = hours * 2 + 2;` //
		`charge = 2 + hours * 2;`
		
		Python
		`charge = hours * 2 + 2` //
		`charge = 2 + hours * 2`
		
		VB.NET
		`charge = hours * 2 + 2` //
		`charge = 2 + hours * 2`
		
		I. Case
		I. Parentheses, unless altering result eg, `hours * (2 + 2)`
			1

Question	Part	Marking guidance	Total marks
04	1	**Mark is for AO2 (apply)** **C** Program B is more efficient than Program A; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
04	2	**2 marks for AO2 (apply)** It will take less time for the computer to execute program B; because fewer lines of code will be executed; // The number of calculations performed is constant in Program B; but increases as the number input gets bigger in Program A; **A.** Program B has fewer variables; so, would use less memory (when executing);	2

Question	Part	Marking guidance	Total marks
05	1	**2 marks for AO1 (recall)** **B** A syntax error is a mistake in the grammar of the code; **D** A syntax error will stop a program from running; **R.** If more than two lozenges shaded	2

Question	Part	Marking guidance	Total marks
05	2	**Mark is for AO2 (apply)** **Mark is for AO3 (refine)** **C#** Line number: 7; Corrected line of code: `Console.WriteLine(numbers[number]);` **Python** Line number: 7; Corrected line of code: `print(numbers[number])` **VB.NET** Line number: 7; Corrected line of code: `Console.WriteLine(numbers(number))` **A**. `WriteLine` changed to `Write` as long as all other required changes have been made	2

Question	Part	Marking guidance	Total marks
05	3	**Mark is for AO2 (apply)** Array // List (of integers);	1

Question	Part	Marking guidance	Total marks
06	1	**Mark is for AO1 (recall)** Removing unnecessary detail (from the problem); **A.** data / information in place of detail for this year only	1

Question	Part	Marking guidance	Total marks

06	2	**2 marks for AO2 (apply)**
		
		Ⓐ Confirm / enter email address;
		
		Ⓑ Log out;
		
		A. any wording with the same meaning
			2

Question	Part	Marking guidance	Total marks

07		**2 marks 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 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;
		
		<u>Program Logic</u>
		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.
		
		<u>C# Example 1 (fully correct)</u>
		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**)
		`}`
			5

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")
```

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

<u>VB.NET Example 2 (fully correct)</u>
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

08		**3 marks for AO3 (design) and 4 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 a selection construct;
		Mark C for the use of a nested selection construct or multiple conditions;
		
		<u>Program Logic</u>
		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
		
		<u>C# Example 1 (fully correct)</u>
		
		`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)**
		`}`
			7

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
09	1	**Mark is for AO2 (apply)** **D** `S`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
09	2	**Mark is for AO2 (apply)** **B** `2`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
09	3	**Mark is for AO2 (apply)** `Sara`; **I.** Case	1

Question	Part	Marking guidance	Total marks
09	4	**2 marks for AO3 (program)** **Mark A** for correct identification of **2, 4**; **Mark B** for correct identification of **1**; **Model Answer** var ← SUBSTRING(**2, 4,** name1) OUTPUT (names[**1**] + var)	2

Question	Part	Marking guidance	Total marks
10	1	**4 marks for AO2 (apply)** • 1 mark for the first value of column `a` **and** the first value of column `b` both correct; • 1 mark for column `a` correctly integer dividing the first value in column `a` by 2 down to 1 and no other values; • 1 mark for minimum of six values in the `b` column, incrementing by one. The number of values in the `b` column must match the number of values in the `a` column; • 1 mark for `OUTPUT` being the final value of `b` and no other values in the output column, and no other values in column `n`; **A.** follow through from column `b` **I.** Different rows used as long as the order within columns is clear **I.** Duplicate values on consecutive rows within a column	4

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

Question	Part	Marking guidance	Total marks
10	2	**Mark is for AO2 (apply)** 1; **R.** the **word** one	1

Question	Part	Marking guidance	Total marks
10	3	**Mark is for AO2 (apply)** 1 mark for giving a new identifier that describes this purpose, eg `count` // `total` // `times` // `numberOfTimes` // `counter`	1

Question	Part	Marking guidance	Total marks
10	4	**2 marks for AO2 (apply)** **Maximum of 2 marks** from: • The `REPEAT…UNTIL` structure tests the condition at the end // the `WHILE…ENDWHILE` structure tests the condition at the beginning; • The `REPEAT…UNTIL` structure will always execute at least once // the `WHILE…ENDWHILE` loop may never execute; • If the value of `n` is 1 (or less) then the `REPEAT…UNTIL` structure will cause the value of `a` / `b` to change, but the `WHILE…ENDWHILE` structure will not; **R.** The `REPEAT…UNTIL` structure repeats lines of code until a condition is true **R.** The `WHILE…ENDWHILE` structure repeats lines of code until a condition is false	2

Question	Part	Marking guidance	Total marks

11	1	**6 marks for AO3 (program)**
		1 mark for each correct item in the correct location
		
		`SUBROUTINE getSize(`**`sampRate, res,`**` seconds)`
		    `size`` ← sampRate * res * seconds`
		    `size ← `**`size / 8`**
		    `RETURN`` size`
		`ENDSUBROUTINE`
		
		`OUTPUT `**`getSize`**`(100, 16, 60)`
		
		I. Case
		R. Incorrect order of parameters
			6

Question	Part	Marking guidance	Total marks
11	2	**Mark is for AO1 (understanding)** A variable that is only accessible / visible within the subroutine; // A variable that only exists while the subroutine is running;	1

Question	Part	Marking guidance	Total marks
11	3	**3 marks for AO1 (understanding)** **Max 3** marks from: • subroutines can be developed in isolation/independently/separately; • easier to discover errors // testing is more effective (than without a subroutine); • subroutines make program code easier to understand; **A.** 'easier to read' for this year only • subroutines make it easier for a team of programmers to work together on a large project; • subroutines make it easier to reuse code;	3

Question	Part	Marking guidance	Total marks
12	1	**6 marks for AO2 (apply)**	6

1 mark for `i` column and `j` column initialised to `0`;
1 mark for rest of `i` **and** `j` columns correct;
1 mark for `temp` column correct;
1 mark for first swap setting `arr[0]` column to `b` **and** `arr[1]` column to `c`;
1 mark for second swap setting `arr[1]` column to `a` **and** `arr[2]` column to `c`;
1 mark for third swap setting `arr[0]` column to `a` **and** `arr[1]` column to `b`;

arr			**i**	**j**	**temp**
[0]	**[1]**	**[2]**			
c	b	a			
			0	0	c
b	c			1	c
	a	c	1	0	b
a	b			1	

I. Different rows used as long as the order within columns is clear
I. Duplicate values on consecutive rows within a column
I. Quotes used around letters
I. Case

Note to Examiners: **A.** missing middle `c` in **`temp`** column

Question	Part	Marking guidance	Total marks
12	2	**Mark is for AO2 (apply)** Sort (the values in order) // bubble sort // put into alphabetical order;	1

Question	Part	Marking guidance	Total marks
12	3	**Mark is for AO2 (apply)** The algorithm will attempt to access an element/item/index in the array that does not exist; // The algorithm will attempt to use an index which is greater than the maximum array index of 2;	1

Question	Part	Marking guidance	Total marks

13	1	**2 marks for AO3 (design), 2 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 idea of inputting a number within the iteration/validation structure;
		Mark B for the use of indefinite iteration;
		
		<u>Program Logic</u>
		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.
		
		<u>C# Example 1 (fully correct)</u>
		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());`
		`}`
		
		<u>C# Example 2 (fully correct)</u>
		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());`
		`}`
		
		<u>C# Example 3 (partially correct – 3 marks)</u>
		1 design mark achieved (**Mark A**)
		`if (position < 1 || position > 100) {` **(C,D)**
		   `Console.Write("Enter card position: ");`
		   `position = Convert.ToInt32(Console.ReadLine());`
		`}`
			4

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
```

		VB.NET Example 4 (partially correct – 3 marks)
		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
13	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

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 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;

Program Logic
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.

C# Example 1 (fully correct)
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

Question	Part	Marking guidance	Total marks

14	1	**4 marks for AO3 (refine)**
		1 mark for initialising `j` to 0 in correct place;
		1 mark for using `i` and `j` as indices in `ticket`;
		1 mark for incrementing `j` by 1 in correct place;
		1 mark for incrementing `i` by 1 in correct place;
		
		A. `i` and `j` in opposite indices in `ticket`
		I. Case
		
		C# Example 1 (fully correct)
		
		`int i = 0;`
		`while (i < 3) {`
		`   int j = 0;`
		`   while (j < 3) {`
		`       ticket[i, j] = generateKeyTerm();`
		`       j = j + 1;`
		`   }`
		`   i = i + 1;`
		`}`
		
		C# Example 2 (fully correct)
		
		`int i = 0;`
		`while (i < 3) {`
		`   int j = 0;`
		`   while (j < 3) {`
		`       ticket[i, j] = generateKeyTerm();`
		`       j++;`
		`   }`
		`   i++;`
		`}`
		
		Python Example 1 (fully correct)
		
		`i = 0`
		`while i < 3:`
		`   j = 0`
		`   while j < 3:`
		`      ticket[i][j] = generateKeyTerm()`
		`      j = j + 1`
		`   i = i + 1`
			4

Python Example 2 (fully correct)

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

VB.NET Example 1 (fully correct)

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

VB.NET Example 2 (fully correct)

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

Question	Part	Marking guidance	Total marks
14	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)
```

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

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

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

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

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

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

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

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

```
Sub checkWinner(ticket)

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

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

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

```
Sub checkWinner(ticket)

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

<u>VB.NET Example 3 (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)
   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