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

Paper 1 Computational thinking and programming skills

Mark scheme

June 2023

Version: Final 1.0

236G8525/1/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.

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	**2 marks for AO1 (recall)** A sequence / series of steps / instructions; (that can be followed) to complete a task / to solve a problem; **A.** set of instructions / steps	2

Question	Part	Marking guidance	Total marks
01	2	**Mark is for AO2 (apply)** **C** `10;` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	3	**Mark is for AO2 (apply)** **D** `San FranciscoAlcatraz Island;` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	4	**Mark is for AO2 (apply)** **D** `traz;` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
01	5	**Mark is for AO2 (apply)** **C** `4;` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
02	1	**Mark is for AO2 (apply)** **A** 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** `0`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
02	3	**Mark is for AO2 (apply)** **C** `4`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
03		**2 marks for AO1 (understanding)** **1 mark** for the advantage, **1 mark** for the expansion point. **Maximum of 2 marks** from: ● Structured programs are easier to read / understand / modify; as they use logical structures // because the code is split into smaller subroutines / chunks / modules / sections; ● Easier to test / debug a program; that is divided into smaller subroutines / chunks / modules / sections; ● Subroutines can be reused; which reduces development time; ● Structured programs are easier to maintain; as they use clear well-documented interfaces // local variables / parameters / return values // as each subroutine is relatively independent of the other; ● Easier to divide projects up between teams; as code is split into subroutines / chunks / modules / sections; **Note to examiners:** both mark points can be taken from different bullets, so long as the explanation is relevant to the advantage eg: Structured programs are easier to understand. This makes it easier to divide projects between teams (2 marks). **Maximum 1 mark** if the explanation is not relevant to the stated advantage. **Maximum 1 mark** if there are two advantages but no explanation of either.	2

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>04</td><td>1</td><td>3 marks for AO2 (apply)

Maximum 2 marks if Output shows numbers or text only with no other errors OR fully correct but contains additional characters.

Maximum 1 mark if Output shows numbers or text only or is inconsistent AND there is at least one error, even if additional characters present.

<table>
<tr><th>First user input</th><th>Second user input</th><th>Third user input</th><th>Output</th></tr>
<tr><td>5</td><td>6</td><td>–1</td><td>Area 30</td></tr>
<tr><td>10</td><td>4</td><td>0</td><td>Volume 0</td></tr>
<tr><td>3</td><td>5</td><td>10</td><td>Volume 150</td></tr>
</table>
I. quotation marks in the Output column</td><td>3</td></tr>
</table>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>04</td><td>2</td><td>Mark is for AO2 (apply)

Maximum of 1 mark from:

● Add validation; A. by example eg check width/length are positive numbers // check height is –1 or a positive number;

● Change data types used in the question to float / single / double / decimal / real for inputs;</td><td>1</td></tr>
</table>

Question	Part	Marking guidance	Total marks
05		**3 marks for AO2 (apply)** **1 mark** for correct first row; **1 mark** for correct second row; **1 mark** for correct third and fourth rows; **Maximum 2 marks** if any errors **I.** different rows used as long as the order within columns is clear **I.** duplicate values on consecutive rows within a column **Note to examiners:** Check vertically as well as horizontally for the effect of duplicate values.	3

a	b	c
0	1	1
1	1	2
1	2	3
2	3	5

Question	Part	Marking guidance	Total marks
06		**4 marks for AO3 (design)** **1 mark** for each correct answer L1 `USERINPUT` L2 `username` L3 `''` **R.** `""` L4 `User not found` **I.** case / spelling mistakes so long as it is clear which option from **Figure 6** has been selected. **Note to Examiners:** If the student has re-written the entire line and added in the correct missing item, award the mark.	4

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>07</td><td></td><td>2 marks for AO3 (design), 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 inputting the number in the group and storing in a variable;
Mark B for using selection;

<u>Program Logic</u>

Mark C for correctly multiplying the number in the group by 15;
Mark D for using an appropriate correct Boolean condition(s) that covers all paths through the problem, eg >=6 // >5 or equivalent;
Mark E for using an appropriate method to reduce the total charge by £5;
Mark F for outputting the final total in a logical place;

Maximum 5 marks if any errors in code.

I. Case
I. Messages or no messages with input statements
I. Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect.</td><td>6</td></tr>
<tr><td></td><td></td><td><u>C# Example 1 (fully correct)</u>
All design marks are achieved (Marks A and B)

<pre>int group = Convert.ToInt32(Console.ReadLine());
int total = group * 15;   (C)
if (group >= 6) {   (D)
   total = total - 5;   (E)
}
Console.WriteLine(total);   (F)</pre>
I. Indentation in C#
A. Write in place of WriteLine

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

<pre>group = int(input())
total = group * 15   (C)
if group >= 6:   (D)
  total = total - 5   (E)
print(total)   (F)</pre></td><td></td></tr>
</table>

		VB.NET Example 1 (fully correct)
		All design marks are achieved (**Marks A** and **B**)
		
		`Dim group As Integer = Console.ReadLine()`
		`Dim total As Integer = group * 15` **(C)**
		`If (group >= 6) Then` **(D)**
		`  total = total - 5` **(E)**
		`End If`
		`Console.WriteLine(total)` **(F)**
		
		I. Indentation in VB.NET
		A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
08		**4 marks for AO1 (understanding)** **Maximum of 4 marks** from: ● The list is (repeatedly) divided into sub-lists (half the size / at the midpoint) until each sub-list is of length 1 (singleton lists) // The list is divided recursively until each sub-list is of length 1 (singleton lists); ● The algorithm compares / sorts individual elements as pairs; ● (After comparing) the individual items are then (repeatedly) merged back together into sub-lists // (After comparing) the sub-lists are then (repeatedly) merged back together; ● (Finally,) one list is produced in the right order (which is the sorted list); **Note to Examiners:** It is perfectly acceptable for the response given to be based on the contents of Figure 7 rather than a generic explanation.	4

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

Question	Part	Marking guidance	Total marks
09	2	**Mark is for AO2 (apply)** **A** `hulk.year ← 2003;` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
09	3	**Mark is for AO2 (apply)** **C** `LEN(filmCollection) - 1;` **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks

09	4	**Mark is for AO2 (apply)**
		
		`antMan.beingShown ← True`
		
		//
		
		`filmCollection[0].beingShown ← True;`
		
		A. `antMan ← Film('Ant-Man', '12A', 2015, True)` **R.** quotation marks around 2015
		
		I. Case
		A. = instead of ←
		R. `Ant-Man`
		R. Quotation marks around `True`
			1

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>10</td><td>1</td><td>5 marks for AO2 (apply)

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

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

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 (person column)
I. minor spelling mistakes in the person column</td><td>5</td></tr>
</table>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>10</td><td>2</td><td>Mark is for AO2 (apply)

C Change line number 7 to: <code>FOR j ← 0 TO 2</code>

R. if more than one lozenge shaded</td><td>1</td></tr>
</table>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>11</td><td>1</td><td>4 marks for AO3 (test)

<table>
<tr><th>Test type</th><th colspan="2">Test data</th><th>validChoice</th><th>difference</th></tr>
<tr><td rowspan="2">Normal</td><td>startYear</td><td>1995</td><td rowspan="2">False</td><td rowspan="2">-1</td></tr>
<tr><td>endYear</td><td>2010</td></tr>
<tr><td rowspan="2">Erroneous</td><td>startYear</td><td>2015</td><td rowspan="2">False</td><td rowspan="2">-1</td></tr>
<tr><td>endYear</td><td>2000</td></tr>
<tr><td rowspan="2">Boundary</td><td>startYear</td><td>2000</td><td rowspan="2">True</td><td rowspan="2">23</td></tr>
<tr><td>endYear</td><td>2023</td></tr>
</table>

1 mark for Normal <code>validChoice</code> result correct;
1 mark for Normal <code>difference</code> result correct;
1 mark for both Erroneous results correct;
1 mark for both Boundary results correct;</td><td>4</td></tr>
</table>

Question	Part	Marking guidance	Total marks
11	2	**Mark is for AO2 (apply)** `IF startYear ≥ 2000 THEN` // change the 'less than' symbol to a 'greater than or equal to' symbol;	1

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>12</td><td>1</td><td>
4 marks for AO2 (apply)
<table>
<tr><th>animalToFind</th><th>validAnimal</th><th>start</th><th>finish</th><th>mid</th></tr>
<tr><td>wolf</td><td>False</td><td>0</td><td>7</td><td>3</td></tr>
<tr><td></td><td></td><td>4</td><td></td><td>5</td></tr>
<tr><td></td><td></td><td>6</td><td></td><td>6</td></tr>
<tr><td></td><td></td><td>7</td><td></td><td>7</td></tr>
<tr><td></td><td>True</td><td></td><td></td><td></td></tr>
</table>
1 mark for correct <code>animalToFind</code>, <code>validAnimal</code> and <code>finish</code> columns and no other values;
1 mark for correct <code>start</code> column and no other values;
1 mark for having 5 as second value in <code>mid</code> column;
1 mark for the rest of <code>mid</code> column being correct and no other values;

Maximum 3 marks if any errors.

I. different rows used as long as the order within columns is clear
I. duplicate values on consecutive rows within a column
</td><td>4</td></tr>
</table>

Question	Part	Marking guidance	Total marks
12	2	**3 marks for AO3 (design), 4 marks for AO3 (program)** **Note to Examiners:** As the question asks them to write a routine for the list `fruits`, if they have used a series of connected selection statements rather than an iteration statement they can still gain marks C and D. **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 asking the user to input a word and storing the input in a variable; **Mark B** for using iteration; **Mark C** for attempting to check **each** index location within the list/array `fruits`; **Program Logic** **Mark D** for a loop which starts at one end of the list/array `fruits` and **could** correctly iterate through each index to the other end; **Mark E** for correctly comparing all 6 fruits against the word to find; **Mark F** for code which correctly **processes** a match; **Mark G** for outputting **only one** of the two messages `True` **and** `False` depending on the result of the match; **Maximum 5 marks** if they have not written their own linear search. **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. **I.** any rewritten code that defines `fruit`	7

C# Example 1 (fully correct)

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

```
string[] fruits = { "banana", "apple", "orange",
"pear", "grape", "pineapple" };
string wordToFind = Console.ReadLine();
bool found = false;
int count = 0;
while (found == false && count < fruits.Length)          (Part of D, Part of F)
{
   if (fruits[count] == wordToFind)                      (E)
   {
      found = true;                                      (F)
   }
   count += 1;
}
if (found == true)                                       (Part of D)
{
   Console.WriteLine("True");                            (Part of G)
}
else
{
   Console.WriteLine("False");                           (Part of G)
}
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

C# Example 2 (fully correct)

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

```
string[] fruits = { "banana", "apple", "orange",
"pear", "grape", "pineapple" };
string wordToFind = Console.ReadLine();
bool found = false;
foreach (string fruit in fruits)                         (Part of D)
{
   if (fruit == wordToFind)                              (E)
   {
      found = true;                                      (Part of F)
      break;                                             (Part of F)
   }
}
if (found == true)                                       (Part of F)
{
    Console.WriteLine("True");                           (Part of G)
}
else
{
    Console.WriteLine("False");                          (Part of G)
}
```

I. Indentation in C#

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

```
fruits = ["banana", "apple", "orange", "pear",
"grape", "pineapple"]
wordToFind = input()
found = False
count = 0
while found == False and count <= len(fruits) - 1:          (Part of D, Part of F)
   if fruits[count] == wordToFind:                          (E)
      found = True                                          (Part of F)
   count += 1                                               (Part of D)
if found == True:
   print("True")                                            (Part of G)
else:
   print("False")                                           (Part of G)
```

Python Example 2 (fully correct)

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

```
fruits = ["banana", "apple", "orange", "pear",
"grape", "pineapple"]
wordToFind = input()
found = False
for fruit in fruits:                                        (D)
   if fruit == wordToFind:                                  (E)
      found = True                                          (Part of F)
      break                                                 (Part of F)
if (found == True):                                         (Part of F)
   print("True")                                            (Part of G)
else:
   print("False")                                           (Part of G)
```

VB.NET Example 1 (fully correct)

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

```
Dim fruits() As String = {"banana", "apple", "orange",
"pear", "grape", "pineapple"}
Dim wordToFind As String = Console.ReadLine()
Dim found As Boolean = False
Dim count As Integer = 0
While found = False And count <= fruits.GetLength(0) - 1
   If fruits(count) = wordToFind Then
      found = True
   End If
   count = count + 1
End While
If found = True Then
   Console.WriteLine("True")
Else
   Console.WriteLine("False")
End If
```

(Part of D, Part of F)
(E)
(Part of F)
(Part of D)
(Part of G)
(Part of 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**, **B** and **C**)

```
Dim fruits() As String = {"banana", "apple",
"orange", "pear", "grape", "pineapple"}
Dim wordToFind As String = Console.ReadLine()
Dim found As Boolean = False
For Each fruit in fruits
   If fruit = wordToFind Then
      found = True
      Exit For
   End If
Next
If found = True Then
   Console.WriteLine("True")
Else
   Console.WriteLine("False")
End If
```

(D)
(E)
(Part of F)
(Part of F)
(Part of D)
(Part of G)
(Part of G)

I. Indentation in VB.NET

A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
12	3	**Mark is for AO2 (apply)** The list / array `fruits` is not ordered / sorted;	1

Question	Part	Marking guidance	Total marks
12	4	**3 marks for AO2 (apply)** **1 mark for each** error corrected. **Maximum 2 marks** if any logic errors **I.** syntax and other minor errors while rewriting, eg spelling, missing colons, brackets. **I.** other terms for `SUBROUTINE` as long as they make sense. **Line 1** `SUBROUTINE diffCurrencies(x)` ; **Line 6** `FOR i ← 7 TO 0 STEP -1` ;;	3

Question	Part	Marking guidance	Total marks
13		**2 marks for AO3 (design), 4 marks for AO3 (program)** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for using the variable `check` within their own code; **Mark B** for using selection or equivalent to check the grid references; **Program Logic** **Mark C** for correctly using an appropriate technique (slicing/indexing/`substring` function) with correct syntax to extract the left **and** right characters of input // for correctly comparing all nine possible valid grid references; **Mark D** for using **one** appropriate correct Boolean condition, eg =”A” // =”2” or equivalent; **Mark E** for having **all** the appropriate correct Boolean conditions to check the letters and numbers **AND** for `check` being set appropriately in all cases; **Mark F** for outputting an appropriate message in a logically appropriate location if their checks have failed; **Maximum 5 marks** if any errors in code. **I.** Case **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect.	6

C# Example 1 (fully correct)

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

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

(Part of C, Part of C) **(D)** **(E)**

(F)

I. Indentation in C#

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

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

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

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

   letter = square[0]
   number = square[1]

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

(Part of C, Part of C)

(D)(E)

(F)

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

VB.NET Example 1 (fully correct)

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

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

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

I. Indentation in VB.NET

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

A. `Write` in place of `WriteLine`

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

VB.NET Example 2 (fully correct)

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

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

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

(Part of C, Part of C)

(D)

(E)

(F)

I. Indentation in VB.NET

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

A. `Write` in place of `WriteLine`

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

Question	Part	Marking guidance	Total marks
14		**3 marks for AO2 (apply)** L1 `1;` L2 `i;` L3 `method;` **Note to Examiners:** If the student has re-written the entire line and added in the correct missing item, award the mark.	3

Question	Part	Marking guidance	Total marks
15		**3 marks for AO3 (design), 5 marks for AO3 (program)** Any solution that does not map to the mark scheme refer to lead examiner **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for storing a user input in a variable with a meaningful name; **Mark B** for using an iteration structure which attempts to pay the bill; **Mark C** for using a selection structure with `ELSE` / `ELSEIF` // use of multiple selection constructs; **Program Logic** **Mark D** for getting the user input for the total amount of the bill (outside the loop) **AND** deducting a payment towards the bill (within the loop); **A.** if there is no loop and both elements are present in the right order. **Mark E** for a mechanism which will correctly terminate the iteration structure, **in all situations**, when the bill is fully paid; **Mark F** for two conditions. One which checks / handles if the amount left to pay is `0` (or less, ie bill is paid), **AND** one which checks if the amount left to pay is less than `0` (for tip); **Mark G** for outputting in an appropriate place `Tip is` and the tip as a number; **R.** if tip is outputted when the amount left to pay is not less than zero **Mark H** for outputting `Bill paid` **and** the amount left to pay in logically appropriate places; **Maximum 7 marks** if any errors in code. **I.** Case **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect. **I.** Messages or no messages with input statements	8

C# Example 1 (fully correct)

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

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

I. Indentation in C#

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

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

A. without rounding / `round()` statements

VB.NET Example 1 (fully correct)

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

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

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

I. Indentation in VB.NET

A. `Write` in place of `WriteLine`

Question	Part	Marking guidance	Total marks
16		**4 marks for AO3 (design), 7 marks for AO3 (program)** Any solution that does not map to the mark scheme refer to lead examiner **Note to Examiners:** For marks **E** and **J** be careful not to penalise the same error twice. For example, if they have used 6 instead of 7 in mark E and then 21 instead of 22 in mark J apply a **DPT** **Program Design** **Note** that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Mark A** for attempting to randomly generate **two** numbers; **Mark B** for use of selection to check the current score against 21; **Mark C** for using iteration to keep rolling the dice; **Mark D** for outputting the dice rolls in appropriate places; **Program Logic** **Mark E** for generating **two** random numbers between 1 and 6 inclusive; **Mark F** for correctly adding the **two** dice values cumulatively to the previous score; **Mark G** for a loop that terminates if the current score is less than 21 **and** player chooses not to roll again; **Mark H** for a correct mechanism to end the game if the player has a score greater than or equal to 21; **Mark I** for a selection statement which correctly checks if the player has lost (final score is greater than 21) **OR** won (final score is 21); **Mark J** for generating a random number between 15 and 21 inclusive in a logically correct place **AND** checking if the result is greater than the final score; **Mark K** for **at least one** correct set of messages output in appropriate places to show whether the user has won or lost; **A.** yes/y, no/n or any other appropriate equivalents **Maximum 10 marks** if any errors in code. **I.** Case **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect. **I.** Messages or no messages with input statements	11

C# Example 1 (fully correct)

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

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

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

I. Indentation in C#

A. `Write` in place of `WriteLine`

Python Example 1 (fully correct)

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

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

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

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

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

VB.NET Example 1 (fully correct)

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

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

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

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

I. Indentation in VB.NET

A. `Write` in place of `WriteLine`