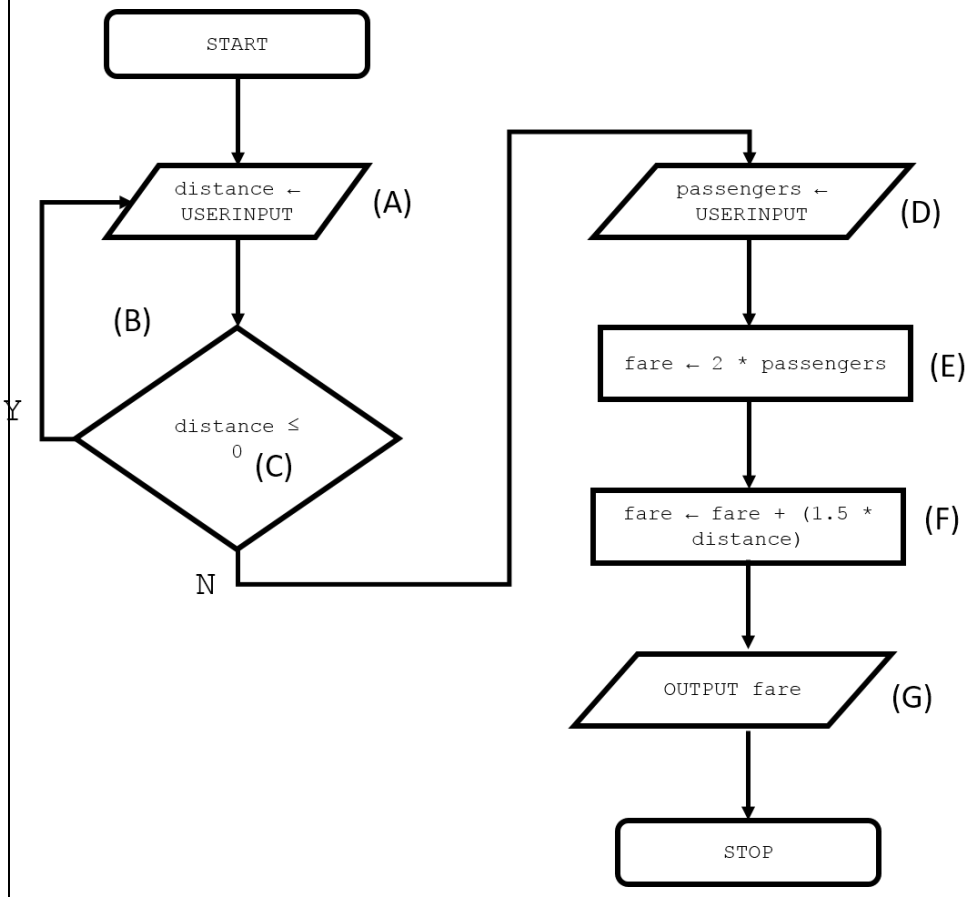
Question	Part	Marking guidance	Total marks
01	1	**Mark is for AO1 (understanding)** **A** (Line number 2) only; **If more than one lozenge shaded then mark is not awarded**	1
01	2	**Mark is for AO1 (understanding)** **C** (Line number 11) only; **If more than one lozenge shaded then mark is not awarded**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

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

Question	Part	Marking guidance	Total marks
02		**8 marks for AO3 (program)** **DPT.** For repeated errors in user input and variable assignment. Mark A for getting user input for the distance and storing in a variable; Mark B for using a WHILE loop or similar to re-prompt for and re-assign the user input; Mark C for using a correct Boolean condition with the validation structure; Mark D for getting user input for the passengers; Mark E for a fare that charges £2 per passenger; Mark F for a fare that charges £1.50 for every kilometre; Mark G for outputting the fare based on E and F (Even if E and/or F have been calculated incorrectly); Mark H if the algorithm is completely correct; **Example 1 (fully correct)** `distance ← USERINPUT` (A) `WHILE distance ≤ 0` (Part of B, C) `  distance ← USERINPUT` (Part of B) `ENDWHILE` `passengers ← USERINPUT` (D) `fare ← 2 * passengers` (E) `fare ← fare + (1.5 * distance)` (F) `OUTPUT fare` (G) (Mark H as completely correct) **Example 2 (fully correct)** `REPEAT` (Part of B) `  distance ← USERINPUT` (A, Part of B) `UNTIL distance > 0` (C) `fare ← (2 * USERINPUT) + (1.5 * distance)` (D, E, F) `OUTPUT fare` (G) (Mark H as completely correct) **Example 3 (fully correct)** `DO` (Part of B) `  distance ← USERINPUT` (A, Part of B) `WHILE NOT (distance > 0)` (C) `fare ← (2 * USERINPUT) + (1.5 * distance)` (D, E, F) `OUTPUT fare` (G) (Mark H as completely correct)	8

Example 4 (fully correct)



(Mark H as completely correct)

Example 5 (7 marks)

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

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

Example 6 (5 marks)

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

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

03	1	**3 marks for AO2 (apply)** 1 mark if column `z` increments by 1 and starts at 0; 1 mark if column `z` has the final value 3; 1 mark if `correct` column is correct; **z** / **correct** 0 / false 1 / true 2 / 3 / / /	3

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

03	3	**Mark is for AO2 (apply)** Second row only; **New Line** / **Tick one box** `IF user = us[z] OR pass = ps[z] THEN` / `IF user = us[z] AND pass = ps[z] THEN` / Tick `IF NOT (user = us[z] AND pass = ps[z]) THEN` /	1

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

Question	Part	Marking guidance	Total marks
04	1	**Mark is for AO2 (apply)** **C** `flourNeeded ← eggsUsed * 100;` **If more than one lozenge shaded then mark is not awarded**	1
04	2	**Mark is for AO2 (apply)** **A** Assignment; **If more than one lozenge shaded then mark is not awarded**	1
04	3	**4 marks for AO3 (program)** Max 3 marks if the answer contains any errors. **1 mark (A)** Indefinite iteration is used; **1 mark (B)** User input is used within the iteration/validation structure **and** the result is stored in the variable eggsUsed; **2 marks (C, D)** A Boolean condition checks the lower bound of eggsUsed is greater than zero/greater than or equal to one **and** the upper bound of eggsUsed is less than or equal to eight/less than nine (even if the **structure** is incorrect). This could possibly be one expression such as `0 < eggsUsed ≤ 8;;` If condition not completely correct then: **1 mark** The Boolean condition checks the lower bound of eggsUsed is greater than zero (even if the structure is incorrect) OR The Boolean condition checks the upper bound of eggsUsed is less than or equal to eight (even if the structure is incorrect) OR The Boolean conditions for the lower and upper bound are joined with the AND operator (even if the structure or the conditions themselves are incorrect); OR A method has been used that does not use a Boolean condition but is largely clear;	

Example 4 mark answer:

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

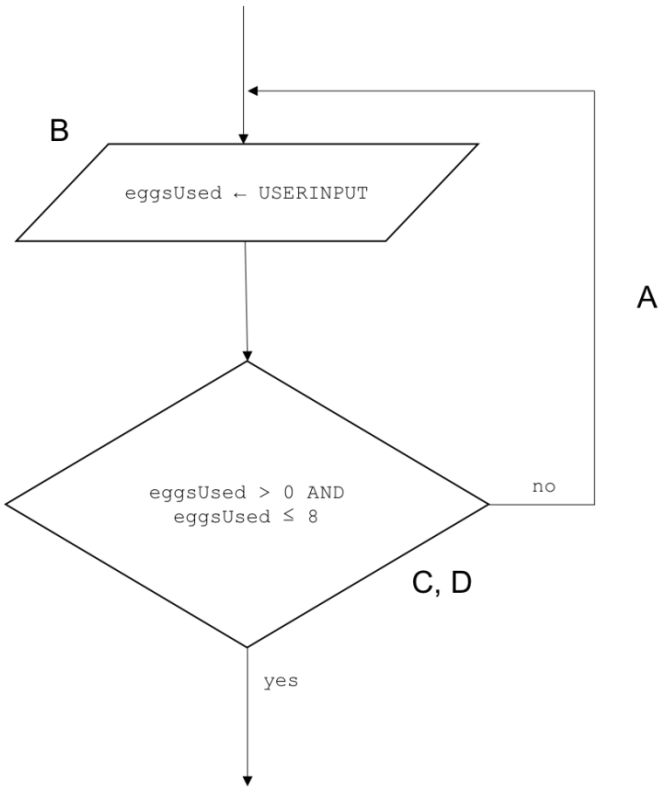
Example 4 mark answer:

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

Example 4 mark answer:

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

Example 4 mark answer:



Question	Part	Marking guidance	Total marks
05	1	**Mark is for AO2 (apply)** **D** `USERINPUT;` **If more than one lozenge shaded then mark is not awarded**	1
05	2	**Mark is for AO2 (apply)** **B** `0;` **If more than one lozenge shaded then mark is not awarded**	1
05	3	**Mark is for AO2 (apply)** **A** `=` ; **If more than one lozenge shaded then mark is not awarded**	1
05	4	**Mark is for AO2 (apply)** **D** `OUTPUT count;` **If more than one lozenge shaded then mark is not awarded**	1
05	5	**Mark is for AO2 (apply)** **B** `i ← i + 1;` **If more than one lozenge shaded then mark is not awarded**	1
05	6	**2 marks for AO2 (apply)** **Maximum of 1 mark if Upper Case Characters given** • 1 mark for a series of more than one correct frequency/value or value/frequency pairs (ignore order of pairs); • 1 mark for all correct pairs in the correct order; Correct answer is: 2 t 2 j 3 e 2 s Other, clear ways to show frequency/value or value/frequency pairs such as '(2, t), (2, j),…' or 't2 j2…'.	2

Question	Part	Marking guidance	Total marks
05	7	**3 marks for AO2 (apply)** Maximum three marks from: • It could be tested with only 1s; • It could be tested with different lengths of input; • It could be tested with an input where the 1s and 0s vary; • It could be tested with an input where the last two numbers are different; • It could be tested with the empty string; • It could be tested with a string of length one; • It could be tested with two runs of 0s separated by a run of 1s / two runs of 1s separated by a run of 0s; • It could be tested with invalid data (such as 1010abc); Any other correct reasoning as long as clearly distinct from other mark points. **R.** not enough tests are carried out.	3
06	1	**Mark is for AO2 (apply)** **C** Selection; **If more than one lozenge shaded then mark is not awarded**	1
06	2	**Mark is for AO2 (apply)** **D** String; **If more than one lozenge shaded then mark is not awarded**	1
06	3	**Mark is for AO2 (apply)** 3//three;	1
06	4	**2 marks for AO2 (apply)** `'no'` followed by `'yes'`; any value that isn't 'no' followed by 'yes' (allow by examples such as 'yes' followed by 'yes'); **R.** if a sequence does not contain two user inputs.	2

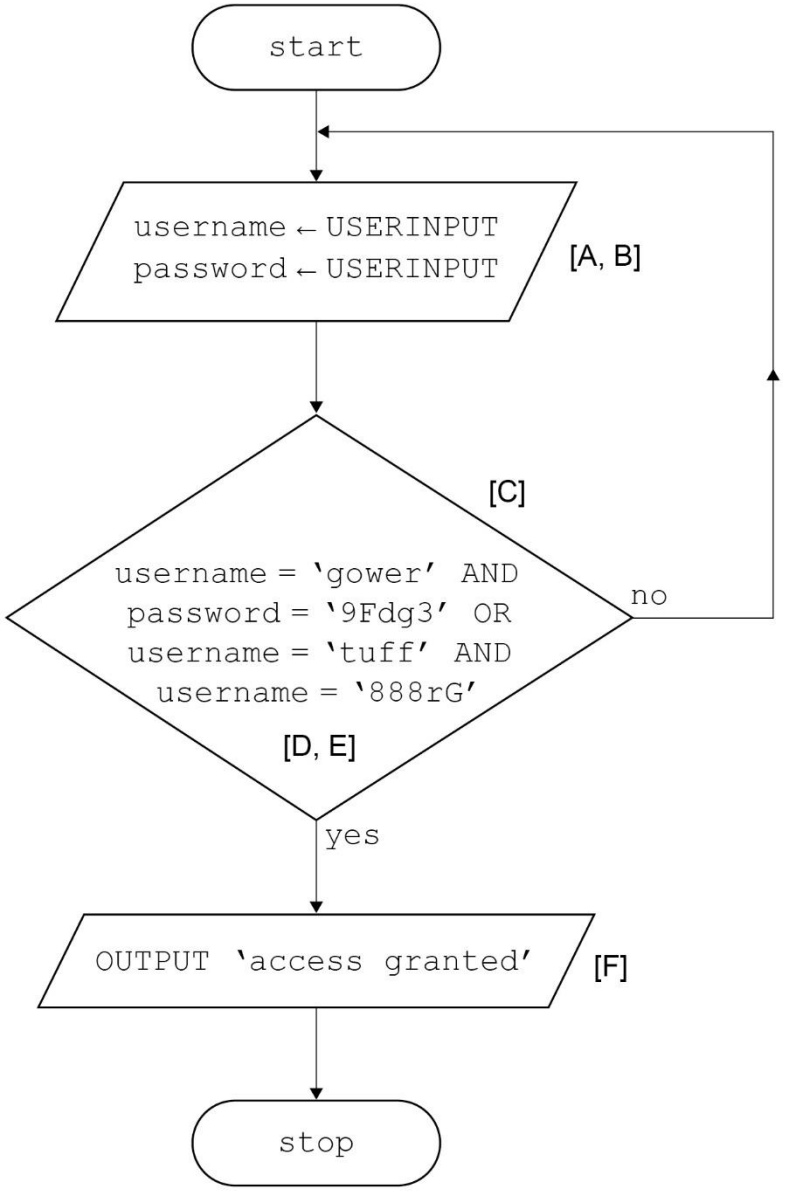
Question	Part	Marking guidance	Total marks
06	5	**3 marks for AO2 (apply)** Maximum three marks overall. **Maximum two marks from each section.** **<u>Reason</u>** • The output message is not descriptive enough/the user is not told what word/words they should use to answer (before user input); • The Boolean expression (at lines 3, 6 and 14) only matches exact values//the program is only written for the exact words `yes` and `no` // a **clear** indication that `y` is not recognised as `yes` or `n` is not recognised as `no`; • A clear explanation of how to fix the problem; **<u>What would happen</u>** Any clear descriptions of what would happen. Line numbers may or may not be included. If the logic and explanation is clear credit the answer. This can include but is not limited to: • Line 3 will only be true if they enter '`no`' // Line 3 will not be true if they enter anything other than '`no`'; • Line 6/14 will only be true if they enter '`yes`' // Line 6/14 will not be true if they enter anything other than '`yes`'; • if they enter '`n`' at line 2 the algorithm will execute an incorrect code block; • if they enter '`y`' at line 5 or line 13 an incorrect message will be output;	3

Qu	Part	Marking guidance	Total marks
07		**6 marks for AO3 (program)** **Mark A** for assigning user input to a variable (username); **Mark B** for assigning user input to a variable (password, the identifier must be different to that used in mark A); **Mark C** for using indefinite iteration and including user input within the iteration structure; **Mark D** for using a Boolean condition that checks the username is `gower` and the password is `9Fdg3` / the username is `tuff` and the password is `888rG`; **Mark E** for using the Boolean `OR` operator for both combinations of username and password, alternatively having sequential `IF` or `ELSE-IF` structures; **Mark F** for outputting the string after the iteration structure; **Max 5 marks** if the algorithm contains any errors. **I.** use of quote marks for usernames or passwords. **I.** minor spelling errors for username or passwords. Example of fully correct answer: `REPEAT` *[part C]* `  username ← USERINPUT` *[A, part C]* `  password ← USERINPUT` *[B, part C]* `UNTIL (username = 'gower' AND password = '9Fdg3') OR (username = 'tuff' AND password = '888rG')` *[D, E]* `OUTPUT 'access granted'` *[F]* Another example of a fully correct answer: `username ← USERINPUT` *[A]* `password ← USERINPUT` *[B]* `WHILE NOT ((username = 'gower' AND password = '9Fdg3') OR (username = 'tuff' AND password = '888rG'))` *[D, E, part C]* `  username ← USERINPUT` *[part C]* `  password ← USERINPUT` *[part C]* `ENDWHILE` `OUTPUT 'access granted'` *[F]*	6

Another example of a fully correct answer:

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

An example of a fully correct flowchart solution:



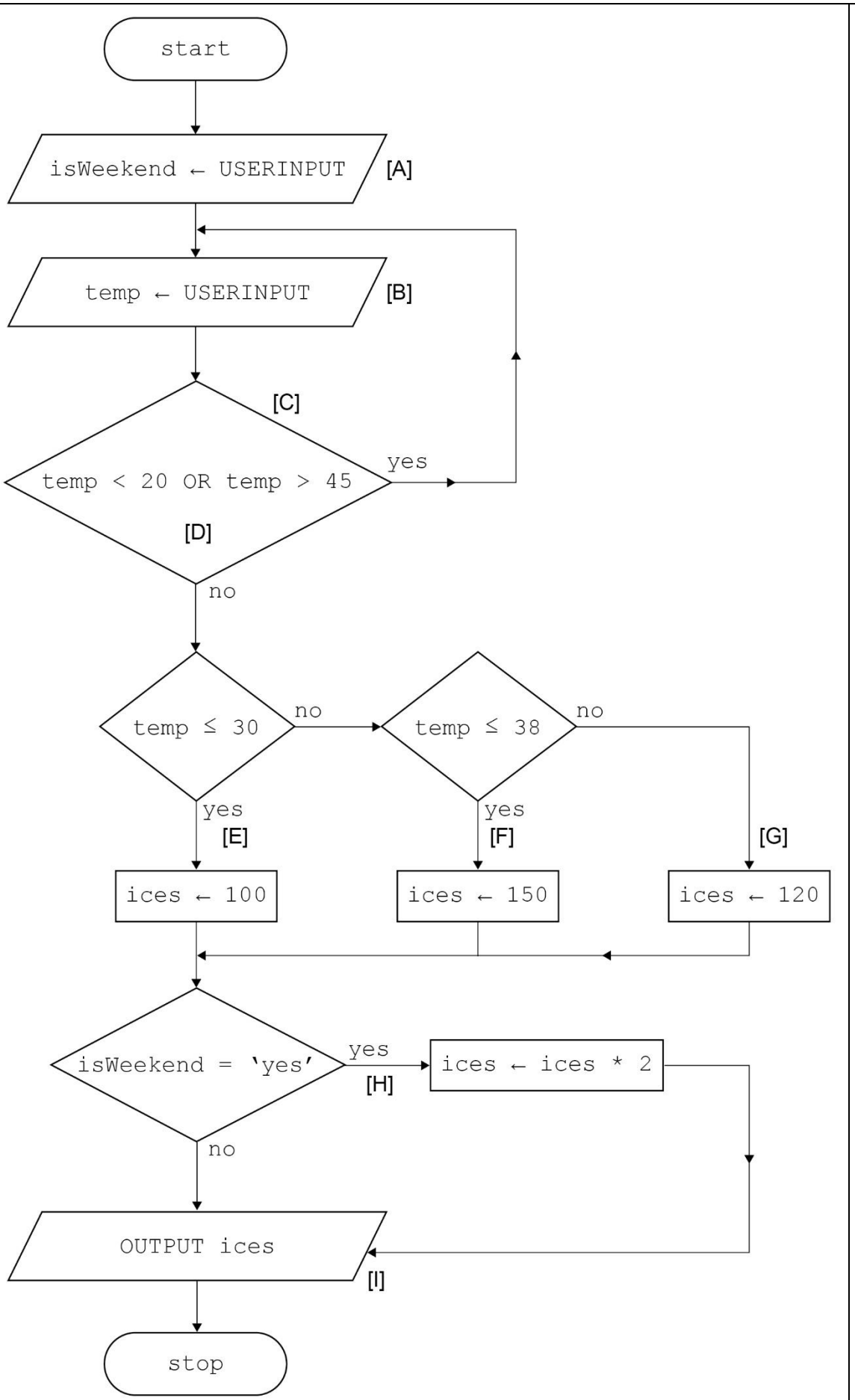
Qu	Part	Marking guidance	Total marks

08		**9 marks for AO3 (program)**
		
		Mark A for assigning user input to a variable (weekend or weekday);
		Mark B for assigning user input to a variable (temperature);
		Mark C for using indefinite iteration to repeatedly input the temperature;
		Mark D for a Boolean condition used to check the temperature between 20 and 45 inclusive;
		Mark E for using selection to set ice creams to be 100 if the temp is between 20 and 30 inclusive;
		Mark F for using selection to set ice creams to be 150 if the temp is between 31 and 38 inclusive;
		Mark G for using selection to set ice creams to be 120 if the temp is higher than 38;
		Mark H for doubling the quantity if it is a weekend (mark A is not required);
		Mark I for **always** outputting the estimated number of ice creams;
		
		Max 8 marks if solution contains any errors.
		
		An example of a fully correct solution:
		
		`isWeekend ← USERINPUT` *[A]*
		`temp ← USERINPUT` *[B]*
		`WHILE temp < 20 OR temp > 45` *[part C, D]*
		`  temp ← USERINPUT` *[part C]*
		`ENDWHILE`
		`IF temp ≤ 30 THEN` *[part E]*
		`  ices ← 100` *[part E]*
		`ELSE IF temp ≤ 38 THEN` *[part F]*
		`  ices ← 150` *[part F]*
		`ELSE` *[part G]*
		`  ices ← 120` *[part G]*
		`ENDIF`
		`IF isWeekend = 'yes' THEN` *[part H]*
		`  ices ← ices * 2` *[part H]*
		`ENDIF`
		`OUTPUT ices` *[part I]*
			9

Another example of a fully correct solution:

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

An example of a fully correct flowchart solution:


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

Question	Part	Marking guidance	Total marks
09	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
09	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
09	3	**Mark is for AO2 (apply)** Array // List (of integers);	1

Question	Part	Marking guidance	Total marks

10		**2 marks 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 using meaningful variable names throughout and for using two variables to store the two email address inputs;
		Mark B for the use of a selection construct // use of multiple selection constructs;
		
		Program Logic
		Mark C for using user input and storing the results in two variables correctly for the first email address and the second email address;
		Mark D for a correct expression that checks if the first entered email address is equal to the second entered email address (or not equal to);
		Mark E for outputting `Do not match` and `Match` in logically separate places such as the IF and ELSE part of selection, and for outputting the email address if both email addresses match;
		
		A. Any suitable alternative messages.
		
		I. Case
		I. Messages or no messages with input statements
		
		Maximum 4 marks if any errors in code.
		
		C# Example 1 (fully correct)
		All design marks are achieved (**Marks A and B**)
		
		`string email1 = Console.ReadLine();` (**Part of C**)
		`string email2 = Console.ReadLine();` (**Part of C**)
		
		`if (email1 != email2) {` (**D**)
		`  Console.WriteLine("Do not match");` (**Part of E**)
		`}`
		`else {`
		`  Console.WriteLine("Match");` (**Part of E**)
		`  Console.WriteLine(email1);` (**Part of E**)
		`}`
			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")
```

VB.NET Example 1 (fully correct)

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

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

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

VB.NET Example 2 (fully correct)

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

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

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

Question	Part	Marking guidance	Total marks

11	1	**2 marks for AO3 (design), 2 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 inputting a number within the iteration/validation structure;
		Mark B for the use of indefinite iteration;
		
		Program Logic
		Mark C for using a Boolean condition that checks the lower or upper bound of `position`;
		Mark D for using a Boolean condition that checks **BOTH** the lower and upper bounds of `position` correctly;
		Marks C and **D** could be one expression eg `0 < position <= 100;`
		
		I. Case
		I. Missing prompts
		
		Maximum 3 marks if any errors in code.
		
		C# Example 1 (fully correct)
		All design marks are achieved (**Marks A and B**)
		`while (position < 1 || position > 100) {` (**C,D**)
		`  Console.Write("Enter card position: ");`
		`  position = Convert.ToInt32(Console.ReadLine());`
		`}`
		
		C# Example 2 (fully correct)
		All design marks are achieved (**Marks A and B**)
		`while (position <= 0 || position >= 101) {` (**C,D**)
		`  Console.Write("Enter card position: ");`
		`  position = Convert.ToInt32(Console.ReadLine());`
		`}`
		
		C# Example 3 (partially correct – 3 marks)
		1 design mark achieved (**Mark A**)
		`if (position < 1 || position > 100) {` (**C,D**)
		`  Console.Write("Enter card position: ");`
		`  position = Convert.ToInt32(Console.ReadLine());`
		`}`
			4

C# Example 4 (partially correct – 3 marks)

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

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

(**Mark C**)

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:
   position = int(input("Enter card position: "))
```

(**C,D**)

Python Example 2 (fully correct)

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

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

(**C,D**)

Python Example 3 (partially correct – 3 marks)

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

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

(**C,D**)

Python Example 4 (partially correct – 3 marks)

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

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

(**C**)

VB.NET Example 1 (fully correct)

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

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

(**C,D**)

VB.NET Example 2 (fully correct)

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

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

(**C,D**)

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

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

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

(**C,D**)

		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

11	2	**2 marks for AO3 (design), 4 marks for AO3 (program)**
		Any solution that does not map to the mark scheme refer to lead examiner
		
		Program Design
		Note that AO3 (design) marks are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.
		
		Mark A for 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)**
		`}`
		`}`
			6

C# Example 2 (fully correct)

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

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

I. Indentation in C#

Python Example 1 (fully correct)

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

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

Python Example 2 (fully correct)

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

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

Python Example 3 (fully correct)

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

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

VB.NET Example 1 (fully correct)

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

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

VB.NET Example 2 (fully correct)

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

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

I. Indentation in VB.NET

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>12</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>12</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
13		**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>14</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>14</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>15</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>

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>15</td><td>2</td><td>Mark is for AO2 (apply)

<code>IF startYear ≥ 2000 THEN</code> // change the ‘less than’ symbol to a ‘greater than or equal to’ symbol;</td><td>1</td></tr>
</table>

Question	Part	Marking guidance	Total marks
16		**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)
   Dim number As String = square(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**

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

17		**2 marks for AO3 (design), 3 marks for AO3 (program)**
		
		Program Design
		Mark A for the use of a selection construct (even if the logic is incorrect);
		Mark B for the correct, consistent use of meaningful variable names throughout (even if the code would not work);
		
		Program Logic
		Mark C for using user input and storing the result in a variable correctly;
		Mark D for a correct expression that checks if the entered password is `'secret'` (even if the syntax is incorrect);
		Mark E for outputting `Welcome` and `Not welcome` correctly in logically separate places such as the `IF` and `ELSE` part of selection;
		
		I. Case of output strings for **Mark E**, but spelling must be correct.
		I. Case of program code
		
		Maximum 4 marks if any errors in code.
		
		Python Example 1 (fully correct)
		All design marks are achieved (**Marks A and B**)
		
		`password = input()` (**C**)
		`if password == 'secret':` (**D**)
		`    print('Welcome')` (**Part of E**)
		`else:`
		`    print('Not welcome')` (**Part of E**)
		
		C# Example (fully correct)
		All design marks are achieved (**Marks A and B**)
		
		`string password;`
		`password = Console.ReadLine();` (**C**)
		`if (password == "secret")` (**D**)
		`{`
		`   Console.WriteLine("Welcome");` (**Part of E**)
		`}`
		`else`
		`{`
		`   Console.WriteLine("Not welcome");` (**Part of E**)
		`}`
		
		I. indentation in C#
		
		VB Example (fully correct)
		All design marks are achieved (**Marks A and B**)
		
		`Dim password As String`
		`password = Console.ReadLine()` (**C**)
			5

Question	Part	Marking guidance	Total marks
18		**3 marks for AO3 (design), 4 marks for AO3 (program)** **Program Design** **Mark A** for the idea of inputting a character and checking if it is lower case (even if the code would not work); **Mark B** for the use of a selection construct (even if the logic is incorrect); **Mark C** for the correct, consistent use of meaningful variable names throughout (even if the code would not work); **Program Logic** **Mark D** for using user input correctly; **Mark E** for storing the result of user input in a variable correctly; **Mark F** for a correct expression/method that checks if the character is lowercase; **Mark G** for outputting `LOWER` and `NOT LOWER` correctly in logically separate places such as the `IF` and `ELSE` part of selection; **I.** Case of output strings for **Mark G**, but spelling must be correct. **I.** Case of program code **Maximum 6 marks** if any errors in code. **Python Example 1 (fully correct)** All design marks are achieved (**Marks A, B and C**) `character = input()` (**D,E**) `if (character >= 'a') and (character <= 'z'):` (**F**) `  print('LOWER')` (**Part of G**) `else:` `  print('NOT LOWER')` (**Part of G**) **Python Example 2 (fully correct)** All design marks are achieved (**Marks A, B and C**) `character = input()` (**D,E**) `if character.islower():` (**F**) `  print('LOWER')` (**Part of G**) `else:` `  print('NOT LOWER')` (**Part of G**)	7

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

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

I. indentation in C#

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

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

I. indentation in VB.NET

Python Example 3 (partially correct – 5 marks)
All design marks are achieved (**Marks A, B and C**)

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

Question	Part	Marking guidance	Total marks

19		**4 marks for AO3 (refine)**
		
		Program Logic
		Mark A: for using a selection structure with else part **or** two selection structures (even if the syntax is incorrect)
		Mark B: for correct condition(s) in selection statement(s) (even if the syntax is incorrect)
		Mark C: for statement that subtracts two from `odd` under the correct conditions (even if the syntax is incorrect)
		Mark D: for `odd` being output and doing one of adding or subtracting two but not both each time loop repeats (even if the syntax is incorrect)
		
		I. while loop from question if included in answer
		I. case of program code
		
		Maximum 3 marks if any errors in code.
		
		Python Example 1 (fully correct)
		
		`print(odd)` (**Part of D**)
		`if number < 0` (**A, B**)
		`  odd = odd - 2` (**C, Part of D**)
		`else:`
		`  odd = odd + 2` (**Part of D**)
		
		C# Example (fully correct)
		
		`Console.WriteLine(odd);` (**Part of D**)
		`if (number < 0)` (**A, B**)
		`{`
		`  odd = odd - 2;` (**C, Part of D**)
		`}`
		`else`
		`{`
		`  odd = odd + 2;` (**Part of D**)
		`}`
		
		I. indentation in C#
			4

VB.Net Example (fully correct)

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

I. indentation in VB.Net

Python Example 2 (partially correct – 3 marks)

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

20		**2 marks for AO3 (test)** 1 mark for each completely correct row to a maximum of 2 marks.	2

Test type	Test data	Expected result
Normal data	5	`Valid choice` message displayed
Invalid data	Any value other than the numbers 1 to 10 inclusive	`Invalid choice` (message displayed)
Boundary data	Any one of 0, 1, 10 or 11	if 1 or 10 given as test data `Valid choice` (message displayed) if 0 or 11 given as test data `Invalid choice` (message displayed)

21	1	**1 mark for AO3 (test)** 2;	1
21	2	**1 mark for AO3 (test)** 5;	1
21	3	**1 mark for AO3 (refine)** Change the < sign to <= // change `num1` to `num1 + 1`; **A.** answers where line of code has been rewritten	1

Question	Part	Marking guidance	Total marks
22		**2 marks for AO3 (design) and 6 marks for AO3 (program)** **Program Design** **Mark A** for using an iterative structure to validate the user input of speed (even if logic is incorrect); **Mark B** for using meaningful variable names **and** suitable data types throughout (speed can be real or integer, breaking distance must be real, the `IsWet` input must be string); **Program Logic** **Mark C** for getting user input for **both** the speed and `IsWet` in appropriate places; **Mark D** for using a WHILE loop or similar to re-prompt for the user input (even if it would not work); **Mark E** for using a correct Boolean condition with the validation structure; **Mark F** for calculating the braking distance correctly (i.e. divided by 5); **Mark G** for using a selection structure to adjust the braking distance calculation if the user input required it (even if it would not work); **Mark H** for outputting the braking distance in a logically correct place; **I.** Case of program code **Maximum 7 marks** if any errors in code. **Python Example (fully correct)** All design marks are achieved (**Marks A and B**) `speed = float(input())` (**Part of C**) `while speed < 10 or speed > 50:` (**D, E**) `  speed = float(input())` (**Part of D**) `braking_distance = speed / 5` (**F**) `IsWet = input()` (**Part of C**) `if IsWet == 'yes':` (**Part of G**) `  braking_distance = braking_distance * 1.5` (**Part of G**) `print(braking_distance)` (**H**)	8

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

```
int intSpeed;
double braking_distance;
string IsWet;
intSpeed = int.Parse(Console.ReadLine());          (Part of C)
while (intSpeed < 10 || intSpeed > 50)             (D, E)
{
   intSpeed = int.Parse(Console.ReadLine());       (Part of D)
}
braking_distance = (double)intSpeed / 5;           (F)
IsWet = Console.ReadLine();                        (Part of C)
if (IsWet == "yes")                                (Part of G)
{
   braking_distance = braking_distance * 1.5;      (Part of G)
}
Console.WriteLine(braking_distance);               (H)
```

I. indentation in C#

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

```
Dim speed As Integer
Dim braking_distance As Decimal
Dim IsWet As String
speed = Console.ReadLine()                         (Part of C)
while speed < 10 Or speed > 50                     (D, E)
   speed = Console.ReadLine()                      (Part of D)
End While
braking_distance = speed / 5                       (F)
IsWet = Console.ReadLine()                         (Part of C)
if IsWet = "yes" Then                              (Part of G)
   braking_distance = braking_distance * 1.5       (Part of G)
End If
Console.WriteLine(braking_distance)                (H)
```

I. indentation in VB.Net

		Python Example (partially correct – 7 marks) All design marks are achieved (**Marks A and B**) `speed = float(input())` (**Part of C**) `while speed <= 10 and speed > 50` (**D, NOT E**) `  speed = float(input())` (**Part of D**) `  braking_distance = speed / 5` (**F**) `IsWet = input()` (**Part of C**) `if IsWet = 'yes'` (**Part of G**) `  braking_distance = braking_distance * 1.5` (**Part of G**) `print(braking_distance)` (**H**)	

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>23</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

C#

```
randomNumber = rGen.Next(1, 101)          ;;
```

Python

```
randomNumber = random.randrange(1, 101)   ;;
```

VB.NET

```
randomNumber = rGen.Next(1, 101)          ;;
```

</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>23</td><td>2</td><td>

2 marks for AO2 (apply)

Test number	Test type	Test data	Expected result
1	Erroneous	150	Invalid number
2	Boundary	0/1/100/101	Invalid number / Valid number entered
3	Normal	Anything between 1 and 100 inclusive	Valid number entered

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 `Invalid number`, if 1 or 100 given as Boundary test data then Expected result must be `Valid number entered`;

</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>23</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
24		**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`