1. A program is being developed that allows users to rate and review movies. A user will enter their rating (out of 10) and a written review for each movie they have watched.

Computational thinking skills are used during the development of the program.

Define the term **abstraction**.

(Total 1 mark)

2. A program is being developed that allows users to rate and review movies. A user will enter their rating (out of 10) and a written review for each movie they have watched.

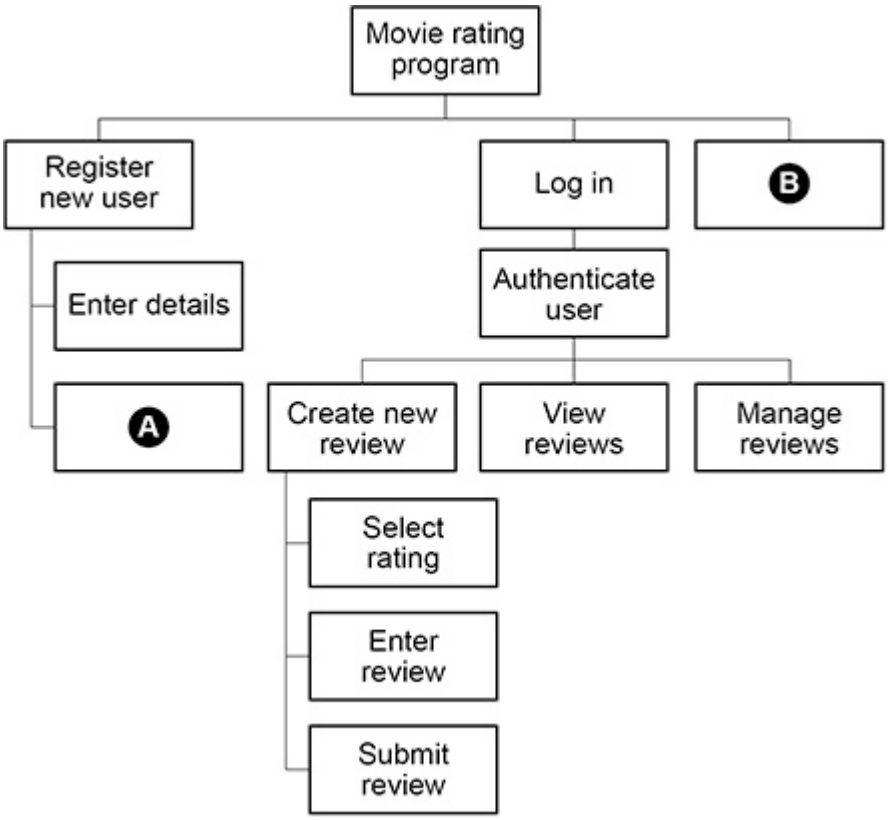
Computational thinking skills are used during the development of the program.

A user will be able to register, log in and log out of the program. When registering, a new user will enter their details, before confirming their email address.

Decomposition has been used to break the problem down into smaller sub-problems.

The chart below represents the design of the program.

Complete the decomposition of this program by stating what should be written in boxes **A** and **B**.



(Total 2 marks)

3. The figure below shows an algorithm represented using pseudo-code.

- Line numbers are included but are not part of the algorithm.

```
1    names ← ['Lily', 'Thomas']
2    name1 ← 'Sarah'
3    name2 ← 'Freddie'
4    OUTPUT name1[0]
5    OUTPUT LEN(names)
6    var ← SUBSTRING(0, 3, name1)
7    OUTPUT var
```

`SUBSTRING` returns part of a string.

For example, `SUBSTRING(3, 5, 'programming')` will return the string `'gra'`.

(a) Shade **one** lozenge which shows the output of **line 4** from the algorithm shown in the figure above.

A `F` ⬭

B `Freddie` ⬭

C `Lily` ⬭

D `S` ⬭

E `Sarah` ⬭

(1)

(b) Shade **one** lozenge which shows the output of **line 5** from the algorithm shown in the figure above.

A `1` ⬭

B `2` ⬭

C `4` ⬭

D `5` ⬭

E `10` ⬭

(1)

(c) State the output of **line 7** from the algorithm shown in the figure above.

(1)

(d) Two extra lines are being added to the end of the algorithm in the figure above.

Fill in the gaps so the output from the new final line will be the string `'Thomasrah'`.

```
var ← SUBSTRING( _____ , _____ , name1)

OUTPUT names[ _______ ] + var
```

(2)

(Total 5 marks)

4. **Figure 1** shows a subroutine represented using pseudo-code.

Figure 1

```
SUBROUTINE calculate(n)
   a ← n
   b ← 0
   REPEAT
      a ← a DIV 2
      b ← b + 1
   UNTIL a ≤ 1
   OUTPUT b
ENDSUBROUTINE
```

The `DIV` operator is used for integer division.

(a) Complete the trace table for the subroutine call `calculate(50)`

You may not need to use all the rows in the table.

n	a	b	OUTPUT
50			

(4)

(b) State the value that will be output for the subroutine call `calculate(1)`

(1)

(c) The identifier for the variable `b` in **Figure 1** was not a good choice.

State a better identifier for this variable that makes the algorithm easier to read and understand.

(1)

(d) A `REPEAT…UNTIL` iteration structure was used in **Figure 1**.

Figure 1 has been included again below.

Figure 1

```
SUBROUTINE calculate(n)
   a ← n
   b ← 0
   REPEAT
      a ← a DIV 2
      b ← b + 1
   UNTIL a ≤ 1
   OUTPUT b
ENDSUBROUTINE
```

Figure 2 shows another subroutine called `calculate` that uses a `WHILE…ENDWHILE` iteration structure.

Figure 2

```
SUBROUTINE calculate(n)
   a ← n
   b ← 0
   WHILE a > 1
      a ← a DIV 2
      b ← b + 1
   ENDWHILE
   OUTPUT b
ENDSUBROUTINE
```

One difference in the way the subroutines in **Figure 1** and **Figure 2** work is:

- the `REPEAT…UNTIL` iteration structure in **Figure 1** loops until the condition is true
- the `WHILE…ENDWHILE` iteration structure in **Figure 2** loops until the condition is false.

Describe **two** other differences in the way the subroutines in **Figure 1** and **Figure 2** work.

(2)

(Total 8 marks)

5. (a) The size of a sound file is calculated using the following formula:

size (in bits) = sampling rate * sample resolution * seconds

To calculate the size **in bytes**, the number is divided by **8**

The algorithm in **Figure 2**, represented using pseudo-code, should output the size of a sound file in **bytes** that has been sampled 100 times per second, with a sample resolution of 16 bits and a recording length of 60 seconds.

A subroutine called `getSize` has been developed as part of the algorithm.

Complete **Figure 2** by filling in the gaps using the items in **Figure 1**.

You will not need to use all the items in **Figure 1**.

Figure 1

bit	byte	getSize	OUTPUT
rate	res	RETURN	sampRate
seconds	size	size + 8	size * 8
size / 8	size MOD 8	SUBROUTINE	USERINPUT

Figure 2

```
SUBROUTINE getSize(_____________, _____________, seconds)

   _____________ ← sampRate * res * seconds

   size ← _____________

   _____________ size

ENDSUBROUTINE

OUTPUT _____________(100, 16, 60)
```

(6)

(b) A local variable called `size` has been used in `getSize`.

Explain what is meant by a local variable in a subroutine.

(1)

(c) State **three** advantages of using subroutines.

(3)

(Total 10 marks)

6. The figure below shows an algorithm represented in pseudo-code. A developer wants to check the algorithm works correctly.

- Line numbers are included but are not part of the algorithm.

```
1       arr[0] ← 'c'
2       arr[1] ← 'b'
3       arr[2] ← 'a'
4       FOR i ← 0 TO 1
5          FOR j ← 0 TO 1
6             IF arr[j + 1] < arr[j] THEN
7                temp ← arr[j]
8                arr[j] ← arr[j + 1]
9                arr[j + 1] ← temp
10            ENDIF
11         ENDFOR
12      ENDFOR
```

(a) Complete the trace table for the algorithm shown in the figure above.

Some values have already been entered. You may not need to use all the rows in the table.

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

(6)

(b) State the purpose of the algorithm.

(1)

(c) The figure above has been included again below.

```
1       arr[0] ← 'c'
2       arr[1] ← 'b'
3       arr[2] ← 'a'
4       FOR i ← 0 TO 1
5          FOR j ← 0 TO 1
6             IF arr[j + 1] < arr[j] THEN
7                temp ← arr[j]
8                arr[j] ← arr[j + 1]
9                arr[j + 1] ← temp
10            ENDIF
11         ENDFOR
12      ENDFOR
```

An earlier attempt at writing the algorithm in the figure above had different code for **lines 4** and **5**.

Lines 4 and **5** of the pseudo-code were:

```
FOR i ← 0 TO 2
   FOR j ← 0 TO 2
```

Explain why the algorithm did not work when the value `2` was used instead of the value `1` on these two lines.

(1)

(Total 8 marks)

7. An algorithm, that uses the modulus operator, has been represented using pseudo-code in **Figure 1**.

- Line numbers are included but are not part of the algorithm.

Figure 1

```
1       i ← USERINPUT
2       IF i MOD 2 = 0 THEN
3          OUTPUT i * i
4       ELSE
5          OUTPUT i
6       ENDIF
```

The modulus operator is used to calculate the remainder after dividing one integer by another.

For example:

- `14 MOD 3` evaluates to `2`
- `24 MOD 5` evaluates to `4`

(a) Shade **one** lozenge that shows the line number where selection is **first** used in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

(b) Shade **one** lozenge that shows the output from the algorithm in **Figure 1** when the user input is `4`

A `0` ⬭

B `2` ⬭

C `4` ⬭

D `8` ⬭

E `16` ⬭

(1)

(c) Shade **one** lozenge that shows the line number where assignment is **first** used in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

(d) Shade **one** lozenge that shows the line number that contains a relational operator in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

Figure 1 has been included again below.

Figure 1

```
1       i ← USERINPUT
2       IF i MOD 2 = 0 THEN
3          OUTPUT i * i
4       ELSE
5          OUTPUT i
6       ENDIF
```

(e) Shade **one** lozenge to show which of the following is a **true** statement about the algorithm in **Figure 1**.

A This algorithm uses a Boolean operator. ⬭

B This algorithm uses a named constant. ⬭

C This algorithm uses iteration. ⬭

D This algorithm uses the multiplication operator. ⬭

(1)

(f) **Figure 2** shows an implementation of the algorithm in **Figure 1** using the C# programming language.

- Line numbers are included but are not part of the program.

Figure 2

```
1       Console.Write("Enter a number: ");
2       int i = Convert.ToInt32(Console.ReadLine());
3       if (i % 2 == 0) {
4          Console.WriteLine(i * i);
5       }
6       else {
7          Console.WriteLine(i);
8       }
```

The program in **Figure 2** needs to be changed so that it repeats five times using **definite** (count controlled) iteration.

Shade **one** lozenge next to the program that does this correctly.

A

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

B

```
for (int x = 0; x < 6; x++) {
   Console.Write("Enter a number: ");
   int i = Convert.ToInt32(Console.ReadLine());
   if (i % 2 == 0) {
      Console.WriteLine(i * i);
   }
   else {
      Console.WriteLine(i);
   }
}
```

C

```
int x = 1;
while (x != 6) {
   Console.Write("Enter a number: ");
   int i = Convert.ToInt32(Console.ReadLine());
   if (i % 2 == 0) {
      Console.WriteLine(i * i);
   }
   else {
      Console.WriteLine(i);
   }
   x = x + 1;
}
```

D

```
int x = 6;
while (x != 0) {
   Console.Write("Enter a number: ");
   int i = Convert.ToInt32(Console.ReadLine());
   if (i % 2 == 0) {
      Console.WriteLine(i * i);
   }
   else {
      Console.WriteLine(i);
   }
   x = x - 1;
}
```

(1)

(Total 6 marks)

8. The figure below shows a C# program that calculates car park charges.

The user inputs their car registration (eg `MA19 GHJ`) and the length of the stay. The program then outputs the charge.

- Line numbers are included but are not part of the program.

```
1     int charge = 0;
2     Console.Write("Enter your car registration: ");
3     string carReg = Console.ReadLine();
4     while (carReg.Length > 8) {
5        string displayMessage = " is not valid";
6        Console.Write(displayMessage);
7        carReg = Console.ReadLine();
8     }
9     Console.Write("Enter your stay in hours: ");
10    int hours = Convert.ToInt32(Console.ReadLine());
11    if (hours < 2) {
12       charge = 0;
13    }
14    else {
15       charge = hours * 2;
16    }
17    Console.WriteLine(charge);
```

(a) Rewrite **line 5** in the figure above to **concatenate** the car registration with the string `" is not valid"`, and store the result in the variable `displayMessage`.

Your answer must be written in C#.

(1)

(b) The charge for parking for two or more hours is changed to include an additional £2 fee.

Rewrite **line 15** in the figure above to show this change.

Your answer must be written in C#.

(1)

(Total 2 marks)

9. The two C# programs in the figure below output the value that is equivalent to adding together the integers between 1 and an integer entered by the user.

For example, if the user entered the integer `5`, both programs would output `15`

Program A

```
Console.Write("Enter a number: ");
int num = Convert.ToInt32(Console.ReadLine());
int total = 0;
for (int i = 1; i < num + 1; i++) {
   total = total + i; }
Console.WriteLine(total);
```

Program B

```
Console.Write("Enter a number: ");
int num1 = Convert.ToInt32(Console.ReadLine());
int num2 = num1 + 1;
num2 = num1 * num2;
num2 = num2 / 2;
Console.WriteLine(num2);
```

(a) Shade **one** lozenge to indicate which of the statements is true about the programs in the figure above.

A Both programs are equally efficient. ⬭

B Program A is more efficient than Program B. ⬭

C Program B is more efficient than Program A. ⬭

(1)

(b) Justify your answer for part (a).

(2)

(Total 3 marks)

10. An algorithm, that uses the modulus operator, has been represented using pseudo-code in **Figure 1**.

- Line numbers are included but are not part of the algorithm.

Figure 1

```
1       i ← USERINPUT
2       IF i MOD 2 = 0 THEN
3          OUTPUT i * i
4       ELSE
5          OUTPUT i
6       ENDIF
```

The modulus operator is used to calculate the remainder after dividing one integer by another.

For example:
- `14 MOD 3` evaluates to `2`
- `24 MOD 5` evaluates to `4`

(a) Shade **one** lozenge that shows the line number where selection is **first** used in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

(b) Shade **one** lozenge that shows the output from the algorithm in **Figure 1** when the user input is `4`

A `0` ⬭

B `2` ⬭

C `4` ⬭

D `8` ⬭

E `16` ⬭

(1)

(c) Shade **one** lozenge that shows the line number where assignment is **first** used in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

(d) Shade **one** lozenge that shows the line number that contains a relational operator in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

Figure 1 has been included again below.

Figure 1

```
1       i ← USERINPUT
2       IF i MOD 2 = 0 THEN
3          OUTPUT i * i
4       ELSE
5          OUTPUT i
6       ENDIF
```

(e) Shade **one** lozenge to show which of the following is a **true** statement about the algorithm in **Figure 1**.

A This algorithm uses a Boolean operator. ⬭

B This algorithm uses a named constant. ⬭

C This algorithm uses iteration. ⬭

D This algorithm uses the multiplication operator. ⬭

(1)

(f) **Figure 2** shows an implementation of the algorithm in **Figure 1** using the Python programming language.

- Line numbers are included but are not part of the program.

Figure 2

```
1       i = int(input("Enter a number: "))
2       if i % 2 == 0:
3          print(i * i)
4       else:
5          print(i)
```

The program in **Figure 2** needs to be changed so that it repeats five times using **definite** (count controlled) iteration.

Shade **one** lozenge next to the program that does this correctly.

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

B	`for x in range(0, 6):`
	`  i = int(input("Enter a number: "))`
	`  if i % 2 == 0:`
	`  print(i * i)`
	`  else:`
	`  print(i)`
		⬭

C	`x = 1`
	`while x != 6:`
	`  i = int(input("Enter a number: "))`
	`  if i % 2 == 0:`
	`  print(i * i)`
	`  else:`
	`  print(i)`
	`  x = x + 1`
		⬭

D	`x = 6`
	`while x != 0:`
	`  i = int(input("Enter a number: "))`
	`  if i % 2 == 0:`
	`  print(i * i)`
	`  else:`
	`  print(i)`
	`  x = x – 1`
		⬭

(1)

(Total 6 marks)

11. The two Python programs in the figure below output the value that is equivalent to adding together the integers between 1 and an integer entered by the user.

For example, if the user entered the integer `5`, both programs would output `15`

Program A

```
print("Enter a number: ")
num = int(input())
total = 0
for i in range(1, num + 1):
   total = total + i
print(total)
```

Program B

```
print("Enter a number: ")
num1 = int(input())
num2 = num1 + 1
num2 = num1 * num2
num2 = num2 // 2
print(num2)
```

(a) Shade **one** lozenge to indicate which of the statements is true about the programs in the figure above.

A Both programs are equally efficient. ⬭

B Program A is more efficient than Program B. ⬭

C Program B is more efficient than Program A. ⬭

(1)

(b) Justify your answer for part (a).

(2)

(Total 3 marks)

12. An algorithm, that uses the modulus operator, has been represented using pseudo-code in **Figure 1**.

- Line numbers are included but are not part of the algorithm.

Figure 1

```
1       i ← USERINPUT
2       IF i MOD 2 = 0 THEN
3          OUTPUT i * i
4       ELSE
5          OUTPUT i
6       ENDIF
```

The modulus operator is used to calculate the remainder after dividing one integer by another.

For example:
- `14 MOD 3` evaluates to `2`
- `24 MOD 5` evaluates to `4`

(a) Shade **one** lozenge that shows the line number where selection is **first** used in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

(b) Shade **one** lozenge that shows the output from the algorithm in **Figure 1** when the user input is `4`

A `0` ⬭

B `2` ⬭

C `4` ⬭

D `8` ⬭

E `16` ⬭

(1)

(c) Shade **one** lozenge that shows the line number where assignment is **first** used in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

(d) Shade **one** lozenge that shows the line number that contains a relational operator in the algorithm in **Figure 1**.

A Line number 1 ⬭

B Line number 2 ⬭

C Line number 3 ⬭

D Line number 4 ⬭

(1)

Figure 1 has been included again below.

Figure 1

```
1       i ← USERINPUT
2       IF i MOD 2 = 0 THEN
3          OUTPUT i * i
4       ELSE
5          OUTPUT i
6       ENDIF
```

(e) Shade **one** lozenge to show which of the following is a **true** statement about the algorithm in **Figure 1**.

A This algorithm uses a Boolean operator. ⬭

B This algorithm uses a named constant. ⬭

C This algorithm uses iteration. ⬭

D This algorithm uses the multiplication operator. ⬭

(1)

(f) **Figure 2** shows an implementation of the algorithm in **Figure 1** using the VB.Net programming language.

- Line numbers are included but are not part of the program.

Figure 2

```
1       Console.Write("Enter a number: ")
2       Dim i As Integer = Console.ReadLine()
3       If i Mod 2 = 0 Then
4          Console.WriteLine(i * i)
5       Else
6          Console.WriteLine(i)
7       End If
```

The program in **Figure 2** needs to be changed so that it repeats five times using **definite** (count controlled) iteration.

Shade **one** lozenge next to the program that does this correctly.

A	`Dim x As Integer = 1`
	`While x <> 6`
	`  Console.Write("Enter a number: ")`
	`  Dim i As Integer = Console.ReadLine()`
	`  If i Mod 2 = 0 Then`
	`  Console.WriteLine(i * i)`
	`  Else`
	`  Console.WriteLine(i)`
	`  End If`
	`  x = x + 1`
	`End While`
		⬭

B	`Dim x As Integer = 6`
	`While x <> 0`
	`  Console.Write("Enter a number: ")`
	`  Dim i As Integer = Console.ReadLine()`
	`  If i Mod 2 = 0 Then`
	`  Console.WriteLine(i * i)`
	`  Else`
	`  Console.WriteLine(i)`
	`  End If`
	`  x = x - 1`
	`End While`
		⬭

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

D	`For x As Integer = 0 To 5`
	`  Console.Write("Enter a number: ")`
	`  Dim i As Integer = Console.ReadLine()`
	`  If i Mod 2 = 0 Then`
	`  Console.WriteLine(i * i)`
	`  Else`
	`  Console.WriteLine(i)`
	`  End If`
	`Next`
		⬭

(1)

(Total 6 marks)

13. The two VB.Net programs in the figure below output the value that is equivalent to adding together the integers between 1 and an integer entered by the user.

For example, if the user entered the integer `5`, both programs would output `15`

Program A

```
Console.Write("Enter a number: ")
Dim num As Integer = Console.ReadLine()
Dim total As Integer = 0
For i = 1 To num
   total = total + i
Next
Console.WriteLine(total)
```

Program B

```
Console.Write("Enter a number: ")
Dim num1 As Integer
num1 = Console.ReadLine()
Dim num2 As Integer = num1 + 1
num2 = num1 * num2
num2 = num2 \ 2
Console.WriteLine(num2)
```

(a) Shade **one** lozenge to indicate which of the statements is true about the programs in the figure above.

A Both programs are equally efficient. ⬭

B Program A is more efficient than Program B. ⬭

C Program B is more efficient than Program A. ⬭

(1)

(b) Justify your answer for part (a).

(2)

(Total 3 marks)

14. Match the computer science process to each correct label.

You should write a label **A**–**F** next to each process.

You should **not** use the same label more than once.

A Abstraction

B Data validation

C Decomposition

D Efficiency

E Random number generation

F Variable assignment

Process	**Label (A–F)**
Breaking down a problem into sub-problems.	
Removing unimportant details.	
Ensuring the user enters data that is allowed, for example within a correct range.	

(Total 3 marks)

15. The algorithm shown in the code below is designed to help an athlete with their training. It uses two subroutines `getBPM` and `wait`:

- `getBPM()` returns the athlete's heart rate in beats per minute from an external input device
- `wait(n)` pauses the execution of the algorithm for `n` seconds, so `wait(60)` would pause the algorithm for 60 seconds.

Line numbers have been included but are not part of the algorithm.

```
1       seconds ← 0
2       rest ← 50
3       REPEAT
4          bpm ← getBPM()
5          effort ← bpm - rest
6          IF effort ≤ 30 THEN
7             OUTPUT 'faster'
8          ELSE
9             IF effort ≤ 50 THEN
10               OUTPUT 'steady'
11            ELSE
12               OUTPUT 'slower'
13            ENDIF
14         ENDIF
15         wait(60)
16         seconds ← seconds + 60
17      UNTIL seconds > 200
```

(a) State the most appropriate data type of the variable `seconds` in the algorithm shown in the code above.

(1)

(b) Explain why `rest` could have been defined as a constant in the algorithm shown in the code above.

(1)

(c) State the line number where iteration is first used in the algorithm shown in the code above.

(1)

(d) Complete the trace table for the algorithm shown in the code above.

Some values have already been entered in the trace table:

- the first value of `seconds`
- the values returned by the subroutine `getBPM` that are assigned to the variable `bpm`.

You may not need to use all rows of the trace table.

seconds	**bpm**	**effort**	**OUTPUT**
0	70		
	80		
	100		
	120		

(4)

(Total 7 marks)

16. A developer is writing a program to convert a sequence of integers that represent playing cards to Unicode text.

The developer has identified that they need to create the subroutines shown in **Figure 1** to complete the program.

Figure 1

Subroutine	**Purpose**
`getSuit(n)`	Returns: • the string '`hearts`' if `n` is `0` • the string '`diamonds`' if `n` is `1` • the string '`spades`' if `n` is `2` • the string '`clubs`' if `n` is `3`.
`getRank(n)`	Returns the number value of the card as a string, for example: • if `n` is `1` then '`ace`' is returned • if `n` is `2` then '`two`' is returned • if `n` is `10` then '`ten`' is returned • if `n` is `11` then '`jack`' is returned.
`convert(cards)`	Returns the complete string representation of the array `cards`. For example: • if `cards` is `[3, 1]`, the string returned would be `'three of diamonds '` • if `cards` is `[1, 0, 5, 2, 7, 0]`, the string returned would be `'ace of hearts five of spades seven of hearts '`.

(a) Explain how the developer has used the structured approach to programming.

(2)

(b) State **two** benefits to the developer of using the three separate subroutines described in **Figure 1** instead of writing the program without using subroutines.

(2)

(c) **Figure 2** shows the subroutine `convert` described in **Figure 1**.

Some parts of the subroutine have been replaced with the labels **L1** to **L5**.

Figure 2

```
SUBROUTINE convert(cards)
   result ← ''
   max ← LEN(cards)
   index ← 0
   WHILE index < L1
      rank ← L2 (cards[index])
      suit ← getSuit(cards[L3 + 1])
      c ← rank + ' of ' + suit + ' '
      result ← result + L4
      index ← index + 2
   ENDWHILE
   RETURN L5
ENDSUBROUTINE
```

State the pseudo-code that should be written in place of the labels in the subroutine written in **Figure 2**.

L1 _____

L2 _____

L3 _____

L4 _____

L5 _____

(5)

(Total 9 marks)

17. (a) This is one row of a bitmap image that uses different shades of grey:

This row is stored using the following numbers to represent the different shades of grey:

56	34	0	99	72	23

The algorithm shown in the code below uses this row.

```
row ← [56, 34, 0, 99, 72, 23]
newRow ← [0, 0, 0, 0, 0, 0]
FOR i ← 0 TO 5
   IF row[i] > 50 THEN
      newRow[i] ← 99
   ENDIF
ENDFOR
```

Complete the trace table for the algorithm shown in the code above. The first values have already been entered. You may not need to use all rows of the trace table.

i	newRow					
	0	1	2	3	4	5
	0	0	0	0	0	0
0						

(3)

(b) State the purpose of the algorithm shown in the code above.

(1)

(Total 4 marks)

18. Develop an algorithm, using either pseudo-code or a flowchart, that checks if the user has entered a string that represents a valid machine code instruction.

The machine code instruction is valid if it contains exactly eight characters **and** all of those characters are either '`0`' or '`1`'.

The algorithm should:

- prompt the user to enter an 8-bit machine code instruction and store it in a variable
- check that the instruction only contains the characters '`0`' or '`1`'
- check that the instruction is exactly eight characters long
- output '`ok`' when the instruction is valid, otherwise it should output '`wrong`'.

For example:

- if the user enters the string '`00101110`' it should output '`ok`'
- if the user enters the string '`11110`' it should output '`wrong`'
- if the user enters the string '`1x011001`' it should output '`wrong`'.

(Total 9 marks)

19. State the comparisons that would be made when the **linear search algorithm** is used to search for the value 8 in the following array (array indices have been included above the array).

0	1	2	3	4	5	6
4	7	8	13	14	15	17

(Total 3 marks)

20. State the comparisons that would be made when the **binary search algorithm** is used to search for the value 8 in the following array (array indices have been included above the array).

0	1	2	3	4	5	6
4	7	8	13	14	15	17

(Total 3 marks)

21. State why binary search is considered a better algorithm than linear search.

(Total 1 mark)

22. The algorithm in the code below is a new search algorithm.

```
arr ← [3, 4, 6, 7, 11, 14, 17, 18, 34, 42]
value ← 21
found ← False
finished ← False
i ← 0
down ← False
WHILE (found = False) AND (finished = False)
   IF arr[i] = value THEN
      found ← True
   ELSE
      IF arr[i] > value THEN
         down ← True
         i ← i – 1
      ELSE
         IF (arr[i] < value) AND (down = True) THEN
            finished ← True
         ELSE
            i ← i + 4
         ENDIF
      ENDIF
   ENDIF
ENDWHILE
```

Complete the trace table for the algorithm in the code above. The first row has been completed for you. You may not need to use all rows of the trace table.

found	**finished**	**i**	**down**
False	False	0	False

(Total 4 marks)

23. The code below shows an algorithm.

```
x ← True
y ← False
IF NOT (x AND y) THEN
   OUTPUT 'A'
   IF NOT((NOT x) OR (NOT y)) THEN
      OUTPUT 'B'
   ELSE
      OUTPUT 'C'
   ENDIF
ELSE
   OUTPUT 'D'
   IF (NOT x) AND (NOT y) THEN
      OUTPUT 'E'
   ELSE
      OUTPUT 'F'
   ENDIF
ENDIF
```

State the output from the algorithm shown in the code above.

(Total 2 marks)

24. Number the following lines of code in order (**1**–**4**) so that they create an algorithm where the final value of the variable `n` is 13.

The `LEFTSHIFT` operator performs a binary left shift.
For example, `4 LEFTSHIFT 2` would left shift the value 4 twice.

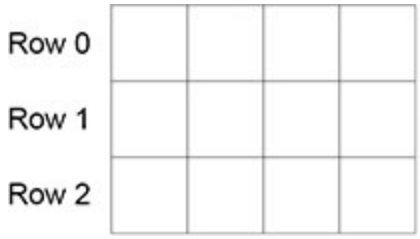
Line of code	**Position (1–4 where 1 is the first line)**
`t ← t - 1`	
`n ← t – n`	
`n ← 2`	
`t ← n LEFTSHIFT 3`	

(Total 3 marks)

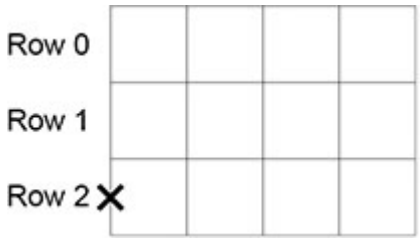
25. The **Algebraic Patent Sewing Machine** is a programmable sewing machine that creates patterns on rows of cloth. It is controlled by writing programs that use the following subroutines:

Subroutine	Description
`gotoRow(n)`	start the sewing machine needle at the left-hand side of row `n`
`move(n)`	move the needle forward by `n` cells without producing a pattern
`shape(s)`	produce shape `s` where `s` can be '`square`' or '`circle`' and move the needle to the next cell
`atEnd()`	returns `True` if the needle is at the end of the row or `False` otherwise

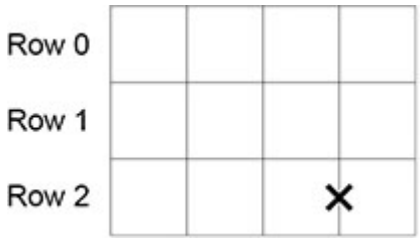
For example, if the cloth looks like this to begin with:



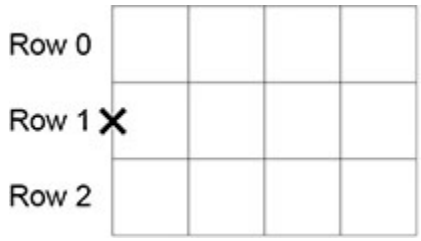
The subroutine call `gotoRow(2)` will place the sewing machine needle at the point shown by the black cross:



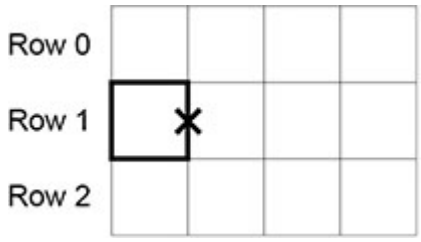
The subroutine call `move(3)` will move the sewing machine needle to the point shown by the black cross:



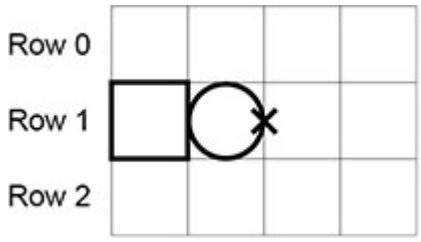
The subroutine call `gotoRow(1)` will move the sewing machine needle to the point shown by the black cross:



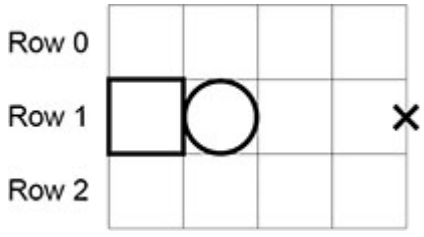
The subroutine call `shape('square')` will draw the following pattern and move the sewing machine needle to the point shown by the black cross:



And finally, the subroutine call `shape('circle')` will draw the following pattern and move the sewing machine needle to the point shown by the black cross:



All of the previous positions of the sewing machine needle would result in the subroutine call `atEnd()` returning `False`, however in the following example `atEnd()` would return `True`:

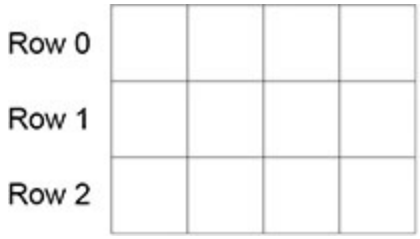


(a) Draw the final pattern after the following algorithm has executed.

```
gotoRow(0)
WHILE atEnd() = False
   shape('square')
   move(1)
ENDWHILE
gotoRow(1)
shape('circle')
move(1)
IF atEnd() = True THEN
   gotoRow(2)
ELSE
   move(1)
ENDIF
shape('square')
```

You should draw your answer on the following grid.

You do not need to show the position(s) of the needle in your answer.



(4)

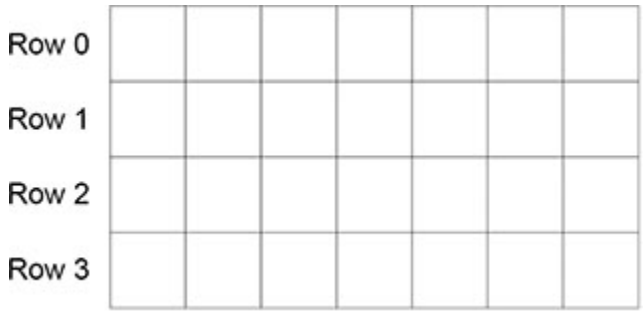
(b) Draw the final pattern after the following algorithm has executed.

This question uses the `MOD` operator. `MOD` calculates the remainder after integer division, for example `7 MOD 5 = 2`.

```
patterns ← ['circle', 'square', 'square', 'circle']
r ← 2
FOR k ← 0 TO 3
   gotoRow(k MOD r)
   move(k + 1)
   shape(patterns[k])
ENDFOR
```

You should draw your answer on the following grid.

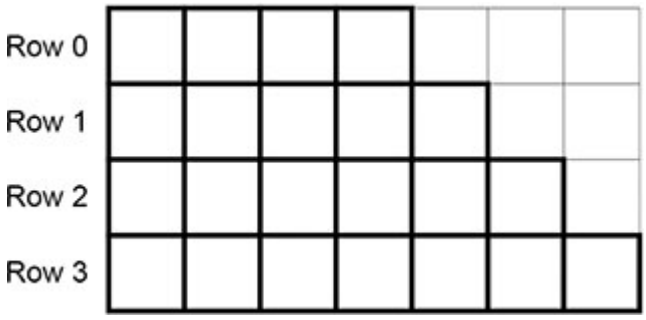
You do not need to show the position(s) of the needle in your answer.



(4)

(c) Develop an algorithm, using either pseudo-code or a flowchart, to produce the pattern shown in the diagram below.

To gain full marks your answer must make appropriate use of iteration.



(4)

(Total 12 marks)

26. The algorithm shown below converts binary data entered as a string by the user into a representation of a black and white image.

The algorithm uses the + operator to concatenate two strings.

Characters in the string are indexed starting at zero. For example `bdata[2]` would access the third character of the string stored in the variable `bdata`

The `MOD` operator calculates the remainder after integer division, for example
`17 MOD 5 = 2`

```
bdata ← USERINPUT
image ← ''
FOR i ← 0 TO LEN(bdata) - 1
     IF bdata[i] = '0' THEN
          image ← image + '*'
     ELSE
          image ← image + '/'
     ENDIF
     IF i MOD 3 = 2 THEN
          OUTPUT image
          image ← ''
     ENDIF
ENDFOR
```

Complete the trace table for the algorithm shown above when the variable `bdata` is given the following value from the user:

`110101`

You may not need to use every row in the table. The algorithm output is not required.

i	image

(Total 3 marks)

27. Describe how the linear search algorithm works.

(Total 3 marks)

28. Develop an algorithm, using either pseudo-code **or** a flowchart, that:

- initialises a variable called `regValid` to `False`
- sets a variable called `regValid` to `True` if the string contained in the variable `reg` is an uppercase `R` followed by the character representation of a single numeric digit.

Examples:

- if the value of `reg` is `R0` or `R9` then `regValid` should be `True`
- if the value of `reg` is `r6` or `Rh` then `regValid` should be `False`

You may wish to use the subroutine `isDigit(ch)` in your answer. The subroutine `isDigit` returns `True` if the character parameter `ch` is a string representation of a digit and `False` otherwise.

(Total 3 marks)

29. The algorithms shown in **Figure 1** and **Figure 2** both have the same purpose.

The operator `LEFTSHIFT` performs a binary shift to the left by the number indicated.

For example, `6 LEFTSHIFT 1` will left shift the number `6` by one place, which has the effect of multiplying the number `6` by two giving a result of 12

Figure 1

```
result ← number LEFTSHIFT 2
result ← result – number
```

Figure 2

```
result ← 0
FOR x ← 1 TO 3
     result ← result + number
ENDFOR
```

(a) Complete the trace table for the algorithm shown in **Figure 1** when the initial value of `number` is 4

You may not need to use all rows of the trace table.

result

(2)

(b) Complete the trace table for the algorithm shown in **Figure 2** when the initial value of `number` is 4

You may not need to use all rows of the trace table.

x	result

(2)

(c) The algorithms in **Figure 1** and **Figure 2** have the same purpose.

State this purpose.

(1)

(d) Explain why the algorithm shown in **Figure 1** can be considered to be a more efficient algorithm than the algorithm shown in **Figure 2**.

(1)

(Total 6 marks)

30. Show the steps involved, for either the bubble sort algorithm **or** the merge sort algorithm, to sort the array shown below so the result is `[1, 4, 5, 8]`

`[8, 4, 1, 5]`

Circle the algorithm you have chosen:

Bubble sort Merge sort

(Total 4 marks)

31. (a) Four subroutines are shown below.

```
SUBROUTINE main(k)
      OUTPUT k
      WHILE k > 1
            IF isEven(k) = True THEN
                  k ← decrease(k)
            ELSE
                  k ← increase(k)
            ENDIF
            OUTPUT k
      ENDWHILE
ENDSUBROUTINE

SUBROUTINE decrease(n)
      result ← n DIV 2
      RETURN result
ENDSUBROUTINE

SUBROUTINE increase(n)
      result ← (3 * n) + 1
      RETURN result
ENDSUBROUTINE

SUBROUTINE isEven(n)
      IF (n MOD 2) = 0 THEN
            RETURN True
      ELSE
            RETURN False
      ENDIF
ENDSUBROUTINE
```

Complete the table showing **all** of the outputs from the subroutine call `main(3)`

The first output has already been written in the trace table. You may not need to use all rows of the table.

Output
3

(4)

(b) Describe how the developer has used the structured approach to programming.

(2)

(Total 6 marks)

32. The subroutine `CODE_TO_CHAR` can be used to convert a character code into the corresponding Unicode character. For example:

`CODE_TO_CHAR(97)` will return the character '`a`'

`CODE_TO_CHAR(65)` will return the character '`A`'

The subroutine `CHAR_TO_CODE` can be used to convert a Unicode character into the corresponding character code. For example:

`CHAR_TO_CODE('a')` will return the integer `97`

`CHAR_TO_CODE('A')` will return the integer `65`

(a) Shade **one** lozenge to show what value would be returned from the subroutine call `CODE_TO_CHAR(100)`

A 'c' ⬭

B 'd' ⬭

C 'e' ⬭

D 'f' ⬭

(1)

(b) State the value that will be returned from the subroutine call:

```
CODE_TO_CHAR(CHAR_TO_CODE('E') + 2)
```

(1)

(c) Write a subroutine `TO_LOWER`, using either pseudo-code **or** a flowchart, that takes an upper case character as a parameter and returns the corresponding lower case character.

For example, if the subroutine `TO_LOWER` is passed the character **'**`A`**'** as a parameter, the subroutine should return the character **'**`a`**'**.

You should make use of the subroutines `CODE_TO_CHAR` and `CHAR_TO_CODE` in your answer.

You can assume that the parameter passed to the subroutine will be in upper case.

(5)

(Total 7 marks)

33. An application allows only two users to log in. Their usernames are stated in the table along with their passwords.

username	password
gower	9Fdg3
tuff	888rG

Develop an algorithm, using either pseudo-code **or** a flowchart, that authenticates the user. The algorithm should:

- get the user to enter their username and password
- check that the combination of username and password is correct and, if so, output the string '`access granted`'
- get the user to keep re-entering their username and password until the combination is correct.

(Total 6 marks)

34. Develop an algorithm, using either pseudo-code **or** a flowchart, that helps an ice cream seller in a hot country calculate how many ice creams they are likely to sell on a particular day. Your algorithm should:

- get the user to enter whether it is the weekend or a weekday
- get the user to enter the temperature forecast in degrees Celsius (they should enter a number between 20 and 45 inclusive; if the number falls outside of this range then they should be made to re-enter another number until they enter a valid temperature)
- calculate the number of ice creams that are likely to be sold using the following information:
  - 100 ice creams are likely to be sold if the temperature is between 20 and 30 degrees inclusive,
  - 150 ice creams are likely to be sold if the temperature is between 31 and 38 degrees inclusive,
  - and 120 ice creams are likely to be sold if the temperature is higher than 38 degrees
- double the estimate if it is a weekend
- output the estimated number of ice creams that are likely to be sold.

(Total 9 marks)

35. A developer has written a set of subroutines to control an array of lights. The lights are indexed from zero. They are controlled using the subroutines in the table.

Subroutine	**Explanation**
`SWITCH(n)`	If the light at index `n` is on it is set to off. If the light at index `n` is off it is set to on.
`NEIGHBOUR(n)`	If the light at index `(n+1)` is on, the light at index `n` is also set to on. If the light at index `(n+1)` is off, the light at index `n` is also set to off.
`RANGEOFF(m, n)`	All the lights between index `m` and index `n` (but **not** including `m` and `n`) are set to off.

Array indices are shown above the array of lights.

For example, if the starting array of the lights is

0	1	2	3
off	on	off	on

Then after the subroutine call `SWITCH(2)` the array of lights will become

0	1	2	3
off	on	on	on

And then after the subroutine call `NEIGHBOUR(0)` the array of lights will become

0	1	2	3
on	on	on	on

Finally, after the subroutine call `RANGEOFF(0, 3)` the array of lights will become

0	1	2	3
on	off	off	on

(a) If the starting array of lights is

0	1	2	3	4	5	6
on	off	off	on	off	off	on

What will the array of lights become after the following algorithm has been followed?

```
a ← 2
SWITCH(a)
SWITCH(a + 1)
NEIGHBOUR(a - 2)
```

Write your final answer in the following array

0	1	2	3	4	5	6

(3)

(b) If the starting array of lights is

0	1	2	3	4	5	6
off	off	on	off	on	on	on

What will the array of lights become after the following algorithm has been followed?

```
FOR a ← 0 TO 2
     SWITCH(a)
ENDFOR
b ← 8
RANGEOFF((b / 2), 6)
NEIGHBOUR(b - 4)
```

Write your final answer in the following array

0	1	2	3	4	5	6

(3)

(c) If the starting array of lights is

0	1	2	3	4	5	6
off	on	off	on	off	on	off

What will the array of lights become after the following algorithm has been followed?

```
a ← 0
WHILE a < 3
     SWITCH(a)
     b ← 5
     WHILE b ≤ 6
          SWITCH(b)
          b ← b + 1
     ENDWHILE
     a ← a + 1
ENDWHILE
```

Write your final answer in the following array

0	1	2	3	4	5	6

(3)

(d) If the starting array of lights is

0	1	2	3	4	5	6
on	on	on	on	on	on	on

Write an algorithm, using **exactly three** subroutine calls, that means the final array of lights will be

0	1	2	3	4	5	6
off	off	off	off	off	off	off

You must use each of the subroutines `SWITCH`, `NEIGHBOUR` and `RANGEOFF` **exactly once** in your answer. If you do not do this you may still be able to get some marks.

(3)

(Total 12 marks)

36. A cake recipe uses 100 grams of flour and 50 grams of sugar for every egg used in the recipe.

The code below shows the first line of an algorithm that will be used to calculate the amount of flour and sugar required based on the number of eggs being used. The number of eggs is entered by the user.

```
eggsUsed ⟵ USERINPUT
```

(a) Shade **one** lozenge to show which of the following lines of code correctly calculates the amount of flour needed in grams.

A `flourNeeded ⟵ USERINPUT` ⬭

B `flourNeeded ⟵ eggsUsed * USERINPUT` ⬭

C `flourNeeded ⟵ eggsUsed * 100` ⬭

D `flourNeeded ⟵ eggsUsed * 50` ⬭

(1)

(b) Shade **one** lozenge to show which programming technique has been used in all of the lines of code in part (a).

A Assignment ⬭

B Indefinite iteration ⬭

C Nested iteration ⬭

D Selection ⬭

(1)

(c) The developer wants to use validation to ensure that the user can only enter a positive number of eggs, ie one egg or more. The maximum number of eggs that can be used in the recipe is eight.

Develop an algorithm, using either pseudo-code or a flowchart, so that the number of eggs is validated to ensure the user is made to re-enter the number of eggs used until a valid number is entered.

You should assume that the user will always enter an integer.

(4)

(Total 6 marks)

37. (a) Complete the trace table for the algorithm shown below for when the user enters the value 750 when prompted.

```
constant PAYLOAD_SIZE ⟵ 250
constant HEADER_SIZE ⟵ 50
OUTPUT 'Enter the number of bits of data to be sent'
dataToBeSent ⟵ USERINPUT
totalSize ⟵ PAYLOAD_SIZE + HEADER_SIZE
numberOfPackets ⟵ 0
REPEAT
    dataToBeSent ⟵ dataToBeSent - totalSize
    numberOfPackets ⟵ numberOfPackets + 1
UNTIL dataToBeSent ≤ 0
```

totalSize	dataToBeSent	numberOfPackets
	750	

(4)

(b) State why both `PAYLOAD_SIZE` and `HEADER_SIZE` from the algorithm did not need to be included in the trace table.

(1)

(c) Shade **one** lozenge to show which of the following best represents the input and output to / from the algorithm in the pseudocode.

A Input: `dataToBeSent`, output: `numberOfPackets` ⬭

B Input: `numberOfPackets`, output: `totalSize` ⬭

C Input: `totalSize`, output: `dataToBeSent` ⬭

(1)

(d) A developer looks at the algorithm and realises that the use of iteration is unnecessary if they use a combination of the `DIV` and `MOD` operators.

- `DIV` calculates integer division, e.g. `11 DIV 4 = 2`
- `MOD` calculates the remainder after integer division, `e.g. 11 MOD 4 = 3`

The programmer realises that she can rewrite the algorithm by replacing the `REPEAT-UNTIL` structure with code that uses selection, `MOD` and `DIV` instead.

Complete this new algorithm by stating the code that should be written in the boxes labelled **A**, **B** and **C**. This new algorithm should calculate the same final result for the variable `numberOfPackets` as the original algorithm.

```
constant PAYLOAD_SIZE ← 250
constant HEADER_SIZE ← 50
OUTPUT 'Enter the number of bits of data to be sent'
dataToBeSent ← USERINPUT
totalSize ← PAYLOAD_SIZE + HEADER_SIZE
numberOfPackets ← dataToBeSent DIV totalSize
IF [ A ] MOD [ B ] > 0 THEN
  numberOfPackets ← [ C ]
ENDIF
```

(3)

(Total 9 marks)

38. Run length encoding (RLE) is a form of compression that creates frequency / data pairs to describe the original data.

For example, an RLE of the bit pattern `00000011101111` could be `6 0 3 1 1 0 4 1` because there are six `0`s followed by three `1`s followed by one `0` and finally four `1`s.

The algorithm below is designed to output an RLE for a bit pattern that has been entered by the user.

Five parts of the code labelled **L1**, **L2**, **L3**, **L4** and **L5** are missing.

- Note that indexing starts at zero.

```
pattern ⟵ L1
i ⟵ L2
count ⟵ 1
WHILE i < LEN(pattern)-1
    IF pattern[i] L3 pattern[i+1] THEN
        count ⟵ count + 1
    ELSE
        L4
        OUTPUT pattern[i]
        count ⟵ 1
    ENDIF
    L5
ENDWHILE
OUTPUT count
OUTPUT pattern[i]
```

(a) Shade **one** lozenge to show what code should be written at point **L1** of the algorithm.

A `OUTPUT` ⬭

B `'RLE'` ⬭

C `True` ⬭

C `USERINPUT` ⬭

(1)

(b) Shade **one** lozenge to show what value should be written at point **L2** of the algorithm.

A `-1` ⬭

B `0` ⬭

C `1` ⬭

C `2` ⬭

(1)

(c) Shade **one** lozenge to show what operator should be written at point **L3** of the algorithm.

A `=` ⬭

B `≤` ⬭

C `<` ⬭

C `≠` ⬭

(1)

(d) Shade **one** lozenge to show what code should be written at point **L4** of the algorithm.

A `count` ⬭

B `count ⟵ count - 1` ⬭

C `count ⟵ USERINPUT` ⬭

C `OUTPUT count` ⬭

(1)

(e) Shade **one** lozenge to show what code should be written at point **L5** of the algorithm.

A `i ⟵ i * 2` ⬭

B `i ⟵ i + 1` ⬭

C `i ⟵ i + 2` ⬭

C `i ⟵ i DIV 2` ⬭

(1)

(f) State a run length encoding of the series of characters `ttjjeeess`

(2)

(g) A developer implements the algorithm and tests their code to check that it is working correctly. The developer tests it only with the input bit pattern that consists of six zeros and it correctly outputs `6 0`.

Using example test data, state **three** further tests that the developer could use to improve the testing of their code.

(3)

(Total 10 marks)

39. A developer creates the algorithm shown below to provide support for users of a new brand of computer monitor (display).

- Line numbers are included but are not part of the algorithm.

```
1     OUTPUT 'Can you turn it on?'
2     ans ⟵ USERINPUT
3     IF ans = 'no' THEN
4        OUTPUT 'Is it plugged in?'
5        ans ← USERINPUT
6        IF ans = 'yes' THEN
7           OUTPUT 'Contact supplier'
8        ELSE
9           OUTPUT 'Plug it in and start again'
10       ENDIF
11    ELSE
12       OUTPUT 'Is it connected to the computer?'
13       ans ⟵ USERINPUT
14       IF ans = 'yes' THEN
15          OUTPUT 'Contact supplier'
16       ELSE
17          OUTPUT 'Connect it to the computer'
18       ENDIF
19    ENDIF
```

(a) Shade **one** lozenge to show which programming technique is used on line 3 of the algorithm.

A Assignment ⬭

B Iteration ⬭

C Selection ⬭

(1)

(b) Shade **one** lozenge to show the data type of the variable `ans` in the algorithm.

A Date ⬭

B Integer ⬭

C Real ⬭

C String ⬭

(1)

(c) Regardless of what the user inputs, the same number of `OUTPUT` instructions will always execute in the algorithm.

State how many `OUTPUT` instructions will execute whenever the algorithm is run.

(1)

(d) The phrase '`Contact supplier`' appears twice in the algorithm.

State the **two** possible sequences of user input that would result in '`Contact supplier`' being output.

(2)

(e) Another developer looks at the algorithm and makes the following statement.

> "At the moment if the user enters 'y' or 'n' they will sometimes get unexpected results. This problem could have been avoided."

Explain why this problem has occurred and describe what would happen if a user entered 'y' or 'n' instead of 'yes' or 'no'.

You may include references to line numbers in the algorithm where appropriate. You do **not** need to include any additional code in your answer.

(3)

(Total 8 marks)

40. State the comparisons that would be made if the binary search algorithm was used to search for the value 30 in the following array (array indices have been included above the array).

0	1	2	3	4	5	6
1	6	14	21	27	31	35

(Total 3 marks)

41. For a binary search algorithm to work correctly on an array of integers, what property must be true about the array?

(Total 1 mark)

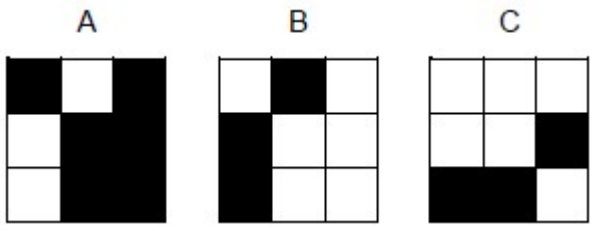
42. A black and white image can be represented as a two-dimensional array where:

- 0 represents a white pixel
- 1 represents a black pixel.

Two images are exact inverses of each other if:

- every white pixel in the first image is black in the second image
- every black pixel in the first image is white in the second image.

For example, B is the inverse of A but C is not the inverse of A:



A developer has started to create an algorithm that compares two 3x3 black and white images, `image1` and `image2`, to see if they are exact inverses of each other.

Complete the algorithm in pseudo-code, ensuring that, when the algorithm ends, the value of the variable `inverse` is `true` if the two images are inverses of each other or `false` if they are not inverses of each other.

The algorithm should work for any 3x3 black and white images stored in `image1` and `image2`.

- Note that indexing starts at zero.

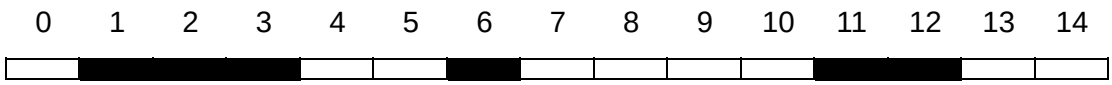
```
image1 ⟵ [   [0, 0, 0],   [0, 1, 1],   [1, 1, 0]   ]
image2 ⟵ [   [1, 1, 1],   [1, 1, 0],   [0, 0, 1]   ]
inverse ⟵ true
i ⟵ 0
WHILE i ≤ 2
    j ⟵ 0
    WHILE j ≤ 2
```

(Total 6 marks)

43. A developer wants to simulate a simple version of the game of Battleships™. The ships are located on a one-dimensional array called `board`. There are always three ships placed on the board:

- one ‘carrier’ that has size three
- one ‘cruiser’ that has size two
- one ‘destroyer’ that has size one.

The size of the board is always 15 squares. A possible starting configuration is shown below where the indices are also written above the board.



The carrier, for example, is found at locations `board[1]`, `board[2]` and `board[3]`.

A player makes a guess to see if a ship (or part of a ship) is located at a particular location. If a ship is found at the location then the player has ‘hit’ the ship at this location.

Every value in the `board` array is 0, 1 or 2.

- The value 0 is used to indicate an empty location.
- The value 1 is used to indicate if a ship is at this location and this location has **not** been hit.
- The value 2 is used to indicate if a ship is at this location and this location has been hit.

The developer identifies one of the sub-problems and creates the subroutine shown in the algorithm below.

```
SUBROUTINE F(board, location)
     h ⟵ board[location]
     IF h = 1 THEN
          RETURN true
     ELSE
          RETURN false
     ENDIF
ENDSUBROUTINE
```

(a) The subroutine uses the values `true` and `false`. Each element of the array `board` has the value `0`, `1` or `2`.

State the most appropriate data type for these values.

Values	Data Type
`true, false`	
`0, 1, 2`	

(2)

(b) The developer has taken the overall problem of the game Battleships and has broken it down into smaller sub-problems.

State the technique that the developer has used.

(1)

(c) The identifier for the subroutine is `F`. This is not a good choice. State a better identifier for this subroutine and explain why you chose it.

(2)

(d) The variable `h` in the subroutine is local to the subroutine. State **two** properties that only apply to local variables.

(2)

(e) Develop a subroutine that works out how far away the game is from ending.

The subroutine should:

- have a sensible identifier
- take the board as a parameter
- work out **and output** how many hits have been made
- work out how many locations containing a ship have yet to be hit and:
  - if 0 then output '`Winner`'
  - if 1, 2 or 3 then output '`Almost there`'.

(11)

(Total 18 marks)

44. Define the term algorithm.

(Total 2 marks)

45. The following are computer science terms (labelled **A – E**).

A assignment
B data type
C decomposition
D efficiency
E input

For each of the definitions in the table, write the label of the most suitable computer science term. Use a label only once.

	Label
Breaking a problem down into a number of sub-problems	
The process of setting the value stored in a variable	
Defines the range of values a variable may take	

(Total 3 marks)

46. The algorithm below has been developed to automate the quantity of dog biscuits to put in a dog bowl at certain times of the day.

- Line numbers are included but are not part of the algorithm.

```
1   time ← USERINPUT
2   IF time = 'breakfast' THEN
3       q ← 1
4   ELSE IF time = 'lunch' THEN
5       q ← 4
6   ELSE IF time = 'dinner' THEN
7       q ← 2
8   ELSE
9       OUTPUT 'time not recognised'
10  ENDIF
11  FOR n ← 1 TO q
12      IF n < 3 THEN
13          DISPENSE_BISCUIT('chewies')
14      ELSE
15          DISPENSE_BISCUIT('crunchy')
16      ENDIF
17  ENDFOR
```

(a) Shade **one** lozenge which shows the line number where selection is **first** used in the algorithm.

A Line number `2` ⬭

B Line number `4` ⬭

C Line number `9` ⬭

D Line number `12` ⬭

(1)

(b) Shade **one** lozenge which shows the line number where iteration is **first** used in the algorithm.

A Line number `1` ⬭

B Line number `8` ⬭

C Line number `11` ⬭

D Line number `13` ⬭

(1)

(c) Shade **one** lozenge which shows how many times the subroutine `DISPENSE_BISCUIT` would be called if the user input is `'breakfast'` in the algorithm.

A 1 subroutine call ⬭

B 2 subroutine calls ⬭

C 3 subroutine calls ⬭

D 4 subroutine calls ⬭

(1)

(d) Shade **one** lozenge which shows the data type of the variable `time` in the algorithm.

A Date/Time ⬭

B String ⬭

C Integer ⬭

D Real ⬭

(1)

(e) State how many times the subroutine `DISPENSE_BISCUIT` will be called with the parameter `'chewies'` if the user input is `'lunch'` in the algorithm.

(1)

(Total 5 marks)

47. The algorithm below is a sorting algorithm.

- Array indexing starts at 0.
- Line numbers are included but are not part of the algorithm.

```
1   arr ← [4, 1, 6]
2   swapsMade ← false
3   WHILE swapsMade = false
4       swapsMade ← true
5       i ← 0
6       WHILE i < 2
7           IF arr[i+1] < arr[i] THEN
8               t ← arr[i]
9               arr[i] ← arr[i+1]
10              arr[i+1] ← t
11              swapsMade ← false
12          ENDIF
13          i ← i + 1
14      ENDWHILE
15  ENDWHILE
```

(a) State the data type of the variable `swapsMade` in the algorithm shown above.

(1)

(b) The identifier `swapsMade` is used in the algorithm shown above.

Explain why this is a better choice than using the identifier `s`.

(2)

(c) Shade **one** lozenge to show which of the following contains the **false** statement about the algorithm.

A The algorithm uses a named constant. ⬭

B The algorithm uses indefinite iteration. ⬭

C The algorithm uses nested iteration. ⬭

(1)

(d) Complete the trace table for the algorithm shown above. Some values have already been entered.

arr			swawpsMade	i	t
[0]	[1]	[2]			
4	1	6	false		

(6)

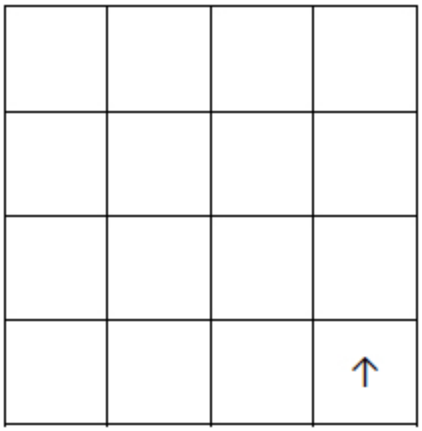
(Total 10 marks)

48. Four separate subroutines have been written to control a robot.

- `Forward(n)` moves the robot `n` squares forward.
- `TurnLeft()` turns the robot 90 degrees left.
- `TurnRight()` turns the robot 90 degrees right.
- `ObjectAhead()` returns `true` if the robot is facing an object in the next square or returns `false` if this square is empty.

(a) Draw the path of the robot through the grid below if the following program is executed (the robot starts in the square marked by the ↑ facing in the direction of the arrow).

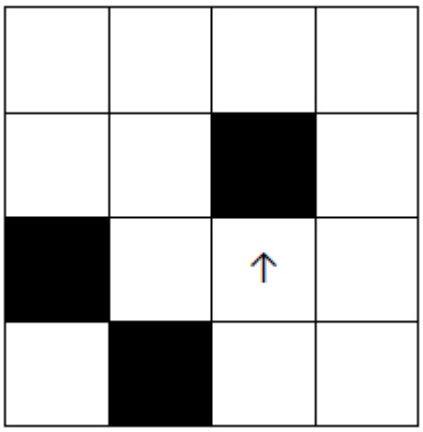
```
Forward(2)
TurnLeft()
Forward(1)
TurnRight()
Forward(1)
```



(3)

(b) Draw the path of the robot through the grid below if the following program is executed (the robot starts in the square marked by the ↑ facing in the direction of the arrow). If a square is black then it contains an object.

```
WHILE ObjectAhead() = true
   TurnLeft()
      IF ObjectAhead() = true THEN
         TurnRight()
         TurnRight()
      ENDIF
   Forward(1)
ENDWHILE
Forward(1)
```



(3)

(Total 6 marks)

49. Fill in the blank arrays to show the steps involved in applying the bubble sort algorithm to the array `[3, 5, 1, 4, 2]`. You only need to show the missing steps where a change is applied to the array.

3	5	1	4	2
1	2	3	4	5

(Total 5 marks)

50. A developer is developing a program for a client. The developer is given the following instructions.

"Many of my friends ask me to walk their dogs for them. All of these friends pay me to do this and the amount I get paid depends on how long I walk their dogs for. If they have more than one dog then I don't charge the owner any extra. I like to walk the dogs in the afternoon when the weather is normally best because I often get colds. I need you to help me keep track of how much I'm owed – fortunately for me all of my friends have different first names so it is really easy to tell them apart. I charge £10 for every 30 minutes of the walk (and I always round this up so 47 minutes would be two half-hour charges or £20)."

(a) The developer needs to remove all of the unnecessary detail from the client's request. Shade the lozenge next to the name for this process.

A Abstraction ⬭

B Conversion ⬭

C Decomposition ⬭

D Validation ⬭

(1)

(b) The developer has decided that the following two points are the only important details from the client's request.

- The charge is based on time and not how many dogs are walked.
- The charge is £10 every 30 minutes.

State **two** other relevant details that the developer has missed.

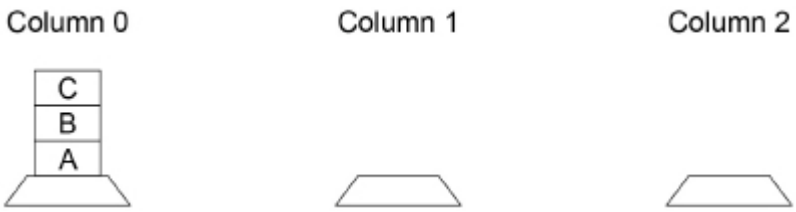
(2)

(Total 3 marks)

51. The following subroutines control the way that labelled blocks are placed in different columns.

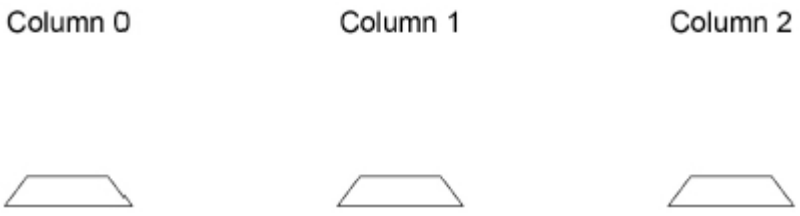
`BLOCK_ON_TOP(column)`	returns the label of the block on top of the column given as a parameter.
`MOVE(source, destination)`	moves the block on top of the `source` column to the top of the `destination` column.
`HEIGHT(column)`	returns the number of blocks in the specified column.

(a) This is how the blocks A, B and C are arranged at the start.



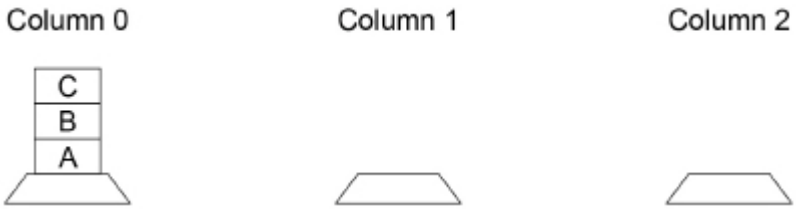
Draw the final arrangement of the blocks after the following algorithm has run.

```
MOVE(0, 1)
MOVE(0, 2)
MOVE(0, 2)
```



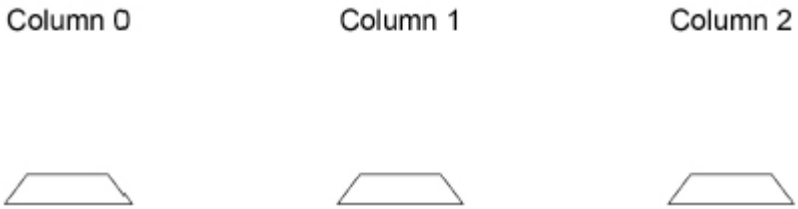
(3)

(b) This is how the blocks A, B and C are arranged at the start.



Draw the final arrangement of the blocks after the following algorithm has run.

```
WHILE HEIGHT(0) > 1
   MOVE(0, 1)
ENDWHILE
MOVE(1, 2)
```



(3)

(Total 6 marks)

52. The following subroutines control the way that labelled blocks are placed in different columns.

`BLOCK_ON_TOP(column)`	returns the label of the block on top of the column given as a parameter.
`MOVE(source, destination)`	moves the block on top of the `source` column to the top of the `destination` column.
`HEIGHT(column)`	returns the number of blocks in the specified column.

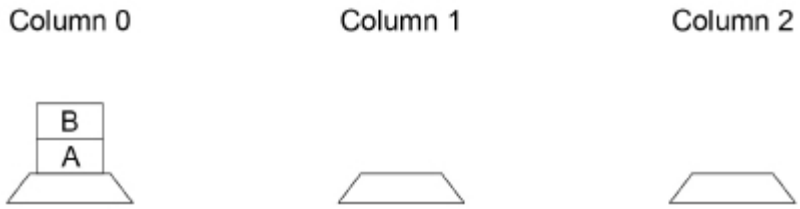
Develop an algorithm using either pseudo-code or a flowchart that will move every block from column 0 to column 1.

Your algorithm should work however many blocks start in column 0. You may assume there will always be at least one block in column 0 at the start and that the other columns are empty.

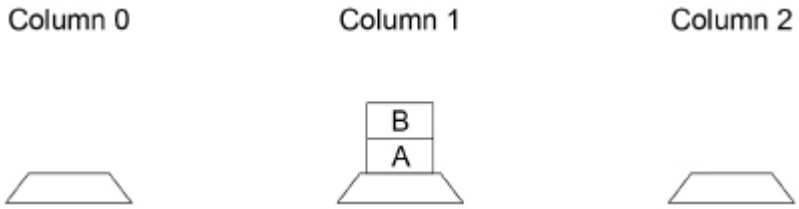
The order of the blocks must be preserved.

The `MOVE` subroutine must be used to move a block from one column to another. You should also use the `HEIGHT` subroutine in your answer.

For example, if the starting arrangement of the blocks is:



Then the final arrangement should have block B above block A:



(Total 4 marks)

53. Define the term algorithm.

(Total 2 marks)

54. The following are computer science terms (labelled **A – E**).

A abstraction
B data type
C decomposition
D efficiency
E input

For each of the definitions in the table, write the label of the most suitable computer science term. Use a label only once.

	Label
Breaking a problem down into a number of sub-problems.	
The process of removing unnecessary detail from a problem.	
Defines the range of values a variable may take.	

(Total 3 marks)

55. The algorithm below is a sorting algorithm.

- Array indexing starts at 0.
- Line numbers are included but are not part of the algorithm.

```
1    arr ← [4, 1, 6]
2    sorted ← false
3    WHILE sorted = false
4       sorted ← true
5       i ← 0
6       WHILE i < 2
7          IF arr[i+1] < arr[i] THEN
8             t ← arr[i]
9             arr[i] ← arr[i+1]
10            arr[i+1] ← t
11            sorted ← false
12         ENDIF
13         i ← i + 1
14      ENDWHILE
15   ENDWHILE
```

(a) State the data type of the variable `sorted` in the algorithm shown above.

(1)

(b) The identifier `sorted` is used in the algorithm shown above.

Explain why this is a better choice than using the identifier `s`.

(2)

(c) Shade **one** lozenge to show which of the following contains the **false** statement about the algorithm above.

A The algorithm uses a named constant ⬭

B The algorithm uses indefinite iteration ⬭

C The algorithm uses nested iteration ⬭

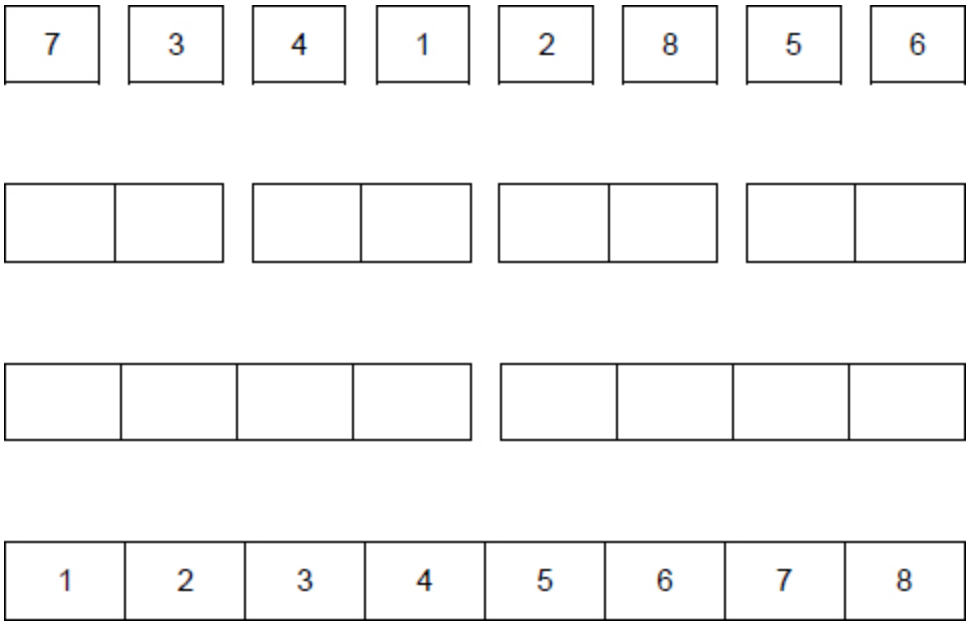
(1)

(d) Complete the trace table for the algorithm shown above. Some values have already been entered.

arr			sorted	i	t
[0]	[1]	[2]			
4	1	6	false		

(6)

(e) Fill in the values in the boxes to show how the merge part of the merge sort algorithm operates. The first and last rows have been completed for you.



(3)

(f) State **one** advantage of the merge sort algorithm compared to the sorting algorithm initially shown.

(1)

(g) A programmer implementing the sorting algorithm decided to create it as a subroutine. Line 1 was removed and the array `arr` was made a parameter of the subroutine.

State **two** reasons why the programmer decided to implement the algorithm as a subroutine.

(2)

(Total 16 marks)

56. The subroutine below is used to authenticate a username and password combination.

- Array indexing starts at 0.
- Line numbers are included but are not part of the algorithm.

```
1    SUBROUTINE Authenticate(user, pass)
2       us ← ['dave', 'alice', 'bob']
3       ps ← ['abf32', 'woof2006', '!@34E$']
4       z ← 0
5       correct ← false
6       WHILE z < 3
7          IF user = us[z] THEN
8             IF pass = ps[z] THEN
9                correct ← true
10            ENDIF
11         ENDIF
12         z ← z + 1
13      ENDWHILE
14      RETURN correct
15   ENDSUBROUTINE
```

(a) Complete the trace table for the following subroutine call:

```
Authenticate('alice', 'woof2006')
```

z	correct

(3)

(b) State the value that is returned by the following subroutine call:

```
Authenticate('bob', 'abf32')
```

(1)

(c) Lines 7 and 8 in the subroutine above could be replaced with a single line. Shade **one** lozenge to show which of the following corresponds to the correct new line.

A `IF user = us[z] OR pass = ps[z] THEN` ⬭

B `IF user = us[z] AND pass = ps[z] THEN` ⬭

C `IF NOT (user = us[z] AND pass = ps[z]) THEN` ⬭

(1)

(d) A programmer implements the subroutine shown above. He replaces line 9 with

```
RETURN true
```

He also replaces line 14 with

```
RETURN false
```

Explain how the programmer has made the subroutine more efficient.

(2)

(Total 7 marks)

57. The following algorithm is used to compare a property of two arrays stored in an array called `arr`.

Note: Line numbers have been included but are not part of the algorithm.

Note: For this algorithm, array indexing starts at 1.

```
1     arr ← [ [3, 2], [4, 3] ]
2     i ← 1
3     h ← 0
4     lenArr ← 2
5     WHILE i ≤ lenArr
6        j ← 1
7        a ← 0
8        WHILE j ≤ lenArr
9           a ← a + arr[i][j]
10          j ← j + 1
11       ENDWHILE
12       IF a > h THEN
13          h ← a
14       ENDIF
15       i ← i + 1
16    ENDWHILE
17    OUTPUT h
```

(a) State the line number where iteration is first used.

(1)

(b) (i) Complete the trace table for this algorithm (the first row has been completed for you).

You may not need to use all the rows in the table.

i	**h**	**j**	**a**
1	0	1	0

(6)

(ii) What does the final value of **h** represent?

(2)

(c) Why could `lenArr` be considered to be a constant in this algorithm?

(1)

(d) Line 1 in the algorithm has been changed to:

```
arr ← [ [3, 2, 1], [4, 3, 1], [1, 1, 1] ]
```

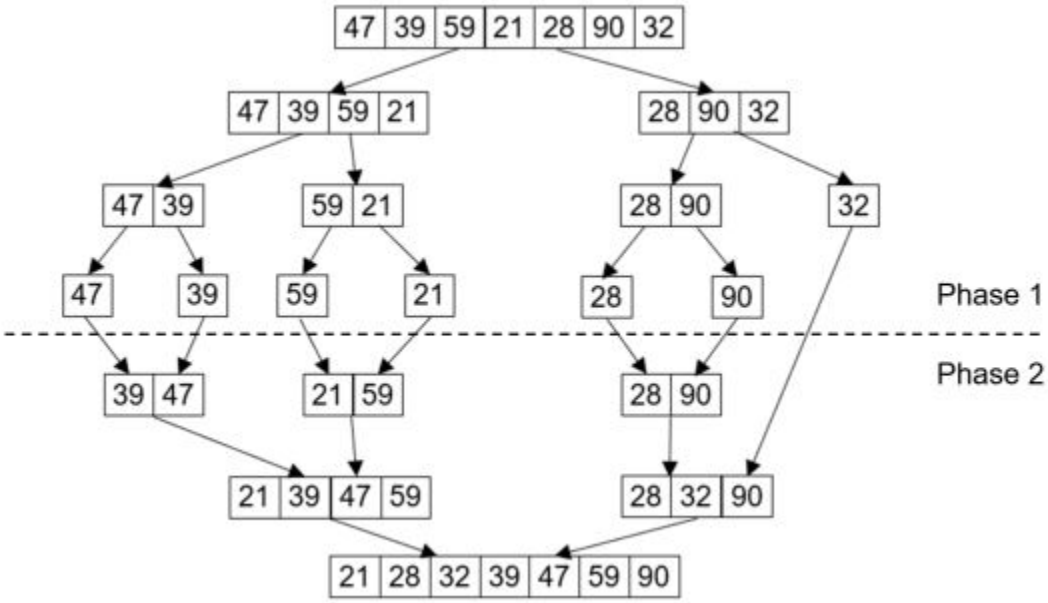
What change will need to be made to line 4 to ensure the algorithm still works as intended?

(1)

(Total 11 marks)

58. The image below shows a merge sort being carried out on a list.

Merge sort being carried out on list



The sort is carried out in two phases. The phases are separated by the dotted line.

(a) Explain the process that is being carried out in **Phase 1** of the sort (above the dotted line).

(3)

(b) Explain the process that is being carried out in **Phase 2** of the sort (below the dotted line).

(3)

(Total 6 marks)

59. State **one** advantage and **one** disadvantage of merge sort when compared to bubble sort.

(Total 2 marks)

60. The code below contains a subroutine that returns a value.

```
SUBROUTINE TotalOut(a, b)
  c ← a + b
  WHILE a < c
    a ← a + 1
    b ← b – a
  ENDWHILE
  RETURN b
ENDSUBROUTINE
```

(a) Complete the trace table below when the subroutine call `TotalOut(3, 4)` is made (you may not need to use all of the rows in the table):

a	b	c

(3)

(b) A programmer mistakenly tries to shorten the subroutine above by replacing the lines:

```
c ← a + b
WHILE a < c
```

With the line:

```
WHILE a < (a + b)
```

Explain why this change is a mistake.

(2)

(c) What value is returned by the subroutine call `TotalOut(x, 0)` where `x` is any positive integer?

(1)

(Total 6 marks)

Mark schemes

1.

Mark is for AO1 (recall)

Removing unnecessary detail (from the problem);

A. data / information in place of detail for this year only

[1]

2.

2 marks for AO2 (apply)

A Confirm / enter email address;

B Log out;

A. any wording with the same meaning

[2]

3.

(a) **Mark is for AO2 (apply)**

D `S;`

R. If more than one lozenge shaded

1

(b) **Mark is for AO2 (apply)**

B `2;`

R. If more than one lozenge shaded

1

(c) **Mark is for AO2 (apply)**

`Sara;`

I. Case

1

(d) **2 marks for AO3 (program)**

Mark A for correct identification of **2, 4**;
Mark B for correct identification of **1**;

Model Answer

```
var ← SUBSTRING(2, 4, name1)
OUTPUT (names[1] + var)
```

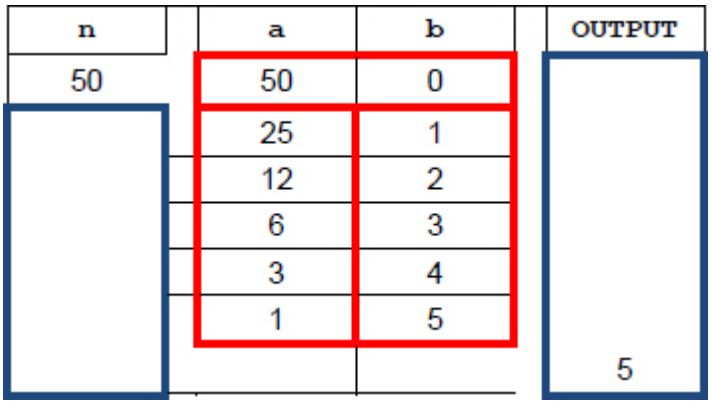
2

[5]

4.

(a) **4 marks for AO2 (apply)**

- 1 mark for the first value of column `a` **and** the first value of column `b` both correct;
- 1 mark for column `a` correctly integer dividing the first value in column `a` by 2 down to 1 and no other values;
- 1 mark for minimum of six values in the `b` column, incrementing by one.
- The number of values in the b column must match the number of values in the a column;
- 1 mark for `OUTPUT` being the final value of `b` and no other values in the output column, and no other values in column `n`; **A.** follow through from column `b`



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

4

(b) **Mark is for AO2 (apply)**

1;

Reject the **word** one

1

(c) **Mark is for AO2 (apply)**

1 mark for giving a new identifier that describes this purpose, eg `count` // `total` // `times` // `numberOfTimes` // `counter`

1

(d) **2 marks for AO2 (apply)**

Maximum of 2 marks from:

- The `REPEAT…UNTIL` structure tests the condition at the end //
- the `WHILE…ENDWHILE` structure tests the condition at the beginning;

- The `REPEAT…UNTIL` structure will always execute at least once //
- the `WHILE…ENDWHILE` loop may never execute;

- If the value of `n` is 1 (or less) then the `REPEAT…UNTIL` structure will cause the value of `a/b` to change, but the `WHILE…ENDWHILE` structure will not;

R. The `REPEAT…UNTIL` structure repeats lines of code until a condition is true
R. The `WHILE…ENDWHILE` structure repeats lines of code until a condition is false

2

[8]

5.

(a) **6 marks for AO3 (program)**
1 mark for each correct item in the correct location

```
SUBROUTINE getSize(sampRate, res, seconds)

    size ← sampRate * res * seconds

    size ← size / 8

    RETURN size

ENDSUBROUTINE


OUTPUT getSize(100, 16, 60)
```

I. Case
R. Incorrect order of parameters

6

(b) **Mark is for AO1 (understanding)**

A variable that is only accessible / visible within the subroutine;
//
A variable that only exists while the subroutine is running;

1

(c) **3 marks for AO1 (understanding)**

Max 3 marks from:

- subroutines can be developed in isolation/independently/separately;
- easier to discover errors // testing is more effective (than without a subroutine);
- subroutines make program code easier to understand; **A.** 'easier to read' for this year only
- subroutines make it easier for a team of programmers to work together on a large project;
- subroutines make it easier to reuse code;

3

[10]

6.

(a) **6 marks for AO2 (apply)**

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

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

I. Different rows used as long as the order within columns is clear
I. Duplicate values on consecutive rows within a column
I. Quotes used around letters
I. Case
Note to Examiners: A. missing middle `c` in **temp** column

6

(b) **Mark is for AO2 (apply)**

Sort (the values in order) // bubble sort // put into alphabetical order;

1

(c) **Mark is for AO2 (apply)**

The algorithm will attempt to access an element/item/index in the array that does not exist;
//
The algorithm will attempt to use an index which is greater than the maximum array index of 2;

1

[8]

7. (a) **Mark is for AO2 (apply)**

B Line number 2;

R. If more than one lozenge shaded

1

(b) **Mark is for AO2 (apply)**

E `16;`

R. If more than one lozenge shaded

1

(c) **Mark is for AO2 (apply)**

A Line number 1;

R. If more than one lozenge shaded

1

(d) **Mark is for AO2 (apply)**

B Line number 2;

R. If more than one lozenge shaded

1

(e) **Mark is for AO2 (apply)**

D This algorithm uses the multiplication operator;

R. If more than one lozenge shaded

1

(f) **Mark is for AO3 (refine)**

C#

A

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

Python

A

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

        print(i)
```

VB.NET

C

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

R. If more than one lozenge shaded

1

[6]

8. (a) **Mark is for AO3 (refine)**

C#

```
string displayMessage = carReg + " is not valid "
```

Python

```
displayMessage = carReg + " is not valid "
```

VB.NET

```
Dim displayMessage As String = carReg + " is not valid " //
Dim displayMessage As String = carReg & " is not valid "
```

I. Case
I. Space between variable outputs
I. Order of strings

1

(b) **Mark is for AO3 (refine)**

C#

```
charge = hours * 2 + 2; //
charge = 2 + hours * 2;
```

Python

```
charge = hours * 2 + 2 //
charge = 2 + hours * 2
```

VB.NET

```
charge = hours * 2 + 2 //
charge = 2 + hours * 2
```

I. Case
I. Parentheses, unless altering result eg, `hours * (2 + 2)`

1

[2]

9.

(a) **Mark is for AO2 (apply)**

C Program B is more efficient than Program A;

R. If more than one lozenge shaded

1

(b) **2 marks for AO2 (apply)**

It will take less time for the computer to execute program B;
because fewer lines of code will be executed;
//
The number of calculations performed is constant in Program B;
but increases as the number input gets bigger in Program A;

A. Program B has fewer variables; so would use less memory (when executing);

2

[3]

10.

(a) **Mark is for AO2 (apply)**

B Line number 2;

R. If more than one lozenge shaded

1

(b) **Mark is for AO2 (apply)**

E `16;`

R. If more than one lozenge shaded

1

(c) **Mark is for AO2 (apply)**

A Line number 1;

R. If more than one lozenge shaded

1

(d) **Mark is for AO2 (apply)**

B Line number 2;

R. If more than one lozenge shaded

1

(e) **Mark is for AO2 (apply)**

D This algorithm uses the multiplication operator;

R. If more than one lozenge shaded

1

(f) **Mark is for AO3 (refine)**

C#

A

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

Python

A

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

VB.NET

C

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

R. If more than one lozenge shaded

1

[6]

11.

(a) **Mark is for AO2 (apply)**

C Program B is more efficient than Program A;

R. If more than one lozenge shaded

1

(b) **2 marks for AO2 (apply)**

It will take less time for the computer to execute program B;
because fewer lines of code will be executed;
//
The number of calculations performed is constant in Program B;
but increases as the number input gets bigger in Program A;

A. Program B has fewer variables; so would use less memory (when executing);

2

[3]

12.

(a) **Mark is for AO2 (apply)**

B Line number 2;

R. If more than one lozenge shaded

1

(b) **Mark is for AO2 (apply)**

E `16;`

R. If more than one lozenge shaded

1

(c) **Mark is for AO2 (apply)**

A Line number 1;

R. If more than one lozenge shaded

1

(d) **Mark is for AO2 (apply)**

B Line number 2;

R. If more than one lozenge shaded

1

(e) **Mark is for AO2 (apply)**

D This algorithm uses the multiplication operator;

R. If more than one lozenge shaded

1

(f) **Mark is for AO3 (refine)**

C#

A

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

Python

A

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

VB.NET

C

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

R. If more than one lozenge shaded

1

[6]

13. (a) **Mark is for AO2 (apply)**

C Program B is more efficient than Program A;

R. If more than one lozenge shaded

1

(b) **2 marks for AO2 (apply)**

It will take less time for the computer to execute program B;
because fewer lines of code will be executed;
//
The number of calculations performed is constant in Program B;
but increases as the number input gets bigger in Program A;

A. Program B has fewer variables; so would use less memory (when executing);

2

[3]

14. **3 marks for AO1 recall**

1 mark for 1 correct label;
2 marks for 2 correct labels;
3 marks for 3 correct labels;

Correct table is:

Process	Label (A–F)
Breaking down a problem into sub-problems.	C
Removing unimportant details.	A
Ensuring the user enters data that is allowed, for example within a correct range.	B

R. all occurrences of a label entered more than once.

[3]

15. (a) **Mark is for AO2**

Integer/int;

A. programming language specific data type

1

(b) **Mark is for AO2**

(The value) doesn't change/vary (after being initialised);

1

(c) **Mark is for AO2**

3 // three;

A. 3rd (line) // third (line);

1

(d) **4 marks for AO2**

1 mark for `seconds` having values 60, 120 and 180 in that order;
1 mark for the final value of `seconds` as 240;
1 mark for the first value of `effort` as 20 **and** the first value of `OUTPUT` as `'faster'`.
1 mark for the last three values in the `effort` column all correct **and** every output correct for these three values of effort;

Max 3 marks if any errors.

I. use of quote marks or minor spelling errors in the `OUTPUT` column.
I. values on different lines as long as the order is correct and no other values have been entered.

Correct table as follows:

seconds	bpm	effort	OUTPUT
0	*70*	20	faster
60	*80*	30	faster
120	*100*	50	steady
180	*120*	70	slower
240			

4

[7]

16. (a) **2 marks for AO2**

Max two marks from the following:

(The developer has…)
decomposed the problem/broken the problem down (into sub-problems);
implemented sub-problems as subroutines;
used interfaces (including parameters and return values);

2

(b) **2 marks for AO1 (understanding)**

Max two marks from the following:

The subroutines will be easier to test/mistakes will be easier to find;
The subroutines can be reused;
The subroutines can be changed without affecting the rest of the program;
The subroutines create better self-documenting code;

2

(c) **5 marks for AO3 (program)**

1 mark for each correct label:

L1 `max` ;

L2 `getRank` ;

L3 `index` ;

L4 `c` ;

L5 `result` ;

5

[9]

17. (a) **3 marks for AO2**

1 mark for `i` column correct;
1 mark for one of indices `0, 3` and `4` assigned the value 99;
1 mark for all of indices `0, 3` and `4` (and no other indices) assigned the value 99;

Max 2 marks if any errors.

Correct table as follows:

i	newRow					
	0	1	2	3	4	5
	0	*0*	*0*	*0*	*0*	*0*
0	99					
1						
2						
3				99		
4					99	
5						

3

(b) **Mark is for AO2**

Converts (row/grey scale image) to black and white // the values 0 and 99 // two colours/shades;

1

[4]

18. **9 marks for AO3 (programming)**

[Mark A] for getting user input and assigning it to a variable;

[Mark B] for using selection to check for the length of user input (even if the Boolean condition is incorrect);

[Mark C] for a correct Boolean condition to check that the length is 8 (or not 8 if opposite logic used) even if not within a selection structure;

***[Mark D]** for iterating over the instruction to check for (in)correct characters;

***[Mark E]** for the iteration structure in **Mark D** isolating every character in the string (even if the subsequent check for validity is incorrect);

[Mark F] for using selection to check if a character is/is not `'0'` or `'1'`;

[Mark G] for a correct Boolean condition checking the character is/is not a `'0'` and/or a `'1'`;

[Mark H] outputting `'ok'` and `'wrong'` based on the length of the user input.

[Mark I] outputting `'ok'` and `'wrong'` based on the characters in the user input.

***A. alternative method for obtaining Mark D and Mark E**
[Mark D] eight selection structures instead of iteration;
[Mark E] ensure every character is checked in **Mark D**;

Max 8 marks if any errors.

An example of a completely correct solution:

```
instruction ← USERINPUT                                  [A]
valid ← True                                        [Part H, Part I]
IF LEN(instruction) ≠ 8 THEN                             [B, C]
  valid ← False                                         [Part H]
ELSE
  FOR i ← 0 TO 7                                         [D, E]
    IF instruction[i] ≠ '0' AND                          [F, G]
        instruction[i] ≠ '1' THEN
      valid ← False                                     [Part I]
    ENDIF
  ENDFOR
ENDIF
IF valid = True THEN
  OUTPUT 'ok'                                       [Part H, Part I]
ELSE
  OUTPUT 'wrong'                                    [Part H, Part I]
ENDIF
```

Another example of a completely correct solution:

```
instruction ← USERINPUT                                  [A]
IF LEN(instruction) = 8 THEN                             [B, C]
   i ← 0                                                 [Part E]
  valid ← True
   WHILE i < 8                                           [D, Part E]
     IF instruction[i] ≠ '0' THEN                        [Part F, Part G]
       IF instruction[i] ≠ '1' THEN                      [Part F, Part G]
         valid ← False
       ENDIF
     ENDIF
     i ← i + 1                                           [Part E]
  ENDWHILE
  IF valid = True THEN
     OUTPUT 'ok'                                         [Part H, Part I]
  ELSE
     OUTPUT 'wrong'                                      [Part I]
  ENDIF
ELSE
  OUTPUT 'wrong'                                         [Part H]
ENDIF
```

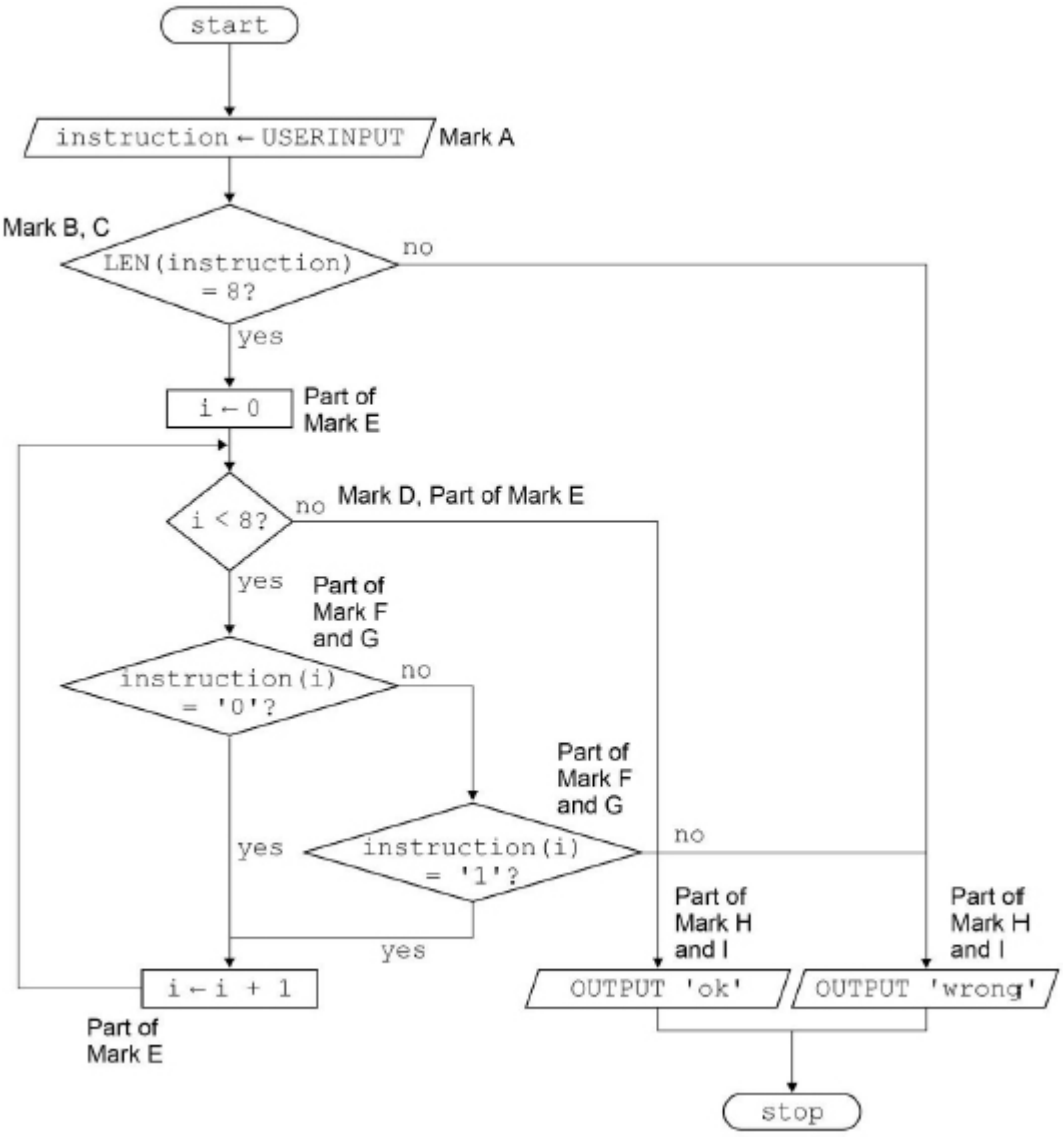
Another example of a completely correct solution:

```
instruction ← USERINPUT                                  [A]
IF LEN(instruction) = 8 THEN                             [B, C]
   valid ← True
  FOR i ← 0 TO 7                                         [D, E]
      IF instruction[i] ≠ '0' THEN                       [Part F, Part G]
        IF instruction[i] ≠ '1' THEN                     [Part F, Part G]
         valid ← False
       ENDIF
     ENDIF
  ENDFOR
  IF valid = True THEN
     OUTPUT 'ok'                                         [Part H, Part I]
  ELSE
     OUTPUT 'wrong'                                      [Part I]
  ENDIF
ELSE
  OUTPUT 'wrong'                                         [Part H]
ENDIF
```

A final example of a completely correct solution that uses a FOR-EACH style loop to iterate over the characters of the string (note that this is not part of the AQA pseudo-code supplement but still perfectly acceptable):

```
instruction ← USERINPUT                                    [A]
valid ← True                                               [Part H, Part I]
IF LEN(instruction) ≠ 8 THEN                               [B, C]
  valid ← False                                            [Part H]
ELSE
  FOR ch IN instruction                                    [D, E]
    IF ch ≠ '0' AND ch ≠ '1' THEN                          [F, G]
      valid ← False                                        [Part I]
    ENDIF
  ENDFOR
ENDIF
IF valid = True THEN
    OUTPUT 'ok'.                                           [Part H, Part I]
ELSE
    OUTPUT 'wrong'                                         [Part H, Part I]
ENDIF
```

An example of a fully correct flowchart solution is:



I. shape of symbols

[9]

19.

3 marks for AO2

(The value 8 is compared to the value) 4; **R.** if not first comparison
(The value 8 is compared to the value) 7; **R.** if not second comparison
(The value 8 is compared to the value) 8; **R.** if not third comparison

Alternatively:

(The value 8 is compared to the) first element of the array;
(The value 8 is compared to) every subsequent value of the array;
When the value 8 is found in the array it returns True;

[3]

20. **3 marks for AO2**

(The value 8 is compared to) 13; **R.** if not first comparison
(The value 8 is compared to) 7; **R.** if not second comparison
(The value 8 is compared to) 8; **R.** if not third comparison

Alternatively:

(The value 8 is compared to the) midpoint of the array;
(The value 8 is compared to the) midpoint of the left subarray ([4, 7, 8]);
(The value 8 is compared to the) midpoint of the right subarray ([8]);

[3]

21. **Mark is for AO1 (understanding)**

It is more efficient // requires fewer steps/comparisons (on average);

[1]

22. **4 marks for AO2**

1 mark for values 4 and 8 in the `i` column (in that order);
1 mark for value 7 as the last value in the `i` column;
1 mark for `down` being set only once to `True;`
1 mark for `finished` being set only once to `True;`

Max 3 marks if any errors.

I. repeated values written in columns
I. exact placing of values as long as the vertical order is correct

Correct table as follows:

found	finished	i	down
False	*False*	*0*	*False*
		4	
		8	True
	True	7	

[4]

23. **2 marks for AO2**

```
A;
C;
```

I. use of quote marks
I. if answers are on the same line or different lines as long as order is clear
R. if more than two characters are stated

[2]

24. **3 marks for AO3 (programming)**

1 mark for 1 correct position;
2 marks for 2 correct positions;
3 marks for 4 correct positions;

R. Any position which is used more than once

Line of code	Position (1–4 where 1 is the first line)
`t ← t - 1`	3
`n ← t - n`	4
`n ← 2`	1
`t ← n LEFTSHIFT 3`	2

[3]

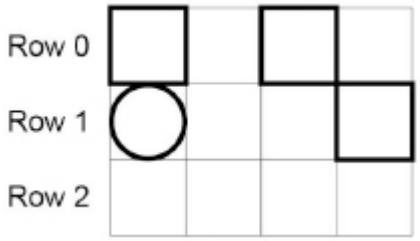
25. (a) **4 marks for AO2**

1 mark for drawing one square at the start of row 0;
1 mark for the remaining 3 cells of row 0 correct;
1 mark for drawing a circle at the start of row 1;
1 mark for the remaining 3 cells of row 1 correct;

I. any marks that indicate the position of the needle.

Max 3 marks if any errors.

The completed pattern is as follows:



4

(b) **4 marks for AO2**

1 mark for drawing a circle in the second cell of row 0 and having no shape in the first cell of row 0;
1 mark for drawing exactly four shapes;

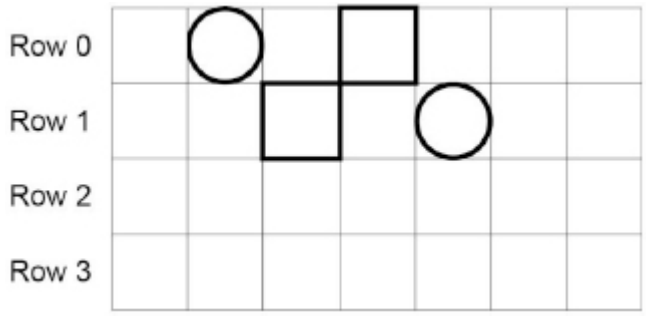
(If more than 4 shapes are drawn, stop marking)

1 mark for drawing a square in the third column of row 1;
1 mark for drawing a square in the fourth column and a circle in the fifth column;

I. any marks that indicate the position of the needle.

Max 3 marks if any errors.

The completed pattern is:



4

(c) **4 marks for AO3 (programming)**

[Mark A] use of the `gotoRow` subroutine with parameters 0, 1, 2 and 3;

[Mark B] use of `shape('square')` to draw four squares in row 0;

[**Mark C]** use of iteration to repeatedly draw the squares;

[Mark D] correct squares drawn in rows 1, 2 and 3;

Max 3 marks if any errors.

An example of a fully correct answer:

```
squares ← 4                                    [Part B, Part D]
FOR row ← 0 TO 3                                   [Part D]
  gotoRow(row)                                       [A]
  FOR x ← 1 TO squares                           [Part B, C]
     shape('square')                             [Part B, Part D]
  ENDFOR
  squares ← squares + 1                            [Part D]
ENDFOR
```

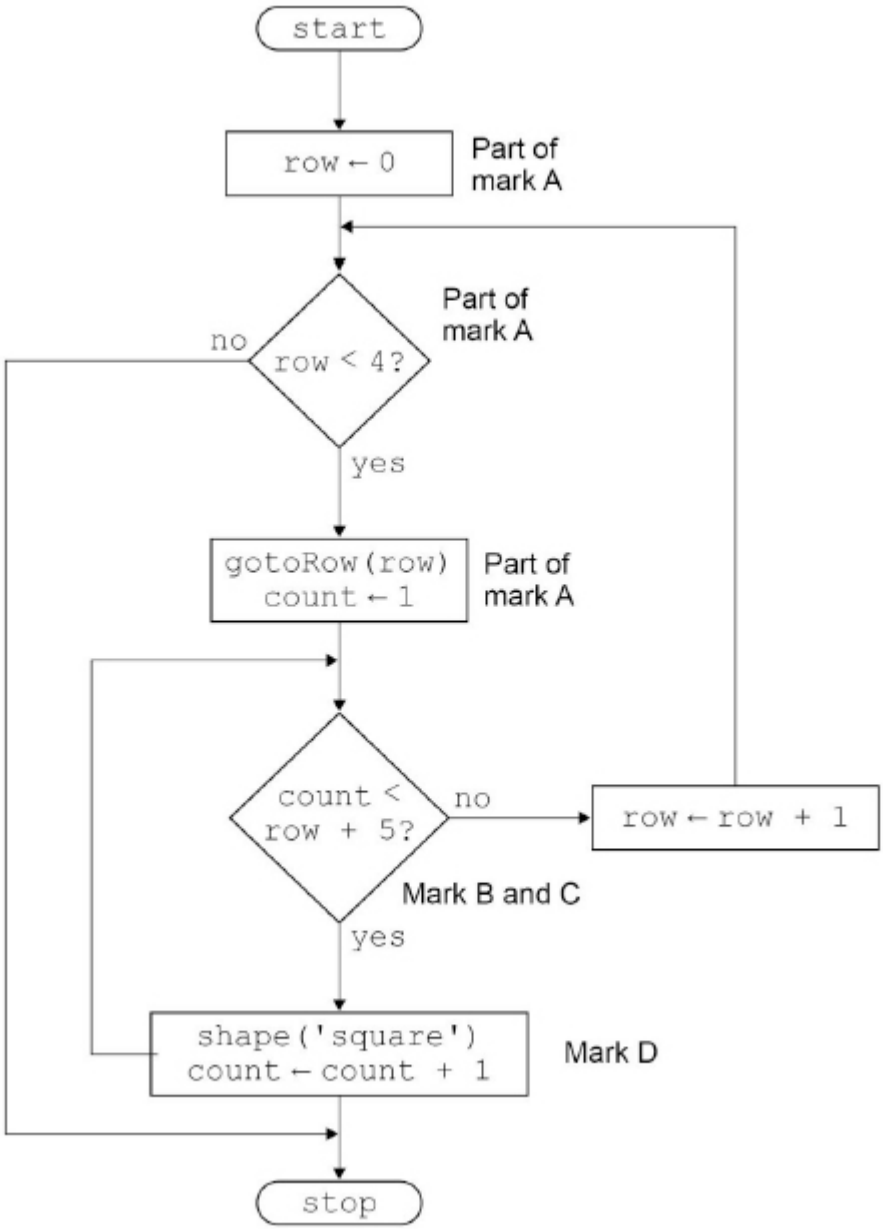
Another example of a fully correct answer:

```
gotoRow(0)                                  [Part A]
FOR x ← 1 TO 4                              [Part B, C]
   shape('square')                          [Part B]
ENDFOR
gotoRow(1)                                  [Part A]
FOR x ← 1 TO 5                              [Part D]
  shape('square')                           [Part D]
ENDFOR
gotoRow(2)                                  [Part A]
FOR x ← 1 TO 6                              [Part D]
  shape('square')                           [Part D]
ENDFOR
gotoRow(3)                                  [Part A]
FOR x ← 1 TO 7                              [Part D]
  shape('square')                           [Part D]
ENDFOR
```

Another example of a fully correct answer:

```
FOR row ← 0 TO 3                            [Part A]
   gotoRow(row)                             [Part A]
   FOR count ← 1 TO row + 4.                [Part B, C]
    shape('square')                         [Part B, D]
   ENDFOR
```

An example of a fully correct flowchart answer:



I. shape of symbols

An example of a partially correct solution is:

```
gotoRow(0)
shape('square')
shape('square')
shape('square')
shape('square')
gotoRow(1)
shape('square')
shape('square')
shape('square')
shape('square')
shape('square')
gotoRow(2)
shape('square')
shape('square')
shape('square')
shape('square')
shape('square')
shape('square')
gotoRow(3)
shape('square')
shape('square')
shape('square')
shape('square')
shape('square')
shape('square')
shape('square')
```

This solution gets marks A, B and D but not mark C as there is no use of iteration.

4

[12]

26. **3 marks for AO2 (apply)**

the `i` column having all values 0-5 in order;
the first three rows of the `image` column;
the last three rows of the `image` column;

Max 2 marks if any additional values given.

i	image
0	/
1	//
2	//*
3	/
4	/*
5	/*/

[3]

27. **3 marks for AO1 (understanding)**

Start at the beginning (of the array/list);
compare each element/item until the value being searched for is found;
or the end of the array/list is reached;

[3]

28. **3 marks for AO3 (program)**

Mark A for setting the variable `regValid` to `True/False` within a selection structure;
Mark B for using a Boolean condition that checks if the first character is an `'R';`
Mark C for using a Boolean condition that checks if the second character is a digit;

Max 2 marks if any errors in the answer.

A. minor changes to variable identifiers if the meaning is still clear.

Example of fully correct answer:

```
regValid ← False                                          [part A]
IF reg[0] = 'R' and isDigit(reg[1]) THEN                  [B,C]
     regValid ← True                                      [part A]
ENDIF
```

Example of another fully correct answer:

```
IF reg[0] = 'R' THEN                                    [B]
    IF isDigit(reg[1]) THEN                             [C]
       regValid ← True                                  [part A]
    ELSE
       regValid ← False                                 [part A]
    ENDIF
ELSE
   regValid ← False                                     [part A]
ENDIF
```

Example of 2 mark answer:

```
IF reg[0] = 'R' or isDigit(reg[1]) THEN                 [B,C]
     regValid ← True                                    [part A]
ELSE
     regValid ← True                                    [part A]
ENDIF
```

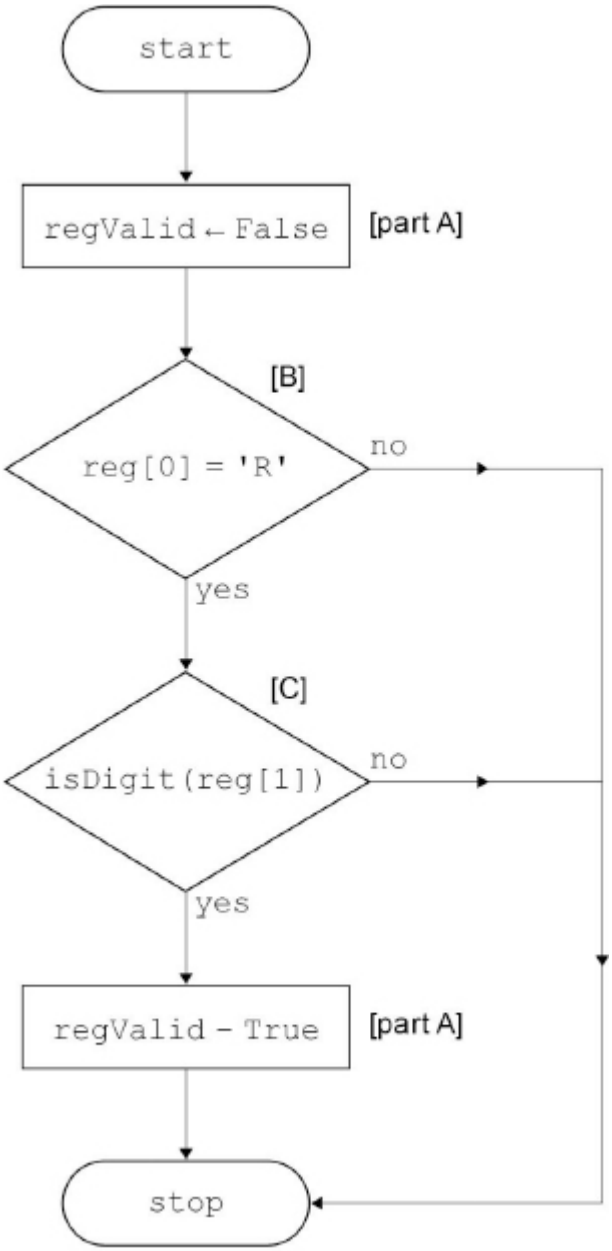
(only 2 marks awarded as the answer contains an error in the Boolean condition)

Example of another 2 mark answer:

```
IF reg[0] = 'R' or isDigit(reg[1]) THEN                 [B,C]
     regValid ← True                                    [part A]
ENDIF
```

(only 2 marks awarded as only part of mark A is given)

Example of a fully correct flowchart solution:



[3]

29. (a) **2 marks for AO2 (apply)**

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

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

The correct table is as follows:

result
16
12

2

(b) **2 marks for AO2 (apply)**

The `x` column fully correct;
The `result` column fully correct;

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

The correct table is as follows: x result

x	result
	0
1	4
2	8
3	12

I. horizontal alignment of values as long as the vertical order of values is correct.

2

(c) **Mark is for AO2 (apply)**

(The purpose of the algorithms is) to multiply the value in `number` by 3;

A. the value 4 instead of `number`.
NE. multiply two numbers.

1

(d) **Mark is for AO2 (apply)**

The algorithm in **Figure 1** uses fewer steps/instructions;

A. the algorithm in **Figure 1** uses fewer variables;
A. the algorithm in **Figure 1** has fewer instructions so will take up less memory;
A. the algorithm in **Figure 1** will execute in less time;
A. opposite statements for **Figure 2**.
NE. reference to number of lines.

1

[6]

30. **4 marks for AO2 (apply)**

Maximum 4 marks from:

If bubble sort chosen then:
8 & 4 are swapped;
1 & 8 are swapped;
5 & 8 are swapped;
1 & 4 are swapped;
swap two consecutive numbers if the left number was greater than the right number;
would repeat passes until no swaps are made/all numbers are sorted // a pass of the array `[1, 4, 5, 8]` requiring no swaps and so the algorithm stops;

or by diagram:

8	*4*	*1*	*5*	; (both lines required)
4	8	1	5	;
4	1	8	5	;
4	1	5	8	;
1	4	5	8	;

R. the final (sorted) array if no prior arrays (excluding `[8, 4, 1, 5]`) are given.

If merge sort chosen then:
separate the array into arrays that contain only one element;;
combine pairs of arrays, ordering the numbers // the values 8 and 4 combine to form the array `[4, 8]` and the value 1 and 5 combine to form the array `[1, 5];`
the arrays `[4, 8]` and `[1, 5]` combine to form the array `[1, 4, 5, 8] /` sorted array `// 4` is compared with 1, 4 is compared with 5, 8 is compared with 5;

Or by diagram (to a max 4 marks):

```
        8, 4, 1, 5                      ; (both lines required)
   8, 4            1, 5
8       4       1       5               ;
   4, 8            1, 5                 ;
        1, 4, 5, 8                      ; [A]
```

R. mark [A] if preceding row not given.

[4]

31. (a) **4 marks for AO2 (apply)**

first (calculated) value of 10;
next calculated value of 5;
next calculated value of 16;
all values of 8, 4, 2 and 1 in that order;

Stop marking at the first incorrect value.
Max of 3 marks if additional outputs are given.

Output
3
10
5
16
8
4
2
1

4

(b) **2 marks for AO1 (understanding)**

Max 2 from:

(The developer has) modularised their code // used subroutines;
(The developer has) decomposed the problem // broken the problem down into sub-problems;
(The developer has) created interfaces (to the subroutines);
(The developer has) used parameters;
(The developer has) used return values;
(The developer has) used local variables;

2

[6]

32. (a) **Mark is for AO1 (understanding)**

B: `'d';`

R. if more than one lozenge shaded.

1

(b) **Mark is for AO2 (apply)**

G;

R. g
I. use of quote marks.

1

(c) **5 marks for AO3 (program)**

Mark A for defining a subroutine with the identifier `TO_LOWER` and one parameter;
Mark B for using `CHAR_TO_CODE` with a variable parameter;
Mark C for adding 32 to the result of mark B;
Mark D for using the result of mark C as a parameter to the `CODE_TO_CHAR` subroutine;
Mark E for returning the value of mark D;
Max 4 marks if any errors in answer.

Example of fully correct answer:

```
SUBROUTINE TO_LOWER(upper)                          [A]
   code ← CHAR_TO_CODE(upper)                       [B]
   code ← code + 32                                 [C]
   lower ← CODE_TO_CHAR(code)                       [D]
   RETURN lower                                     [E]
ENDSUBROUTINE
```

Another example of a fully correct answer:

```
SUBROUTINE TO_LOWER(upper)                          [A]
   code ← CHAR_TO_CODE(upper)                       [B]
   RETURN CODE_TO_CHAR(code + 32)                   [C,D,E]
ENDSUBROUTINE
```

Another example of a fully correct answer:

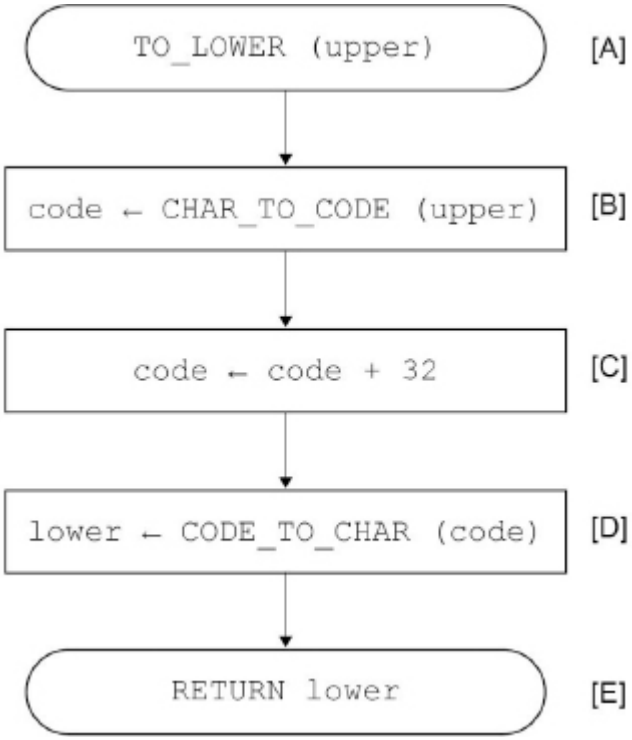
```
SUBROUTINE TO_LOWER(upper)                          [A]
   RETURN CODE_TO_CHAR(CHAR_TO_CODE(upper) + 32)    [B,C,D,E]
ENDSUBROUTINE
```

Example of a 4 mark answer:

```
SUBROUTINE TO_LOWER(upper)                          [A]
   code ← CHAR_TO_CODE(character)                   [B]
   code ← code + 32                                 [C]
   lower ← CODE_TO_CHAR(code)                       [D]
   RETURN lower                                     [E]
ENDSUBROUTINE
```

(only 4 marks awarded as answer contains an error where parameter to `CHAR_TO_CODE` is different to parameter for `TO_LOWER`)

Example of a fully correct flowchart solution:



5

[7]

33. **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                             [D, E]
       password = '9Fdg3') OR
      (username = 'tuff' AND
       password = '888rG')
OUTPUT 'access granted'                                   [F]
```

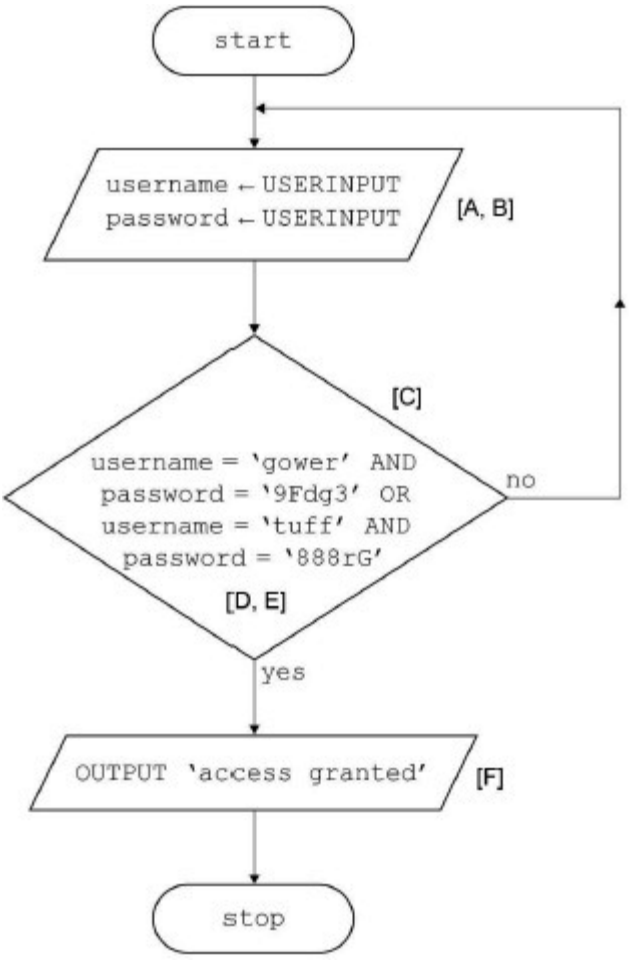
Another example of a fully correct answer:

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

Another example of a fully correct answer:

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

An example of a fully correct flowchart solution:



[6]

34. **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.

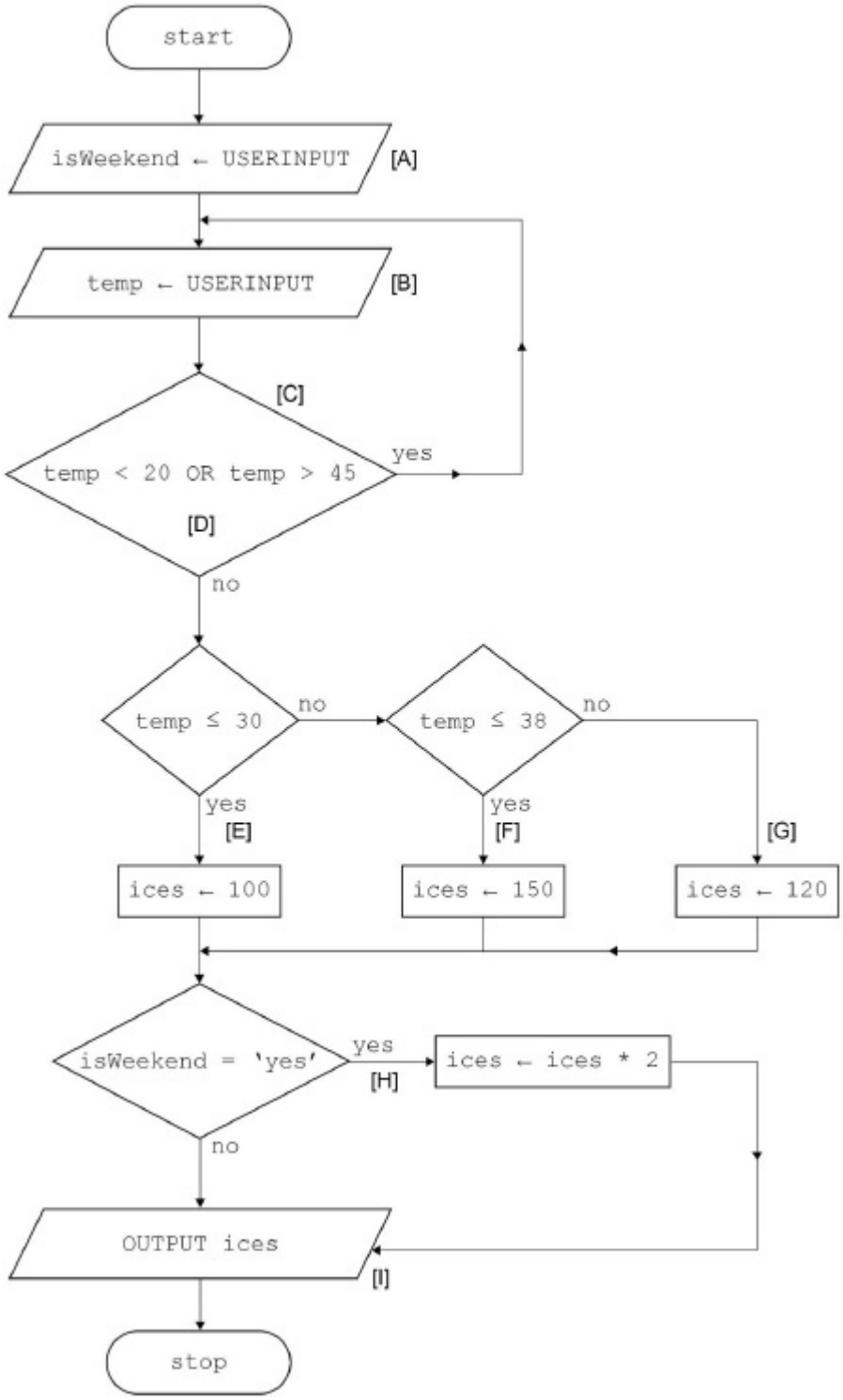
An example of a fully correct solution:

```
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]
```

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:



[9]

35.

(a) **3 marks for AO2 (apply)**

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

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

0	1	2	3	4	5	6
off	off	on	off	off	off	on

3

(b) **3 marks for AO2 (apply)**

1 mark for indices 0, 1 and 2 set to on, on and off respectively;
1 mark for index 4 set to off;
1 mark for index 5 set to off;

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

0	1	2	3	4	5	6
on	on	off	off	off	off	on

3

(c) **3 marks for AO2 (apply)**

1 mark for index 0 set to on and index 1 set to off;
1 mark for index 2 set to on;
1 mark for indices 5 and 6 set to off and on respectively;

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

0	1	2	3	4	5	6
on	off	on	on	off	off	on

3

(d) **3 marks for AO3 (program)**

3 marks if each of the subroutines is used correctly exactly once to produce the correct final array;;;

2 marks if the subroutines are used correctly to produce the correct final array but three subroutines are not used or a subroutine is used more than once;;

1 mark if at least two subroutines (possibly the same) are used correctly but the final array is incorrect;

A. 1 mark for `RANGEOFF(-1, 7);`

First full mark example answer:
```
RANGEOFF(0, 6)
NEIGHBOUR(0)
SWITCH(6)
```

Second full mark example answer:
```
RANGEOFF(0, 6)
SWITCH(6)
NEIGHBOUR(0)
```

An example 2 mark answer (not all subroutines are used):
```
RANGEOFF(0, 6)
SWITCH(6)
SWITCH(0)
```

3

[12]

36.

(a) **Mark is for AO2 (apply)**

C `flourNeeded ⟵ eggsUsed * 100;`
If more than one lozenge shaded then mark is not awarded

1

(b) **Mark is for AO2 (apply)**

A Assignment;
If more than one lozenge shaded then mark is not awarded

1

(c) **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)
```

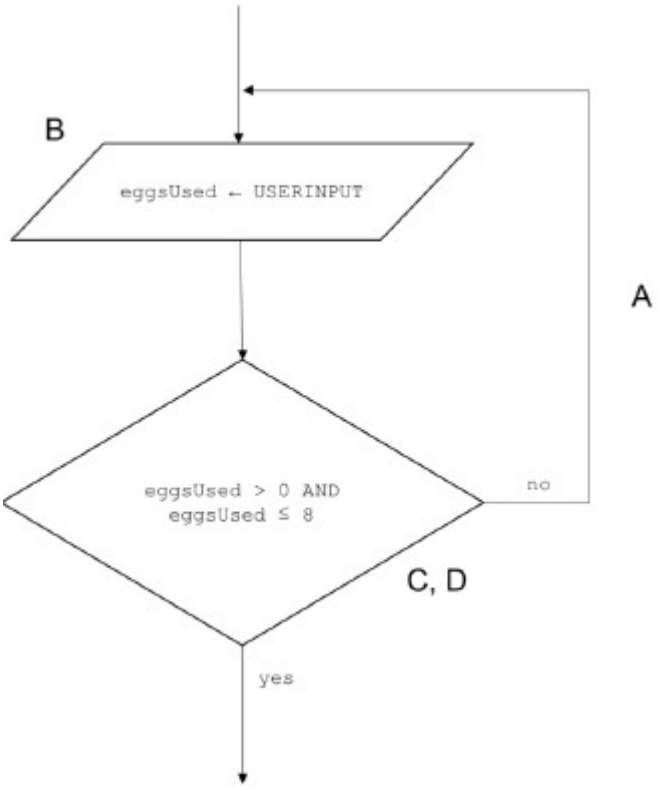
Example 4 mark answer:

```
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:



4

[6]

37. (a) **4 marks for AO2 (apply)**

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

Correct table is:

totalSize	dataToBeSent	numberOfPackets
300	*750*	0
	450	1
	150	2
	-150	3

A. follow through for incorrect `totalSize`

4

(b) **Mark is for AO2 (apply)**

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

1

(c) **Mark is for AO2 (apply)**

A Input: `dataToBeSent`. output: `numberOfPackets`;

If more than one lozenge shaded then mark is not awarded

1

(d) **3 marks for AO3 (program)**

A `dataToBeSent`;

B `totalSize`;

C `numberOfPackets + 1`;

A. `numberOfPackets++ for C`;

I. case and minor spelling mistakes;

3

[9]

38.

(a) **Mark is for AO2 (apply)**

D `USERINPUT`;
If more than one lozenge shaded then mark is not awarded

1

(b) **Mark is for AO2 (apply)**

B `0`;
If more than one lozenge shaded then mark is not awarded

1

(c) **Mark is for AO2 (apply)**

A `=` ;
If more than one lozenge shaded then mark is not awarded

1

(d) **Mark is for AO2 (apply)**

D `OUTPUT count`;
If more than one lozenge shaded then mark is not awarded

1

(e) **Mark is for AO2 (apply)**

B `i ⟵ i + 1`;
If more than one lozenge shaded then mark is not awarded

1

(f) **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

(g) **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

[10]

39. (a) **Mark is for AO2 (apply)**

C Selection;
If more than one lozenge shaded then mark is not awarded

1

(b) **Mark is for AO2 (apply)**

D String;
If more than one lozenge shaded then mark is not awarded

1

(c) **Mark is for AO2 (apply)**

3//three;

1

(d) **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

(e) **3 marks for AO2 (apply)**

Maximum three marks overall.
Maximum two marks from each section.

Reason

- The output message is not descriptive enough/the user is not told what word/words they should use to answer (before user input);
- The Boolean expression (at lines 3, 6 and 14) only matches exact values//the program is only written for the exact words yes and no // a **clear** indication that y is not recognised as yes or n is not recognised as no;
- A clear explanation of how to fix the problem;

What would happen

Any clear descriptions of what would happen. Line numbers may or may not be included. If the logic and explanation is clear credit the answer.

This can include but is not limited to:

- Line 3 will only be true if they enter 'no' // Line 3 will not be true if they enter anything other than 'no';
- Line 6 / 14 will only be true if they enter 'yes' // Line 6 / 14 will not be true if they enter anything other than 'yes';
- if they enter 'n' at line 2 the algorithm will execute an incorrect code block;
- if they enter 'y' at line 5 or line 13 an incorrect message will be output;

3

[8]

40. **3 marks for AO2 (apply)**

Stop marking at the first error.

(Compare) 30 with 21 / position 3;
(Compare) 30 with 31 / position 5;
(Compare) 30 with 27 / position 4;

[3]

41. **1 mark for AO1 (understanding)**

(The array) must be ordered / sorted;

[1]

42. **6 marks for AO3 (program)**

Any fully correct answer should get 6 marks even if it does not map exactly to the following mark points.

Maximum 5 marks if the answer contains any errors.

Mark A: using a selection statement in the nested `WHILE` loop;
Mark B: using a Boolean condition that tests for equality//inequality of the `image1` and `image2` variables;
Mark C: indexing either `image1` or `image2` using the variables `i` and `j`;
Mark D: assigning false to inverse within the selection if logically correct throughout the code (if assigned true then check for correctness);
Mark E: incrementing `j` in the relevant place;
Mark F: incrementing `i` in the relevant place.

Example 6 mark answer:

```
image1 ⟵ [ [0, 0, 0], [0, 1, 1], [1, 1, 0] ]
image2 ⟵ [ [1, 1, 1], [1, 1, 0], [0, 0, 1] ]
inverse ⟵ true
i ⟵ 0
WHILE i ≤ 2
    j ⟵ 0
    WHILE j ≤ 2
        IF image1[ i ][ j ] = image2[ i ][ j ] THEN          (A,B,C)
            inverse ⟵ false                                  (D)
        ENDIF
        j ⟵ j + 1                                            (E)
    ENDWHILE
    i ⟵ i + 1                                                (F)
ENDWHILE
```

[6]

43. (a) **2 marks for AO1 (understanding)**

Correct table is:

Values	**Data type**
`true, false`	Boolean;
`0, 1, 2`	Integer;

A. Bool / bool / boolean instead of Boolean

A. Int / int instead of integer

2

(b) **Mark is for AO1 (recall)**

Decomposition;

A. Top-down design;

1

(c) **2 marks for AO2 (apply)**

1 mark for giving a new identifier that describes this purpose, e.g. `notHit` (alternatively award this mark if the explanation is incorrect but the identifier describes the purpose stated in the answer);
1 mark for explaining the purpose of the subroutine is to see if a hit has been made at the specified location;

2

(d) **2 marks for AO2**

(A local variable) is only accessible//declarable//within scope (in the subroutine);
(A local variable) only exists while the subroutine / program block is executing;

2

(e) **11 marks for AO3 (program)**

Any fully correct answer should get 11 marks even if it does not map exactly to the following mark points.

Max 10 marks if the answer includes any errors.

Mark A:for creating a subroutine with an identifier that defines its purpose;
Mark B: for passing the board as a parameter;
Mark C: for using iteration to loop over all (15) locations in the board;
Mark D: for using indices (or similar) to identify the value of each cell//a `FOR-EACH` loop used correctly;
Mark E: for using selection to ascertain if a cell is a hit (value 2);
Mark F: for incrementing a variable that stores how many hits have been made;
Mark G: for ascertaining the number of cells yet to be hit (value 1), possibly by using the subroutine `F;`
Mark H: for suitable variable initialisation;
Mark I: for outputting ‘Winner’ if the number yet to be hit is zero;
Mark J: or outputting ‘Almost there’ if the number yet to be hit is 1–3 inclusive;
Mark K: for outputting the Mark F variable;

A. For marks I, J and K accept returning the number of hits and messages in place of outputting to the screen on this occasion only.

Example of complete correct answer:

```
SUBROUTINE howFarAwayFromEnding(board)    [A, B]
   hits ← 0                               [part H]
   yetToBeHit ← 0                         [part H]
   FOR x ← 0 TO 14                        [C]
      IF board[x] = 2 THEN                [D, E]
         hits ← hits + 1                  [F]
      ELSE
         IF board[x] = 1 THEN             [part G]
            yetToBeHit ← yetToBeHit + 1   [part G]
         ENDIF
      ENDIF
   ENDFOR
   OUTPUT hits                            [K]
   IF yetToBeHit = 0 THEN                 [part I]
      OUTPUT 'Winner'                     [part I]
   ELSE IF yetToBeHit < 4 THEN            [part J]
      OUTPUT 'Almost there'               [part J]
   ENDIF
ENDSUBROUTINE
```

Example of complete correct answer that uses `FOREACH`

```
SUBROUTINE howFarAwayFromEnding(board)    [A, B]
   hits ← 0                               [part H]
   yetToBeHit ← 0                         [part H]
   FOREACH cell IN board                  [C, D]
      IF cell = 2 THEN.                   [E]
         hits ← hits + 1                  [F]
      ELSE
         IF cell = 1 THEN                 [part G]
            yetToBeHit ← yetToBeHit + 1   [part G]
         ENDIF
      ENDIF
   ENDFOREACH
   OUTPUT hits                            [K]
   IF yetToBeHit = 0 THEN                 [part I]
      OUTPUT 'Winner'                     [part I]
   ELSE IF yetToBeHit < 4 THEN            [part J]
      OUTPUT 'Almost there'               [part J]
   ENDIF
ENDSUBROUTINE
```

Example of complete correct answer that doesn't use Mark G variable:

```
SUBROUTINE howFarAwayFromEnding(board)    [A, B]
   hits ← 0                               [part H]
   FOR x ← 0 TO 14                        [C]
      IF board[x] = 2 THEN                [D, E]
         hits ← hits + 1                  [F]
      ENDIF
   ENDFOR
   OUTPUT hits                            [K]
   IF (6 - hits) = 0 THEN          [part G, part I]
      OUTPUT 'Winner'                     [part I]
   ELSE IF (6 - hits) < 4 THEN     [part G, part J]
      OUTPUT 'Almost there'               [part J]
   ENDIF
ENDSUBROUTINE
```

11

[18]

44. **2 marks for AO1 (recall)**

A sequence of steps/instructions;
that can be followed to complete a task;

A. Different wording with similar meaning

[2]

45. **3 marks for AO1 (recall)**

One mark for each correct distinct label.

If the answers given were, for example, C, C, B then award only 1 mark for the B as the C is duplicated. Likewise if C, C, C was the answer then no marks would be given. The correct table is:

	Label
Breaking a problem down into a number of sub-problems	C
The process of setting the value stored in a variable	A
Defines the range of values a variable may take	B

A. If actual terms are written out instead of labels
R. All instances of duplicate labels

[3]

46. (a) **Mark is for AO2 (apply)**

A Line number 2;
R. If more than one lozenge shaded

1

(b) **Mark is for AO2 (apply)**

C Line number 11;
R. If more than one lozenge shaded

1

(c) **Mark is for AO2 (apply)**

A 1 subroutine call;
R. If more than one lozenge shaded

1

(d) **Mark is for AO2 (apply)**

B String;
R. If more than one lozenge shaded

1

(e) **Mark is for AO2 (apply)**

2//twice//two;

1

[5]

47. (a) **Mark is for AO2 (apply)**

Boolean//bool;
I. Case

1

(c) **2 marks for AO2 (apply)**

(The identifier) `swapsMade` describes the purpose//role//meaning of the variable;
this makes the algorithm easier to understand//maintain//follow;

or

(The identifier) `s` does not describe the purpose//role//meaning of the variable;
this makes the algorithm harder to understand//maintain//follow;

2

(c) **Mark is for AO2 (apply)**

A The algorithm uses a named constant;
R. If more than one lozenge shaded

1

(d) **6 marks for AO2 (apply)**

1 mark for column `arr[0]` correct;
1 mark for column `arr[1]` correct;
1 mark for column `arr[2]` correct **only if** `arr[0]` and `arr[1]` are correct;
1 mark for `swapsMade` column correct;
1 mark for `i` column correct;
1 mark for `t` column correct;

Arr			swapsMade	i	t
0	1	2			
4	1	6	false		
			true	0	
1	4		false		4
				1	
				2	
			true	0	
				1	
				2	

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

6

[10]

48. (a) **3 marks for AO2 (apply)**

Mark as follows:

1 mark for the robot moving to **both** squares marked **A**;
1 mark for the robot moving to the square marked **B**;
1 mark for the robot moving to the square marked **C**;

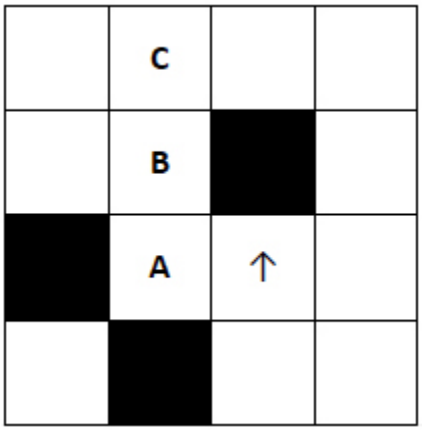
		C	
		B	**A**
			A
			↑

3

(b) **3 marks for AO2 (apply)**

Mark as follows:

1 mark for the robot moving to the square marked **A**;
1 mark for the robot moving to the square marked **B**;
1 mark for the robot moving to the square marked **C**;



3

[6]

49. **5 marks for AO2 (apply)**

1 mark for each correct change (allow follow on);

The correct sequence is:

3	**1**	**5**	4	2
3	1	**4**	**5**	2
3	1	4	**2**	**5**
1	**3**	4	2	5
1	3	**2**	**4**	5

[5]

50. (a) **1 mark for AO1 (recall)**

A Abstraction;

R. if more than one lozenge shaded

1

(b) **2 marks for AO2 (apply)**

All friends have different first names;
The time is rounded up to the nearest half-hour;

2

[3]

51. (a) **3 marks for AO2 (apply)**

1 mark for C written once and in column 1;
1 mark for A and B written once and both in column 2 (in any order);
1 mark for A and B written once and in correct positions in column 2;

Column 0	Column 1	Column 2
	C	A B

3

(b) **3 marks for AO2 (apply)**

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

Column 0	Column 1	Column 2
A	C	B

3

[6]

52. **4 marks for AO3 (design)**

Mark A for using a `WHILE` loop or similar to move from column 0 to column 2;
Mark B for a Boolean condition that detects when column 0 is empty;
Mark C for using a second `WHILE` loop or similar to move the result from A and B into column 1 (both the loop and the associated Boolean condition need to be correct to gain this mark);

or

Mark A for using a `FOR` loop or similar to move from column 0 to column 2;
Mark B for ascertaining the terminating value for the `FOR` loop;
Mark C for using a second `FOR` loop or similar to move the result from A and B into column 1 (both the loop and the associated terminating value need to be correct to gain this mark);

and

Mark D for using the subroutines correctly throughout, i.e. called with appropriate parameters and return values handled correctly;

A. Minor spelling errors such as `HIEGHT` for `HEIGHT`
I. Case

Example 1

```
WHILE HEIGHT(0) > 0                 (Part of A, B)
   MOVE(0, 2)                       (Part of A)
ENDWHILE
WHILE HEIGHT(2) > 0                 (Part of C)
   MOVE(2, 1)                       (Part of C)
ENDWHILE
```

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

Example 2

```
DO                                  (Part of A)
   MOVE(0, 2)                       (Part of A)
WHILE HEIGHT(0) > 0                 (Part of A,B)
DO                                  (Part of C)
   MOVE(2, 1)                       (Part of C)
WHILE HEIGHT(2) > 0                 (Part of C)
```

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

Example 3

```
REPEAT                              (Part of A)
   MOVE(0, 2)                       (Part of A)
UNTIL HEIGHT(0) = 0                 (Part of A,B)
REPEAT                              (Part of C)
   MOVE(2, 1)                       (Part of C)
WHILE HEIGHT(2) = 0                 (Part of C)
```

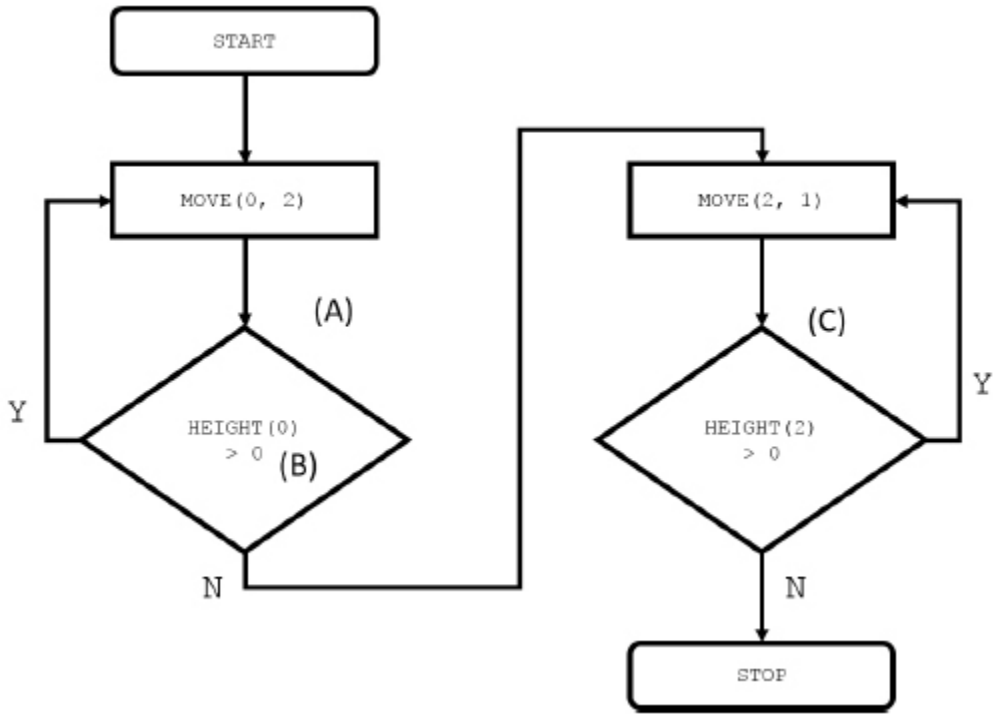
(`MOVE` and `HEIGHT` are used correctly throughout so **D**.)

Example 4

```
number_of_blocks ← HEIGHT(0)        (Part of B)
FOR x ← 0 TO number_of_blocks       (Part of A, Part of B)
   MOVE(0, 2)                       (Part of A)
ENDFOR
FOR x ← 0 TO number_of_blocks       (Part of C)
   MOVE(2, 1)                       (Part of C)
ENDFOR                              (Part of C)
```

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

Example 5



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

[4]

53.

2 marks for AO1 (recall)

A sequence / number / set of steps / instructions;
that can be followed to complete a task / to solve a problem;

A. Different wording with similar meaning

[2]

54.

3 marks for AO1 (recall)

One mark for each correct distinct label.

If the answers given were, for example, C, C, B then award only 1 mark for the B as the C is duplicated. Likewise if C, C, C was the answer then no marks would be given. The correct table is:

	Label
Breaking a problem down into a number of sub-problems.	C
The process of removing unnecessary detail from a problem.	A
Defines the range of values a variable may take.	B

A. If actual terms are written out instead of labels
R. All instances of duplicate labels

[3]

55. (a) **Mark is for AO2 (apply)**

Boolean // bool;

I. Minor spelling mistakes

(1)

(b) **2 marks for AO2 (apply)**

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

or
(The identifier) `s` does not describe the purpose // role // meaning of the variable;
this makes the algorithm harder to understand // maintain // follow;

(2)

(c) **Mark is for AO2 (apply)**

A (The algorithm uses a named constant.) only;
If more than one lozenge shaded then mark is not awarded

(1)

(d) **6 marks for AO2 (apply)**

1 mark for column `arr[0]` correct;
1 mark for column `arr[1]` correct;
1 mark for column `arr[2]` correct only if `arr[0]` and `arr[1]` are correct;
1 mark for `sorted` column correct;
1 mark for `i` column correct;
1 mark for `t` column correct;

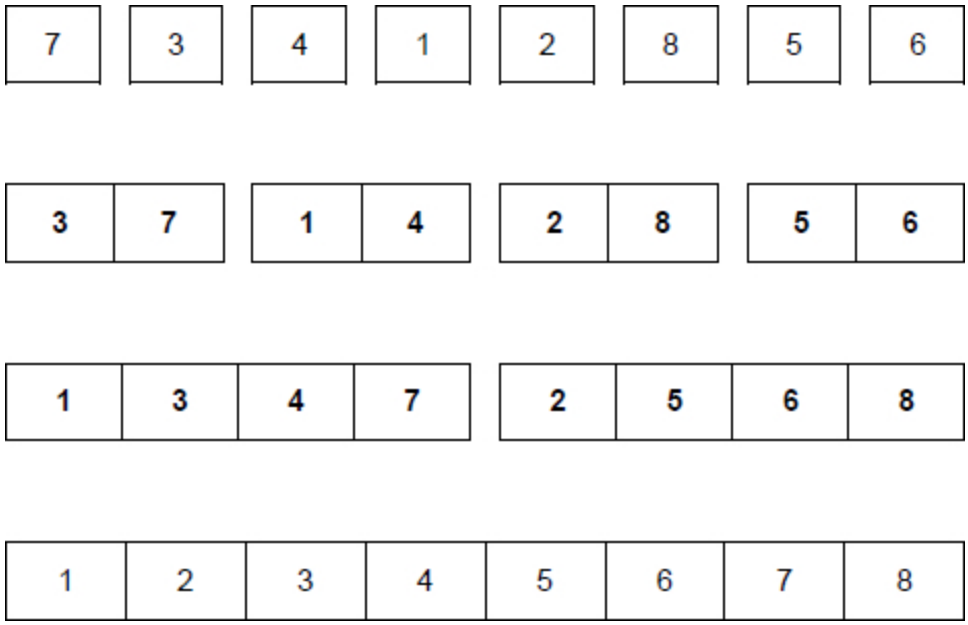
Arr			sorted	i	t
0	1	2			
4	1	6	false		
			true	0	
1	4		false		4
				1	
				2	
			true	0	
				1	
				2	

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

(6)

(e) **3 marks for AO2 (apply)**

1 mark if pairwise comparisons are made in the second row but allow for one pairwise comparison error;
1 mark if pairwise comparisons are made in the third row but allow for one pairwise comparison error (allow follow through from previous row);
1 mark if all correct;



3

(f) **Mark is for AO1 (understanding)**

It is more (time) efficient //
It will usually take fewer steps;

A. quicker // it will take less time as long as the answer has been qualified.

1

(g) **2 marks for AO2 (apply)**

Maximum of 2 from:
It allows the code to be (more easily) reused;
It can be used to sort any array (not just the one on line 1);
It would be easier to test;
The code could be changed // updated without affecting the overall program;
Makes the program easier to read//understand;

A. Any other creditable answer as long as they are clearly distinct from the other responses.

2

[16]

56. (a) **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

(b) **Mark is for AO2 (apply)**

false;

I. Case

1

(c) **Mark is for AO2 (apply)**

Second row only;

`IF user = us[z] OR pass = ps[z] THEN`	
`IF user = us[z] AND pass = ps[z] THEN`	✓
`IF NOT (user = us[z] AND pass = ps[z]) THEN`	

1

(d) **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

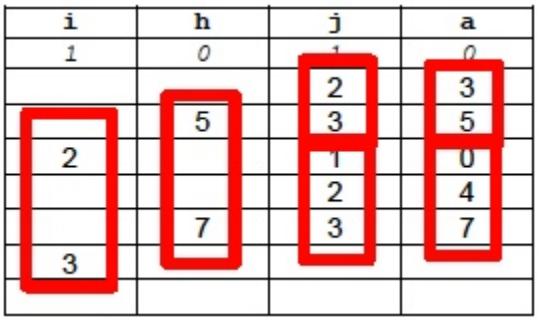
2

[7]

57. (a) 5;

1

(b) (i) 1 mark for all correct values of `i` in the correct order (only 2 and 3);
1 mark for all correct values of `h` in the correct order (only 5 and 7);
1 mark for second and third values of `j` (2 and 3);
1 mark for last three values of `j` in the correct order (1, 2 and 3);
1 mark for second and third values of `a` (3 and 5);
1 mark for last three values of `a` in the correct order (0, 4 and 7);



I. Different rows used as long as the order within columns is clear and repeated values in columns.

6

(ii) (`h` represents) the higher value;
of the sum of the arrays (within `arr`);

2

(c) (Its value) does not change;

1

(d) `LEN(arr)` instead of 2//
`lenArr ← 3`//
2 changed to 3//
Value changed to 3;

1

[11]

58. (a) **3 marks for AO2 (apply)**

The list is being divided into shorter lists;
The list is split at the (approximate) middle item;
Each sublist is (approximately) half the length of the list it was created from;
The subdivision process terminates when each sublist is of length 1;

Max 3

3

(b) **3 marks for AO2 (apply)**

The lists are being merged together;
When two lists are merged, the items in them are put into the new list in order;
Eventually one list is produced (which is the sorted list);

Max 3

3

[6]

59. **2 marks for AO1 (knowledge and understanding)**

Advantage of merge sort:

Algorithm can sort a list more quickly (in most cases); **A.** algorithm is more efficient

Disadvantage of merge sort (**Max 1**):

Algorithm requires additional storage space during the sort; **A.** more memory is needed
Algorithm is more difficult to code; **A.** algorithm requires more lines of code

[2]

60. (a) **3 marks for AO2 (apply)**

Mark as follows:
1 mark for all correct values in the correct order for column a;
1 mark for all correct values in the correct order for column b;
1 mark for column c having only the value 7;

The completed trace table should have these values although the candidate may have entered the values on different rows (do not penalise as long as the order of the values is correct).

a	b	c
3	4	7
4	0	
5	−5	
6	−11	
7	−18	

3

(b) **2 mark for AO2 (apply)**

1 mark for each correct answer to a maximum of two.

The value of c is constant / does not change;
(a+b) / a / b may change;
There is a logical error;

2

(c) **Mark is for AO2 (apply)**
0;

1

[6]