0 1 The algorithm in **Figure 1** has been developed to automate the quantity of dog biscuits to put in a dog bowl at certain times of the day. The algorithm contains an error.

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

Figure 1

```
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        a ← 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
```

0 1 . 1 Shade **one** lozenge which shows the line number where selection is **first** used in the algorithm shown in **Figure 1**.

[1 mark]

A Line number 2 ○

B Line number 4 ○

C Line number 9 ○

D Line number 12 ○

0 1 . 2 Shade **one** lozenge which shows the line number where iteration is **first** used in the algorithm shown in **Figure 1**.

[1 mark]

A Line number 1 ○

B Line number 8 ○

C Line number 11 ○

D Line number 13 ○

0 1 . 3 Shade **one** lozenge which shows how many times the subroutine `DISPENSE_BISCUIT` would be called if the user input is `'breakfast'`.

[1 mark]

A 1 subroutine call ○

B 2 subroutine calls ○

C 3 subroutine calls ○

D 4 subroutine calls ○

0 1 . 4 Shade **one** lozenge which shows the data type of the variable `time` in the algorithm shown in **Figure 1**.

[1 mark]

A Date/Time ○

B String ○

C Integer ○

D Real ○

0 1 . 5 State how many times the subroutine `DISPENSE_BISCUIT` will be called with the parameter `'chewies'` if the user input is `'lunch'`.

[1 mark]

____

0 1 . 6 State how many possible values the result of the comparison `time = 'dinner'` could have in the algorithm shown in **Figure 1**.

[1 mark]

____

0 1 . 7 The programmer realises they have made a mistake. State the line number of the algorithm shown in **Figure 1** where the error has been made.

[1 mark]

____

0 1 . 8 Write **one** line of code that would correct the error found in the algorithm in **Figure 1**.

[1 mark]

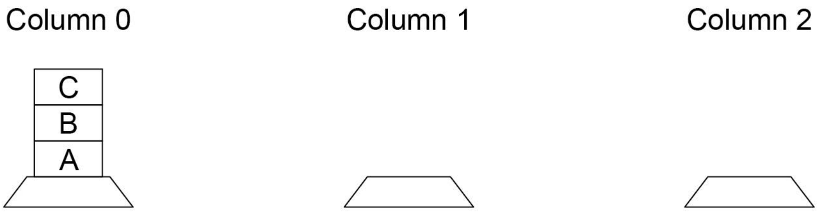
____

____

0 2 The following subroutines control the way that labelled blocks are placed in different columns.

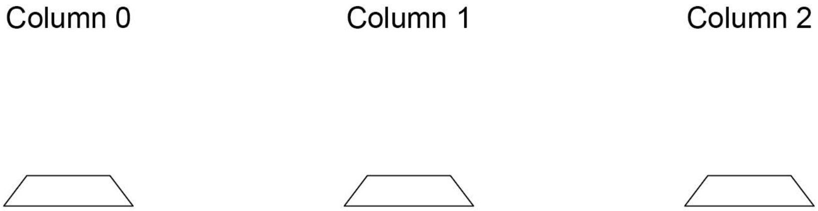
`BLOCK_ON_TOP(column)`	returns the label of the block on top of the column given as a parameter.
`MOVE(source, destination)`	moves the block on top of the `source` column to the top of the `destination` column.
`HEIGHT(column)`	returns the number of blocks in the specified column.

0 2 . 1 This is how the blocks A, B and C are arranged at the start.



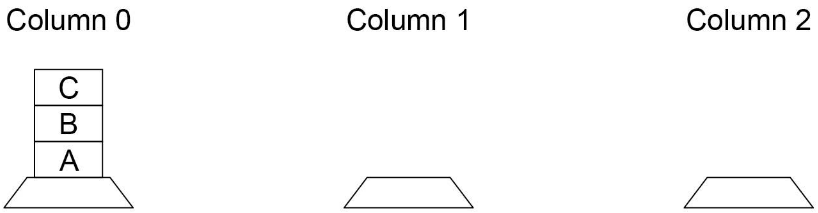
Draw the final arrangement of the blocks after the following algorithm has run.

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



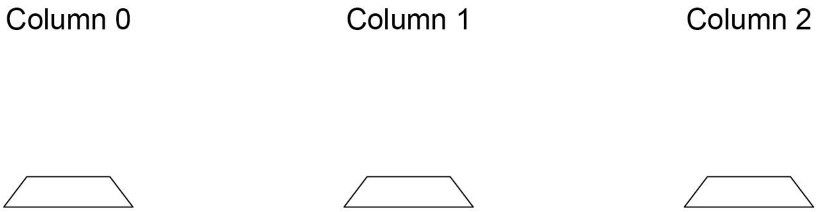
[3 marks]

0 2 . 2 This is how the blocks A, B and C are arranged at the start.



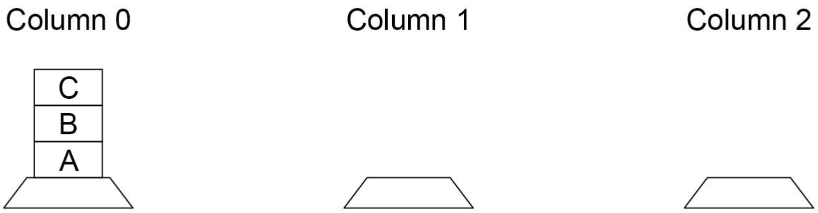
Draw the final arrangement of the blocks after the following algorithm has run.

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



[3 marks]

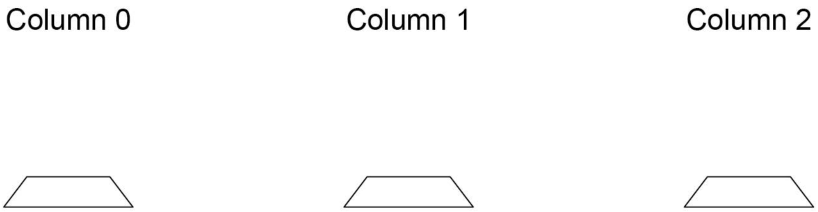
0 2 . 3 This is how the blocks A, B and C are arranged at the start.



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

```
FOR c ← 0 TO 2
   IF BLOCK_ON_TOP(0) = 'B' THEN
      MOVE(0, (c+1) MOD 3)
   ELSE
      MOVE(0, (c+2) MOD 3)
   ENDIF
ENDFOR
```

This algorithm uses the MOD operator which calculates the remainder resulting from integer division. For example, `13 MOD 5 = 3`.



[3 marks]

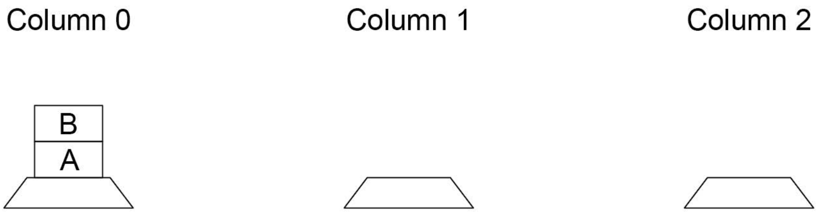
0 2 . 4 Develop an algorithm using either pseudo-code or a flowchart that will move every block from column 0 to column 1.

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.

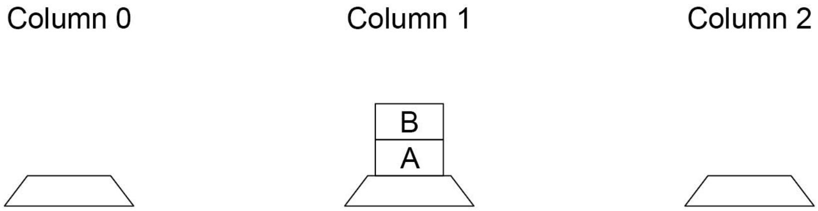
The order of the blocks must be preserved.

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:



[5 marks]

Turn over for the next question

0 3 The subroutine in **Figure 3** 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.

Figure 3

```
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
```

0 3 . 1 Complete the trace table for the following subroutine call:

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

[3 marks]

z	**correct**

0 3 . 2 State the value that is returned by the following subroutine call:

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

[1 mark]

____

0 3 . 3 Lines 7 and 8 in **Figure 3** could be replaced with a single line. Shade **one** lozenge to show which of the following corresponds to the correct new line.

[1 mark]

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` ○

0 3 . 4 A programmer implements the subroutine shown in **Figure 3**. 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 marks]

____

____

____

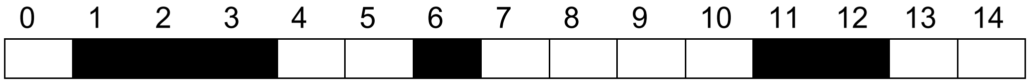
____

0 4 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 in **Figure 9** where the indices are also written above the board.

Figure 9



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 **Figure 10**.

Figure 10

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

0 4 . 1 The subroutine in **Figure 10** 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.

[2 marks]

Values	**Data type**
`true`, `false`	
`0`, `1`, `2`	

0 4 . 2 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 mark]

0 4 . 3 The identifier for the subroutine in **Figure 10** is `F`. This is not a good choice. State a better identifier for this subroutine and explain why you chose it.

[2 marks]

New subroutine identifier:

Explanation:

0 4 . 4 The variable `h` in the subroutine in **Figure 10** is local to the subroutine. State **two** properties that only apply to local variables.

[2 marks]

0 4 . 5 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 marks]

0 5 . 1 Four subroutines are shown in **Figure 7**.

Figure 7

```
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.

[4 marks]

Output
3

0 5 . 2 Describe how the developer has used the structured approach to programming in **Figure 7**.

[2 marks]

____

____

____

____

0 6 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 **Table 2**.

Table 2

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

0 6 . 1 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

[3 marks]

0	1	2	3	4	5	6

0 6 . 2 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

[3 marks]

0	1	2	3	4	5	6

0 6 . 3 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

[3 marks]

0	1	2	3	4	5	6

0 6 . 4 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 marks]

___

___

___

0 7 **Figure 9** shows a subroutine represented using pseudo-code.

Figure 9

```
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.

0 7 . 1 Complete the trace table for the subroutine call `calculate(50)`

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

[4 marks]

n	a	b	OUTPUT
50			

0 7 . 2 State the value that will be output for the subroutine call `calculate(1)`

[1 mark]

0 7 . 3 The identifier for the variable `b` in **Figure 9** was not a good choice.

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

[1 mark]

0 7 . 4 A REPEAT…UNTIL iteration structure was used in **Figure 9**.

Figure 9 has been included again below.

Figure 9

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

Figure 10 shows another subroutine called calculate that uses a WHILE…ENDWHILE iteration structure.

Figure 10

```
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 9** and **Figure 10** work is:

- the REPEAT…UNTIL iteration structure in **Figure 9** loops until the condition is true
- the WHILE…ENDWHILE iteration structure in **Figure 10** loops until the condition is false.

Describe **two** other differences in the way the subroutines in **Figure 9** and **Figure 10** work.

[2 marks]

1 ____

____

____

2 ____

____

____

Turn over for the next question

0 8 . 1 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 12**, 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 12** by filling in the gaps using the items in **Figure 11**.

You will not need to use all the items in **Figure 11**.

[6 marks]

Figure 11

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

Figure 12

```
SUBROUTINE getSize(____________, ____________, seconds)

    ______________ ← sampRate * res * seconds

    size ← ______________

    ______________ size

ENDSUBROUTINE


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

0 8 . 2 A local variable called `size` has been used in `getSize`.

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

[1 mark]

0 8 . 3 State **three** advantages of using subroutines.

[3 marks]

1

2

3

Turn over for the next question

0 9 A program is being written to simulate a computer science revision game in the style of bingo.

At the beginning of the game a bingo ticket is generated with nine different key terms from computer science in a 3 x 3 grid. An example bingo ticket is provided in **Figure 15**.

Figure 15

CPU	ALU	Pixel
NOT gate	Binary	LAN
Register	Cache	Protocol

The player will then be prompted to answer a series of questions.

If an answer matches a key term on the player's bingo ticket, then the key term will be marked off automatically.

0 9 . 1 **Figure 16** shows an incomplete C# program to create a bingo ticket for a player.

The programmer has used a two-dimensional array called `ticket` to represent a bingo ticket.

The program uses a subroutine called `generateKeyTerm`. When called, the subroutine will return a random key term, eg `"CPU"`, `"ALU"`, `"NOT gate"` etc.

Complete the C# program in **Figure 16** by filling in the five gaps.

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

[4 marks]

Figure 16

```
1   string[,] ticket = new string[,] {{"","",""},
                                      {"","",""},
                                      {"","",""}};

2   int i = 0;
3   while (i < 3) {

4       int j = ____ ;
5       while (j < 3) {

6           ticket[ ____ , ____ ] = generateKeyTerm();

7           _______________;
8       }

9       _______________;
10  }
```

0 9 . 2 Each time a player answers a question correctly the `ticket` array is updated; if their answer is in the `ticket` array then it is replaced with an asterisk (*).

An example of the `ticket` array containing key terms and asterisks is shown in **Figure 17**.

Figure 17

	0	1	2
0	CPU	ALU	*
1	*	*	LAN
2	Register	Cache	*

Write a subroutine in C# called `checkWinner` that will count the number of asterisks.

The subroutine should:

- take the `ticket` array as a parameter
- count the number of asterisks in the `ticket` array
- output the word `Bingo` if there are nine asterisks in the array
- output the total number of asterisks if there are fewer than nine asterisks in the array.

You **must** write your own count routine and not use any built-in count function that might be available in C#.

You **should** use meaningful variable name(s) and C# syntax in your answer.

The answer grid below contains vertical lines to help you indent your code.

[8 marks]

1 0 Explain **one** advantage of the structured approach to programming.

[2 marks]

1 1 **Figure 5** shows an algorithm represented using pseudo-code.

The algorithm is for a simple authentication routine.

The pseudo-code uses a subroutine `getPassword` to check a username:

- If the username exists, the subroutine returns the password stored for that user.
- If the username does not exist, the subroutine returns an empty string.

Parts of the algorithm are missing and have been replaced with the labels L1 to L4.

Figure 5

```
login ← False
REPEAT
   username ← ''
   WHILE username = ''
      OUTPUT 'Enter username: '
      username ← L1
   ENDWHILE
   password ← ''
   WHILE password = ''
      OUTPUT 'Enter password: '
      password ← USERINPUT
   ENDWHILE
   storedPassword ← getPassword(L2)
   IF storedPassword = L3 THEN
      OUTPUT 'L4'
   ELSE
      IF password = storedPassword THEN
         login ← True
      ELSE
         OUTPUT 'Try again.'
      ENDIF
   ENDIF
UNTIL login = True
OUTPUT 'You are now logged in.'
```

Figure 6

`-1`	`OUTPUT`	`0`
`username`	`True`	`SUBROUTINE`
`1`	`User not found`	`''`
`USERINPUT`	`password`	`Wrong password`

State the items from **Figure 6** that should be written in place of the labels in the algorithm in **Figure 5**.

You will not need to use all the items in **Figure 6**.

[4 marks]

L1 ______________________________

L2 ______________________________

L3 ______________________________

L4 ______________________________

Turn over for the next question

1 2 The algorithm in **Figure 2** 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.

Figure 2

```
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
```

1 2 . 1 Shade **one** lozenge which shows the line number where selection is **first** used in the algorithm shown in **Figure 2**.

[1 mark]

A Line number 2 ○

B Line number 4 ○

C Line number 9 ○

D Line number 12 ○

1 2 . 2 Shade **one** lozenge which shows the line number where iteration is **first** used in the algorithm shown in **Figure 2**.

[1 mark]

A Line number 1 ○

B Line number 8 ○

C Line number 11 ○

D Line number 13 ○

1 2 . 3 Shade **one** lozenge which shows how many times the subroutine `DISPENSE_BISCUIT` would be called if the user input is `'breakfast'` in **Figure 2**.

[1 mark]

A 1 subroutine call ○

B 2 subroutine calls ○

C 3 subroutine calls ○

D 4 subroutine calls ○

1 2 . 4 Shade **one** lozenge which shows the data type of the variable `time` in the algorithm shown in **Figure 2**.

[1 mark]

A Date/Time ○

B String ○

C Integer ○

D Real ○

1 2 . 5 State how many times the subroutine `DISPENSE_BISCUIT` will be called with the parameter `'chewies'` if the user input is `'lunch'` in **Figure 2.**

[1 mark]

____

Turn over for the next question

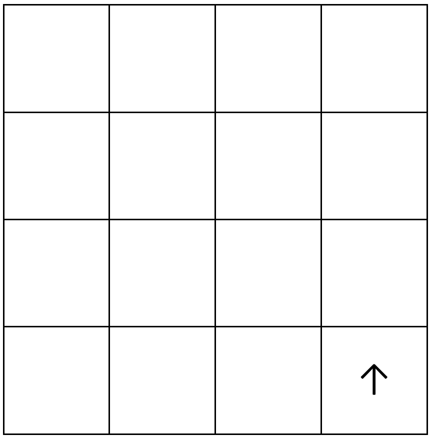
1 3 Four separate subroutines have been written to control a robot.

- `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.

1 3 . 1 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)
```

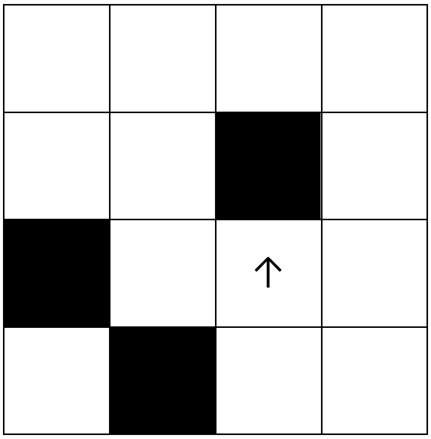
[3 marks]



1 3 . 2 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 marks]



Turn over for the next question

1 4 State **two** benefits of developing solutions using the structured approach.

[2 marks]

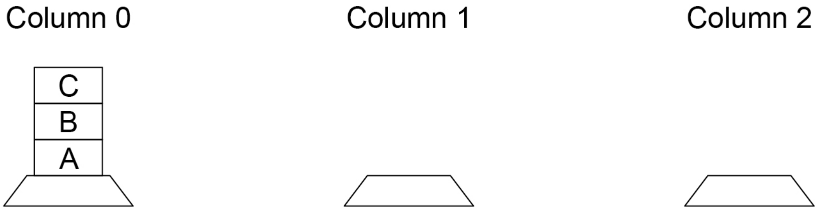
15 The following subroutines control the way that labelled blocks are placed in different columns.

`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.

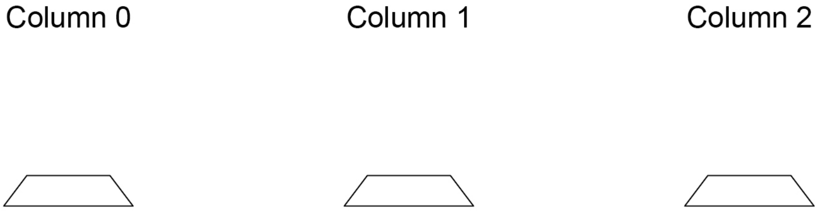
`HEIGHT(column)` returns the number of blocks in the specified column.

15 . 1 This is how the blocks A, B and C are arranged at the start.



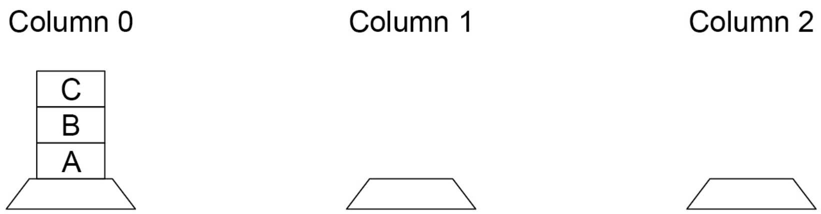
Draw the final arrangement of the blocks after the following algorithm has run.

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



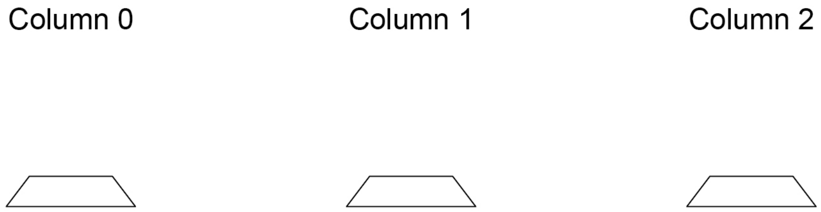
[3 marks]

1 5 . 2 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 marks]

Turn over for the next question

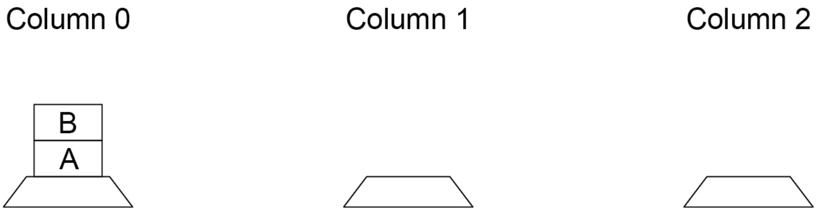
15.3 Develop an algorithm using either pseudo-code or a flowchart that will move every block from column 0 to column 1.

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.

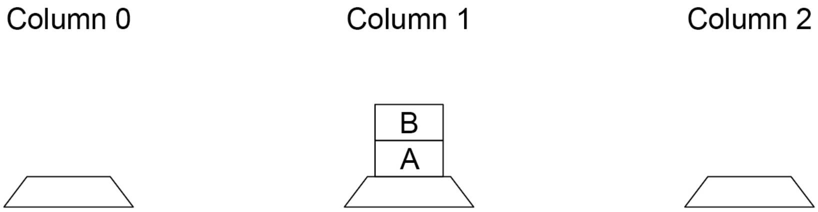
The order of the blocks must be preserved.

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:



[4 marks]

1 6 A programmer has written the C# program in **Figure 5** to add up the numbers between one and five.

Figure 5

```
int total = 0;
for (int number = 1; number < 6; number++)
{
  total = total + number;
}
Console.WriteLine(total);
```

The program needs to be changed so that it also multiplies all of the numbers between one and five.

Shade **one** lozenge next to the program that will do what the programmer wants.

[1 mark]

A	`int total = 0;`
	`int product = 1;`
	`for (int number = 1; number < 6; number++)`
	`{`
	`  total = total + number;`
	`  product = total * number;`
	`}`
	`Console.WriteLine(total);`
	`Console.WriteLine(product);`
		○

B	`int total = 0;`
	`int product = 1;`
	`for (int number = 1; number < 6; number++)`
	`{`
	`  total = total + number;`
	`  product = product * number;`
	`}`
	`Console.WriteLine(total);`
	`Console.WriteLine(product);`
		○

C	`int total = 0;`
	`int product = 1;`
	`for (int number = 1; number < 6; number++)`
	`{`
	`  total = total + number;`
	`  product = product * total;`
	`}`
	`Console.WriteLine(total);`
	`Console.WriteLine(product);`
		○

D	`int total = 0;`
	`int product = 1;`
	`for (int number = 1; number < 6; number++)`
	`{`
	`  total = total + number;`
	`  product = (total + product) * number;`
	`}`
	`Console.WriteLine(total);`
	`Console.WriteLine(product);`
		○

17 **Figure 8** shows a C# program.

Figure 8

```
static void First(int p1, int p2, int p3)
{
   int v1 = p2 + p3;
   Console.WriteLine(Second(v1, p1));
}

static int Second(int p1, int p2)
{
   int v1 = p1 + p2;
   if (v1 > 12)
   {
      v1 = v1 + Third(p1);
   }
   return v1;
}

static int Third(int p1)
{
   if (p1 > 3)
   {
      return 2;
   }
   else
   {
      return 0;
   }
}
```

17.1 State what will be displayed by the `Console.WriteLine` statement when the subroutine `First` is called with the values `3`, `4` and `4` for the parameters `p1`, `p2` and `p3`

[1 mark]

17.2 State what will be displayed by the `Console.WriteLine` statement when the subroutine `First` is called with the values `3`, `4` and `8` for the parameters `p1`, `p2` and `p3`

[1 mark]

1 8 A program is being written to solve a sliding puzzle.

- The sliding puzzle uses a 3 x 3 board.
- The board contains eight tiles and one blank space.
- Each tile is numbered from 1 to 8
- On each turn, a tile can only move one position up, down, left, or right.
- A tile can only be moved into the blank space if it is next to the blank space.
- The puzzle is solved when the tiles are in the correct final positions.

Figure 10 shows an example of how the tiles might be arranged on the board at the start of the game with the blank space in the position (0, 1).

Figure 11 shows the correct final positions for the tiles when the puzzle is solved.

The blank space (shown in black) is represented in the program as number `0`

Figure 10

		column		
		0	1	2
	0	4	■	2
row	1	1	7	6
	2	5	3	8

Figure 11

		column		
		0	1	2
	0	1	2	3
row	1	4	5	6
	2	7	8	■

Table 3 describes the purpose of three subroutines the program uses.

Table 3

Subroutine	**Purpose**
`getTile(row, column)`	Returns the number of the tile on the board in the position `(row, column)` For example: • `getTile(1, 0)` will return the value `5` if it is used on the board in **Figure 12** • `getTile(1, 2)` will return the value `0` if it is used on the board in **Figure 12**.
`move(row, column)`	Moves the tile in position `(row, column)` to the blank space, if the blank space is next to that tile. If the position `(row, column)` is not next to the blank space, no move will be made. For example: • `move(0, 2)` would change the board shown in **Figure 12** to the board shown in **Figure 13** • `move(2, 0)` would not make a move if used on the board shown in **Figure 12**.
`displayBoard()`	Displays the board showing the current position of each tile.

Figure 12

		column		
		0	1	2
	0	1	7	4
row	1	5	8	■
	2	6	2	3

Figure 13

		column		
		0	1	2
	0	1	7	■
row	1	5	8	4
	2	6	2	3

1 8 . 1 The C# program shown in **Figure 14** uses the subroutines in **Table 3**, on page 25.

The program is used with the board shown in **Figure 15**.

Figure 14

```
if (getTile(1, 0) == 0)
{
   move(2, 0);
}
if (getTile(2, 0) == 0)
{
   move(2, 1);
}
displayBoard();
```

Figure 15

column

	0	1	2
0	1	8	3
row 1	■	7	5
2	4	2	6

Complete the board to show the new positions of the tiles after the program in **Figure 14** is run.

[2 marks]

column

	0	1	2
0			
row 1			
2			

Figure 16 shows part of a C# program that uses the `getTile` subroutine from **Table 3**, on page 25.

The program is used with the board shown in **Figure 17**.

Figure 16

```
int ref1, ref2;
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (getTile(i, j) == 0)
        {
            ref1 = i;
            ref2 = j;
        }
    }
}
```

Figure 17

			column	
		0	1	2
	0	4	7	6
row	1	3	8	1
	2	■	5	2

18.2 Which **two** of the following statements about the program in **Figure 16** are **true** when it is used with the board in **Figure 17**?

Shade **two** lozenges.

[2 marks]

A Nested iteration is used. ○

B The final value of `ref1` will be `0` ○

C The number of comparisons made between `getTile(i, j)` and `0` will be nine. ○

D The outer loop, `for (int i = 0; i < 3; i++)`, will execute nine times. ○

E The values of `i` and `j` do not change when the program is executed. ○

Figure 16 and **Figure 17** are repeated below.

Figure 16

```
int ref1, ref2;
for (int i = 0; i < 3; i++)
{
   for (int j = 0; j < 3; j++)
   {
      if (getTile(i, j) == 0)
      {
         ref1 = i;
         ref2 = j;
      }
   }
}
```

Figure 17

		column		
		0	1	2
	0	4	7	6
row	1	3	8	1
	2	■	5	2

1 8 . 3 Explain the purpose of the **first** iteration structure in the program in **Figure 16**.

[1 mark]

1 8 . 4 Explain the purpose of the **second** iteration structure in the program in **Figure 16**.

[1 mark]

1 8 . 5 State the purpose of the program in **Figure 16**.

[1 mark]

1 8 . 6 **Table 4** shows a description of the `getTile` subroutine previously described in more detail in **Table 3**, on page 25.

Table 4

Subroutine	**Purpose**
`getTile(row, column)`	Returns the number of the tile on the board in the position `(row, column)`

Figure 18 and **Figure 19** show example boards.

Figure 18

		column		
		0	1	2
	0	5	2	■
row	1	1	3	4
	2	6	7	8

Figure 19

		column		
		0	1	2
	0	2	3	4
row	1	5	1	■
	2	7	8	6

Write a C# program to:

- check that in the first row:
  - o the second tile number is one more than the first tile number
  - o the third tile number is one more than the second tile number
- display `Yes` when the row meets both conditions above
- display `No` when the row does not meet both conditions above.

For example:

- for the board in **Figure 18**, the program would display `No`
- for the board in **Figure 19**, the program would display `Yes`

You **must** use the `getTile` subroutine in your C# code.

You **should** use meaningful variable name(s) and C# syntax in your answer.

The answer grid below contains vertical lines to help you indent your code accurately.

[4 marks]

1 8 . 7 **Table 5** describes the purpose of another two subroutines the program uses.

Table 5

Subroutine	**Purpose**
`solved()`	Returns `true` if the puzzle has been solved. Otherwise returns `false`
`checkSpace(row, column)`	Returns `true` if there is a blank space next to the tile on the board in the position `(row, column)` Otherwise returns `false`

Table 6 shows a description of the `move` subroutine previously described in more detail in **Table 3**, on page 25.

Table 6

Subroutine	**Purpose**
`move(row, column)`	Moves the tile in position `(row, column)` to the blank space, if the blank space is next to that tile. If the position `(row, column)` is not next to the blank space, no move will be made.

Write a C# program to help the user solve the puzzle.

The program should:

- get the user to enter the row number of a tile to move
- get the user to enter the column number of a tile to move
- check if the tile in the position entered is next to the blank space
  - o if it is, move that tile to the position of the blank space
  - o if it is not, output `Invalid move`
- repeat these steps until the puzzle is solved.

You **must** use the subroutines in **Table 5** and **Table 6**.

You **should** use meaningful variable name(s) and C# syntax in your answer.

The answer grid opposite contains vertical lines to help you indent your code accurately.

[6 marks]

19.1 State **one** property of local variables that is **not** true for all variables.

[1 mark]

19.2 Using C#, write a subroutine to help a museum review the number of visitors in a month.

The subroutine must:

- have the identifier `countDays`
- have the number of days a museum was open in the last month as a parameter
- get the user to enter the number of visitors to the museum for each of those days
- count how many of those days the museum had more than 200 visitors
- return the count.

You **should** use meaningful variable name(s) and C# syntax in your answer.

The answer grid below contains vertical lines to help you indent your code.

[6 marks]
