**Q1.**

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)**

**Q2.**

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)**

**Q3.**

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  to .

**Figure 2**

SUBROUTINE convert(cards)

  result ← ''

  max ← LEN(cards)

  index ← 0

  WHILE index < 

    rank ←  (cards[index])

    suit ← getSuit(cards[ + 1])

    c ← rank + ' of ' + suit + ' '

    result ← result + 

    index ← index + 2

  ENDWHILE

  RETURN 

ENDSUBROUTINE

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

  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(5)**

**(Total 9 marks)**

**Q4.**

(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:



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)**

**Q5.**

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)**

**Q6.**

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)**

**Q7.**

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)**

**Q8.**

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q9.**

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)**

**Q10.**

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)**

**Q11.**

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 |  |

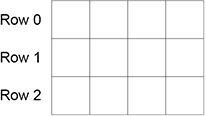
**(Total 3 marks)**

**Q12.**

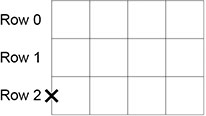
The **Algebraic Patent Sewing Machine** is a programmable sewing machine that creates patterns on rows of cloth. It is controlled by writing programs that use the following subroutines:

|  |  |
| --- | --- |
| **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 |

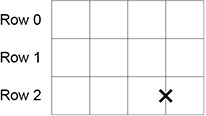
For example, if the cloth looks like this to begin with:



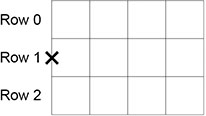
The subroutine call gotoRow(2) will place the sewing machine needle at the point shown by the black cross:



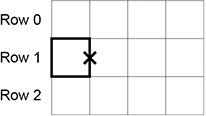
The subroutine call move(3) will move the sewing machine needle to the point shown by the black cross:



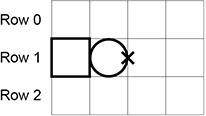
The subroutine call gotoRow(1) will move the sewing machine needle to the point shown by the black cross:



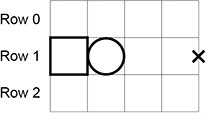
The subroutine call shape('square') will draw the following pattern and move the sewing machine needle to the point shown by the black cross:



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

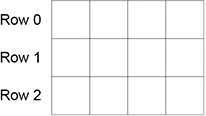
  move(1)

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)

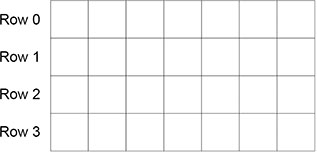
  move(k + 1)

  shape(patterns[k])

ENDFOR

You should draw your answer on the following grid.

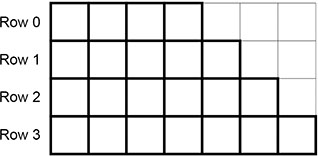
You do not need to show the position(s) of the needle in your answer.



**(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)**

**Q13.**

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)**

**Q14.**

Describe how the linear search algorithm works.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 3 marks)**

**Q15.**

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)**

**Q16.**

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)**

**Q17.**

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

Steps:

|  |
| --- |
|  |

**(Total 4 marks)**

Mark schemes

**Q1.**

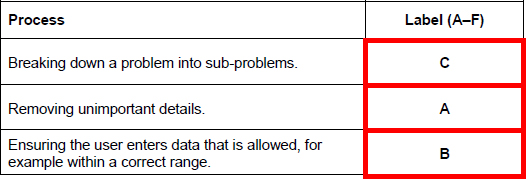
**3 marks for AO1 recall**

1 mark for 1 correct label;

2 marks for 2 correct labels;

3 marks for 3 correct labels;

Correct table is:



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

**[3]**

**Q2.**

(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'.

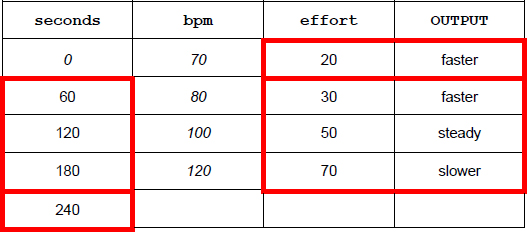
1 mark for the last three values in the effort column all correct **and** every output correct for these three values of effort;

**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:



**4**

**[7]**

**Q3.**

(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]**

**Q4.**

(a)  **3 marks for AO2**

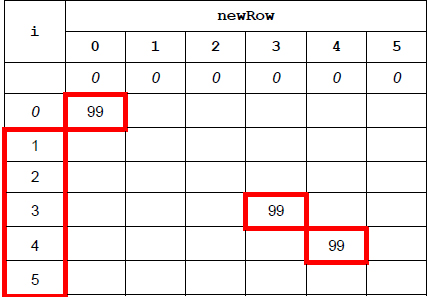
1 mark for i column correct;

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:



**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]**

**Q5.**

**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 |  |

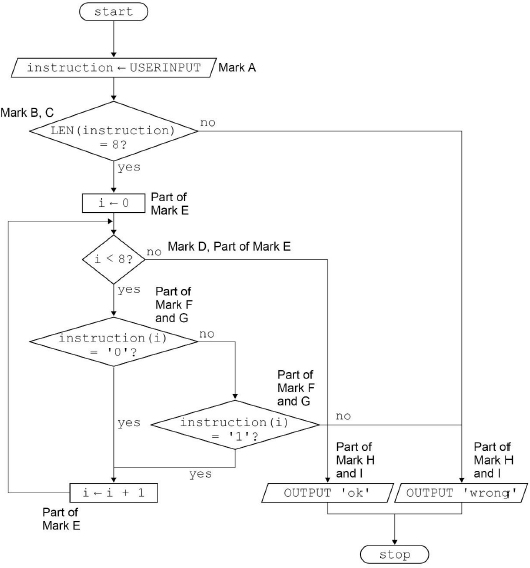
Another example of a completely correct solution:

|  |  |
| --- | --- |
| 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]**

**Q6.**

**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]**

**Q7.**

**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]**

**Q8.**

**Mark is for AO1 (understanding)**

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

**[1]**

**Q9.**

**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;

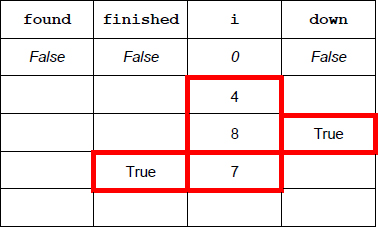
1 mark for finished being set only once to True;

**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:



**[4]**

**Q10.**

**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]**

**Q11.**

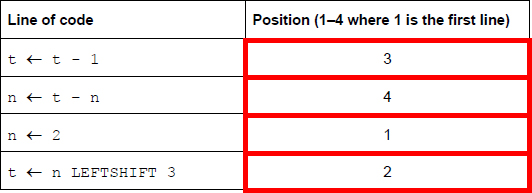
**3 marks for AO3 (programming)**

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



**[3]**

**Q12.**

(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;

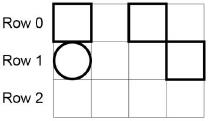
1 mark for drawing a circle at the start of row 1;

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)

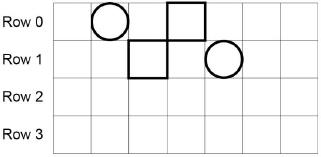
1 mark for drawing a square in the third column of row 1;

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 |  |

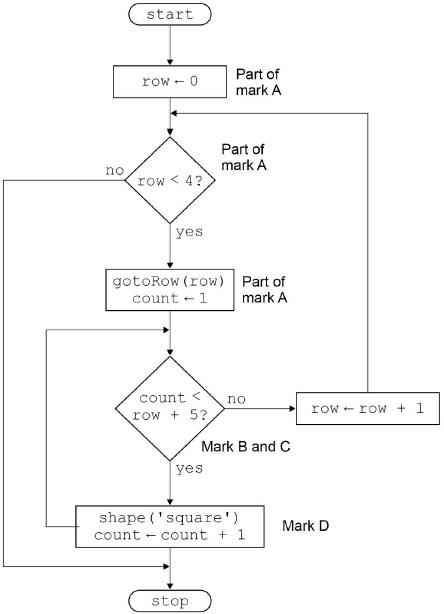
Another example of a fully correct answer:

|  |  |
| --- | --- |
| 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]**

**Q13.**

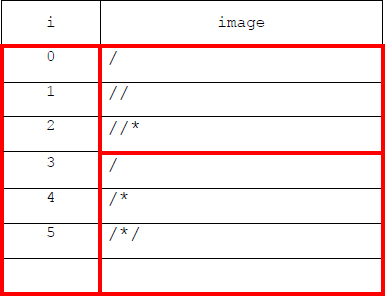
**3 marks for AO2 (apply)**

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.



**[3]**

**Q14.**

**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]**

**Q15.**

**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  regValid ← True | *[B,C]*  *[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 |  |

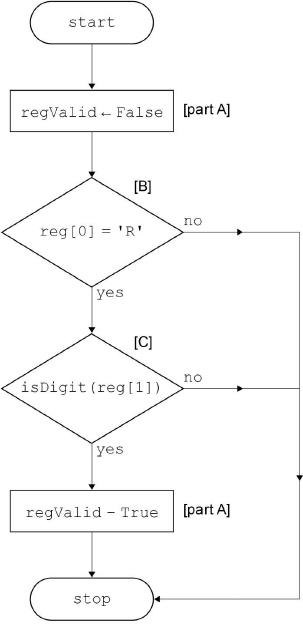
(only 2 marks awarded as the answer contains an error in the Boolean condition)

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]**

**Q16.**

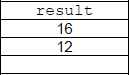
(a)  **2 marks for AO2 (apply)**

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:



**2**

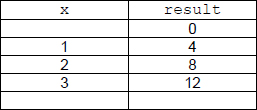
(b)  **2 marks for AO2 (apply)**

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



**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]**

**Q17.**

**4 marks for AO2 (apply)**

Maximum 4 marks from:

**If bubble sort chosen then:**

8 & 4 are swapped;

1 & 8 are swapped;

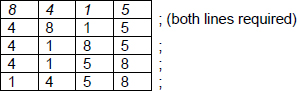
5 & 8 are swapped;

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:*

**

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

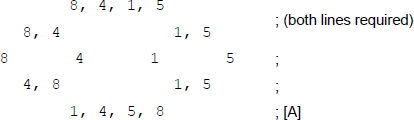
**If merge sort chosen then:**

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):



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

**[4]**