**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.**

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)**

**Q5.**

(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)**

**Q6.**

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)

Value returned  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(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)**

**Q7.**

A developer needs to store data about thousands of songs in a program. She needs to be able to hold information on every song’s title, singer and year of release.

Explain how the developer could use a combination of an array and records to store this information.

In your answer you should refer to the data types that would be used by the developer.

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

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

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

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

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

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

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

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

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

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

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

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

**(Total 4 marks)**

**Q8.**

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)**

**Q9.**

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)**

**Q10.**

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)**

**Q11.**

The pseudo-code below assigns two string values to two variables.

title ← 'computer science'

level ← 'gcse'

(a)  Shade **one** lozenge that shows the length of the contents of the variable level in the pseudo-code.

|  |  |  |
| --- | --- | --- |
| **A** | 1 |  |
| **B** | 2 |  |
| **C** | 3 |  |
| **D** | 4 |  |

**(1)**

(b)  Shade **one** lozenge that shows the result of concatenating the variable title with the variable level in the pseudo-code.

|  |  |  |
| --- | --- | --- |
| **A** | 'computer science gcse' |  |
| **B** | 'Computer Science GCSE' |  |
| **C** | 'computersciencegcse' |  |
| **D** | 'computer sciencegcse' |  |

**(1)**

(c)  Shade **one** lozenge to show which of the following strings is a substring of the variable title in the pseudo-code.

|  |  |  |
| --- | --- | --- |
| **A** | 'compsci' |  |
| **B** | 'puters' |  |
| **C** | 'sci' |  |
| **D** | 'tersci' |  |

**(1)**

(d)  The Unicode character code of title[0], which is 'c', is 99.

Shade **one** lozenge to show the Unicode character code of the character level[3] in the pseudo-code.

|  |  |  |
| --- | --- | --- |
| **A** | 95 |  |
| **B** | 99 |  |
| **C** | 101 |  |
| **D** | 103 |  |

**(1)**

**(Total 4 marks)**

**Q12.**

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)**

**Q13.**

A program has been written in Python to display all the odd integers between 1 and the largest odd number smaller than an integer entered by the user. The program is shown below.

odd = 1

number = int(input("Enter an integer: "))

while odd != number:

  print(odd)

  odd = odd + 2

print("Finished!")

The program works correctly if the integer entered by the user is an odd, positive integer. For example, if 7 is entered the program correctly displays the values 1, 3 and 5

The program does not work correctly if an odd integer less than 1 is entered by the user. For example, when -7 is entered the program should display the values 1, -1, -3 and -5 but it does not do this.

Using Python only, change the program code inside the while loop so that it will work correctly for any odd integer entered by the user.

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

**(Total 4 marks)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Mark schemes

**Q1.**

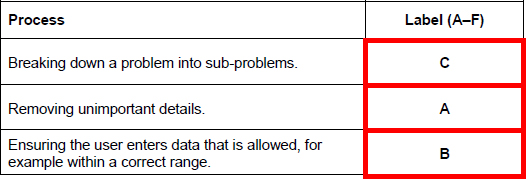
**3 marks for AO1 recall**

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'.

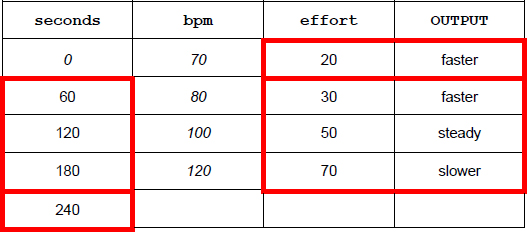
1 mark for the last three values in the effort column all correct **and** every output correct for these three values of effort;

**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.**

**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 |  |

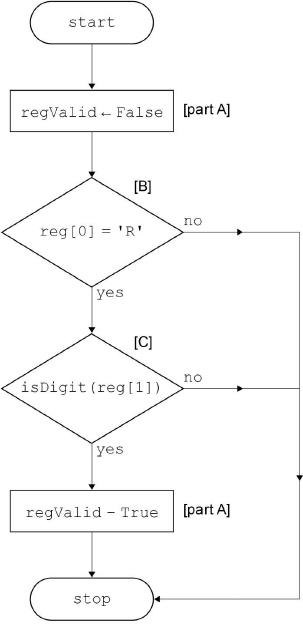
(only 2 marks awarded as the answer contains an error in the Boolean condition)

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]**

**Q5.**

(a)  **4 marks for AO2 (apply)**

first (calculated) value of 10;

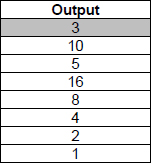
next calculated value of 5;

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.



**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]**

**Q6.**

(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:

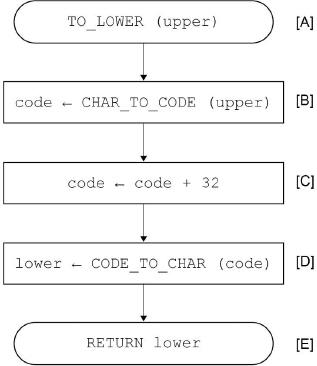
|  |  |
| --- | --- |
| SUBROUTINE TO\_LOWER(upper) | *[A]* |
| RETURN CODE\_TO\_CHAR(CHAR\_TO\_CODE(upper) + 32) | *[B,C,D,E]* |
| ENDSUBROUTINE |  |

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]**

**Q7.**

**4 marks for AO2 (apply)**

A record could be used to store the data of one song;

An array could store all of the songs/records;

One mark for one of the following, two marks for all three:

•   The song title could be a string

•   The singer could be a string

•   The year of release could be an integer/date.

**[4]**

**Q8.**

**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]* |

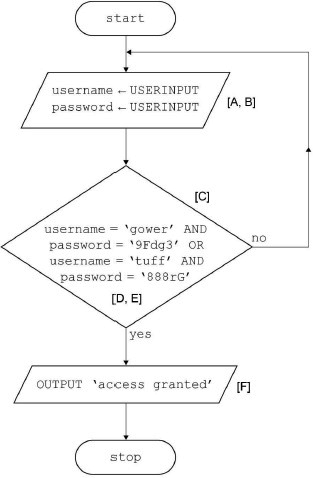
Another example of a fully correct answer:

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

**Q9.**

**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.

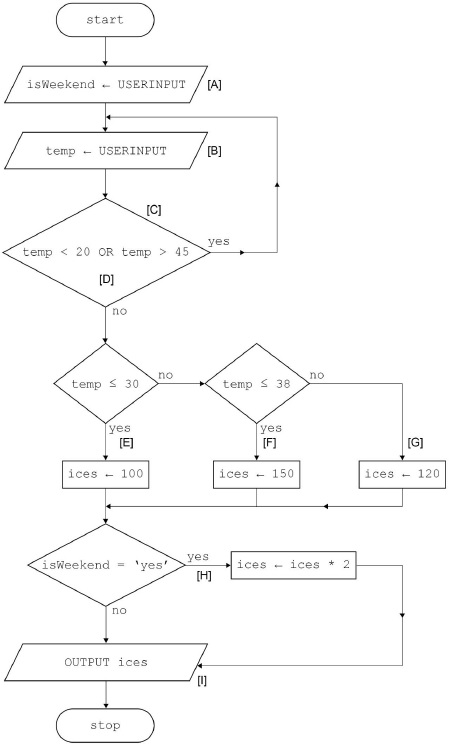
An example of a fully correct solution:

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

**Q10.**

(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.



**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.



**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.



**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]**

**Q11.**

(a)  **Mark is for AO2 (apply)**

**D** 4;

**If more than one lozenge shaded then mark is not awarded**

**1**

(b)  **Mark is for AO2 (apply)**

**D** 'computer sciencegcse';

**If more than one lozenge shaded then mark is not awarded**

**1**

(c)  **Mark is for AO2 (apply)**

**C** 'sci';

**If more than one lozenge shaded then mark is not awarded**

**1**

(d)  **Mark is for AO2 (apply)**

**C** 101;

**If more than one lozenge shaded then mark is not awarded**

**1**

**[4]**

**Q12.**

(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   eggsUsed ⟵ USERINPUT  UNTIL eggsUsed > 0 AND eggsUSED ≤ 8 | (A)  (B)  (C, D) |

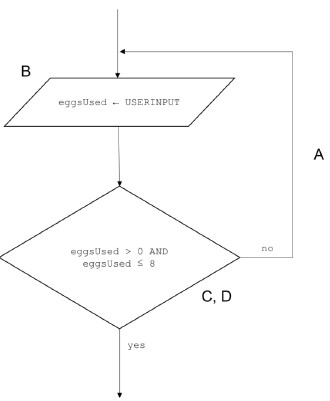
Example 4 mark answer:

|  |  |
| --- | --- |
| DO   eggsUsed ⟵ USERINPUT  WHILE eggsUsed < 1 OR eggsUSED > 8 | (A)  (B)  (C, D) |

Example 4 mark answer:

|  |  |
| --- | --- |
| REPEAT   eggsUsed ⟵ USERINPUT  UNTIL 0 < eggsUSED ≤ 8 | (A)  (B)  (C, D) |

Example 4 mark answer:



**4**

**[6]**

**Q13.**

**4 marks for AO3 (refine)**

**Program Logic**

**Mark A** for using a selection structure with else part **or** two selection structures (even if the syntax is incorrect)

**Mark B** for correct condition(s) in selection statement(s) (even if the syntax is incorrect)

**Mark C** for statement that subtracts two from odd under the correct conditions (even if the syntax is incorrect)

**Mark D** for odd being output and doing one of adding or subtracting two but not both each time loop repeats (even if the syntax is incorrect)

**I.** while loop from question if included in answer

**I.** case of program code

**Maximum 3 marks** if any errors in code.

**Python Example 1 (fully correct)**

|  |  |
| --- | --- |
| print(odd) | **(Part of D)** |
| if number < 0 | **(A, B)** |
| odd = odd – 2 | **(C, Part of D)** |
| else: |  |
| odd = odd + 2 | **(Part of D)** |

**Python Example 2 (partially correct – 3 marks)**

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

**[4]**