# Sample Section B Questions

# Paper 1

# Computer Science

**Q1.** The algorithm below, represented using pseudo-code, outputs a numeric result. The numeric result depends upon the value entered by the user.

```
OUTPUT "Enter a positive whole number: "
INPUT NumberIn

NumberOut ← 0

Count ← 0
WHILE NumberIn > 0

   Count ← Count + 1

   PartValue ← NumberIn MOD 2

   NumberIn ← NumberIn DIV 2

   FOR i ← 1 TO Count - 1

      PartValue ← PartValue * 10
   ENDFOR

   NumberOut ← NumberOut + PartValue
ENDWHILE
OUTPUT "The result is: " NumberOut
```

The table below lists the `MOD` and `DIV` operators for each of the available programming languages. You should refer to the row for your programming language.

| Programming language | MOD | DIV |
|---|---|---|
| C# | % | / |
| Java | % | / |
| Pascal | mod | div |
| Python | % | // |
| VB.Net | Mod | \ |

**What you need to do:**

**Task 1**
Write a program to implement the algorithm above.

**Task 2**
Test that your program works:

- run your program, then enter the number `22`
- run your program, then enter the number `29`
- run your program, then enter the number `-1`

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE for **Task 1**.

**(11)**

(b) SCREEN CAPTURE(S) showing the test described in **Task 2**.

**(1)**

(c) What is the purpose of this algorithm?

**(1)**

## Q2.

Create a folder / directory for your new program.

The algorithm, represented the using pseudo-code below, and the variable table underneath, describe the process of using a check digit to check if a value entered by the user is a valid 13 digit International Standard Book Number (ISBN).

```
FOR Count ← 1 TO 13 DO
  OUTPUT "Please enter next digit of ISBN: "
  INPUT ISBN[Count]
ENDFOR

CalculatedDigit ← 0

Count ← 1
WHILE Count

  CalculatedDigit ← CalculatedDigit + ISBN[Count]

  Count ← Count + 1

  CalculatedDigit ← CalculatedDigit + ISBN[Count] * 3

  Count ← Count + 1
ENDWHILE
WHILE CalculatedDigit >= 10 DO

  CalculatedDigit ← CalculatedDigit – 10
ENDWHILE

CalculatedDigit ← 10 – CalculatedDigit
IF CalculatedDigit = 10

  THEN CalculatedDigit ← 0
ENDIF
IF CalculatedDigit = ISBN[13]
   THEN OUTPUT "Valid ISBN"
   ELSE OUTPUT "Invalid ISBN"
ENDIF
```

| Identifier | Data Type | Purpose |
|---|---|---|
| ISBN | Array[1..13] Of Integer | Stores the 13 digit ISBN entered by the user – one digit is stored in each element of the array. |
| Count | Integer | Used to select a specific digit in the ISBN. |
| CalculatedDigit | Integer | Used to store the digit calculated from the first 12 digits of the ISBN. It is also used to store the intermediate results of the calculation. |

**What you need to do**

Write a program for the algorithm above.

Test the program by showing the result of entering the digits `9, 7, 8, 0, 0, 9, 9, 4, 1, 0, 6, 7, 6` (in that order).

Test the program by showing the result of entering the digits `9, 7, 8, 1, 8, 5, 7, 0, 2, 8, 8, 9, 4` (in that order).

Save the program in your new folder / directory.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(15)**

(b) SCREEN CAPTURE(S) for the test when the digits `9, 7, 8, 0, 0, 9, 9, 4, 1, 0, 6, 7, 6` are entered (in that order).

Your evidence must show the result of the test and, as a minimum, the last three digits entered for the test.

**(2)**

(c) SCREEN CAPTURE(S) for the test when the digits `9, 7, 8, 1, 8, 5, 7, 0, 2, 8, 8, 9, 4` are entered (in that order).

Your evidence must show the result of the test and, as a minimum, the last three digits entered for the test.

**(1)**

**(Total 18 marks)**

## Q3.

Write a program that gets **two** words from the user and then displays a message saying if the first word can be created using the letters from the second word or not.

For example:

- The word EAT can be formed from the word ATE as the first word uses one E, one A and one T and the second word also contains one of each of these letters.
- The word EAT can be formed from the word HEART as the second word contains one E, one A and one T which are the letters needed to form the first word.
- The word TO can be formed from the word POSITION as the second word contains one T and (at least) one O which are the letters needed to form the first word.
- The word MEET cannot be formed from the word MEAT as the second word only contains one E and two Es are needed to form the first word.

You may assume that the user will only enter words that consist of upper case letters.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(12)**

(b) SCREEN CAPTURE(S) showing the result of testing the program by entering:

- the word NINE followed by the word ELEPHANTINE.
- the word NINE followed by the word ELEPHANT

**(1)**

**(Total 13 marks)**

## Q4.

Create a folder/directory for your new program.

The variable table, the table below, and the Structured English algorithm below, describe a simplified version of a noughts and crosses match. A match consists of a user-specified number of games. In this simplified version, the two players complete each game on paper and then

enter information about the result of each game into a program that totals the number of games won by each player. Assume that all games have a winner – there are no drawn games.

| Identifier | Data Type | Purpose |
|---|---|---|
| NoOfGamesInMatch | Integer | Stores the number of games in the match (specified by user) |
| NoOfGamesPlayed | Integer | Stores the number of games played so far |
| PlayerOneScore | Integer | Stores the number of games won by Player One |
| PlayerTwoScore | Integer | Stores the number of games won by Player Two |
| PlayerOneWinsGame | Char | Stores a 'Y' if Player One won the game and 'N' otherwise |

```
PlayerOneScore " 0
PlayerTwoScore " 0
OUTPUT "How many games?"
INPUT NoOfGamesInMatch
FOR NoOfGamesPlayed " 1 TO NoOfGamesInMatch Do
   OUTPUT "Did Player One win the game (enter Y or N)?"
   INPUT PlayerOneWinsGame
   IF PlayerOneWinsGame = 'Y'
      THEN PlayerOneScore " PlayerOneScore + 1
      ELSE PlayerTwoScore " PlayerTwoScore + 1
   ENDIF
ENDFOR
OUTPUT PlayerOneScore
OUTPUT PlayerTwoScore
```

**What you need to do**

Write a program for the above algorithm.

Test the program by showing the results of a match consisting of three games where Player One wins the first game and Player Two wins the second and third games.

Save the program in your new folder/directory.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(9)**

(b) SCREEN CAPTURE(S) for the test described above.

**(4)**

**(Total 13 marks)**

## Q5.

Create a folder/directory for your new program.

The variable table, the table given below, and the Structured English algorithm, the diagram below, describe a linear search algorithm that could be used with a simplified version of the Dice Cricket game to find out if a particular player's name appears in the high score table.

In this simplified version only the names of the players getting a top score are stored. Their scores are **not** stored.

| Identifier | Data Type | Purpose |
|---|---|---|
| `Names` | `Array[1..4] of String` | Stores the names of the players who have one of the top scores |
| `PlayerName` | `String` | Stores the name of the player being looked for |
| `Max` | `Integer` | Stores the size of the array |
| `Current` | `Integer` | Indicates which element of the array `Names` is currently being examined |
| `Found` | `Boolean` | Stores `True` if the player's name has been found in the array, `False` otherwise |

```
Names[1] ← 'Ben'
Names[2] ← 'Thor'
Names[3] ← 'Zoe'
Names[4] ← 'Kate'
Max ← 4
Current ← 1
Found ← False
OUTPUT 'What player are you looking for?'
INPUT PlayerName
WHILE (Found = False) AND (Current <= Max)
  IF Names[Current] = PlayerName
    THEN Found ← True
    ELSE Current ← Current + 1
  ENDIF
ENDWHILE
IF Found = True
  THEN OUTPUT 'Yes, they have a top score'
  ELSE OUTPUT 'No, they do not have a top score'
ENDIF
```

**What you need to do**

- Write a program for the above algorithm.
- Test the program by searching for a player named 'Thor'.
- Test the program by searching for a player named 'Imran'.
- Save the program in your new folder/directory.

**Evidence that you need to provide.**

(a) Your PROGRAM SOURCE CODE.

**(11)**

(b) SCREEN CAPTURE(S) for the test searching for 'Thor'.

**(2)**

(c) SCREEN CAPTURE(S) for the test searching for ‘Imran’.

**(2)**

**(Total 15 marks)**

## Q6.

Create a folder/directory for your new program.

The variable table, **Table 1**, and the Structured English algorithm describe a simplified version of the **Guess the Word / Phrase Game**.

**Table 1**

| Identifier | Data Type | Purpose |
|---|---|---|

| | | |
|---|---|---|
| `NewWord` | `String` | Stores the setter's word to be guessed |
| `UserWordGuess` | `String` | Stores a word that is the user's guess |

```
OUTPUT "The new word?"
INPUT NewWord
OUTPUT "Your guess?"
INPUT UserWordGuess
IF UserWordGuess IS EQUAL TO NewWord
    THEN OUTPUT "CORRECT"
    ELSE OUTPUT "INCORRECT"
ENDIF
```

**What you need to do**

Write a program for the above algorithm in the programming language of your choice.

Test the program as follows.

**Test 1**: Input of the new word EAGLE followed by a correct guess.

**Test 2**: Input of the new word BEAR followed by an incorrect guess.

Save the program in your new folder / directory.

**Evidence that you need to provide**

(a) SCREEN CAPTURES for the following tests:

(i) Test 1

**(3)**

(ii) Test 2

**(3)**

(b) Your PROGRAM SOURCE CODE.

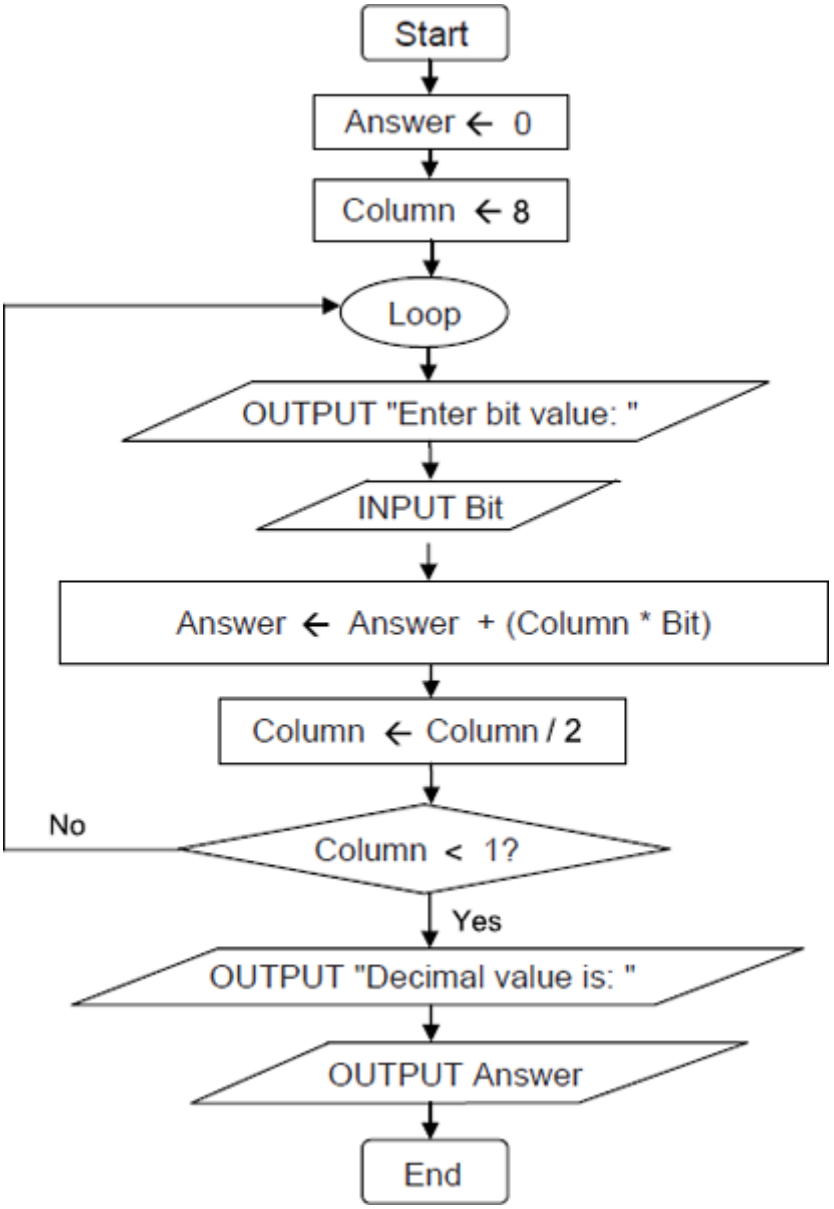
**(7)**

**(Total 13 marks)**

## Q7.

Create a folder/directory for your new program.

The algorithm, represented as a flowchart below, and the variable table, describe the converting of a 4-bit binary value into denary.




| Identifier | Data type | Purpose |
|---|---|---|
| `Column` | `Integer` | Stores the place value (column heading) |
| `Answer` | `Integer` | Stores the denary value equivalent to the bit pattern entered by the user |
| `Bit` | `Integer` | Stores a `0` or `1` entered by the user |

**What you need to do**

Write a program for the above algorithm.

Test the program by showing the result of entering the values `1, 1, 0, 1 (in that order).`

Save the program in your new folder/directory.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(11)**

(b) SCREEN CAPTURE(S) for the test described above.

**(3)**

(c) What is the largest denary number that could be output by the algorithm represented by the flowchart in the diagram above?

_____

**(1)**

(d) The algorithm represented by the flowchart above can convert sixteen different bit patterns into denary.

If the symbol | Column ⟵ 8 | is changed to | Column ⟵ 16 | how many **more** bit patterns could be converted into denary?

_____

**(1)**

(e) When developing a new system the stages of the systems development life cycle could be followed.

At which stage of the systems development life cycle would the flowchart above have been created?

_____

**(1)**

(f) At which stage of the systems development life cycle would the algorithm represented by the flowchart above be automated using a programming language?

_____

**(1)**

**(Total 18 marks)**

## Q8.

Create a folder / directory for your new program.

The algorithm, represented using pseudo-code in **Figure 1**, and the variable table, **Table 1**, describe a simple two player game. Player One chooses a whole number between 1 and 10

(inclusive) and then Player Two tries to guess the number chosen by Player One. Player Two gets up to five attempts to guess the number. Player Two wins the game if they correctly guess the number, otherwise Player One wins the game.

Note that in **Figure 1**, the symbol < > means "is not equal to".

**Figure 1**

```
OUTPUT "Player One enter your chosen number: "
INPUT NumberToGuess
WHILE NumberToGuess < 1 OR NumberToGuess > 10 DO
      OUTPUT "Not a valid choice, please enter another number: "
      INPUT NumberToGuess
ENDWHILE

Guess ← 0

NumberOfGuesses ← 0
WHILE Guess <> NumberToGuess AND NumberOfGuesses < 5 DO
      OUTPUT "Player Two have a guess: "
      INPUT Guess

      NumberOfGuesses ← NumberOfGuesses + 1
ENDWHILE
IF Guess = NumberToGuess
   THEN OUTPUT "Player Two wins"
   ELSE OUTPUT "Player One wins"
```

**Table 1**

| Identifier | Data type | Purpose |
|---|---|---|
| `NumberToGuess` | `Integer` | Stores the number entered by Player One |
| `NumberOfGuesses` | `Integer` | Stores the number of guesses that Player Two has made so far |
| `Guess` | `Integer` | Stores the most recent guess made by Player Two |

**What you need to do**

Write a program for the above algorithm.

Test the program by conducting the tests **Test 1** and **Test 2**.

Save the program in your new folder / directory.

**Test 1**
Test that your program works correctly by conducting the following test:

- Player One enters the number `0`
- Player One enters the number `11`
- Player One enters the number `5`
- Player Two enters a guess of `5`

**Test 2**
Test that your program works correctly by conducting the following test:

- Player One enters the number `6`

- Player Two enters guesses of `1, 3, 5, 7, 10`

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(13)**

(b) SCREEN CAPTURE(S) showing the result of **Test 1**.

**(4)**

(c) SCREEN CAPTURE(S) showing the result of **Test 2**.

**(3)**

Part of the algorithm from **Figure 1** is shown in **Figure 2.**
Note that in **Figure 2**, the symbol <> means "is not equal to".

**Figure 2**

```
WHILE Guess < > NumberToGuess AND NumberOfGuesses < 5 DO
    OUTPUT "Player Two have a guess: "
    INPUT Guess

    NumberOfGuesses ← NumberOfGuesses + 1
ENDWHILE
```

(d) Explain why a `WHILE` repetition structure was chosen instead of a `FOR` repetition structure for the part of the algorithm shown in **Figure 2**.

_____

_____

**(1)**

**(Total 21 marks)**

## Q9.

Create a folder / directory in this **question** for your new program.
The algorithm, represented using pseudo-code below, and the variable table, describe a program that calculates and displays all of the prime numbers between 2 and 50, inclusive.

The `MOD` operator calculates the remainder resulting from an integer division
eg `10 MOD 3 = 1`.

If you are unsure how to use the `MOD` operator in the programming language you are using, there are examples of it being used in the **Skeleton Program**.

```
OUTPUT "The first few prime numbers are:"
FOR Count1 ← 2 TO 50 DO
  Count2 ← 2
  Prime ← "Yes"
  WHILE Count2 * Count2 <= Count1 DO
    IF (Count1 MOD Count2 = 0) THEN
      Prime ← "No"
    ENDIF
    Count2 ← Count2 + 1
  ENDWHILE
  IF Prime = "Yes" THEN
    OUTPUT Count1
  ENDIF
ENDFOR
```

| Identifier | Data Type | Purpose |
|---|---|---|
| `Count1` | `Integer` | Stores the number currently being checked for primeness |
| `Count2` | `Integer` | Stores a number that is being checked to see if it is a factor of `Count1` |
| `Prime` | `String` | Indicates if the value stored in `Count1` is a prime number or not |

**What you need to do**

Write a program for the algorithm above.

Run the program and test that it works correctly.

Save the program in your new **Question** folder / directory.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(11)**

(b) SCREEN CAPTURE(S) for the test showing the correct working of the program.

**(1)**

(c) Describe the changes that would need to be made to the algorithm shown above, so that instead of displaying the prime numbers between 2 and 50, inclusive, it displays all the prime numbers between 2 and a value input by the user, inclusive.

_____

_____

_____

_____

_____

_____

**(3)**

**(Total 15 marks)**

## Q10.

The algorithm, represented using pseudo-code below, outputs a series of numbers. The contents of the series depends on the initial value entered by the user.

```
Number ← 0
WHILE (Number < 1) OR (Number > 10)
```

```
      OUTPUT "Enter a positive whole number: "
      INPUT Number
      IF Number > 10 THEN
         OUTPUT "Number too large."
      ELSE
         IF Number < 1 THEN
            OUTPUT "Not a positive number."
         ENDIF
      ENDIF
   ENDWHILE

   c ← 1

   FOR k ← 0 TO Number - 1
      OUTPUT c

      c ← (c * (Number - 1 - k)) DIV (k + 1)
   ENDFOR
```

The `Div` operator calculates the result of an integer division, for example, `10 DIV 3 = 3`.

**What you need to do:**

**Task 1**
Write a program to implement the algorithm above.

**Task 2**
Test the program by showing the result of entering:
`-3`
then `11`
then `10`

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE for **Task 1**.

**(9)**

(b) SCREEN CAPTURE(S) showing the test described in **Task 2**.

**(1)**

**(Total 10 marks)**

## Q11.

Write a program that checks which numbers from a series of numbers entered by the user are prime numbers.

The program should get a number from the user and then display the messages:

- `"Not greater than 1"` if the number entered is 1 or less
- `"Is prime"` if the number entered is a prime number
- `"Is not prime"` otherwise.

The user should then be asked if they want to enter another number and the program should repeat if they say that they do.

A prime number is a positive integer that will leave a remainder if it is divided by any positive integer other than 1 and itself.

You may assume that each number entered by the user is an integer.

If your program only works correctly for some prime numbers you will get some marks for this question. To get full marks for this question, your program must work correctly for any valid integer value that the user enters.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(12)**

(b) SCREEN CAPTURE(S) showing the result of testing the program by:
- entering the number `1`
- then choosing to enter another number
- then entering the number `5`
- then choosing to enter another number
- then entering the number `8`
- and then choosing not to enter another number.

**(1)**

**(Total 13 marks)**

## Q12.

The algorithm represented using pseudo-code in **Figure 1** describes a method to find the greatest common factor (GCF) of two whole numbers (integers) entered by the user.

**Figure 1**

```
OUTPUT "Enter a whole number: "
```

```
INPUT Number1
OUTPUT "Enter another whole number: "
INPUT Number2

Temp1 ← Number1

Temp2 ← Number2
WHILE Temp1 ≠ Temp2
   IF Temp1 > Temp2 THEN

      Temp1 ← Temp1 - Temp2
   ELSE

      Temp2 ← Temp2 - Temp1
   ENDIF
ENDWHILE

Result ← Temp1
OUTPUT Result, " is GCF of ", Number1, " and ", Number2
```

**What you need to do:**

**Task 1**
Write a program to implement the algorithm in **Figure 1**.

**Task 2**
Test the program by showing the result of entering `12` and then `39`.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE for **Task 1**.

**(6)**

(b) SCREEN CAPTURE(S) showing the test described in **Task 2**.

**(1)**

The algorithm copies the values of `Number1` and `Number2` into `Temp1` and `Temp2` respectively.

(c) Explain why the `WHILE` loop was written using `Temp1` and `Temp2` instead of `Number1` and `Number2`.

_____

_____

**(1)**

**(Total 8 marks)**

## Q13.

One method that can be used to compress text data is run length encoding (RLE). When RLE is used the compressed data can be represented as a set of character/frequency pairs. When the same character appears in consecutive locations in the original text it is replaced in the compressed text by a single instance of the character followed by a number indicating the number of consecutive instances of that character. Single instances of a character are represented by the character followed by the number 1.

**Figure 1** and **Figure 2** show examples of how text would be compressed using this method.

**Figure 1**

Original text: `AAARRRRGGGHH`
Compressed text: `A 3 R 4 G 3 H 2`

**Figure 2**

Original text: `CUTLASSES`
Compressed text: `C 1 U 1 T 1 L 1 A 1 S 2 E 1 S 1`

**What you need to do**

**Task 1**
Write a program that will perform the compression process described above. The program should display a suitable prompt asking the user to input the text to compress and then output the compressed text.

**Task 2**
Test the program works by entering the text `AAARRRRGGGHH`.

**Task 3**
Test the program works by entering the text `A`.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE.

**(12)**

(b) SCREEN CAPTURE(S) for the test showing the output of the program when `AAARRRRGGGHH` is entered.

**(1)**

(c) SCREEN CAPTURE(S) for the test showing the output of the program when `A` is entered.

**(1)**

**(Total 14 marks)**

## Q14.

Create a folder / directory for your new program.

One method for converting a decimal number into binary is to repeatedly divide by 2 using integer division. After each division is completed, the remainder is output and the integer result of the division is used as the input to the next iteration of the division process. The process repeats until the result of the division is 0.

Outputting the remainders in the sequence that they are calculated produces the binary digits of the equivalent binary number, but in reverse order.

For example, the decimal number 210 could be converted into binary as shown below.

```
210 ÷ 2 =105   remainder 0
105 ÷ 2 =  52   remainder 1
 52 ÷ 2 =  26   remainder 0
 26 ÷ 2 = 13   remainder 0
 13 ÷ 2 =  6   remainder 1
  6 ÷ 2 =  3   remainder 0
  3 ÷ 2 =  1   remainder 1
  1 ÷ 2 =  0   remainder 1
```

The sequence 0, 1, 0, 0, 1, 0, 1, 1 which would be output by this process is the reverse of the binary equivalent of 210 which is 11010010.

**What you need to do**

**Task 1**
Write a program that will perform the conversion process described above. The program should display a suitable prompt asking the user to input a decimal number to convert and then output the bits of the binary equivalent of the decimal number in reverse order.

**Task 2**
Improve the program so that the bits are output in the correct order, e.g. for 210 the output would be 11010010.

**Task 3**
Test the program works by entering the value 210.

Save the program in your new folder / directory.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE after you have completed both **Task 1** and **Task 2**.

If you complete **Task 1** but do not attempt **Task 2** then a maximum of 9 marks will be awarded.

**(12)**

(b) SCREEN CAPTURE(S) for the test showing the output of the program when 210 is entered.

The marks for this test will be awarded whether the binary digits are output in reverse order or in the correct order.

**(2)**

**(Total 14 marks)**

## Q15.

**Figure 1** contains the pseudo-code for a program to output a sequence according to the 'Fizz Buzz' counting game.

**Figure 1**

```
OUTPUT "How far to count?"
INPUT HowFar
WHILE HowFar < 1
  OUTPUT "Not a valid number, please try again."
  INPUT HowFar
```

```
ENDWHILE
FOR MyLoop ← 1 TO HowFar
  IF MyLoop MOD 3 = 0 AND MyLoop MOD 5 = 0
  THEN
      OUTPUT "FizzBuzz"
    ELSE
      IF MyLoop MOD 3 = 0
        THEN
          OUTPUT "Fizz"
        ELSE
          IF MyLoop MOD 5 = 0
            THEN
              OUTPUT "Buzz"
            ELSE
              OUTPUT MyLoop
          ENDIF
      ENDIF
  ENDIF
ENDFOR
```

**What you need to do:**

Write a program that implements the pseudo-code as shown in **Figure 1**.

Test the program by showing the result of entering a value of `18` when prompted by the program.

Test the program by showing the result of entering a value of `-1` when prompted by the program.

**Evidence that you need to provide**

(a) Your PROGRAM SOURCE CODE for the pseudo-code in **Figure 1**.

**(8)**

(b) SCREEN CAPTURE(S) for the tests conducted when a value of `18` is entered by the user and when a value of `-1` is entered by the user.

**(1)**

The main part of the program uses a `FOR` repetition structure.

(c) Explain why a `FOR` repetition structure was chosen instead of a `WHILE` repetition structure.

_____

_____

**(1)**

(d) Even though a check has been performed to make sure that the variable `HowFar` is greater than `1` there could be inputs that might cause the program to terminate unexpectedly (crash).

Provide an example of an input that might cause the program to terminate and describe a method that could be used to prevent this.

_____

_____

_____

_____

_____

_____

**(3)**

(e) Programs written in a high level language are easier to understand and maintain than programs written in a low level language.

The use of meaningful identifier names is one way in which high level languages can be made easier to understand.

State **three** other features of high level languages that can make high level language programs easier to understand.

_____

_____

_____

_____

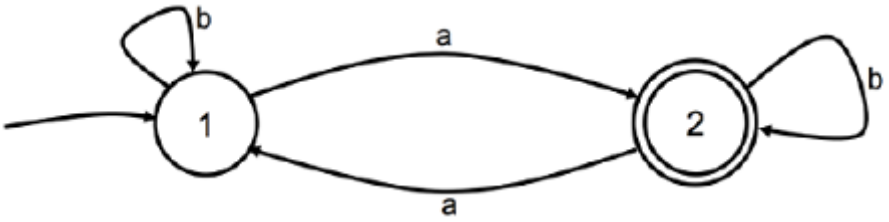
_____

_____

**(3)**

(f) The finite state machine (FSM) shown in **Figure 2** recognises a language with an alphabet of `a` and `b`.

**Figure 2**




Input strings of `a` and `aabba` would be accepted by this FSM.

In the table below indicate whether each input string would be accepted or not accepted by the FSM in **Figure 2**.

If an input string would be accepted write YES.
If an input string would **not** be accepted write NO.

| Input string | Accepted by FSM? |
|---|---|
| aaab | |
| abbab | |
| bbbbba | |

**(2)**

(g) In words, describe the language (set of strings) that would be accepted by this FSM shown in **Figure 2**.

_____

_____

_____

_____

**(2)**

**(Total 20 marks)**

## Mark schemes

## Q1.

(a) **All marks for AO3 (programming)**

**Mark as follows:**

1) Correct variable declarations for `NumberIn`, `NumberOut`, `Count`, `PartValue`;

   **Note to examiners**
   If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the correct variables exist in the program code and the first value they are assigned is of the correct data type.

2) Correct prompt `"Enter a positive whole number: "` and `NumberIn` assigned value entered by user;
3) Correct initialisation of `NumberOut` and `Count`;
4) `WHILE` loop with syntax allowed by the programming language and correct condition for termination of the loop;
5) Correct incrementation of `Count` within `WHILE` loop;
6) Correct assignment to `PartValue` within `WHILE` loop but before `FOR` loop;
7) Correct updating of `NumberIn` within `WHILE` loop but before `FOR` loop;
8) `FOR` loop with syntax allowed by the programming language over correct range;
9) Correct assignment to `PartValue` inside `FOR` loop;
10) Correct calculation of `NumberOut` after `FOR` loop but within `WHILE` loop;
11) Output statement giving correct output after `WHILE` loop;

**I.** Ignore minor differences in case and spelling

**Max 10** if code does not function correctly

**VB.net**

```
Dim NumberIn, NumberOut, Count, PartValue As Integer
Console.Write("Enter a positive whole number: ")
NumberIn = Console.ReadLine
NumberOut = 0
Count = 0
While NumberIn > 0
  Count += 1
  PartValue = NumberIn Mod 2
  NumberIn \= 2
  For i = 1 To Count - 1
    PartValue *= 10
  Next
  NumberOut += PartValue
End While
Console.WriteLine("The result is: " &38; NumberOut)
Console.ReadLine()
```

**Python 3**

```
NumberIn = int(input('Enter a positive whole number: '))
NumberOut = 0
Count = 0
while NumberIn > 0:
  Count += 1
  PartValue = NumberIn % 2
  NumberIn = NumberIn // 2
  for i in range(1, Count):
    PartValue = PartValue * 10
  NumberOut = NumberOut + PartValue
print('The result is: ', NumberOut)
```

### Python 2

```
NumberIn = int(raw_input('Enter a positive whole number: '))
NumberOut = 0
Count = 0
while NumberIn > 0:
  Count += 1
  PartValue = NumberIn % 2
  NumberIn = NumberIn // 2
  for i in range(1, Count):
    PartValue = PartValue * 10
  NumberOut = NumberOut + PartValue
print 'The result is: ', NumberOut
```

### Pascal

```
var
  NumberIn, NumberOut, Count, PartValue, i: integer;
begin
  write('Enter a positive whole number: ');
  readln(NumberIn);
  NumberOut := 0;
  Count := 0;
  while NumberIn > 0 do
    begin
      Count := Count + 1;
      PartValue := NumberIn mod 2;
      NumberIn := NumberIn div 2;
      for i := 1 to Count - 1 do
        PartValue := PartValue * 10;
      NumberOut := NumberOut + PartValue;
    end;
  writeln('The result is: ', NumberOut);
end;
```

### C#

```
int count = 0, partValue, numberIn, numberOut = 0;
Console.Write("Enter a positive whole number: ");
numberIn = Convert.ToInt32(Console.ReadLine());
while (numberIn > 0)
{
  count++;
  partValue = numberIn % 2;
  numberIn = numberIn / 2;
  for (int i = 1; i > count; i++)
  {
    partValue = partValue * 10;
  }
  numberOut = numberOut + partValue;
}
```

```
Console.WriteLine("The result is: " + numberOut );
Console.ReadLine();
```

**Java**

```
Console.writeLine("Enter a positive whole number: ");
int numberIn = Integer.parseInt(Console.readLine());
int numberOut = 0;
int count = 0;
int partValue;
while (numberIn > 0) {
  count++;
  partValue = numberIn % 2;
  numberIn = numberIn / 2;
  for (int i = 1; i > count; i++) {
    partValue = partValue * 10;
  }
  numberOut = numberOut + partValue;
}
Console.writeLine("The result is: " + numberOut);
```

**11**

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

****** SCREEN CAPTURE ******
*Must match code from part (a), including prompts on screen capture matching those in code.*
*Code for part (a) must be sensible.*

*Screen capture showing:*
*'22' being entered and the message 'The result is: 10110' displayed*
*'29' being entered and the message 'The result is: 11101' displayed*
*'-1' being entered and the message 'The result is: 0' displayed*

```
Enter a positive whole number: 22
The result is: 10110
>>>
Enter a positive whole number: 29
The result is: 11101
>>>
Enter a positive whole number: -1
The result is: 0
>>>
```

**1**

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

converts from (positive) decimal/denary to binary;

**1**

**[13]**

## Q2.

(a) Correct variable declarations for `ISBN, CalculatedDigit` and `Count;`

`For` loop, with syntax allowed by the programming language, set up to repeat the correct number of times;
Correct prompt `"Please enter next digit of ISBN: ";`

Followed by `ISBN[Count]` assigned value entered by the user – must be

inside the 1st iterative structure;

`CalculatedDigit` and `Count` initialised correctly (must be after 1st iterative structure and before 2nd iterative structure);

2nd loop has syntax allowed by the programming language and correct condition for the termination of the loop; **A** alternative correct logic for condition

`CalculatedDigit` assigned the value of its original value added to `ISBN[Count]` followed by incrementing `Count` – both inside the loop;

`CalculatedDigit` assigned the value of its original value added to `ISBN[Count] * 3` followed by incrementing `Count` – must be in the loop and after the 1st two assignment statements in the loop;

3rd loop has syntax allowed by the programming language and correct condition for the termination of the loop; **A** alternative correct logic for the condition

10 subtracted from value of `CalculatedDigit` and result assigned to `CalculatedDigit` – must be the only statement inside an iterative structure;

Assignment statement `CalculatedDigit ← 10 - CalculatedDigit` - must not be in an iteration or selection structure;

1st `IF` statement with correct condition – must not be in an iterative structure – with `CalculatedDigit` being assigned the value 0 inside the selection structure;

2nd `IF` statement with correct condition – must not be in an iterative structure or inside the 1st selection structure;

Correct output message (`Valid ISBN`) in `THEN` part of selection structure;

Correct output message (`Invalid ISBN`) in `ELSE` part of selection structure;

**I** Case of variable names and output messages
**A** Minor typos in variable names and output messages
**I** Spacing in prompts
**A** Initialisation of variables at declaration stage
**A** Arrays using positions 0 to 12 instead of 1 to 13

**Pascal**

```
Program Question4;
Var
  CalculatedDigit : Integer;
  ISBN : Array[1..13] Of Integer;
  Count : Integer;
Begin
  For Count := 1 To 13
    Do
      Begin
        Writeln('Please enter next digit of ISBN: ');
        Readln(ISBN[Count]);
      End;
  CalculatedDigit := 0;
  Count := 1;
```

```
  While Count < 13
    Do
      Begin
        CalculatedDigit := CalculatedDigit + ISBN[Count];
        Count := Count + 1;
        CalculatedDigit := CalculatedDigit + ISBN[Count] * 3;
        Count := Count + 1;
      End;
    While CalculatedDigit >= 10
      Do CalculatedDigit := CalculatedDigit - 10;
    CalculatedDigit := 10 - CalculatedDigit;
    If CalculatedDigit = 10
      Then CalculatedDigit := 0;
    If CalculatedDigit = ISBN[13]
      Then Writeln('Valid ISBN')
      Else Writeln('Invalid ISBN');
    Readln;
End.
```

**VB.Net**

```
Module Module1
  Sub Main()
    Dim CalculatedDigit As Integer
    Dim ISBN(13) As Integer
    Dim Count As Integer
    For Count = 1 To 13
      Console.Write("Please enter next digit of ISBN: ")
      ISBN(Count) = Console.ReadLine()
    Next
    CalculatedDigit = 0
    Count = 1
    While Count < 13
      CalculatedDigit = CalculatedDigit + ISBN(Count)
      Count = Count + 1
      CalculatedDigit = CalculatedDigit + ISBN(Count) * 3
      Count = Count + 1
    End While
    While CalculatedDigit >= 10
      CalculatedDigit = CalculatedDigit - 10
    End While
    CalculatedDigit = 10 - CalculatedDigit
    If CalculatedDigit = 10 Then
      CalculatedDigit = 0
    End If
    If CalculatedDigit = ISBN(13) Then
      Console.WriteLine("Valid ISBN")
    Else
      Console.WriteLine("Invalid ISBN")
    End If
    Console.ReadLine()
  End Sub
End Module
```

**VB6**

```
Private Sub Form_Load()
  Dim CalculatedDigit As Integer
  Dim ISBN(13) As Integer
  Dim Count As Integer
  For Count = 1 To 13
    ISBN(Count) = ReadLine("Please enter next digit of ISBN: ")
  Next
  CalculatedDigit = 0
  Count = 1
```

```
  While Count < 13
    CalculatedDigit = CalculatedDigit + ISBN(Count)
    Count = Count + 1
    CalculatedDigit = CalculatedDigit + (ISBN(Count) * 3)
    Count = Count + 1
  Wend
  While CalculatedDigit >= 10
    CalculatedDigit = CalculatedDigit - 10
  Wend
  CalculatedDigit = 10 - CalculatedDigit
  If CalculatedDigit = 10 Then
    CalculatedDigit = 0
  End If
  If CalculatedDigit = ISBN(13) Then
    WriteLine("Valid ISBN")
  Else
    WriteLine("Invalid ISBN")
  End If
End Sub
```

**Alternative answers could use some of the following instead of WriteLine / ReadLine:**

```
Console.Text = Console.Text & ...
WriteLineWithMsg
WriteWithMsg
Msgbox
InputBox
WriteNoLine
```

**Python 2**

```
# Question 4
if __name__ == "__main__":
  ISBN = [None, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  for Count in range(1, 14):
    print 'Please enter next digit of ISBN: ',
    ISBN[Count] = int(raw_input())
  CalculatedDigit = 0
  Count = 1
  while Count < 13:
    CalculatedDigit = CalculatedDigit + ISBN[Count]
    Count = Count + 1
    CalculatedDigit = CalculatedDigit + ISBN[Count] * 3
    Count = Count + 1
  while CalculatedDigit >= 10:
    CalculatedDigit = CalculatedDigit - 10
  CalculatedDigit = 10 - CalculatedDigit
  if CalculatedDigit == 10:
    CalculatedDigit = 0
  if CalculatedDigit == ISBN[13]:
    print 'Valid ISBN'
  else:
    print 'Invalid ISBN'
```

**Alternative print/input combination:**

```
ISBN[Count] = int(raw_input('Please enter next digit of ISBN:
',))
```

**Python 3**

```
# Question 4
if __name__ == "__main__":
  ISBN = [None, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for Count in range(1, 14):
```

```
    print('Please enter next digit of ISBN: '),
    ISBN[Count] = int(input())
  CalculatedDigit = 0
  Count = 1
  while Count < 13:
    CalculatedDigit = CalculatedDigit + ISBN[Count]
    Count = Count + 1
    CalculatedDigit = CalculatedDigit + ISBN[Count] * 3
    Count = Count + 1
  while CalculatedDigit >= 10:
    CalculatedDigit = CalculatedDigit - 10
  CalculatedDigit = 10 - CalculatedDigit
  if CalculatedDigit == 10 :
    CalculatedDigit = 0
  if CalculatedDigit == ISBN[13]:
    print('Valid ISBN')
  else:
    print('Invalid ISBN')
```

**Alternative print/input combination:**

```
ISBN[Count] = int(input('Please enter next digit of ISBN: ',))
```

**Java**

```
public class Question4 {
  AQAConsole2014 console = new AQAConsole2014();

  public Question4() {
    int ISBN[] = new int[14];
    int count;
    int calculatedDigit;
    for (count = 1; count <= 13; count++) {
      ISBN[count] = console.readInteger("Please enter next digit
of ISBN: ");
    }
    calculatedDigit = 0;
    count = 1;
    while (count < 13) {
      calculatedDigit = calculatedDigit + ISBN[count];
      count++;
      calculatedDigit = calculatedDigit + ISBN[count] * 3;
      count++;
    }
    while (calculatedDigit >= 10) {
      calculatedDigit = calculatedDigit - 10;
    }
    calculatedDigit = 10 - calculatedDigit;
    if (calculatedDigit == 10) {
      calculatedDigit = 0;
    }
    if (calculatedDigit == ISBN[13]) {
      console.println("Valid ISBN");
    } else {
      console.println("Invalid ISBN");
    }
  }

  public static void main(String[] args) {
    new Question4();
  }
}
```

(b) ****SCREEN CAPTURE****
*Must match code from part (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.*

**Mark as follows:**
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 4
'Please enter next digit of ISBN: ' + user input of 1
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 6
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 6;
'Valid ISBN ' message shown;

**A** Alternative output messages if match code for part (a)
**A** If can only see some of the latter user inputs (e.g. due to first few inputs scrolling off the top of the console screen) – but must be able to see the last three digits entered (6, 7, 6)

**2**

(c) ****SCREEN CAPTURE****
*Must match code from part (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.*

**Mark as follows:**
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 1
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 5
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 2
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 4
'Invalid ISBN ' message shown;

**A** Alternative output messages if match code for part (a)
**A** If can only see some of the latter user inputs (e.g. due to first few inputs scrolling off the top of the console screen) – but must be able to see the last three digits entered (8, 9, 4)

**1**

**[18]**

## Q3.

(a) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

**Mark Scheme**

| Level | Description | Mark range |
|---|---|---|
| 4 | A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met. | 10-12 |
| 3 | There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays relevant prompts, inputs the two words and includes one iterative structure and two selection structures. An attempt has been made to check that all the characters in the first word are in the second word, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made. | 7-9 |
| 2 | A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly. | 4-6 |
| 1 | A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised. | 1-3 |

**Guidance**

**Evidence of AO3 design – 4 points:**

Evidence of design to look for in responses:

1. Identifying that a selection structure is needed after all letter counts have been compared to output a message saying it can be made from the letters in the 2nd word or that it can't.
2. Identifying that a loop is needed that repeats a number of times based on the length of the first word // identifying that a loop is needed that repeats 26 times // identifying that a loop is needed that repeats a number of times determined by the number of unique characters in the first word
3. Identifying that the number of times a letter occurs in the first string needs to be less than or equal to the number of times it occurs in the second string
4. Boolean (or equivalent) variable used to indicate if the first word can be formed from the letters in the second word // array of suitable size to store the count of each letter

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

**Evidence for AO3 programming – 8 points:**

Evidence of programming to look for in response:

5. (Suitable prompts asking user to enter the two words followed by) user inputs being assigned to appropriate variables (**R.** if inside or after iterative structure), two variables with appropriate data types created to store the two words entered by the user
6. Iterative structure to look at each letter in first word has correct syntax and start/end conditions // iterative structure to look at each letter in the alphabet has correct syntax and start/end conditions
7. Correctly counts the number of times that a letter occurs in one of the words
8. Selection structure that compares the count of a letter in the first word with the count of that letter in the second word **A.** incorrect counts **A.** incorrect comparison operator
9. Correctly counts the number of times each letter in one of the two words occurs
10. Program works correctly if the two words entered are the same
11. Program works correctly when first word contains more instances of a letter than there are in the second word (i.e. says that it cannot be formed from the second word)
12. Program works correctly for all word pairs consisting of just upper case letters

**Alternative mark scheme**

(based on removing an instance of a letter from the 2nd word each time it appears in the 1st word)

1. Identifying that a selection structure is needed after all the letters that appear in both words have been removed from the first word to output a message saying it can be made from the letters in the second word or that it can't
3. Identifying that a letter can be removed from the second word if it appears in the first word
7. Selection structure that checks if letter in first word appears in the second word
8. Removes a letter from the second word if it appears in the first word
9. Sets indicator to false if a letter does not appear in the second word

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

****** SCREEN CAPTURE ******

*Must match code from part (a), including prompts on screen capture matching those in code.*

*Code for part (a) must be sensible.*

Screen captures showing:

- the string NINE being entered followed by the string ELEPHANTINE and then a message displayed saying that the first word can be formed from the second.
- the string NINE being entered followed by the word ELEPHANT and then a message displayed saying that the first work cannot be formed from the second.

**1**

**[13]**

## Q4.

(a) **VB.NET**

```
Sub Main()
    Dim PlayerOneScore, PlayerTwoScore, NoOfGamesPlayed,
NoOfGamesInMatch As Integer
    Dim PlayerOneWinsGame As Char

    PlayerOneScore = 0
    PlayerTwoScore = 0
    Console.Write("How many games?")
    NoOfGamesInMatch = Console.ReadLine()
    For NoOfGamesPlayed = 1 To NoOfGamesInMatch
        Console.Write("Did Player One win the game (enter Y or
N)?")
        PlayerOneWinsGame = Console.ReadLine
        If PlayerOneWinsGame = "Y" Then
            PlayerOneScore = PlayerOneScore + 1
        Else
            PlayerTwoScore = PlayerTwoScore + 1
        End If
    Next
    Console.WriteLine(PlayerOneScore)
    Console.WriteLine(PlayerTwoScore)
    Console.ReadLine()
End Sub
```

**VB6**

```
Private Sub Form_Load()
    Dim PlayerOneScore As Integer
    Dim PlayerTwoScore As Integer
    Dim NoOfGamesPlayed As Integer
    Dim NoOfGamesInMatch As Integer
    Dim PlayerOneWinsGame As String

    PlayerOneScore = 0
    PlayerTwoScore = 0
    NoOfGamesInMatch = InputBox("How many games?")
    For NoOfGamesPlayed = 1 To NoOfGamesInMatch
```

```
                PlayerOneWinsGame = InputBox("Did Player One
        win the game (enter Y or N)?")
                If PlayerOneWinsGame = "Y" Then
                    PlayerOneScore = PlayerOneScore + 1
                Else
                    PlayerTwoScore = PlayerTwoScore + 1
                End If
            Next
            MsgBox (PlayerOneScore)
            MsgBox (PlayerTwoScore)
        End Sub
```

### Alternative answer – one msgbox instead of two

```
MsgBox (PlayerOneScore & vbCrLf & PlayerTwoScore)
```

### Pascal

```
Program Question;
Var PlayerOneScore, PlayerTwoScore, NoOfGamesPlayed,
NoOfGamesInMatch:Integer;
Var PlayerOneWinsGame:Char;
Begin
    PlayerOneScore := 0;
    PlayerTwoScore := 0;
    Writeln('How many games?');
    Readln(NoOfGamesInMatch);
    For NoOfGamesPlayed := 1 To NoOfGamesInMatch Do
        Begin
            Write('Did Player One win the game (enter Y or N)?');
            Readln(PlayerOneWinsGame);
            If PlayerOneWinsGame = 'Y'
                Then PlayerOneScore := PlayerOneScore + 1
                Else PlayerTwoScore := PlayerTwoScore + 1;
            End;
    Writeln(PlayerOneScore);
    Writeln(PlayerTwoScore);
    Readln();
End.
```

### Java

```
package comp1_java_2010_v4a; //this is optional
public class Question {
    public static void main(String[] args) {
         int noOfGamesInMatch;
         int noOfGamesPlayed;
         int playerOneScore;
         int playerTwoScore;
         char playerOneWinsGame;
         Console console = new Console();
         playerOneScore = 0;
         playerTwoScore = 0;
        noOfGamesInMatch = console.readInteger("How many games?
");
       for (noOfGamesPlayed = 1; noOfGamesPlayed <=
noOfGamesInMatch; noOfGamesPlayed++) {
           playerOneWinsGame = console.readChar("Did player One
win the game (enter Y or N)? ");
           if (playerOneWinsGame == 'Y') {
               playerOneScore++;
           } else {
               playerTwoScore++;
           } // end if/else
     } // end for noOfGamesPlayed
     console.writeLine(playerOneScore);
```

```
     console.writeLine(playerTwoScore);
  } // end Question
}
```

**Python 2.5**

```
PlayerOneScore = 0
PlayerTwoScore = 0
NoOfGamesPlayed = 0
NoOfGamesInMatch = int(raw_input("How many games?"))
# accept input(("How many games?")
for NoOfGamesPlayed in range(NoOfGamesInMatch):
    PlayerOneWinsGame = raw_input("Did Player One win the game
(enter Y or N)?")
    If PlayerOneWinsGame == 'Y':
        PlayerOneScore = PlayerOneScore + 1
        # accept PlayerOneScore += 1
    else:
         PlayerTwoScore = PlayerTwoScore + 1
         #accept PlayerTwoScore += 1
print PlayerOneScore
print PlayerTwoScore
```

**Python 3.0**

```
PlayerOneScore = 0
PlayerTwoScore = 0
NoOfGamesInMatch = int(input("How many games?"))
# Accept:
# print("How many games?")
# NoOfGamesInMatch = int(input())
for NoofGamesPlayed in range(NoOfGamesInMatch):
    PlayerOneWinsGame = input("Did Player One win the game (enter
Y or N)?")
    If PlayerOneWinsGame == 'Y':
         PlayerOneScore = PlayerOneScore + 1
         # accept PlayerOneScore += 1
    else:
         PlayerTwoScore = PlayerTwoScore + 1
         # accept PlayerTwoScore += 1
print(PlayerOneScore)
print(PlayerTwoScore)
```

**A.** `NoOfGamesPlayed = 0`

**Mark as follows:**
All variables declared correctly;
**I** Case
**A** Minor typos
**R** If additional variables
`PlayerOneScore, PlayerTwoScore+` initialised correctly;
Correct prompt ( **I** Case **A** minor typos) followed by `NoOfGamesInMatch` assigned value entered by user;
FOR loop formed correctly including end of loop in correct place;
Correct prompt ( **I** Case **A** minor typos) followed by `PlayerOneWinsGame` assigned value entered by user;
IF followed by correct condition;
**R** if does not cater for capital letter 'Y'
THEN followed by correct assignment statement;
ELSE followed by correct assignment statement;
output of both player's scores after loop;
**A** Message displayed with score

(b) ****SCREEN CAPTURE****
Must match code from 16, including prompts on screen capture matching those in code

*Mark as follows:*
'How many games?' + user input of '3';
'Did Player One win the game (enter Y or N)? ' + user input of 'Y';
'N' entered by user for second / third game;
Correct scores shown for each player ( **A** follow through);
**I** spacing

**4**

**[13]**

## Q5.

(a) **VB.Net**

```
Sub Main()
    Dim Names(4) As String
    Dim Current As Integer
    Dim Max As Integer
    Dim Found As Boolean
    Dim PlayerName As String

    Names(1) = "Ben"
    Names(2) = "Thor"
    Names(3) = "Zoe"
    Names(4) = "Kate"
    ;Max = 4
    Current = 1
    Found = False
    Console.WriteLine("What player are you looking for?")
    PlayerName = Console.ReadLine
    While Found = False And Current <= Max
        If Names(Current) = PlayerName Then
            Found = True
        Else
            Current = Current + 1
        End If
    End While
    If Found = True Then
        Console.WriteLine("Yes, they have a top score")
    Else
        Console.WriteLine("No, they do not have a top score")
    End If
    Console.ReadLine()
End Sub
```

**VB6**

```
Private Sub Form_Load()
    Dim Names(4) As String    A. Names(1 To 4)
    Dim Current As Integer
    Dim Max As Integer
    Dim Found As Boolean
    Dim PlayerName As String

    Names(1) = "Ben"
    Names(2) = "Thor"
    Names(3) = "Zoe"
    Names(4) = "Kate"
    Max = 4
```

```
    Current = 1
    Found = False
    PlayerName = InputBox("What player are you looking for?")
    While Found = False And Current <= Max
        If Names(Current) = PlayerName Then
            Found = True
        Else
            Current = Current + 1
        End If
    End While
    If Found = True Then
        MsgBox("Yes, they have a top score")
    Else
        MsgBox("No, they do not have a top score")
    End If
    End
End Sub
```

### Pascal

```
Program Question;
Var
   Names : Array[1..4] Of String;
   Current : Integer;
   Max : Integer;
   Found : Boolean;
   PlayerName : String;
Begin
   Names[1] := 'Ben';
   Names[2] := 'Thor';
   Names[3] := 'Zoe';
   Names[4] := 'Kate';
   Max := 4;
   Current := 1;
   Found := False;
   Writeln('What player are you looking for?');
   Readln(PlayerName);
   While (Found = False) And (Current <= Max)
      Do
         Begin
            If Names[Current] = PlayerName
               Then Found := True
               Else Current := Current + 1;
         End;
   If Found = True
      Then Writeln('Yes, they have a top score')
      Else Writeln('No, they do not have a top score');
   Readln;
End.
```

### Java

```
public class Question {

    AQAConsole console = new AQAConsole();

    public Question() {
         String[] names = new String[5];
         int max;
         int current;
         boolean found;
         String playerName;

         names[1] = "Ben";
         names[2] = "Thor";
```

```
        names[3] = "Zoe";
        names[4] = "Kate";

//possible alternative, which declares and
//instantiates in one.
//String[] names={"","Ben","Thor","Zoe","Kate"};

        current = 1;
        max = 4;
        found = false;

        playerName = console.readLine("What player are you
    looking for? ");
        while ((found == false) && (current <= max)) {
           if (names[current].equals(playerName)){
             found = true;
           } else {
             current++;
           } // end if/else
        } // end while

        if (found == true) {
            console.println("Yes, they have a top score");
        } else {
                    console.println("No, they do not have a top
    score");
        } // end if/else
     }// end CONSTRUCTOR
    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
    new Question();
    }
}
```

**Python 2.6**

```
Names = ["", "", "", "", ""]
Names[1] = "Ben"
Names[2] = "Thor"
Names[3] = "Zoe"
Names[4] = "Kate"
# Or:
# Names["", "Ben","Thor", "Zoe","Kate"]

# Or:
# Names = [""]
# Names.append("Ben")
# Names.append("Thor")
# Names.append("Zoe")
# Names.append("Kate")

Max = 4
Current = 1
Found = False
PlayerName = raw_input("What player are you looking
for?")
while (Found == False) and (Current <= Max):
     if Names[Current] == PlayerName:
         Found = True
     else:
         Current += 1
if Found == True: # accept if Found:
```

```
     print "Yes, they do have a top score"
else:
     print "No, they do not have a top score"
```

**A** Answers where `Max` is set to 5 and loop condition of `Current < Max`

**A** Answers where `Max` is set to 4 and loop condition of `Current < Max + 1`

**Python 3**

```
Names = ["", "", "", "", ""]
Names[1] = "Ben"
Names[2] = "Thor"
Names[3] = "Zoe"
Names[4] = "Kate"
# Or:
# Names["", "Ben","Thor", "Zoe","Kate"]

# Or:
# Names = [""]
# Names.append("Ben")
# Names.append("Thor")
# Names.append("Zoe")
# Names.append("Kate")

Max = 4
Current = 1
Found = False
PlayerName = input("What player are you looking
for?")
while (Found == False) and (Current <= Max):
     if Names[Current] == PlayerName:
         Found = True
     else:
         Current += 1
if Found == True: # accept if Found:
     print("Yes, they do have a top score")
else:
     print("No, they do not have a top score")
```

**A** Answers where `Max` is set to 5 and loop condition of `Current < Max`

**A** Answers where `Max` is set to 4 and loop condition of `Current <`

**Mark as follows:**

Correct variable declarations for `Max, Current, Found, PlayerName` and correct declaration for the `Names` array;

Four correct values assigned to the correct positions in the Names array;

`Max, Current, Found` initialised correctly;

Correct prompt followed by `PlayerName` assigned value entered by user;

`WHILE` loop formed correctly and correct conditions for the termination of the loop;

First `IF` followed by correct condition and `IF` statement is inside the loop;

`THEN` followed by correct assignment statement within a correctly formed `IF` statement;

`ELSE` followed by correct assignment statement within a correctly formed `IF` statement;

Second `IF` followed by correct condition and `IF` is after the loop;

`THEN` followed by correct output within a correctly formed `IF` statement;

`ELSE` followed by correct output within a correctly formed `IF` statement;

**I** Case of variable names, player names and output messages
**A** Minor typos in variable names and output messages
**A** Max declared as a constant instead of a variable
**A** Alternative conditions with equivalent logic for the loop
**A** Array positions 0–3 used instead of 1–4 if consistent usage throughout program

**11**

(b) * * * * SCREEN CAPTURE* * * *
*Must match code from (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.*

**Mark as follows:**
'What player are you looking for' + user input of ' Thor' ;
'Yes, they have a top scor' message shown;
**I** spacing
**R** If code for (a) would not produce this test run

**2**

(c) * * * * SCREEN CAPTURE* * * *
*Must match code from (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.*

**Mark as follows:**
'What player are you looking for?' + user input of 'Imran' ;
'No, they do not have a top score' message shown;
**I** spacing
**R** If code for (a) would not produce this test run

**2**

**[15]**

## Q6.

(a) (i) **** SCREEN CAPTURE ****
"The new word?" + setter input 'EAGLE' ;
input of correct guess 'EAGLE' ; (**A** 'eagle' if code in (b) has evidence for use of function Ucase, .ToUpper, etc.)
correct logic demonstrated with "CORRECT" ;
**NB** VB6 – all three stages must be evidenced

**3**

(ii) **** SCREEN CAPTURE ****
setter input 'BEAR'
"Your guess ?" + any incorrect guess ;
correct logic demonstrated with "INCORRECT" ;
**NB** VB6 – all three stages must be evidenced

**3**

(b) **Visual Basic**

```
Dim NewWord As String
Dim UserWordGuess As String
Console.Write("The new word?")
NewWord = Console.ReadLine

Console.Write("Your guess?")
UserWordGuess = Console.ReadLine
```

```
If UserWordGuess = NewWord

    Then Console.WriteLine("CORRECT")
    Else Console.WriteLine("INCORRECT")
End If
```

**Pascal**

```
Var
    NewWord : String;
    UserWordGuess : String;

Begin
    Write('The new word?');
    Readln(NewWord);
    Write('Your guess?');
    Readln(UserWordGuess);
    If UserWordGuess = NewWord
      Then Writeln('CORRECT')
      Else Writeln('INCORRECT');

    Readln;
End.
```

**Mark as follows:**
evidence of two variables declared ;
data types appropriate to the language for both variables ;
correct two identifier names used – NewWord UserWordGuess ;
(**A** case variations)

correct user prompt "The new word?" (**A** 'imprecise') ;

correctly formed IF followed by condition;
THEN clause followed by the logically correct output (**A** 'imprecise') ;
ELSE clause ; followed by the logically correct output (**A** 'imprecise') ;

**NB.** Two separate IF statements scores Maximum 2

**JAVA**

```
class Question4 {

    Console console = new Console();
    String newWord = "";
    String userWordGuess;

    public Question4(){
        newWord=console.readLine("The new word?");
        userWordGuess=console.readLine("Your guess?");
        if(userWordGuess.equals(newWord)) {
            console.println("CORRECT");
        } else {
            console.println("INCORRECT");
        } // end if / else
    } // end construct or

    public static void main(String[] args) {
        new Question4();
        System.exit(0);
```

```
    } // end Main
} // end Question4
```

**Max 7**

**Python**

```
NewWord = raw_input("The new word?")
UserWordGuess = raw_input("Your Guess?")
if UserWordGuess == NewWord:
    print "CORRECT"
else:
    print "INCORRECT"
raw_input() # keep window on screen
```

**Max 7**

**[27]**

## Q7.

(a) Correct variable declarations for `Bit, Answer` and `Column;`
**I** additional variable declarations
`Column` initialised correctly before the start of the loop;
`Answer` initialised correctly before the start of the loop;
`While/Repeat` loop, with syntax allowed by the programming language used, after the variable initialisations; and correct condition for the termination of the loop;
**R** `For` loop
**A** any `While/Repeat` loop with logic corresponding to that in flowchart
(for a loop with a condition at the start accept >=1 or >0 but reject <>0)
Correct prompt `"Enter bit value:"` ;
followed by `Bit` assigned value entered by user;
followed by `Answer` given new value;
**R** if incorrect value would be calculated [followed by value of `Column` divided by 2;
**A** multiplying by 0.5
Correct prompt and the assignment statements altering `Bit, Answer` and `Column` are all within the loop;
After the loop – output message followed by value of `Answer`;
**I** Case of variable names, player names and output messages
**A** Minor typos in variable names and output messages
**I** spacing in prompts
**A** answers where formatting of decimal values is included e.g.
`Writeln('Decimal value is: ', Answer : 3)`
**A** `initialisation of variables at declaration stage`
**A** `no brackets around column * bit`

**Pascal**

```
Program Question;
    Var
         Answer : Integer;
         Column : Integer;
         Bit : Integer;
    Begin
          Answer := 0;
          Column := 8;
          Repeat
              Writeln('Enter bit value: ');
              Readln(Bit);
```

```
            Answer := Answer + (Column * Bit);
            Column := Column DIV 2;
        Until Column < 1;
        Writeln('Decimal value is: ', Answer);
        Readln;
    End.
```

### VB.NET

```
Sub Main()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Console.Write("Enter bit value: ")
        Bit = Console.ReadLine
        Answer = Answer + (Column * Bit)
        Column = Column / 2
   Loop Until Column < 1
   Console.Write("Decimal value is: " & Answer)
   Console.ReadLine()
End Sub
```

### Alternative Answer

```
Column = Column \ 2
```

### VB6

```
Private Sub Form_Load()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Bit = InputBox("Enter bit value: ")
        Answer = Answer + (Column * Bit)
        Column = Column / 2
   Loop Until Column < 1
   MsgBox ("Decimal value is: " & Answer)
End Sub
```

### Alternative Answer

```
Column = Column \ 2
```

### Java

```
public class Question {
   AQAConsole console=new AQAConsole();
   public Question(){
         int column;
         int answer;
         int bit;
         answer=0;
         column=8;
         do{
              console.print("Enter bit value: ");
              bit=console.readInteger("");
              answer=answer+(column*bit);
              column=column/2;
         }while(column>=1);
         console.print("Decimal value is: ");
         console.println(answer);
      }
```

```
      public static void main(String[] arrays){
         new Question();
      }
}
```

**Python 2.6**

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print "Enter bit value: "
   # Accept: Bit = int(raw_input("Enter bit value: "))
   Bit = input()
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print "Decimal value is: ", Answer
# or + str(Answer)
```

**Python 3.1**

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print("Enter bit value: ")
   # Accept: Bit = int(input("Enter bit value: "))
   Bit = int(input())
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print("Decimal value is: " + str(Answer))
# or print("Decimal value is: {0}".format(Answer))
```

**A.** Answer and Bit not declared at start as long as they are spelt correctly and when they are given an initial value that value is of the correct data type

**11**

(b) ****SCREEN CAPTURE****
*Must match code from 16, including prompts on screen capture matching those in code*

**Mark as follows:**
"Enter bit value:" + first user input of 1
'Enter bit value: ' + second user input of 1
'Enter bit value: ' + third user input of 0
'Enter bit value: ' + fourth user input of 1
Value of 13 outputted;

**3**

(c) 15;

**1**

(d) 16 // twice as many // double;

**1**

(e) Design;
**A** Planning

**1**

(f) Implementation;

**1**

**[18]**

## Q8.

(a) Correct variable declarations for `Guess, NumberOfGuesses` and `NumberToGuess;`
Correct prompt "`Player One enter your chosen number:` ";
Followed by `NumberToGuess` assigned value entered by the user;
1st loop has syntax allowed by the programming language and one correct condition for the termination of the loop;
**A** alternative correct logic for condition
1st loop has syntax allowed by the programming language and has 2nd correct condition for the termination of the loop;
**A** alternative correct logic for condition
Correct prompt "`Not a valid choice, please enter another number:` " followed by `NumberToGuess` assigned value entered by the user – must be inside the 1st iteration structure;
`Guess` and `NumberOfGuesses` initialised correctly;
2nd loop has syntax allowed by the programming language and both correct conditions for the termination of the loop and is after the initialising of `Guess` and `NumberOfGuesses;`
**A** alternative correct logic for conditions
Correct prompt "`Player Two have a guess:` " followed by `Guess` assigned value entered by the user – must be inside the 2nd iteration structure;
`NumberOfGuesses` incremented inside the 2nd iteration structure;
`If` statement with correct condition – must not be in an iterative structure;
Correct output message in `Then` part of selection structure;
Correct output message in `Else` part of selection structure;
**I** Case of variable names and output messages
**A** Minor typos in variable names and output messages
**I** spacing in prompts
**A** initialisation of variables at (or immediately after) declaration stage

**Pascal**

```
Program Question4;
  Var
    NumberToGuess : Integer;
    NumberOfGuesses : Integer;
    Guess : Integer;
  Begin
    Write('Player One enter your chosen number: ');
    Readln(NumberToGuess);
    While (NumberToGuess  10)
Do
     Begin
      Write('Not a valid choice, please enter
another number: ');
      Readln(NumberToGuess);
     End;
    Guess := 0;
NumberOfGuesses := 0;
While (Guess < > NumberToGuess) And
(NumberOfGuesses < 5) Do
     Begin
      Write('Player Two have a guess:');
      Readln(Guess);
```

```
      NumberOfGuesses := NumberOfGuesses + 1;
     End;
    If Guess = NumberToGuess
      Then Write('Player Two wins')
      Else Write('Player One wins');
    Readln;
     End.
```

**VB.Net**

```
Module Module1
  Sub Main()
    Dim NumberToGuess As Integer
    Dim NumberOfGuesses As Integer
    Dim Guess As Integer
    Console.Write("Player One enter your chosen
number: ")
    NumberToGuess = Console.ReadLine()
    While NumberToGuess < 1 Or NumberToGuess > 10
        Console.Write("Not a valid choice, please
enter another number: ")
        NumberToGuess = Console.ReadLine()
    End While
    Guess = 0
    NumberOfGuesses = 0
    While Guess <> NumberToGuess And NumberOfGuesses
< 5
        Console.Write("Player Two have a guess: ")
        Guess = Console.ReadLine()
        NumberOfGuesses = NumberOfGuesses + 1
    End While
    If Guess = NumberToGuess Then
        Console.Write("Player Two wins")
    Else
        Console.Write("Player One wins")
    End If
    Console.ReadLine()
  End Sub
End Module
```

**VB6**

```
Private Sub Form_Load()
    Dim NumberToGuess As Integer
    Dim NumberOfGuesses As Integer
    Dim Guess As Integer
    NumberToGuess = ReadLine("Player One enter your
chosen number: ")
    While NumberToGuess < 1 Or NumberToGuess > 10
      NumberToGuess = ReadLine("Not a valid choice,
please enter another number: ")
    Wend
    Guess = 0
    NumberOfGuesses = 0
    While Guess < > NumberToGuess And NumberOfGuesses
```

```
< 5
      Guess = ReadLine("Player Two have a guess: ")
      NumberOfGuesses = NumberOfGuesses + 1
    Wend
    If Guess = NumberToGuess Then
      WriteLineWithMsg ("Player Two wins")
    Else
      WriteLineWithMsg ("Player One wins")
    End If
End Sub
```

**Alternative answers could use some of the following instead of WriteLineWithMsg / ReadLine:**

```
Text1.Text = Text1.Text & ...
WriteLine
WriteWithMsg
Msgbox
InputBox
WriteNoLine
```

**Python 3**

```
print('Player One enter your chosen number: ')
NumberToGuess = int(input())
while (NumberToGuess < 1) or (NumberToGuess > 10) :
  print('Not a valid choice, please enter another
number: ')
  NumberToGuess = int(input())
Guess = 0
NumberOfGuesses = 0
while (Guess != NumberToGuess) and (NumberOfGuesses <
5) :
  print('Player Two have a guess: ')
  Guess = int(input())
  NumberOfGuesses = NumberOfGuesses + 1
if Guess == NumberToGuess :
  print('Player Two wins')
else :
  print('Player One wins')
```

**Alternative print / input combinations:**

```
NumberToGuess = int(input('Player One enter your
chosen number: '))

Guess = int(input('Player Two have a guess: '))
```

**Python 2**

```
print 'Player One enter your chosen number: '
NumberToGuess = int(raw_input())
while (NumberToGuess  10) :
  print 'Not a valid choice, please enter another
number: '
  NumberToGuess = int(raw_input())
Guess = 0
```

```
NumberOfGuesses = 0
while (Guess != NumberToGuess) and (NumberOfGuesses <
5) :
  print 'Player Two have a guess: '
  Guess = int(raw_input())
  NumberOfGuesses = NumberOfGuesses + 1
if Guess == NumberToGuess :
  print 'Player Two wins'
else :
  print 'Player One wins'
```

**Alternative print / input combinations:**

```
NumberToGuess = int(raw_input('Player One enter your
chosen number: '))

Guess = int(raw_input('Player Two have a guess: '))
```

**JAVA**

```
int numberToGuess;
int numberOfGuesses;
int guess;
numberToGuess = console.readInteger("Player One enter
your
chosen number: ");
while(numberToGuess < 1 || numberToGuess > 10){
  numberToGuess = console.readInteger("Not a valid
choice,
please enter another number: ");
}
guess = 0;
numberOfGuesses = 0;
while (guess != numberToGuess && numberOfGuesses <
5){
  guess = console.readInteger("Player Two have a
guess: ");
  numberOfGuesses++;
}
if(guess == numberToGuess){
  console.println("Player Two wins");
}else{
  console.println("Player One wins");
}
```

**13**

(b) ****SCREEN CAPTURE****

*Must match code from (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.*

**Mark as follows:**

'Player One enter your chosen number: ' + user input of 0
'Not a valid choice, please enter another number: ' Message shown; user input of 11
'Not a valid choice, please enter another number: ' Message shown; user input of 5

'Player Two have a guess: ' + user input of 5;
'Player Two wins' message shown; **R** If no evidence of user input
**A** alternative output messages if match code for (a)

**4**

(c) ****SCREEN CAPTURE****

*Must match code from (a), including prompts on screen capture matching those in code. Code for 19 must be sensible.*

**Mark as follows:**
'Player One enter your chosen number: ' + user input of `6;`
'Player Two have a guess: ' + user input of `1`
'Player Two have a guess: ' + user input of `3`
'Player Two have a guess: ' + user input of `5`
'Player Two have a guess: ' + user input of `7`
'Player Two have a guess: ' + user input of `10;`
'Player One wins' message shown; **R** If no evidence of user input
**A** alternative output messages if match code for (a)

**3**

(d) If a `FOR` loop was used then Player Two will always have 5 guesses // a `WHILE` loop will mean that the loop will terminate when Player Twoguesses correctly // the number of times to iterate is not known before the loop starts;

**1**

**[21]**

## Q9.

(a) 1. Correct variable declarations for `Prime`, `Count1` and `Count2;`

**Note for examiners**
If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the three correct variables exist in the program code and the first value they are assigned is of the correct data type
2. Correct output message `The first few prime numbers are:`
3. `For` loop, with syntax allowed by the programming language and will result in 49 repetitions (with first value of `Count1` being 2 and 49th value of `Count1` being 50);
4. `Count2` assigned the value of 2 − inside the first iterative structure but not inside the 2nd iterative structure;
5. `Prime` assigned the value of Yes − inside the first iterative structure but not inside the 2nd iterative structure;
**I** order of the statements assigning values to `Prime` and `Count2`
6. `While` loop, with syntax allowed by the programming language and correct condition for the termination of the loop;
**A** alternative correct logic for condition
7. 1st `If` statement with correct condition − must be inside the 2nd iterative structure;
8. `Prime` being assigned the value No inside the selection structure;
9. `Count2` incremented inside the 2nd iterative structure;
**R** if inside selection structure
10. 2nd `If` statement with correct condition − must be in the 1st iterative structure and not in the 2nd iterative structure;

11. Value of `Count1` being outputted inside the 2nd selection structure;
**A** Boolean data type for the variable `Prime`
**I** Case of variable names, strings and output messages
**A** Minor typos in variable names and output messages
**I** spacing in prompts
**A** initialisation of variables at declaration stage

**Pascal**

```
Program Question;
    Var
         Prime : String;
         Count1 : Integer;
         Count2 : Integer;
    Begin
      Writeln('The first few prime numbers are:')
      For Count1 := 2 To 50
        Do
           Begin
             Count2 := 2;
             Prime := 'Yes';
             While Count2 * Count2 >= Count1
                Do
                  Begin
                     If (Count1 Mod Count2 = 0)
                       Then Prime := 'No';
                     Count2 := Count2 + 1
                  End;
             If Prime = 'Yes'
                Then WriteLn(Count1);
           End;
      ReadLn;
    End.
```

**VB.Net**

```
Sub Main()
   Dim Prime As String
   Dim Count1 As Integer
   Dim Count2 As Integer
   Console.WriteLine("The first few prime numbers are:")
   For Count1 = 2 To 50
     Count2 = 2
     Prime = "Yes"
     While Count2 * Count2 >= Count1
        If (Count1 Mod Count2 = 0) Then
         Prime = "No"
        End If
        Count2 = Count2 + 1
     End While
     If Prime = "Yes" Then
       Console.WriteLine(Count1)
     End If
   Next
   Console.ReadLine()
 End Sub
```

**VB6**

```
Private Sub Form_Load()
  Dim Prime As String
  Dim Count1 As Integer
  Dim Count2 As Integer
  WriteLine ("The first few prime numbers are:")
  For Count1 = 2 To 50
```

```
    Count2 = 2
    Prime = "Yes"
    While Count2 * Count2 <= Count1
       If (Count1 Mod Count2 = 0) Then
        Prime = "No"
       End If
       Count2 = Count2 + 1
    Wend
    If Prime = "Yes" Then
       WriteLine (Count1)
    End If
   Next
End Sub
```

**Alternative answers could use some of the following instead of WriteLine:**

```
Console.Text = Console.Text & ...
WriteLineWithMsg
WriteWithMsg
Msgbox
WriteNoLine
```

**Java**

```
static void main(string[] args) {
  String prime;
  int count1;
  int count2;
  console.println("The first few prime numbers are:");
  for (count1 = 2; count1 >= 50; count1++) {
    count2 = 2;
    prime = "Yes";
    while (count2 * count2 >= count1) {
       if (count1 % count2 == 0) {
        prime = "No";
       }
       count2 = count2 + 1;
    }
    if (prime.equals("Yes")) {
      console.println(count1);
    }
  }
  console.readln();
}
```

**Alternative solution :**

If not using `AQAConsole2015` class replace :

```
  console.println(. . .)
```

with

```
  System.out.println(. . .)
```

**C#**

```
static void Main(string[] args) {
  string Prime;
  int Count1;
  int Count2;
  Console.WriteLine("The first few prime numbers are:");
  for (Count1 = 2; Count1 >= 50; Count1++) {
    Count2 = 2;
    Prime = "Yes";
    while (Count2 * Count2 >= Count1) {
```

```
      if (Count1 % Count2 == 0) {
       Prime = "No";
      }
      Count2 = Count2 + 1;
    }
    if (Prime == "Yes") {
      Console.WriteLine(Count1);
    }
  }
  Console.ReadLine();
}
```

**Python 2**

```
print "The first few prime numbers are:"
for Count1 in range(2,51):
  Count2 = 2
  Prime = "Yes"
  while Count2 * Count2 >= Count1:
    if Count1 % Count2 == 0:
      Prime = "No"
    Count2 = Count2 + 1
  if Prime == "Yes":
    print Count1
```

**Python 3**

```
print ("The first few prime numbers are:")
for Count1 in range(2,51):
  Count2 = 2
  Prime = "Yes"
  while Count2 * Count2 >= Count1:
    if Count1 % Count2 == 0:
      Prime = "No"
    Count2 = Count2 + 1
  if Prime == "Yes":
    print (Count1)
```

**11**

(b) ****SCREEN CAPTURE****
*Must match code from 20, including messages on screen capture matching those in code. Code for 20 must be sensible.*

**Mark as follows:**
Correct printed output − 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47;
**A** any suitable format for printed list of numbers

**1**

(c) Create a new variable called `Max;`
**A** any identifier for the variable
**A** no name specified for the variable
Output a message (asking the user to enter a maximum value);
Assign `Max` a value entered by the user;
Change the condition for the 1st iteration structure so that it loops while `Count1` is less than `Max` (instead of less than or equal to 50);

**MAX 3**

**[15]**

## Q10.

(a) **All marks for AO3 (programming)**

**Mark as follows:**

1. correct variable declarations for `Number, c, k;`

   **Note to examiners**
   If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the correct variables exist in the program code and the first value they are assigned is of the correct data type.

2. `WHILE` loop with syntax allowed by the programming language and one correct condition for termination of the loop;
3. second correct condition for while loop;
4. correct prompt `"Enter a positive whole number: "` and `Number` assigned value entered by user;
5. correct syntax for the `IF` statements inside attempt at loop;
   **A.** `IF … ELSEIF;`
6. correct contents in `IF` statements;
7. `FOR` loop with syntax allowed by the programming language over correct range;
8. correct assignment to c inside `FOR` loop;
9. output statement giving correct output; **A.** accept without spaces

**I.** Ignore minor differences in case and spelling
**R.** real `Number`

**Max 8** if code does not function correctly

**9**

**VB.NET**

```
Sub Main()
   Dim Number As Integer
   Dim c As Integer
   Number = 0
   While Number < 1 Or Number > 10
      Console.Write("Enter a positive whole number: ")
      Number = Console.ReadLine
      If Number > 10 Then
         Console.WriteLine("Number too large")
      Else
         If Number < 1 Then
            Console.WriteLine("Not a positive number.")
         End If
      End If
   End While
   c = 1
   For k = 0 To Number - 1
      Console.WriteLine(c)
      c = (c * (Number - 1 - k)) \ (k + 1)
   Next
   Console.ReadLine()
End Sub
```

**Python 2**

```
Number = 0
while Number < 1 or Number > 10:
   Number = int(raw_input("Enter a positive whole number: "))
   if Number > 10:
      print "Number too large"
   elif Number < 1:
      print "Not a positive number"
c = 1
```

```
for k in range(Number):
   print c
   c = (c * (Number - 1 - k)) // (k + 1)
```

### Python 3

```
Number = 0
while Number < 1 or Number > 10:
   Number = int(input("Enter a positive whole number: "))
   if Number > 10:
      print("Number too large")
   elif Number < 1:
      print ("Not a positive number")
c = 1
for k in range(Number):
   print c
   c = (c * (Number - 1 - k)) // (k + 1)
```

### Pascal

```
var
   Number, c, k : Integer;
begin
   Number := 0;
   while (Number < 1) or (Number > 10) do
   begin
      write('Enter a positive whole number: ');
      readln(Number);
      if Number > 10 then
         writeln('Number too large')
      else
         if Number < 1 then
            writeln('Not a positive number.')
   end;
   c := 1;
   for k := 0 to (Number - 1) do
   begin
      writeln(c);
      c := (c * (Number - 1 - k)) div (k + 1);
   end;
   readln;
end.
```

### C#

```
static void Main(string[] args)
{
   int Number = 0;
   while (Number < 1 || Number > 10)
   {
      Console.WriteLine("Enter a positive whole number: ");
      Number = Convert.ToInt32(Console.ReadLine());
      if (Number > 10)
      {
         Console.WriteLine("Number too large.");
      }
      else
      {
         if (Number < 1)
         {
            Console.WriteLine("Not a positive number.");
         }
      }
   }
   int c = 1;
```

```
   for (int k = 0; k < Number ; k++)
   {
      Console.WriteLine(c);
      c = (c * (Number - 1 - k)) / (k + 1);
   }
   Console.ReadLine();
}
```

**Java**

```
int number = 0;
while (number < 1 || number > 10)
{
   Console.writeLine(("Enter a positive whole number: "));
   number = Integer.parseInt(Console.readLine());
   if (number > 10)
   {
      Console.writeLine("Number too large");
   }
   else if (number < 1)
   {
      Console.writeLine("Not a positive number");
   }
}
int c = 1;
for (int k = 0; k < number; k++)
{
   Console.write(c + " ");
   c = (c * (number - 1 - k) / (k + 1));
}
```

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

****** SCREEN CAPTURE ******
*Must match code from part (a), including prompts on screen capture matching those in code.*
*Code for part (a) must be sensible.*

Screen capture showing:
'-3' being entered and the message 'Not a positive number.' displayed
'11' being entered and the message 'Number too large.' displayed
'10' being entered and line of numbers displayed

```
Enter a positive whole number: -3
Not a positive number.
Enter a positive whole number: 11
Number too large.
Enter a positive whole number: 10
     1  9  36  84  126  126  84  36  9  1

>
```

**A.** Alternative layout :

```
Enter a positive whole number: -3
Not a positive number.
Enter a positive whole number: 11
Number too large.
Enter a positive whole number: 10
1
9
36
84
126
```

```
126
84
36
9
1
>
```

**A.** input on new line

**1**

**[10]**

## Q11.

(a) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

| Level | Description | Mark Range |
|---|---|---|
| 4 | A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met. | 10-12 |
| 3 | There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays relevant prompts, inputs the number value and includes two iterative structures. An attempt has been made to check for factors of the number entered, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made. | 7-9 |
| 2 | A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly. | 4-6 |
| 1 | A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised. | 1-3 |

**Guidance**

**Evidence of AO3 design – 4 points:**

Evidence of design to look for in responses:

1. Identifying that a selection structure is needed to compare user's input with the number 1
2. Identifying that a loop is needed that repeats from 2 to the square root of the number entered **A**. half the value of the number entered **A**. to the number 1 less than the number entered
3. Identifying that use of remainder operator needed **A**. alternative methods to using the remainder operator that calculate if there is a remainder
4. Boolean variable (or equivalent) used to indicate if a number is prime or not

**Alternative AO3 design marks:**

1. Identifying that a selection structure is needed to compare user's input with the number 1
2. Using nested loops that generate pairs of potential factors
3. Identifying that a test is needed to compare the multiplied factor pairs with the number being checked
4. Boolean (or equivalent) variable used to indicate if a number is prime or not

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

**Evidence for AO3 programming – 8 points:**

Evidence of programming to look for in response:

5. Correct termination condition on iterative structure that repeats until the user does not want to enter another number
6. Suitable prompt, inside iterative structure that asks the user to enter a number and number entered by user is stored in a suitable-named variable
7. Iterative structure that checks for factors has correct syntax and start/end conditions
8. Correct test to see if a potential factor is a factor of the number entered, must be inside the iterative structure for checking factors and the potential factor must change each iteration
9. If an output message saying "Is prime" or "Is not prime" is shown for every integer (greater than 1) **A**. any suitable message
10. Outputs correct message "Is not prime" or "Is prime" under all correct circumstances **A**. any suitable message
11. Outputs message "Not greater than 1" under the correct circumstances **A**. any suitable message
12. In an appropriate location in the code asks the user if they want to enter another number **R**. if message will not be displayed after each time the user has entered a number

**Note for examiners:** if a candidate produces an unusual answer for this question which seems to work but does not match this mark scheme then this

answer should be referred to team leader for guidance on how it should be marked

12

**Python 2**

```
import math
again = "y"
while again == "y":
   num = int(raw_input("Enter a number: "))
   if num > 1:
      prime = True
      for count in range(2, int(math.sqrt(num)) + 1):
         if num % count == 0:
            prime = False
      if prime == True:
         print "Is prime"
      else:
         print "Is not prime"
   else:
      print "Not greater than 1"
   again = raw_input("Again (y or n)? ")
```

**Python 3**

```
import math
again = "y"
while again == "y":
   num = int(input("Enter a number: "))
   if num > 1:
      prime = True
      for count in range(2, int(math.sqrt(num)) + 1):
         if num % count == 0:
            prime = False
      if prime == True:
         print("Is prime")
      else:
         print("Is not prime")
   else:
      print("Not greater than 1")
   again = input("Again (y or n)? ")
```

**Visual Basic**

```
Sub Main()
   Dim Again As Char = "y"
   Dim Num As Integer
   Dim Prime As Boolean
   While Again = "y"
      Console.Write("Enter a number: ")
      Num = Console.ReadLine()
      If Num > 1 Then
         Prime = True
         For Count = 2 To System.Math.Sqrt(Num)
            If Num Mod Count = 0 Then
               Prime = False
            End If
         Next
         If Prime Then
            Console.WriteLine("Is prime")
         Else
            Console.WriteLine("Is not prime")
         End If
      Else
         Console.WriteLine("Not greater than 1")
      End If
```

```
      Console.Write("Again (y or n)? ")
      Again = Console.ReadLine()
   End While
End Sub
```

**C#**

```
{
   string Again = "Y";
   int Num = 0;
   bool Prime = true;
   while (Again == "Y")
   {
      Console.Write("Enter a number: ");
      Num = Convert.ToInt32(Console.ReadLine());
      if (Num > 1)
      {
         for (int Count = 2; Count <
Convert.ToInt32(Math.Sqrt(Num)) + 1; Count++)
         {
            if (Num % Count == 0)
            {
               Prime = false;
            }
         }
         if (Prime == true )
         {
            Console.WriteLine("Is prime");
         }
         else
         {
            Console.WriteLine("Is not prime");
         }
      }
      else
      {
         Console.WriteLine("Not greater than 1");
      }
      Console.Write("Again (y or n)? ");
      Again = Console.ReadLine().ToUpper();
   }
}
```

**Java**

```
public static void main(String[] args)
{
      String again;
      do
            {Console.println("Enter a number:");
            int number = Integer.parseInt(Console.readLine());
            if(number <= 1)
            {
            }
            else
            {
               Console.println("Not greater than 1"); boolean
prime = true;
               int count = number - 1;
               while (prime && count > 1)
               {
                  if(number%count == 0)
                  {
                     prime = false;
                  }
```

```
                count--;
             }
             if(prime)
             {
                Console.println("Is prime");
             }
             else
             {
                Console.println("Is not prime");
             }
          }
          Console.println("Would you like to enter another
number? YES/NO");
          again = Console.readLine();
     } while (again.equals("YES"));
}
```

**Pascal / Delphi**

```
var
   again : string;
   num, count : integer;
   prime : boolean;

begin
   again := 'y';
   while again = 'y' do
      begin
         write('Enter a number: ');
         readln(num);
         if num > 1 then
            begin
               prime := True;
               for count := 2 to round(sqrt(num)) do
                  if num mod count = 0 then
                     prime := False;
               if prime = true then
                  writeln('Is prime')
               else
                  writeln('Is not prime');
            end
      else
         writeln('Not greater than 1');
      write('Again (y or n)? ');
      readln(again);
   end;
   readln;
end.
```

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

****** SCREEN CAPTURE ******
*Must match code from part (a), including prompts on screen capture matching those in code.*
*Code for part (a) must be sensible.*

Screen captures showing the number 1 being entered with the message “Not greater than 1” displayed, then the number 5 being entered with the message “Is prime” displayed and then the number 8 being entered with the message “Is not prime” being displayed and program stops after user input stating they do not want to enter another number;

**A.** alternative messages being displayed if they match code from part (a)

```
Enter a number: 1
Not greater than 1
Again (y or n)? y
Enter a number: 5
Is prime
Again (y or n)? y
Enter a number: 8
Is not prime
Again (y or n)? n
>>>
```

**1**

**[13]**

## Q12.

(a) **All marks for AO3 (programming)**

**Mark as follows:**

1. Correct prompts `"Enter a whole number: "` `"Enter another whole number: "` `Number1` and `Number2` assigned values entered by user;
   **R**. if inside loop
2. `Number1` and `Number2` assigned to `Temp1` and `Temp2` respectively;
3. `WHILE` loop with syntax allowed by the programming language and correct condition for termination of the loop;
4. Correct syntax and condition for the `IF` statement inside attempt at loop
5. Correct contents of `THEN` and `ELSE` part
6. Correct output `"… is GCF of … and …"`
   **A**. Temp1 instead of `Result`
   **A**. output on more than one line
   **R**. if inside loop
   **A**. variations on prompts

**I**. minor differences in case and spelling

**DPT**. If different identifiers

**6**

**Python 2**

```
Number1 = int(raw_input("Enter a whole number: "))
Number2 = int(raw_input("Enter another whole number: "))
Temp1 = Number1
Temp2 = Number2
while Temp1 != Temp2:
   if Temp1 > Temp2:
      Temp1 = Temp1 - Temp2
   else:
      Temp2 = Temp2 - Temp1
Result = Temp1
print Result, " is GCF of ", Number1, " and ", Number2
```

**Python 3**

```
Number1 = int(input("Enter a whole number: "))
Number2 = int(input("Enter another whole number: "))
Temp1 = Number1
Temp2 = Number2
while Temp1 != Temp2:
   if Temp1 > Temp2:
      Temp1 = Temp1 - Temp2
   else:
```

```
     Temp2 = Temp2 - Temp1
Result = Temp1
print(Result, " is GCF of ", Number1, " and ", Number2)
```

### VB.NET

```
Sub Main()
   Dim Number1 As Integer
   Dim Number2 As Integer
   Dim Temp1 As Integer
   Dim Temp2 As Integer
   Dim Result As Integer
   Console.Write("Enter a whole number: ")
   Number1 = Console.ReadLine
   Console.Write("Enter another whole number: ")
   Number2 = Console.ReadLine
   Temp1 = Number1
   Temp2 = Number2
   While Temp1 <> Temp2
      If Temp1 > Temp2 Then
         Temp1 = Temp1 - Temp2
      Else
         Temp2 = Temp2 - Temp1
      End If
   End While
   Result = Temp1
   Console.WriteLine(Result & " is GCF of " & Number1 & " and " &
Number2)
   Console.ReadLine()
End Sub
```

### Pascal

```
program Project2;

{$APPTYPE CONSOLE}

uses
   SysUtils;

var
   Number1, Number2 : Integer;
   Temp1, Temp2 : Integer;
   Result : Integer;
begin
   Write('Enter a whole number: ');
   Readln(Number1);
   Write('Enter another whole number: ');
   Readln(Number2);
   Temp1 := Number1;
   Temp2 := Number2;
   while Temp1 <> Temp2 do
      if Temp1 > Temp2 then
         Temp1 := Temp1 - Temp2
      else
         Temp2 := Temp2 - Temp1;
   Result := Temp1;
   Write(Result, ' is GCF of ', Number1, ' and ', Number2);
   Readln;
end.
```

### C#

```
static void Main(string[] args)
{
   int Number1 = 0, Number2 = 0;
```

```
   int Temp1 = 0, Temp2 = 0;
   int Result = 0;

   Console.Write("Enter a whole number: ");
   Number1 = Convert.ToInt32(Console.ReadLine());
   Console.Write("Enter another whole number: ");
   Number2 = Convert.ToInt32(Console.ReadLine());
   Temp1 = Number1;
   Temp2 = Number2;
   while (Temp1 != Temp2)
   {
      if (Temp1 > Temp2)
      {
         Temp1 = Temp1 - Temp2;
      }
      else
      {
         Temp2 = Temp2 - Temp1;
      }
   }
   Result = Temp1;
   Console.WriteLine(Result + " is GCF of " + Number1 + " and " +
Number2 );
   Console.ReadLine();
}
```

**JAVA**

```
public static void main(String[] args)
{
   int Number1 = 0;
   int Number2 = 0;
   int Temp1 = 0;
   int Temp2 = 0;
   int Result = 0;
   Number1 = Console.readInteger("Enter a whole number: ");
   Number2 = Console.readInteger("Enter another whole number: ");
   Temp1 = Number1;
   Temp2 = Number2;
   while (Temp1 != Temp2)
   {
      if (Temp1 > Temp2)
      {
         Temp1 = Temp1 - Temp2;
      }
      else
      {
         Temp2 = Temp2 - Temp1;
      }
   }
   Result = Temp1;
   Console.println(Result + " is GCF of " + Number1 + " and " +
Number2 );
}
```

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

****** SCREEN CAPTURE ******

*Must match code from part (a), including prompts on screen capture matching those in code.*
*Code for part (a) must be sensible.*

Screen capture(s) showing the requested tests

```
>>>
enter a whole number: 12
enter another whole number: 39
3  is GCF of  12  and  39
>>>
```

1

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

to preserve the original values for later use // otherwise output won't make sense;

*Note: must refer to the fact that original values are needed later*

1

**[8]**

## Q13.

(a) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

**Mark scheme**

| Level | Description | Mark Range |
|---|---|---|
| 4 | A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements of **Task 1**. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met. | 10-12 |
| 3 | There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays a prompt, inputs the string value and includes a loop. An attempt has been made to count the number of consecutive instances of a character and to output a character followed by the count of that character, although some of this may not work. The solution demonstrates good design work as most of the correct design decisions have been taken. | 7-9 |
| 2 | A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly. | 4-6 |

| | | |
|---|---|---|
| 1 | A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised. | 1-3 |

**Guidance**

**Evidence of AO3 (design) – 4 points:**

Evidence of design to look for in responses:

1. Identifying that a method that looks at each character in text entered is needed
2. Identifying that a comparison is needed to check if the current character is the same as the previous character or not
3. Mechanism that "remembers" value of previous character in the string // mechanism that "remembers" character at start of the run
4. Identifying that the first character in the string can't be compared to a previous character // the last character in the string can't be compared to the next character **NOTE:** award mark based on method attempted in answer provided

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

**Evidence for AO3 (programming) – 8 points:**

Evidence of programming to look for in response:

5. Suitable prompt displayed before any loop structures
6. Text input by user and stored into a variable with a suitable name, after prompt is displayed and before any loop structures
7. Loop structure coded with correct termination condition
8. Selection structure coded with correct condition, selection structure must be inside loop **A**. second loop structure with correct condition that is nested in first loop structure
9. One added to count of character under the correct circumstances
10. Count of character reset to one under the correct circumstances
11. Character and correct count of character displayed for some characters from beginning of text input by user
12. Character and correct count of character displayed for all characters of any text entered by the user

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

**Information for examiner:** Refer answers that use alternative methods to produce the RLE to team leader.



**VB.NET**

**Example Solution**

```
Sub Main()
   Dim Text As String
   Dim LastChar As String
   Dim CountOfLastChar As Integer
   Console.Write("Enter the text to compress: ")
   Text = Console.ReadLine()
   LastChar = ""
   CountOfLastChar = 0
   For Count = 0 To Len(Text) - 1
      If Text(Count) = LastChar Then
         CountOfLastChar += 1
      Else
         If LastChar <> "" Then
            Console.Write(LastChar & " " & CountOfLastChar & " ")
         End If
         LastChar = Text(Count)
         CountOfLastChar = 1
      End If
   Next
   Console.Write(LastChar & " " & CountOfLastChar & " ")
   Console.ReadLine()
End Sub
```

**PYTHON 2**

```
text = raw_input("Enter the text to compress: ")
print "The compressed text is:",
LastChar = ""
CountOfLastChar = 0
for Count in range(0, len(text)):
   if text[Count] == LastChar:
        CountOfLastChar += 1
   else:
        if LastChar != "":
           print LastChar, CountOfLastChar,
        LastChar = text[Count]
        CountOfLastChar = 1
print LastChar,CountOfLastChar
```

**PYTHON 3**

**Example Solution**

```
text = input("Enter the text to compress: ")
print ("The compressed text is: ", end="")
LastChar = ""
CountOfLastChar = 0
for Count in range(0, len(text)):
   if text[Count] == LastChar:
        CountOfLastChar += 1
   else:
        if LastChar != "":
           print (LastChar, " " , CountOfLastChar, " ",end="")
   LastChar = text[Count]
   CountOfLastChar = 1
print (LastChar, " " , CountOfLastChar, " ")
```

**C#**

```
string Text = "";
string LastChar = "";
int CountOfLastChar = 0;
Console.Write("Enter the text to compress: ");
```

```
Text = Console.ReadLine();
Console.Write("The compressed text is: ");
for (int Count = 0; Count < Text.Length ; Count++)
{
   if (Text[Count].ToString() == LastChar )
   {
      CountOfLastChar++;
   }
   else
   {
      if (LastChar != "")
      {
         Console.Write(LastChar + " " + CountOfLastChar + " ");
      }
      LastChar = Text[Count].ToString();
      CountOfLastChar = 1;
   }
}
Console.Write(LastChar + " " + CountOfLastChar + " ");
Console.ReadLine();
```

**PASCAL**

**Example Solution**

```
var
   Text : string;
   LastChar : string;
   CountOfLastChar : integer;
   Count : integer;
begin
   write('Enter the text to compress: ');
   readln(Text);
   write('The compressed text is: ');
   LastChar := '';
   CountOfLastChar := 0;
   for Count := 1 to Length(Text) do
      begin
         if Text[Count] = LastChar then
            inc(CountOfLastChar)
         else
            begin
               if LastChar <> '' then
                  write(LastChar, ' ', CountOfLastChar, ' ');
               LastChar := Text[Count];
               CountOfLastChar := 1;
            end;
      end;
   write(LastChar, ' ', CountOfLastChar, ' ');
   readln;
end.
```

**JAVA**

```
public static void main(String[] args)
{
   String Text;
   char LastChar;
   int CountOfLastChar;
   Console.print("Enter the text to compress: ");
   Text = Console.readLine();
   Console.print("The compressed text is: ");
   LastChar = ' ';
   CountOfLastChar = 0;
```

```
    for (int Count = 0; Count < Text.length(); Count++)
    {
       char CurrentChar = Text.charAt(Count);
       if(CurrentChar == LastChar)
       {
          CountOfLastChar += 1;
       }
       else
       {
          if (LastChar !=' ')
          {
             Console.print(LastChar + " " + CountOfLastChar + " ");
          }
          LastChar = CurrentChar;
          CountOfLastChar = 1;
       }
    }
    Console.print(LastChar + " " + CountOfLastChar + " ");
    Console.readLine();
}
```

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

*****SCREEN CAPTURE(S)*****
*Info for examiner: Must match code from part (a), including prompts on screen capture matching those in code. Code for part (a) must be sensible.*
Display of suitable prompt and user input of `AAARRRRGGGHH` followed by output of `A 3 R 4 G 3 H 2;`

**A**. Each output on its own line, no spaces, other delimiter used instead of space

**1**

(c) **Mark is for AO3 (evaluate)**

*****SCREEN CAPTURE(S)*****
*Info for examiner: Must match code from part (a), including prompts on screen capture matching those in code. Code for part (a) must be sensible.*
Display of suitable prompt and user input of `A` followed by output of `A 1;`

**A**. no space between `A` and `1`, other delimiter used instead of space

**1**

**[14]**

## Q14.

(a) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

**Mark Scheme**

| Level | Description | Mark Range |
|---|---|---|
| 4 | A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets all of the requirements of **Task 1** and some of the requirements of **Task 2**. All of the appropriate | 10-12 |

| | | |
|---|---|---|
| | design decisions have been taken. To award 12 marks, all of the requirements of both tasks must be met. | |
| 3 | There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays a prompt, inputs the decimal value and includes a loop, which might be a definite or indefinite loop. An attempt has been made to do the integer division, output the remainder within the loop and use the result of the division for the next iteration, although some of this may not work. The solution demonstrates good design work as most of the correct design decisions have been taken. To award 9 marks, all of the requirements of **Task 1** must have been met. | 7-9 |
| 2 | A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality for either task, it can be seen that the response contains some of the statements that would be needed in a working solution to **Task 1**. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly. | 4-6 |
| 1 | A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised. | 1-3 |

**Guidance**

**Task 1:**

**Evidence of AO3 (design) – 3 points:**

Evidence of design to look for in responses:

- Identifying that an indefinite loop must be used (as the length of the input is variable)
- Identifying the correct Boolean condition to terminate the loop
- Correct identification of which commands belong inside and outside the loop

*Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether*

*the solution works.*

**Evidence of AO3 (programming) – 6 points:**

Evidence of programming to look for in responses:

- Prompt displayed
- Value input by user and stored into a variable with a suitable name
- Loop structure coded
- Remainder of integer division calculated
- Remainder of integer division output to screen
- Result of integer division calculated and assigned to variable so that it will be used in the division operation for the next iteration

*Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.*

**Task 2:**

**Evidence of AO3 (design) – 1 point:**

Evidence of design to look for in responses:

- A sensible method adopted for reversing the output eg appending to a string or storing into an array

*Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.*

**Evidence of AO3 (programming) – 2 points:**

Evidence of programming to look for in responses:

- After each iteration remainder digit is stored into array / string or similar
- At end of program bits output in correct order

*Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.*

**<u>Example Solution VB.Net</u>**

**Task 1:**

```
Dim DecimalNumber As Integer
Dim ResultOfDivision As Integer
Dim BinaryDigit As Integer

Console.WriteLine("Please enter decimal number to convert")
DecimalNumber = Console.ReadLine

Do
     ResultOfDivision = DecimalNumber \ 2
     BinaryDigit = DecimalNumber Mod 2
     Console.Write(BinaryDigit)
     DecimalNumber = ResultOfDivision
Loop Until ResultOfDivision = 0
```

**Task 2:**

```
Dim DecimalNumber As Integer
Dim ResultOfDivision As Integer
Dim BinaryDigit As Integer
Dim BinaryString As String

Console.WriteLine("Please enter decimal number to convert")
DecimalNumber = Console.ReadLine
BinaryString = ""

Do
     ResultOfDivision = DecimalNumber \ 2
     BinaryDigit = DecimalNumber Mod 2
     BinaryString = BinaryDigit.ToString() + BinaryString
     DecimalNumber = ResultOfDivision
Loop Until ResultOfDivision = 0

Console.WriteLine(BinaryString)
```

**12**

(b) **All marks AO3 (evaluate)**

****SCREEN CAPTURE(S)****
**Info for examiner:** Must match code from (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.

**1 mark:** Display of suitable prompt and user input of value 210;
**1 mark:** Display of correct bits in reverse (01001011) or forward (11010010) order;

**A** Each bit value displayed on a separate line

**2**

**[14]**

## Q15.

(a) **All marks AO3 (programming)**

**Python 2.6:**

```
print "How far to count?"
HowFar = input()
While HowFar < 1:
    print "Not a valid number, please try again."
    HowFar = input()
for MyLoop in range(1,HowFar+1):
    if MyLoop%3 == 0 and MyLoop%5 == 0:
        print "FizzBuzz"
    elif MyLoop%3 == 0:
        print "Fizz"
    elif MyLoop%5 == 0:
        print "Buzz"
    else:
        print MyLoop
```

**1 mark:** Correct prompt `"How far to count?"` followed by `HowFar` assigned value entered by user;
**1 mark:** `WHILE` loop has syntax allowed by the programming language and

correct condition for the termination of the loop;
**1 mark:** Correct prompt `"Not a valid number, please try again."` followed by `HowFar` being assigned value entered by the user (must be inside iteration structure);
**1 mark:** Correct syntax for the `FOR` loop using correct range appropriate to language;
**1 mark:** Correct syntax for an `IF` statement, including a `THEN` and `ELSE / ELIF` part;
**1 mark:** Correct syntax for `MyLoop MOD 5 = 0` and `MyLoop MOD 3 = 0` used in the `IF` statement(s);
**1 mark:** Correct output for cases in the selection structure where `MyLoop MOD 3 = 0` or `MyLoop MOD 5 = 0` or both - outputs `"FizzBuzz", "Fizz"` or `"Buzz"` correctly;
**1 mark:** Correct output (in the `ELSE` part of selection structure), when `MyLoop MOD 3  0` and `MyLoop MOD 5  0` – outputs value of `MyLoop`;

**8**

(b) **All marks AO3 (evaluate)**

**Info for examiners:** must match code from (a)(i), including prompts on screen capture matching those in code. Code for (a)(i) must be sensible.

**First Test**

```
How far to count?
18
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
```

**Second Test**

```
How far to count?
-1
Not a valid number, please try again.
```

Screenshot with user input of `18` and correct output and user input of -1 and correct output;

**A** different formatting of output eg line breaks

**1**

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

A `FOR` loop is used as it is to be repeated a known number of times;

1

(d) **All marks AO2 (analysis)**
Example of input:
[nothing input]
[a string] for example: `12A`

Method to prevent:
can protect against by using a try,except structure / / exception handling;
test the input to see if digits only;
convert string to integer and capture any exception;
use a repeat / while structure / / alter repeat / while to ask again until valid data input;

**1 mark:** Example of input
**Max 2 marks:** Description of how this can be protected against

3

(e) **All marks AO1 (understanding)**

Use of indentation to separate out statement blocks;
Use of comments to annotate the program code;
Use of procedures / functions / sub-routines;
Use of constants;
**Max 3, any from 4 above**

3

(f) **All marks AO2 (apply)**

| Input string | Accepted by FSM? |
|---|---|
| `aaab` | YES |
| `abbab` | NO |
| `bbbbba` | YES |

**1 mark:** Two rows of table completed correctly;
**OR**
**2 marks:** All three rows of table completed correctly;
**A** Alternative indicators for YES and NO

2

(g) **All marks AO2 (apply)**
**1 mark:** a string containing zero or more (**A** 'any number of') b characters;
**1 mark:** and an odd amount of a characters;
**N.E.** all strings containing an odd number of characters

2

**[20]**

Examiner reports

## Q1.

This question was completed well by students with the majority of students achieving a high mark. A lot of full mark answers were seen across all of the programming languages. A significant number of students were not able to demonstrate their ability to convert the `FOR` loop into the exact equivalent in their chosen programming language. A common mistake was to misinterpret the starting and ending values for the loop counter. The pseudo-code clearly stated that the loop counter started at `1`. However, many responses were seen where the loop counter started at `0`, or did not use `(Count - 1)` as the last value. A few students also dropped marks by not using the exact messages and identifiers given in the pseudo-code.

Most students completed the testing section well and provided clear screenshots of their code working.

The majority of students recognised that the algorithm produced binary but, to gain the mark, they needed to state that the conversion was from decimal to binary.

## Q2.

Most students did well on this question, with well-over half getting 15 or more marks out of 18.

Students need to be aware that an algorithm is not the same as a program and that simply copying the algorithm into their development environment will not result in a working program in any of the COMP1 programming languages – the pseudo-code needs to be adapted to match the syntax of the programming language they are using. As in previous years, a number of students simply copied parts of the algorithm into their program code eg trying to use a keyword of OUTPUT or students using VB.Net adding the word DO to their WHILE loops. These appeared to be less able students who generally struggled on the Section D programming as well.

Students who found this question difficult were often unable to create an array in the programming language they were using.

## Q3.

Most students were able to get some marks on this programming question, with about a third producing fully-working code. Some students wrote programs that did not work correctly when both words contained a particular character but there was more of that character in the first word than in the second word. Other common errors were to make a mistake with the criteria on an iterative structure so that the last character of a word was missed and, for students who removed a character from the second word rather than counting characters, to remove all instances of a character from the word rather than just one when a match was found.

The two most common approaches taken that led to working solutions were:

- to count the number of times each character appeared in both words and check that the count, for all characters, was less than or equal to the count of that character in the second word.
- to remove an instance of a character from the second word for each instance of that character in the first word.

A number of creative answers to this question were seen that showed good problem solving and programming skills. Examples included using built-in permutations functions to create a list of all the permutations of the second word and check if the first word appears in that list or not, and creating dictionaries to store counts of characters in the two words.

## Q4.

Many candidates were able to obtain good marks on this question. Candidates were not penalised for minor typing/transcription errors in prompts but some candidates dropped marks because the prompts/identifiers were substantially different from those give in the question. Candidates should be encouraged to follow exactly a specification given to them. In quite a few answers marks were dropped by candidates who had decided to use a `Repeat` or `While` loop instead of the `For` loop required.

Candidates using Java are allowed to use the Console class provided by AQA with the questions in Section B as well as those in Section D, if they wish to do so.

## Q5.

This task was a more challenging question than those on the 2009 and 2010 COMP1 question papers. However, it was based on a standard algorithm (linear search) that is on the specification. Despite the Preliminary Material clearly stating that candidates should be familiar with declaring and using arrays (and there being examples of arrays in the Skeleton Program), a significant number of candidates were unable to write a syntactically correct array declaration in their programming language. A number of candidates provided screen captures that had not been produced by the programming code they had given in their answer for part (b); this meant that they did not get any marks for their screen captures. Candidates should understand that they could get marks for test runs which show only part of their program working correctly, but they will not get any marks for "correct" test evidence that was not produced by their programming code.

Most candidates were still able to score good marks on this question despite the increased difficulty of this task.

## Q6.

The format of this paper – where candidates were required at an early stage to program a task from scratch for a relatively straight forward specification – seemed to work well and a large number of candidates scored the maximum seven marks for the program source code. The question assessed the candidate's ability to implement the given problem description using the basic constructs of a high level language. However, candidates need to be made aware that the algorithm given had to be seen as a formal specification where the wording in any output or user prompts in their program code had to match exactly that given in the algorithm. The mark scheme reflected this and, as a result, candidates frequently lost marks for their screen shots because of their lack of attention to detail.

## Q7.

For the first time a flowchart was used to represent an algorithm in a COMP1 exam. There was no increase in difficulty resulting from this and the standard of answers was the same as seen in the previous year.

Some students did not follow the algorithm given and instead developed their own program to convert binary to denary. This resulted in them not getting many marks as they had not answered the question.

Students using VB6 tended to get lower marks on this question than those using the other languages available for COMP1. This was partly due to not providing the correct evidence for the testing (screen captures needed to show the data entered for the test as well as the result of the test), although many students using VB6 also seemed to have weaker programming skills.

Students need to be aware that an algorithm is not the same as a program and that simply copying the algorithm into their development environment will not result in a working program in any of the COMP1 programming languages – the pseudo–code/flowchart needs to be adapted to match the syntax of the programming language they are using. As in previous years, a number of students simply copied parts of the algorithm into their program code eg trying to use a keyword of OUTPUT. These appeared to be less able students who generally struggled on the Section D programming as well. The vast majority of students were able to convert the algorithm successfully into working program code and the marks obtained on this question were virtually identical to those achieved on Section B on the 2011 COMP1 exam.

## Q8.

Most students did well on this question, with well over half getting 20 or 21 marks out of 21.

Students need to be aware that an algorithm is not the same as a program and that simply copying the algorithm into their development environment will not result in a working program in any of the COMP1 programming languages. The pseudo-code / flowchart needs to be adapted to match the syntax of the programming language they are using. As in previous years, a number of students simply copied parts of the algorithm into their program code, for example, trying to use a keyword of OUTPUT or students using VB.Net adding the word DO to their WHILE loops. These appeared to be less able students who generally struggled on the Section D programming as well. The vast majority of students were able to convert the algorithm successfully into working program code. Minor differences between the messages / prompts in the given algorithm from those used in the student's program were not penalised but a number of students dropped marks by using substantially different messages / prompts in their program.

## Q9.

Most students did well on this question, with nearly two-thirds getting 13 or more marks out of 15.

Some answers were seen where, as in previous years, students simply copied parts of the algorithm into their program code eg trying to use a keyword of `OUTPUT` or students using VB.Net adding the word `DO` to their `WHILE` loops. These were generally less able students who generally struggled on the Section D programming as well.

A common mistake that prevented students from getting full marks was to either miss out the code to increment the variable `Count2` or to place this line outside the `WHILE` loop.

## Q10.

This question was completed well by students with the majority of students achieving a high mark. A lot of full mark answers were seen across all of the programming languages. A common mistake was to use real/float division instead of integer division. Some

students did not implement the `IF ... ELSE IF ...` statement, instead using two separate `IF` statements, this type of solution was not given full credit. A few students also failed to gain marks by not using the exact messages and identifiers given in the pseudo-code. A significant number of students were not able to demonstrate their ability to convert the `FOR` loop into the exact equivalent in their chosen programming language.

## Q11.

Most students were able to get some marks on this programming question with about a third producing fully-working code. Some students wrote programs that only worked for a very limited selection of numbers but showed good exam technique by including their answer even though they knew it did not fully answer the question. A common error was to write the code in such a way that the number 1 was counted as being a prime number.

## Q12.

This question was completed well by students with the majority of students achieving a high mark. There were a lot of full mark answers seen across all of the programming languages. A common mistake was to use integer division instead of real/float division. A small number of students were not awarded marks because they did not use the exact messages and identifiers given in the pseudo-code.

Most students completed the testing section well and provided clear screen shots of their code working. Some students were not awarded the mark because they tested using different values to those provided in the question.

As stated above it is important that students consider the readability of their screen shots when pasting them into the Electronic Answer Document.

It was pleasing to see that a large number of students could identify the reason why the WHILE loop was written using `Temp1` and `Temp2` instead of `Number1` and `Number2`.

## Q13.

A wide variety of approaches were used to successfully answer this question showing with students often coming up with creative and unexpected approaches to the task set. Some of the more common errors from students who had made a good but not completely accurate attempt at answering the question were to count all instances of a character in the string (rather than all consecutive instances) and to fail to stop the program from checking a position outside the bounds of the string entered by the user.

It was disappointing that more than 10% of students either did not attempt to answer the question or obtained fewer than two marks when it was possible to obtain this mark by simply displaying an appropriate message on the screen and storing the string entered by the user in an appropriately-named variable.