**Q1.**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 =105 105 ÷ 2 =  52  52 ÷ 2 =  26 26 ÷ 2 = 13 13 ÷ 2 =  6 6 ÷ 2 =  3 3 ÷ 2 =  1 1 ÷ 2 =  0 | remainder 0 remainder 1 remainder 0 remainder 0 remainder 1 remainder 0 remainder 1 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)**

**Q2.**

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)**

**Q3.**

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)**