A level Computer Science

Induction Tasks

Summer 2023

Part 1 – Investigation

You must choose **one** of the three topics below and create a presentation that can be viewed by other students in the group to teach them about that topic.

- **Data Compression and Encryption** – this must cover the following:
    - Lossy and lossless data compression
    - Run length encoding vs dictionary-based methods of data compression
    - Caesar and Vernam cipher methods of data encryption
- **Programming Languages** – this must cover the following:
    - High level and low level languages
    - Compilation vs interpretation
    - Assembly language and embedded systems
- **Internal Components of a Computer** – this must cover the following:
    - Components of a processor and how they work together
    - Factors affecting the performance of a processor
    - Von Neumann vs Harvard architectures

Part 2 – Programming

Complete as many of the programs from the pseudo shown on the following 8 pages. Test each one using the data provided.

You must provide evidence of each program that you write:

1. Screen shot of the Python code that you have written
2. Screen shot of the program running using the data provided

The screen shots should be put into a document (Word or Google Doc) with a brief commentary explaining what each screen shot shows.

NB. Include evidence of partly made programs and explain which parts you were unable to complete – we can then look at these in the first lessons back in September.

The variable table, **Table 2**, and the Structured English algorithm describe a simplified version of the **Guess the Word/Phrase Game**.

Table 2

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.

June 2010

The variable table, **Table 2**, and the Structured English algorithm, **Figure 4**, 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.

Table 2

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

Figure 4

```
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.

June 2011

The variable table, **Table 4**, and the Structured English algorithm, **Figure 4**, 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.

Table 4

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

Figure 4

```
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'.

June 2012

The algorithm, represented as a flowchart in **Figure 4**, and the variable table, **Table 3**, describe the converting of a 4-bit binary value into denary.

Figure 4

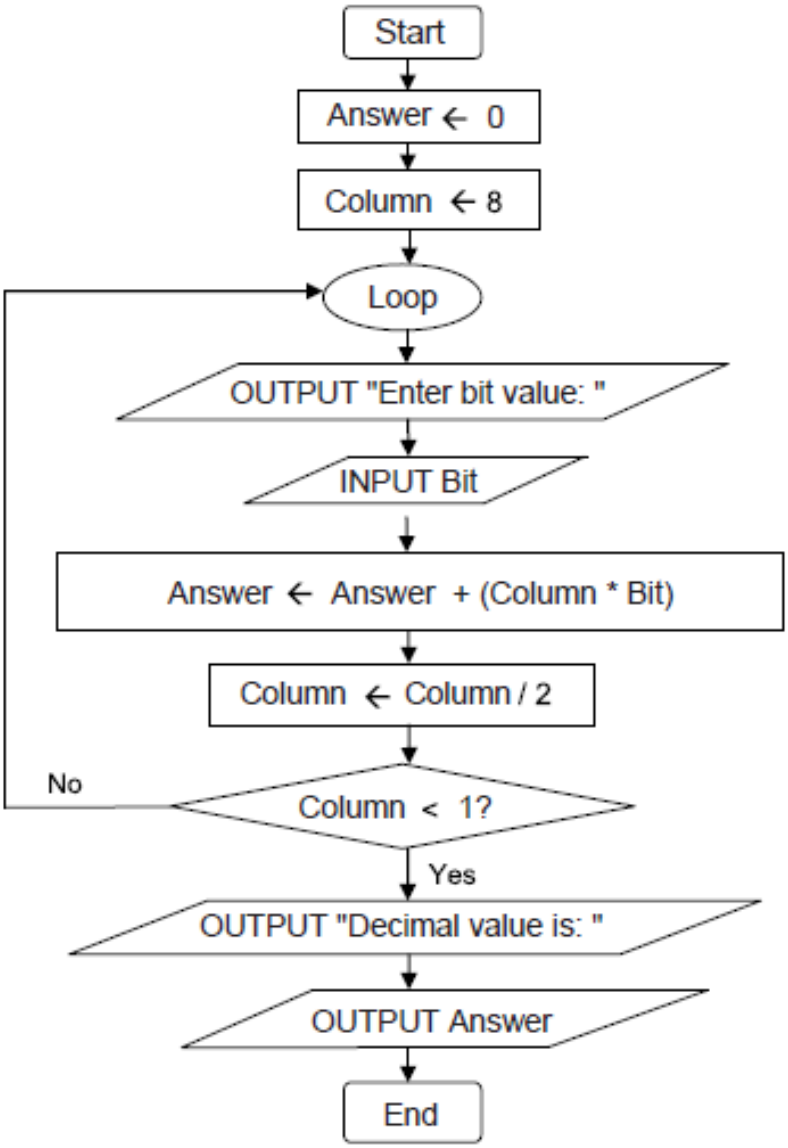

Table 3

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).

June 2013

The algorithm, represented using pseudo-code in **Figure 4**, and the variable table, **Table 3**, 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 4**, the symbol <> means "is not equal to".

Figure 4

```
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 3

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 **Question4** 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`

June 2014

The algorithm, represented using pseudo-code in **Figure 5**, and the variable table, **Table 3**, 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).

Figure 5

```
FOR Count ← 1 TO 13 DO
  OUTPUT "Please enter next digit of ISBN: "
  INPUT ISBN[Count]
ENDFOR
CalculatedDigit ← 0
Count ← 1
WHILE Count < 13 DO
  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
```

Table 3

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 in **Figure 5**.

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).

The algorithm, represented using pseudo-code in **Figure 4**, and the variable table, **Table 3**, 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**.

Figure 4

```
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
```

Table 3

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 in **Figure 4**.

Run the program and test that it works correctly.

June 2016

The algorithm, represented using pseudo-code in **Figure 4**, and the variable table, **Table 2**, describe a program that outputs an **estimate** for a particular calculation.

Figure 4

```
OUTPUT "Enter a number:"
INPUT N
F ← 16.0
IF N >= 1.0
   THEN
      X ← N
      WHILE X * X - N > 1.0 AND F - 1.0 > 1.0 DO
         L ← X
         X ← X ÷ F
         WHILE X * X <= N DO
            F ← F - 0.1
            X ← L ÷ F
         ENDWHILE
      ENDWHILE
      OUTPUT X
   ELSE
      OUTPUT "Not a number greater than or equal to 1"
ENDIF
```

Table 2

Identifier	Data type
X	Real number
F	Real number
L	Real number
N	Real number

What you need to do

Write a program for the algorithm in **Figure 4**.

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

Save the program in your new **Question5** folder/directory.

Test 1
Test that your program works correctly by entering the number 0.1

Test 2
Test that your program works correctly by entering the number 4.1