Ch15

Software Development

CHAPTER ASSIGNMENT · 20 MARKS

Name _____ Class _____ No. _____ Date _____

Answer every question in the space given. Show your working. You may use the chapter page to help you.

Questions from *Computer Science for Cambridge International AS & A Level Coursebook*, 2nd edition, Sylvia Langfield and Dave Duddell (Cambridge University Press), for classroom use.

Tasks and questions from the chapter

DP **Discussion Point:** Do you think that all programs can be totally error-free?

15.01 Draw a structure chart for the following module: Input a number of km, output the equivalent number of miles.

15.02 Amend the structure chart for the number-guessing game (Figure 15.04) to include repeated guesses until the player guesses the secret number. The output should include the number of guesses made.

15.03 Draw a structure chart for the following problem: A user attempts to log on with a user ID. User IDs and passwords are stored in two 1D arrays (lists). The algorithm searches the list of user IDs and looks up the password in the password list. The user is given three chances to input the correct password. If the correct password is entered, a suitable message is output. If the third attempt is incorrect, a warning message is output.

15.04

(1) Write pseudocode to implement the structure chart from Figure 12.03 (for the average of two numbers).

(2) Write pseudocode to implement the structure chart from Figure 12.04 (for the number-guessing game).

(3) Amend the pseudocode from Worked Example 13.05 to implement the interface shown in the structure chart from Figure 15.06.

DP **Discussion Point:** The full rules of Connect 4 are that a diagonal of four tokens also is a winning line. Where in Figure 15.06 should the module to check for a diagonal be added? What parameters are required for this module? Does this additional module require further stepwise refinement?

15.05 Write a program that simulates the intruder detection system in Worked Example 15.01.

Q15.01 What is the output from the FSM represented by the state-transition diagram in Figure 15.10, when the input is 0101?

EQ15.01 **Extension Question 15.01:** Does the FSM in Figure 15.10 work for converting a negative binary number into its positive equivalent?

15.06 Design a trace table for the following algorithm:

```
FUNCTION ConvertFromHex(HexString : STRING) RETURNS INTEGER
    DECLARE ValueSoFar, HexValue, HexLength, i : INTEGER
    DECLARE HexDigit : CHAR
    ValueSoFar <- 0
    HexLength <- Length(HexString)
    FOR i <- 1 TO HexLength
        HexDigit <- HexString[i]
        CASE OF HexDigit
            'A': HexValue <- 10
            'B': HexValue <- 11
            'C': HexValue <- 12
            'D': HexValue <- 13
            'E': HexValue <- 14
            'F': HexValue <- 15
           OTHERWISE HexValue <- StringToInt(HexDigit)
        ENDCASE
        ValueSoFar <- ValueSoFar * 16 + HexValue
    NEXT i
    RETURN ValueSoFar
ENDFUNCTION
```

Dry-run the function call `ConvertFromHex('A5')` by completing the trace table.

DP **Discussion Point:** Do you think that a program tester will find errors the programmer did not know about? You can try out the idea by letting your friends test a program that you think works perfectly.

15.07 Look at the programs you wrote in Chapter 14.

(1) Design test data for the number-guessing game (Task 14.09.2).

(2) Design test data for the Connect 4 game (Task 14.11).

15.08

(1) Design a trace table for the following algorithm:

```
INPUT BinaryString
StringLength <- Length(BinaryString)
FOR i <- 1 TO StringLength
    Bit <- BinaryString[i]
    BitValue <- IntegerValue(Bit) // convert string to integer
    DenaryValue <- DenaryValue + 2 + BitValue
NEXT i
```

(2) Dry-run the algorithm using '101' as the input. Complete the trace table.

(3) The result should be 5. Can you find the error in the code and correct it?

15.09 Design the algorithm to simulate the computer playing the part of Player X in Connect 4.

15.10 Analyse the pseudocode below and make amendments to enhance maintainability.

```
FUNCTION GetPositiveNumber
    DECLARE n : INTEGER
    OUTPUT "Enter a positive number: "
    INPUT n
    RETURN n
ENDFUNCTION
// main program
REPEAT
    Number1 <- GetPositiveNumber
    IF Number1 <= 0
        THEN
            OUTPUT "Not a positive number: "
    ENDIF
UNTIL Number1 > 0
REPEAT
    Number2 <- GetPositiveNumber
    IF Number2 <= 0
        THEN
            OUTPUT "Not a positive number: "
    ENDIF
UNTIL Number2 > 0
```

RP **Reflection Point:** Have you used dry-running for programs you have written? You can check your trace table if you add output statements at key points in your program. You can then compare the program output with the contents of your trace table.

Exam-style questions

1 Consider this code for a function:

```
FUNCTION Binary(Number : INTEGER) RETURNS STRING
    DECLARE BinaryString : STRING
    DECLARE PlaceValue : INTEGER
    BinaryString <- ''  // empty string
    PlaceValue <- 8
    REPEAT
        IF Number >= PlaceValue
            THEN
                BinaryString <- BinaryString & '1' // concatenates two strings
                Number <- Number - PlaceValue
            ELSE
                BinaryString <- BinaryString & '0'
        ENDIF
        PlaceValue <- PlaceValue DIV 2
    UNTIL Number = 0
    RETURN BinaryString
ENDFUNCTION
```

(a) Complete the trace table for the function call `Binary(11)`.

Number	BinaryString	PlaceValue	Number >= PlaceValue
11	''	8	

State the return value. [5]

(b) Complete the trace table for the function call `Binary(10)`.

(i)

Number	BinaryString	PlaceValue	Number >= PlaceValue
10	''	8	

State the return value. [3]

(ii) The algorithm is supposed to convert a denary integer into the equivalent binary number, stored as a string of 0s and 1s. Explain the result of each dry-run and what needs changing in the given algorithm. [3]

2 A procedure to output a row in a tally chart has been written using pseudocode:

```
PROCEDURE OutputTallyRow(NumberToDraw : INTEGER)
    IF Count > 0
        THEN
            FOR Count <- 1 TO NumberToDraw
                IF (Count MOD 5) = 0
                    THEN
                        OUTPUT('\') // every 5th bar slants the other way
                    ELSE
                        OUTPUT('/')
                ENDIF
            NEXT Count
    ENDIF
    OUTPUT NewLine // move to next row
ENDPROCEDURE
```

Suggest suitable test data that will test the procedure adequately. Justify your choices in each case. [9]

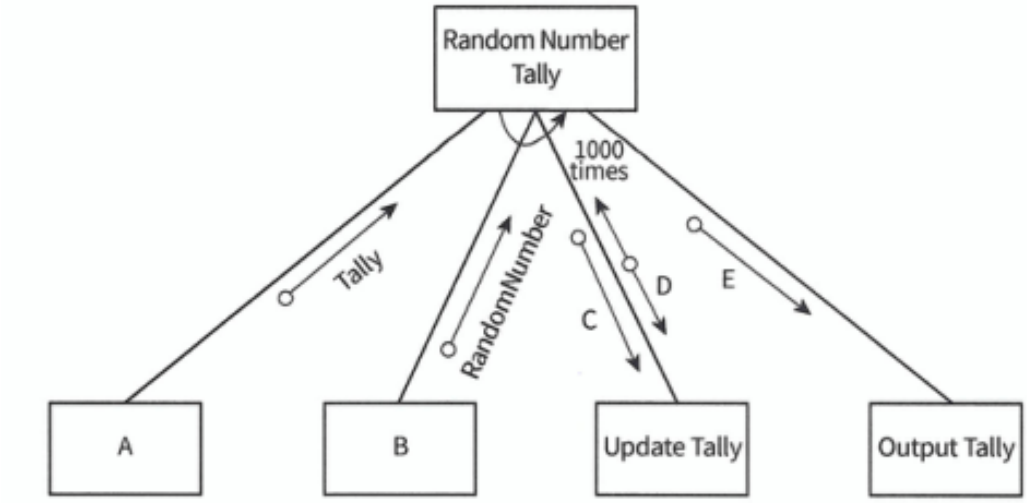
3 A random number generator is to be tested to see whether all numbers within the range 1 to 20 are generated equally frequently. The structured English version of the algorithm is

```
Initialise a tally for the numbers 1 to 20
Repeatedly generate numbers in range 1 to 20
For each number generated, increment the relevant count
Calculate how often each number should be generated (expected frequency)
Output expected frequency
Output the list of numbers as a table with actual frequency
```

The identifiers required are:

Identifier	Data Type	Explanation
Tally	Array[1 : 20] OF INTEGER	1D array to store the count of how many times each number has been generated
RandomNumber	INTEGER	The random number generated
NumberOfTests	INTEGER	The number of times a random number is to be generated (1000 in this example)
ExpectedFrequency	INTEGER	The number of times any one number would be generated if all numbers are generated equally frequently (1000/20 in this example)

(a) Complete the structure chart below by naming the labels A to E. [5]



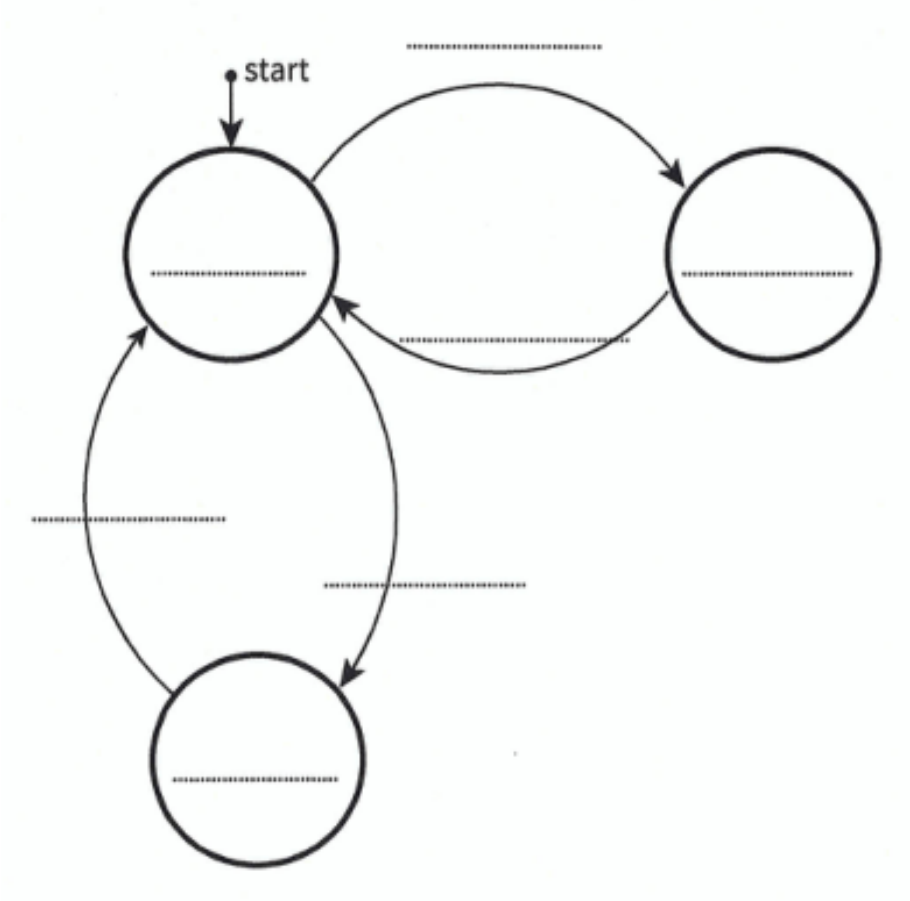
(b) Develop pseudocode from the structure chart. **[12]**

4 A car park has a barrier at the exit. The starting position of the barrier is lowered. When a car wants to exit the car park, the driver has to insert a coin into a coin slot at the barrier. The barrier raises and allows the car to drive out of the car park. After the car has passed through the barrier, the barrier lowers. In case of emergency, a member of staff can open the barrier using a remote control. The barrier will remain open until the remote control is used again to lower the barrier.

The barrier has three states: lowered, raised and open. The transition from one state to another is as shown in the state-transition table:

Current state	Event	Next state
Barrier lowered	Coin inserted	Barrier raised
Barrier lowered	Open remotely	Barrier open
Barrier open	Close remotely	Barrier lowered
Barrier raised	Car has exited	Barrier lowered

Complete the state-transition diagram for the barrier: [7]



End of assignment · Check your answers against the chapter at gurujazzy.com/subjects/computer-science/as-a-level/software-development/