OCR

RECOGNISING ACHIEVEMENT

OXFORD CAMBRIDGE AND RSA EXAMINATIONS

General Certificate of Education Advanced Subsidiary Level

COMPUTING 2507

Structured Practical Computing Tasks

Issued September 2004
Maximum mark **120**

INSTRUCTIONS TO CANDIDATES

- You should attempt all tasks, working independently from other candidates.
- There are no time limitations on the tasks other than that they must be submitted by the appropriate internal deadline set by the Candidate's Centre. This deadline will reflect the need for the Centre to complete marking of the tasks and submit the marks to OCR by the required date.
- There are no restrictions on computing facilities, hardware or software, that may be used.
- You are strongly advised to keep all your working notes as these may be required by the moderator.
- Reasons for answers to tasks are expected to form part of the work submitted.

Notice to candidates

1 The work which you submit for assessment must be your own.

However, you may:

(a) quote from books or any other sources: if you do, you must state which ones you have used;

(b) receive guidance from someone other than your teacher: if so you must tell your teacher, who will record the nature of the assistance given to you.

2 If you copy from someone else or allow another candidate to copy from you, or if you cheat in any other way, **you may be disqualified from at least the subject concerned**.

3 When you hand in your coursework for assessment, you will be required to sign that you have understood and followed the coursework and portfolio requirements for the subject.

ALWAYS REMEMBER – YOUR WORK MUST BE YOUR OWN

This question paper consists of 8 printed pages.

Registered Charity Number: 1066969
[Turn over

Task 1 [43 marks]

This is a software development task and an implementation task.

Courses at a university consist of modules. Each module is taught by one or more lecturers and each lecturer teaches at least one module. Each module has a book list consisting of one or more books. A book may be used on more than one module.

Modules have a code and a name. The code consists of two letters followed by 4 digits, for example CP1051 is called An Introduction to Visual Basic.

Each lecturer has a telephone extension number and an office; these may be shared with other lecturers. The telephone extension is a four-digit number and all offices have a five-character identification consisting of two letters and three digits, for example, MU217. Other than the names of lecturers, no personal details of lecturers are to be held.

A database is required that holds details of lecturers, modules and books. It has already been decided to hold the data in the following five tables.

Lecturer – holds details of each lecturer.

Module – holds details of each module.

LecturerModule – acts as a link between the Lecturer and Module tables.

Book – holds details of each book.

ModuleBook – acts as a link between the Module and Book tables.

(a) (i) Design the Lecturer table. You must state the name, data type and purpose of each attribute in the table. You should also specify the key for the table.

You may use a screen dump (or dumps) of your design as evidence. [5]

(ii) Design the Module table. You must state the name, data type and purpose of each attribute in the table. You should also specify the key for the table.

You may use a screen dump (or dumps) of your design as evidence. [3]

(iii) Design the LecturerModule table. You must state the name, data type and purpose of each attribute in the table. You should also specify the key for the table.

You may use a screen dump (or dumps) of your design as evidence. [3]

(iv) Design the Book table. You must state the name, data type and purpose of each attribute in the table. You should also specify the key for the table.

You may use a screen dump (or dumps) of your design as evidence. [4]

(v) Design the ModuleBook table. You must state the name, data type and purpose of each attribute in the table. You should also specify the key for the table.

You may use a screen dump (or dumps) of your design as evidence. [3]

(b) Create validation checks to ensure that

(i) the office identification is valid;

(ii) the module code is valid;

(iii) the telephone extension number is valid;

(iv) only valid data is put in the LecturerModule table;

(v) only valid data is put in the ModuleBook table.

You should include evidence that you have created all five validations. [5]

(c) Create data for your five tables. You should include at least 5 modules, 10 books and 15 lecturers. Make sure that the data is suitable for testing the rest of this task.

You may use screen dumps of your tables as evidence. [10]

Parts **(d)** and **(e)** ask you to produce reports. Marks will be awarded for the layout and contents of the reports.

(d) Create a report that lists the details of all the lecturers for each module.

Include a copy of the report as evidence. You should also include evidence showing how your solution created the report. [5]

(e) Create a report that lists full details of all the books used by a lecturer. The identity of the lecturer should be input by the user when the report is requested.

Include evidence that the user has input the Lecturer ID and a copy of the report produced. You should also include evidence showing how your solution created the report. [5]

Task 2 [25 Marks]

This is an algorithm trace task. No implementation is required.

Read the following algorithm.

```
READ Size, Start, Step
Row = 1
Col = (Size + 1) / 2
Count = 1
Number = Start
REPEAT
    Array(Row, Col) = Number
    Number = Number + Step
    Count = Count + 1
    NewRow = Row –1
    IF NewRow < 1 THEN
        NewRow = Size
    END IF
    NewCol = Col – 1
    IF NewCol < 1 THEN
        NewCol = Size
    END IF
    IF Array(NewRow, NewCol) NOT EMPTY THEN
        NewRow = Row + 1
        NewCol = Col
        IF NewRow > Size THEN
            NewRow = 1
        END IF
    END IF
    Row = NewRow
    Col = NewCol
UNTIL Count > Size * Size
```

(a) (i) Copy and complete Table 2.1 to show successive values of each of the variables when the input data is

3, 1, 1.

Size																
Start																
Step																
Row																
Col																
Count																
Number																
Array(1,1)																
Array(1,2)																
Array(1,3)																
Array(2,1)																
Array(2,2)																
Array(2,3)																
Array(3,1)																
Array(3,2)																
Array(3,3)																
NewRow																
NewCol																

Table 2.1 [10]

(ii) Copy and complete Table 2.2 to show the results of your trace.

		Col 1	Col 2	Col 3
	1			
Row	2			
	3			

Table 2.2 [1]

(b) The above algorithm produces a pattern for entering numbers into a square array. Use this pattern to complete a copy of Table 2.3 if the input is

7, 3, 2.

		Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7
	1							
	2							
	3							
Row	4							
	5							
	6							
	7							

Table 2.3 [9]

(c) **(i)** Explain what would happen if the input data were

8, 1, 1. [2]

(ii) Explain how to modify the algorithm so that it provides an appropriate response to this data. [3]

Task 3 [31 marks]

This is a software development task and an implementation task.

This task is to be solved by using a high-level language of your choice. You should state the name of the language used at the start of your solution.

A manufacturer of games has decided to design a new snakes and ladders board consisting of 100 squares. The manufacturer has asked a systems analyst to design a simulation of the game using different designs. Also, the systems analyst has been told that there must be exactly 10 snakes and 10 ladders. The number of squares that a snake sends a player back must lie between 10 and 30. The number of squares that a ladder sends a player forward must lie between 10 and 30.

The systems analyst has decided to represent the board by using a one-dimensional array with 100 cells numbered from 1 to 100. Initially each cell is to be set to zero.

When the game starts the user is asked to input the positions of the snakes. This is to be done by entering, for each snake, the square on which its head is to be and the number of squares a player must go back when landing on a snake's head. For each snake, the number of squares to go back is then to be stored in the cell representing the head of the snake, as a negative integer.

This is repeated for the ladders by inputting the squares for the feet of the ladders and the number of squares a player must go forward when landing on the foot of a ladder. For each ladder, the number of squares to go forward should be stored as a positive integer in the cell representing the foot of the ladder.

Part of the array may look similar to that in Fig. 3.1. This indicates that there is a snake's head in cell 51 and its tail is in cell 41. The bottom of a ladder is in cell 40 and its top is in cell 52. The remaining squares do not contain any snakes or ladders.

Cell Index	40	41	42	43	44	45	46	47	48	49	50	51	52	53
Data	12	0	0	0	0	0	0	0	0	0	0	–10	0	0

Fig. 3.1

You should annotate all your code and use meaningful names throughout.

(a) Create annotated code that initialises the board ready for the user input. [2]

(b) Create an interface that allows the user to input the data as described above. Your code should validate the input and store the data as described above. You need only check for values that satisfy the criteria given above. [5]

The rules of this game of Snakes and Ladders are

- On each turn, a player moves forward the number of squares shown when a 6-sided die is thrown.
- If a player lands on the head of a snake, the player must move to the tail of the snake.
- If a player lands on the foot of a ladder, the player must move to the top of the ladder.
- If a move would result in the player going off the board, no move is made.

You need only simulate the moves of a **single** player for the rest of this task.

(c) Create annotated code, in a high-level language, that simulates a game of snakes and ladders and outputs the total number of throws. [8]

(d) Modify your solution so that it asks a user to input the number of games to be played. Your solution should then play that number of games and output the average number of 'throws' per game. [4]

(e) The user now wishes to be able to enter the number of snakes and the number of ladders before entering their details. The number of each has to be in the range 5 to 10.

Modify your solution so that the user can enter the number of snakes and the number of ladders and their details. [6]

(f) The systems analyst now specifies the following restrictions on data entry.

- A square can only contain one of
  - the top of a ladder
  - the foot of a ladder
  - the head of a snake
  - the tail of a snake
- No ladder goes off the board
- No snake goes off the board

Modify your program to ensure it handles these restrictions. Annotate your modifications to show how you have solved this task. You should provide sample runs to show the results of using **one** set of valid data and **one** set of invalid data for **each** restriction. [6]

Task 4 [21 Marks]

This is an algorithm trace task. No implementation is required.

Read the following algorithm for the recursive function called Mystery. X( ) is a one-dimensional array.

```
Function Mystery(X( ), N)

    IF N = 1 THEN
        Mystery = 1
    ELSE
        Te mp = Mystery(X( ), N - 1)
        IF X(N) >= X(Temp) THEN
            Mystery = N
        ELSE
            Mystery = Temp
        ENDIF
    ENDIF

End Function Mystery
```

(a) Write down the exact output when the function is called with

(i) Output 'The answer is: ', Mystery(A( ), 5)

if A(1) = 6, A(2) = 10, A(3) = 6, A(4) = 18, A(5) = 15.

(ii) Output 'The answer is: ', Mystery(A( ), 5)

if A(1) = 16, A(2) = 18, A(3) = 12, A(4) = 15, A(5) = 18. [2]

(b) The function is called with

Mystery(A( ), 3)

where A(1) = 28, A(2) = 20 and A(3) = 28.

Produce a diagram showing each step of the algorithm and the function calls. [9]

(c) Describe the purpose of the algorithm. [3]

(d) Write an iterative algorithm that does the same task as the recursive algorithm. [7]