REGISTRATION CENTRE NUMBER	CENTRE NAME	
CANDIDATE'S FULL NAMES		
CANDIDATE'S IDENTIFICATION NUMBER	SUBJECT CODE **0795**	PAPER NUMBER **3**

FOLI

HERI

FOR OFFICIAL USE ONLY (Candidate Random Code): ▶	

GENERAL CERTIFICATE OF EDUCATION BOARD

General Certificate of Education Examination

ADVANCED LEVEL

SUBJECT TITLE **COMPUTER SCIENCE**	SUBJECT CODE **0795**	PAPER NUMBER **3**
	EXAMINATION DATE: **JUNE 2025**	

Duration: Two Hours

Enter the information required in the shaded boxes above.

Do not write in pencil.

Carry out ALL the tasks given. For your guidance, the approximate mark for each part of a task is indicated in brackets.

Great importance is attached to the accuracy, layout and labelling of drawings and computer generated outputs.

You are reminded of the necessity for good English and orderly presentation of your answers.

Record all your answers in the spaces provided in this question booklet. Also record in your question booklet any information requested or that you believe would make it easier to understand how you carried out tasks or answered questions. Blank pages have been provided at the end of this booklet in case you need additional space for your answers or rough work.
Make sure all your answers, including printed works, are submitted with your question booklet.
*When an imperative programming language is required to write program code, either Standard **[ISO]** Pascal or the **[ANSI]** C programming language may be used.*

If need be, supervisors will assist you in recording details of intermediate work carried out on the computer.

FOR EXAMINERS' USE ONLY	
Marked by:.....	***SCORE***
Signature: ..... ***Date:***	
Checked by:.....	
Signature: ..... ***Date:***	

Turn Over

SECTION A: DATABASE SYSTEMS

Database design and implementation

An e-commerce company wants to develop a database to manage its data. The database should store information about customers, orders, products and suppliers.

Customer (CustomerID, Name, email, Address)
Product (ProductID, Product Name, Price)
Supplier (SupplierID, SupplierName, Address)
Order (Order Date, Total)

Task 1

1. In your answer booklet, produce an entity relation diagram in 3rd normal form. Underline key fields therein.
(5 marks)

2. Write SQL queries that will enable you to add the column **phoneNumber** to the table **Customer**, and **Description** to table **Product**.
(2 marks)

.....

.....

.....

.....

.....

.....

3. Use a suitable DBMS of your choice to implement the database design in 1 above; print a copy of your database.
(2 marks)

4. Insert sample data into the tables created and print a copy each of product, customer and order; see the tables below.
(3 marks)

5. Implement SQL queries to perform the following operations, and then copy them into their respective spaces below:
   a) Retrieve the names of all customers who have placed an order.

.....

.....

.....

.....

(1 mark)

b) Retrieve the names of all the products that Jane supplied. Use the join operator to obtain your answer.

.....

.....

.....

.....

(2 marks)

c) Print a copy of each query. **(1 mark)**

d) Retrieve the names of all customers who have ordered a product supplied by Ousman. **(3 marks)**

.....

.....

.....

.....

.....

.....

e) Print a copy of this query. **(1 mark)**

CustomerID	Customer Name	Email	Address
001	John Che	Johnc@gmail.com	Bekora
002	Jane Fuh	Jane@gmail.com	Bafang
003	Jane Fuh	Fuh@yahoo.com	Bafang
004	Shutang Ben	Shutang@hotmail.com	Kejum

ProductID	Product Name	SupplierID	Price
001	Caps	01	2500
002	Shirts	01	1500
003	Shoes	02	2000

SupplierID	Supplier Name	Address
1	Ousman	Njombe
2	Manka'a	Sangmelima
3	Ngimfack	Banteng
4	Manka'a	Sangmelima

OrderID	CustomerID	Order Date
1	001	20-03-2025
2	002	25-03-2025

SECTION B: PROGRAM DEVELOPMENT (30 marks)

Using your preferred programming language (PL), C or Pascal, and integrated program development environment (IDE), carry out the following programming tasks. Note that comments within questions start from a double slash (//) to the end-of-line, or are between / and */.*

Game of Life Task Description

Conways' Game of Life (or just "Life") is a "cellular automaton" which evolves live and dead cells within a grid (or board) here represented as a 2-D array. Grid cells are either alive or dead and evolve in successive generations as determined by their neighbours. The neighbours of any given cell are horizontally, vertically or diagonally adjacent to it. The following rules are applied to the current (or initial) generation of cells in order to obtain the cell values of the next generation.

1) Any live cell with fewer than two live neighbours dies as if caused by under population.
2) Any live cell with two or three live neighbours lives on to the next generation.
3) Any live cell with more than three live neighbours dies, as if by overpopulation.
4) Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

For example, Figure 1(a) is a 4-by-4 grid for the initial stage (or current generation) of a "Life" cellular automaton and Figure 1(b) is the next generation obtained by obeying the above rules.

```
Initial Stage:
 ----- ----- ----- -----
|  1  |  1  |  0  |  0  |
 ----- ----- ----- -----
|  1  |  0  |  0  |  0  |
 ----- ----- ----- -----
|  0  |  0  |  1  |  1  |
 ----- ----- ----- -----
|  1  |  1  |  1  |  1  |
 ----- ----- ----- -----
```

Figure. 1: (a) Initial Stage

```
Next Generation:
 ----- ----- ----- -----
|  1  |  1  |  0  |  0  |
 ----- ----- ----- -----
|  1  |  0  |  1  |  0  |
 ----- ----- ----- -----
|  1  |  0  |  0  |  1  |
 ----- ----- ----- -----
|  0  |  1  |  0  |  1  |
 ----- ----- ----- -----
```

(b) Next Generation

Task 2: Coding Algorithms **(17 marks)**

We develop program fragments to display and evolve successive generations of "Life" from a 2-D array. Arrays `A[Row][Col]` and `B[Row][Col]` are used to store the current and next generation of the board's state. `Col` and `Row` are global variables used to store the size of the arrays and are both initially set to 4. You are to develop executable code from the algorithms given in each subtask. For each subtask, make sure the coding is correctly done.

1) Subprogram `row_line()` draws a horizontal line of dashes as follows. It outputs a new line then, for each column, outputs five dashes (- - - - -). It then outputs a second new line. **(2 marks)**

2) Implement the [LCG] random number generator (RNG) given in Figure 2. Follow the directives given in it, in italics. You will later call function `rand_init(s)` in order to initialise the RNG. Similarly, you will later call function `rand_no()` repeatedly in order to return successive random numbers. **(4 marks)**

```
Define the global constant:
RAND_SEED = 7  //default random number seed.

Define the global variable with an initial default value:
Rand_x <- RAND_SEED // default initial value.

Define functions rand_init(s) and rand_no().
// seed (initialise) the RNG with
// integer s, where 0 < s < 16.
function Rand_init (s)≡
      Rand_x <- s
endfunc

// return the next random number.
function Rand_no () ≡
   rand_x = (5 * rand_x + 3) mod 16
   return (rand_x mod 12) // return 0 or 1
endfunc
```

Figure 2. An LCG Random Number Generator.

3) Subprogram `Gen_cells(A, row, col)` takes as arguments an array `A`, for the grid, and `row` and `col` for its size. It returns a `row`-by-`col` board state in which each cell is randomly initialized to either ALIVE (represented by a **1**) or DEAD (represented by a **0**) via a random number generator. **(2 marks)**

```
subprogam Gen_cells(A, row, col)≡
begin
   for i from 1 to row do
      for j from 1 to col do
       A[i][j]= rand_no()      // randomly insert a 0 or 1:
      endfor
   endfor
endsubprogram
```

4) Subprogram `count_live_neighs (A, r, c)` counts the number of live neighbours in array `A` for the cell at row `r` and column `c`. **(3 marks)**

```
count_live_neighs(A, r, c) ≡
begin
   i, j, count : Integer
   for i from r-1 to r+1 do
      for j from c-1 to c+1 do
         if not ((i less than 0 OR j is less than 0) OR
                 (i greater than or equal to Row) OR
                 (j greater than or equal to Col)
                 )          // ensure cell indices are valid
         then
            if(A[i][j] EQUAL 1) then
               increment count
            endif
         endif
      endfor
   endfor
   return count
endproc
```

5) Subprogram `Next_Gen(A, B)` generates into array `B` the next generation cells from array `A`. **(4 marks)**

```
Next_Gen(A,B) ≡
begin
   for i from 1 to Row do
      for j from 1 to Col do
         n_live = count_live_neighs(A,i,j)
         if(A[i][j] EQUAL 1 AND
               (n_live EQUAL 2  OR  n_live EQUAL 3))
         then
            Set B[i][j] to 1
         else if(A[i][j] EQUAL 0 AND n_live EQUAL 3) then
            Set B[i][j] to 1
         else
            Set B[i][j] to 0
         endif
      endfor
   endfor
endproc
```

6) Make sure the subprograms at least compile. Save your work, then print your source code. **(2 marks)**

Task 3: Finalising with Displays **(13 marks)**

We exploit the algorithms from Task 2 to display and evolve the various generations of "Life". Specifically, call existing subprograms, where applicable, to complete each task.

7) In this answer booklet, exploit the algorithms in Task 2, to write the following:

(a) Subprogram `PrintMat(A, row, col)` to displays the board's initial state. **(2 marks)**

(b) Subprogram `PrintNextGen(B, row, col)` to displays the board's next generation. **(2 marks)**

8) In your IDE, develop executable subprograms from the algorithms in question (7). Your subprogram should display a vertical bar (|) and appropriate number of spaces before each value printed, and a final bar at the end of the row. Use subprogram `row_line()` to draw lines between rows. Your grid layout should be similar to those in Figure 1. **(3 marks)**

9) Write a main program that calls the following subprogram in order to print and evolve the cellular automaton: `Rand_init(s)`, `Gen_cells(A,row,col)`, `PrintMat(A,row,col)`, `NextGen(A,B)`, and `PrintNextGen(B,row,col)`.

Note: *See Figure 2 for constraints on argument* `s` in function `Rand°init`. **(3 marks)**

10) Print a copy of the entire source code. **(1 mark)**

11) Modify your program from Task 3(8) so that the row and column dimensions of the board is 8. Make sure the program works correctly. Then run it and screen-capture its initial and next generation stages, save them in the file called `Size8Output`, and then print a copy. **(2 marks)**