Task 0: ADDRESS TRANSLATION

You can only be eligible for bonus marks if you attempt the whole assignment

Consider the following C++ code fragment and answer the following questions:
const int City = 6, School = 3, Class = 4;
int enrolled_Students [City][School][Class];

A. What is represented by enrolled_Students [2][1][3]? Explain it.

B. Compute the logical address of enrolled_Students [2][1][3]. Suppose the base address of the table is $0x100_{(h)}$ in C++? Show complete steps. Also show the addressable location with the help of a diagram to support your calculations. Show final address in hexadecimal form. Marks will not be awarded for the direct answer.

Hint: dimensions of enrolled_Students are 6 x 3 x 4

Task 1: DYNAMIC 1D ARRAY

A. Write a C++ program that dynamically creates an array of N elements. The program should take size and input in the array from the user. Then the program calculates average of the values in the array.

Note: Array elements should be accessed using pointer notation only.

Task 2: DYNAMIC 2D ARRAY

A. Write a program to create a dynamic 2D array (or matrix), i.e., array is created on heap using new operator. The program gets the following inputs from the user.

- Size of the matrix, i.e., number of rows and columns
- Values to be stored on each index of the matrix, i.e., ask user to fill the matrix.

Afterwards, the program should display the matrix row-wise and column-wise.

B. Write a program to multiply two matrices and return the resultant matrix.

Note: Array elements should be accessed using pointer notation only.

Task 3: SORTING AND MERGING

A. Write a program that takes N integer arrays (of varying sizes) as input. You have to ensure that user enter these arrays in ascending order, if user enters incorrectly display a prompt to read input in correct format. Write a C++ program to produce an array that merges elements of all arrays in descending order, but it also needs to remove duplicates.

Example:

```
Array A_1: {1, 3, 5, 6}
Array A_2: {1, 2, 4, 8}
Array A_3: {5, 6, 7, 8}
Array A_4: {23, 24, 94, 108}
Array A_5: {1, 2, 2, 23, 24, 67, 1234}
Merged array: {1234, 108, 94, 67, 24, 23, 8, 7, 6, 5, 4, 3, 2, 1}
```

Note: Final array should be created/merged in sorted order.

Task 4: EMPLOYEE PERFORMANCE REPORTING

Write a program that calculates performance based salary of customer support representatives (CSRs) in an organization:

- **csrID (Customer support representative ID):** An array of seven strings to hold employee identification numbers. The array should be initialized with the following numbers: CSR_01, CSR_02, CSR_03, CSR_04, CSR_05, CSR_06, CSR_07
- **csrName:** An array to hold the name of each CSR, input at run time
- **hours:** Number of hours worked by each CSR, input at run time
- **complaintsResolved:** Number of complaints successfully resolved by each CSR, input at run time
- **payRate:** An array of seven to hold each employee s hourly pay rate, where payRate is calculated as following:

  payRate = $25 + 25*(complaintsResolved by each CSR/total complaints resolved)
- **wages**: An array hold each employee’s wages wages = hours * payRate

The program should relate the data in each array through the subscripts. For example, the number in element 0 of the hours array should be the number of hours worked by the CSR whose identification number is stored in element 0 of the csrID array. The program should do the following:

- Display each csrID and ask the user to enter names, hours and complaintsResolved.
- It should then calculate the payRate and gross wages for that CSR and store them in the payRate and wages array, respectively.

- After the data has been entered for all the CSRs, the program should display each CSR's identification number, name and gross wages.
- Program should also display options to display the top N CSRs on the basis of different criteria, including: number of complaintsResolved, number of hours worked.

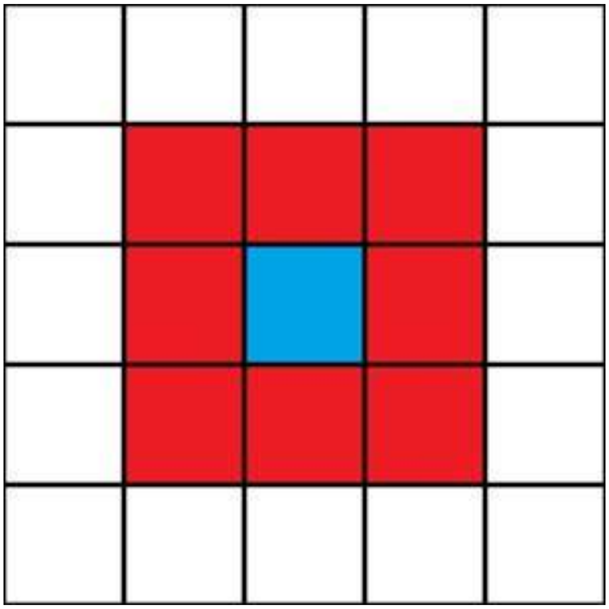
Note: Ensure all necessary input validation, for example, hours, complaintsResolved cannot be negative numbers.

Task 5: GAME OF LIFE

Game of Life (or just "Life") is not really a game. There's no winning or losing or destroying your opponent mentally and spiritually. Life is a "cellular automaton" - a system of cells that live on a grid, where they live, die and evolve according to the rules that govern their world.

Life's simple, elegant rules give rise to astonishingly complex emergent behavior. It is played on a 2-D grid Each square in the grid contains a cell, and each cell starts the game as either "alive" or "dead". Play proceeds in rounds. During each round, each cell looks at its 8 immediate neighbors and counts up the number of them that are currently alive.

Make a type char 30 x 30 2D grid. Randomly assign active and dead cells. Active cells will have value '*' and dead cell will have value ' '.



In Above diagram Blue cell is the current cell whereas Red cells are its neighboring cells The cell then updates its own liveness according to 4 rules:

1. Any live cell with 0 or 1 live neighbors becomes dead, because of underpopulation
2. Any live cell with 2 or 3 live neighbors stays alive, because its neighborhood is just right
3. Any live cell with more than 3 live neighbors becomes dead, because of overpopulation
4. Any dead cell with exactly 3 live neighbors becomes alive, by reproduction

Run your code for infinite rounds and observe the pattern changing

And that’s all there is to Life. These 4 rules give rise to some unbelievably complex and beautiful patterns, and an equally unbelievable quantity of analysis by Life devotees intent on discovering new ones.