

For Assignment 1 Question 2, it is not required to be answered.

Hints for Assignment 1 Question 1:

Comparison of *A and **A for Matrix Multiplication in C

In C programming, matrices can be represented and manipulated using dynamic memory allocation in two common ways: as a single pointer (double *A or int *A) for a contiguous 1D array simulating a 2D matrix, or as a pointer to a pointer (double **A or int **A) for a true 2D array with row pointers. Both approaches are suitable for matrix multiplication (e.g., multiplying A ($m \times n$) by B ($n \times p$) to get C ($m \times p$)), but they differ in allocation, access, memory efficiency, performance, and complexity.

1. Conceptual Representation

- double *A (Single Pointer, Contiguous Allocation):
- Treats the matrix as a flat 1D array in row-major order.
- The entire matrix is stored in one contiguous block of memory.
- Suitable for simulating 2D access via index calculations (e.g., $A[i * \text{cols} + j]$ for element at row i , column j).
- This is like a "flattened" 2D array.
- double **A (Pointer to Pointer, Non-Contiguous Rows):
- Represents a true 2D array as an array of row pointers.
- Each row is allocated separately, so rows may not be contiguous in memory.
- Allows natural 2D array syntax (e.g., $A[i][j]$).

2. Memory Allocation

- double *A:
- Single malloc call for the entire matrix.
- Example for matrix A ($m \times n$):
`double *A = (double *)malloc(m * n * sizeof(double));`
- Total memory: Exactly $m * n * \text{sizeof(double)}$ bytes (e.g., 8 bytes per double).
- Simpler: One allocation, one free(A).

// Input dimensions

```
printf("Enter rows and columns for matrix A (m n): ");
scanf("%d %d", &m, &n);
printf("Enter columns for matrix B (p): ");
scanf("%d", &p);
```

```
// Input elements for matrix A
printf("Enter elements of matrix A (%d x %d):\n", m, n);
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        printf("A[%d][%d] = ", i, j);
        scanf("%lf", &A[i * n + j]);
    }
}
```