Instructions:

- This lab is based on the topics: Structures, Pointers and Dynamic Memory Allocation.
- You should save each program with the file name as specified against each problem as <Lab#>-<Assignment#>-<Roll#>.c. For example, **10-01-24NA10006.c** to save the Program to 1st assignment in Lab 10 with Roll Number 24NA10006.
- You should upload each program to the Moodle system. Also, copy + paste your programs to the text window on the test page.
- A few test cases against each problem are given for your reference, including but not limited to.
- There are three problems and the maximum time allowed is 150 minutes.

Problem 1:

a. Define a structure **Point** to store a 2D point.
   Create a method **read_point()** to read the coordinate values of a point.
b. Define another structure to store three 2D points of a triangle.
   Create a method **read_traingle()** to read three points of a triangle.
   Also define a method **area()** to calculate the area of a triangle.
c. In your **main()**, write statements to create a triangle and print the coordinates of the triangle.
   In addition, check if the three points are lying on a straight line.

$[(5+5)+(5+5+10)+5=35]$

Test cases:

#	INPUT (Points)	OUTPUT
1	0 0 4 0 0 3	Triangle coordinates: (0, 0), (4, 0), (0, 3) Area: 6.0 Collinearity: Not Collinear
2	0 0 1 1 2 2	Triangle coordinates: (0, 0), (1, 1), (2, 2) Area: 0.0 Collinearity: Collinear
3	-3 -2 1 4 4 -3	Triangle coordinates: (-3, -2), (1, 4), (4, -3) Area: 23 Collinearity: Not Collinear
4	0 0 3 3 0 0	Triangle coordinates: (0, 0), (3, 3), (0, 0) Area: 0.0 Collinearity: Collinear

```c
// Code Creator: Nishkal Prakash (nishkal@iitkgp.ac.in)
// Code to compute the area of a triangle and check if the three points
are lying on a straight line
#include <stdio.h>
#include <math.h>

// The structure Point to store a 2D point
struct Point {
    int x, y;
};

// The structure to store three 2D points of a triangle
struct Triangle {
    struct Point p1, p2, p3;
};

// Method to read the coordinate values of a point
void read_point(struct Point *p) {
    scanf("%d %d", &p->x, &p->y);
}

// Method to read three points of a triangle
void read_traingle(struct Triangle *t) {
    read_point(&t->p1);
    read_point(&t->p2);
    read_point(&t->p3);
}

// Method to calculate the area of a triangle
float area(struct Triangle t) {
    return abs((t.p1.x * (t.p2.y - t.p3.y) + t.p2.x * (t.p3.y - t.p1.y)
+ t.p3.x * (t.p1.y - t.p2.y)) / 2.0);
}

int main() {
    struct Triangle t;
    // Reading the coordinates of the triangle
    read_traingle(&t);
    // Printing the coordinates of the triangle
    printf("Triangle coordinates: (%d, %d), (%d, %d), (%d, %d)\n",
t.p1.x, t.p1.y, t.p2.x, t.p2.y, t.p3.x, t.p3.y);
    // Printing the area of the triangle
    printf("Area: %.1f\n", area(t));
    // Checking if the three points are lying on a straight line
    if (area(t) == 0)
        printf("Collinearity: Collinear\n");
    else
        printf("Collinearity: Not Collinear\n");
    return 0;
}
```

Problem 2:

A vector is an *n*-dimensional data, which can be stored in an array of integers of size *n*. For example, *v=ix+jy+kz* can be stored as *[x,y,z]* where *i,j, and k* are three dimensions. You have to write a program to do the following.

- **a.** Define a structure to store a vector of dimension *n>=2*. The value of *n* is known at the run-time of your program.
- **b.** Using your structure definition, read two vectors, say *A* and *B* from the user.
- **c.** For the two vectors *A* and *B*, find the cosine angle between the two vectors *A* and *B*. Hint: $\cos\Theta = (A.B)/(|A||B|)$, where *A.B* denotes scalar product of two vectors *A* and *B*, and $|A|, |B|$ denotes the magnitude of a vector *A* and *B* respectively.
- **d.** Decide if the two vectors are parallel or perpendicular to each other.

(Assume the vectors will be of the same dimension)

$[5 + 5 + (15 + 5) + 5 = 35]$

Test cases:

#	INPUT (Vector A and Vector B)	OUTPUT
1	2 1 0 2 0 1	Cosine angle: 0 Angle in degrees: 90° Relation: Perpendicular
2	3 1 2 3 3 2 4 6	Cosine angle: 1 Angle in degrees: 0° Relation: Parallel
3	3 1 0 0 3 1 1 0	Cosine angle: 0.707 Angle in degrees: 45° Relation: Neither Parallel nor Perpendicular
4	4 1 1 1 1 4 5 5 5 5	Cosine angle: 1.000 Angle in degrees: 0 Relation: Parallel

```c
// Code Creator: Nishkal Prakash (nishkal@iitkgp.ac.in)
// Code to calculate the cosine angle between two vectors and check if they are parallel or perpendicular
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// A structure to store a vector of dimension n>=2
struct vector {
    int n;
    int *data;
};

// Method to read the vector
void read_vector(struct vector *v) {
    scanf("%d", &v->n);
    v->data = (int *)malloc(v->n * sizeof(int));
    for (int i = 0; i < v->n; i++)
        scanf("%d", &v->data[i]);
}

// Method to calculate the dot product of two vectors
int dot_product(struct vector A, struct vector B) {
    int sum = 0;
    for (int i = 0; i < A.n; i++)
        sum += A.data[i] * B.data[i];
    return sum;
}

// Method to calculate the magnitude of a vector
float magnitude(struct vector A) {
    int sum = 0;
    for (int i = 0; i < A.n; i++)
        sum += A.data[i] * A.data[i];
    return (float)sqrt(sum);
}

int main() {
    struct vector A, B;
    // Reading the two vectors
    read_vector(&A);
    read_vector(&B);
    // Calculating the cosine angle between the two vectors
    int dot = dot_product(A, B);
    float magA = magnitude(A);
    float magB = magnitude(B);
    float cosine = dot / (float)(magA * magB);
    // Printing the cosine angle and the angle in degrees
    printf("Cosine angle: %.3f\n", cosine);
    printf("Angle in degrees: %.0f\n", acos(cosine) * 180 / 3.14159);
    // Calculating abs_c = absoulte value of cosine
    float abs_c = (cosine < 0) ? -cosine : cosine;
    // Checking if the two vectors are parallel or perpendicular
    if ((1-abs_c) <= 0.0001)
        printf("Relation: Parallel\n");
    else if (abs_c <= 0.0001)
        printf("Relation: Perpendicular\n");
    else
        printf("Relation: Neither Parallel nor Perpendicular\n");
    return 0;
}
```

Problem 3:

You have to write a C-program to multiply two matrices A_{m*n} and B_{n*p} where $m*n$ and $n*p$ denotes the size of the matrices in usual notation. Do the following.

a. Define the structure which would appropriately store all information of a matrix. Assume that your matrix will store integer values only.
Create a method **create()** to initialize values reading from the keyboard.

b. Write a function **matrixMultiplication()**, where you should pass two matrices A_{m*n} and B_{n*p}, and it will return a matrix $C_{m*p}=A_{m*n}$ X B_{n*p}.
Here, X denotes the matrix multiplication.

(NOTE: Extra 20 marks will be given for writing sufficient comments in your programs)

Test cases:

#	INPUT (Matrix A and Matrix B)	OUTPUT (Resultant Matrix C)
1	Matrix A (2x3): 1 2 3 4 5 6 Matrix B (3x2): 7 8 9 10 11 12	Matrix C (2x2): 58 64 139 154
2	Matrix A (3x3): 1 0 2 0 3 0 4 5 6 Matrix B (3x3): 7 8 9 10 11 12 13 14 15	Matrix C (3x3): 33 36 39 30 33 36 156 171 186
3	Matrix A (2x2): 1 2 3 4 Matrix B (2x2): 2 0 1 2	Matrix C (2x2): 4 4 10 8
4	Matrix A (1x3): 3 4 2 Matrix B (3x2): 13 9 8 7 6 4	Matrix C (1x2): 83 63

$[(8 + 7) + (10 + 15) = 40]$

```c
// Code Creator: Nishkal Prakash (nishkal@iitkgp.ac.in)
// Code to multiply two matrices
#include <stdio.h>
#include <stdlib.h>

// Structure to store matrix
struct Matrix {
    int row, col;
    int **matrix;
};
// Method to create matrix
void create(struct Matrix *m) {
    printf("Matrix (%dx%d):\n", m->row, m->col);
    m->matrix = (int **)malloc(m->row * sizeof(int *));
    for (int i = 0; i < m->row; i++) {
        m->matrix[i] = (int *)malloc(m->col * sizeof(int));
        for (int j = 0; j < m->col; j++) {
            scanf("%d", &m->matrix[i][j]);
        }
    }
}
// Function to multiply two matrices
struct Matrix matrixMultiplication(struct Matrix a, struct Matrix b) {
    struct Matrix c;
    c.row = a.row;
    c.col = b.col;
    // Allocate memory for matrix c
    c.matrix = (int **)malloc(c.row * sizeof(int *));
    for (int i = 0; i < c.row; i++) {
        c.matrix[i] = (int *)malloc(c.col * sizeof(int));
        for (int j = 0; j < c.col; j++) {
            c.matrix[i][j] = 0;
            for (int k = 0; k < a.col; k++) {
                c.matrix[i][j] += a.matrix[i][k] * b.matrix[k][j];
            }
        }
    }
    return c;
}
// Main method
int main() {
    struct Matrix a, b, c;
    printf("Enter the size of matrix A (row col): ");
    scanf("%d %d", &a.row, &a.col);
    // Create matrix a
    create(&a);
    printf("Enter the size of matrix B (row col): ");
    scanf("%d %d", &b.row, &b.col);
    // Create matrix b
    create(&b);
    // Check if matrix multiplication is possible
    if (a.col != b.row) {
        printf("Matrix multiplication is not possible\n");
        return 0;
    }
    // Multiply matrix a and b
    c = matrixMultiplication(a, b);
    // Print matrix c
    printf("Matrix C (%dx%d):\n", c.row, c.col);
    for (int i = 0; i < c.row; i++) {
        for (int j = 0; j < c.col; j++) {
            printf("%d ", c.matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```