#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define MAX 100 // Maximum matrix size

typedef struct {

    int row; // Row index for the element in the result matrix

    int col; // Column index for the element in the result matrix

    int n;   // Number of columns in Matrix A (or rows in Matrix B)

    int (\*matrixA)[MAX];

    int (\*matrixB)[MAX];

    int (\*result)[MAX];

} ThreadData;

// Function to compute a single element in the result matrix

void\* multiplyElement(void\* arg) {

    ThreadData\* data = (ThreadData\*)arg;

    int sum = 0;

    for (int k = 0; k < data->n; k++) {

        sum += data->matrixA[data->row][k] \* data->matrixB[k][data->col];

    }

    data->result[data->row][data->col] = sum;

    pthread\_exit(0);

}

int main() {

    int rowsA, colsA, rowsB, colsB;

    int matrixA[MAX][MAX], matrixB[MAX][MAX], result[MAX][MAX];

    pthread\_t threads[MAX \* MAX];

    ThreadData threadData[MAX \* MAX];

    int threadCount = 0;

    // Input dimensions for Matrix A and Matrix B

    printf("Enter number of rows for Matrix A: ");

    scanf("%d", &rowsA);

    printf("Enter number of columns for Matrix A: ");

    scanf("%d", &colsA);

    printf("Enter number of rows for Matrix B: ");

    scanf("%d", &rowsB);

    printf("Enter number of columns for Matrix B: ");

    scanf("%d", &colsB);

    // Validate dimensions for multiplication

    if (colsA != rowsB) {

        printf("Matrix multiplication not possible: columns of Matrix A must equal rows of Matrix B.\n");

        return -1;

    }

    // Input elements of Matrix A

    printf("Enter elements of Matrix A:\n");

    for (int i = 0; i < rowsA; i++) {

        for (int j = 0; j < colsA; j++) {

            scanf("%d", &matrixA[i][j]);

        }

    }

    // Input elements of Matrix B

    printf("Enter elements of Matrix B:\n");

    for (int i = 0; i < rowsB; i++) {

        for (int j = 0; j < colsB; j++) {

            scanf("%d", &matrixB[i][j]);

        }

    }

    // Create threads for each element in the result matrix

    for (int i = 0; i < rowsA; i++) {

        for (int j = 0; j < colsB; j++) {

            threadData[threadCount].row = i;

            threadData[threadCount].col = j;

            threadData[threadCount].n = colsA; // Shared dimension

            threadData[threadCount].matrixA = matrixA;

            threadData[threadCount].matrixB = matrixB;

            threadData[threadCount].result = result;

            pthread\_create(&threads[threadCount], NULL, multiplyElement, &threadData[threadCount]);

            threadCount++;

        }

    }

    // Wait for all threads to finish

    for (int i = 0; i < threadCount; i++) {

        pthread\_join(threads[i], NULL);

    }

    // Print the result matrix

    printf("Resultant Matrix after Multiplication:\n");

    for (int i = 0; i < rowsA; i++) {

        for (int j = 0; j < colsB; j++) {

            printf("%d ", result[i][j]);

        }

        printf("\n");

    }

    return 0;

}