#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#define MAX 100 // Maximum matrix size

typedef struct {

int row;

int col;

int n; // For multiplication: shared dimension

int operation; // 0 for addition, 1 for multiplication

int (\*matrixA)[MAX];

int (\*matrixB)[MAX];

int (\*result)[MAX];

} ThreadData;

// Function to perform matrix operation for a single element

void\* performOperation(void\* arg) {

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

if (data->operation == 0) { // Addition

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

data->matrixA[data->row][data->col] + data->matrixB[data->row][data->col];

} else if (data->operation == 1) { // Multiplication

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;

char operation[10];

// Input matrix dimensions

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

scanf("%d", &rowsA);

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

scanf("%d", &colsA);

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

scanf("%d", &colsB);

// 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 < colsA; i++) {

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

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

}

}

// Input operation

printf("Choose operation (add/multiply): ");

scanf("%s", operation);

// Validate operation and dimensions

if (strcmp(operation, "add") == 0) {

if (rowsA != colsA || rowsA != rowsB || colsA != colsB) {

printf("Addition requires both matrices to have the same dimensions.\n");

return -1;

}

} else if (strcmp(operation, "multiply") == 0) {

if (colsA != rowsB) {

printf("Multiplication requires columns of Matrix A to match rows of Matrix B.\n");

return -1;

}

} else {

printf("Invalid operation. Choose 'add' or 'multiply'.\n");

return -1;

}

// Create threads for each element in the result matrix

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

for (int j = 0; j < (strcmp(operation, "add") == 0 ? colsA : colsB); j++) {

threadData[threadCount].row = i;

threadData[threadCount].col = j;

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

threadData[threadCount].matrixA = matrixA;

threadData[threadCount].matrixB = matrixB;

threadData[threadCount].result = result;

threadData[threadCount].operation = strcmp(operation, "add") == 0 ? 0 : 1;

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

threadCount++;

}

}

// Wait for all threads to complete

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

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

}

// Display result matrix

printf("Result Matrix:\n");

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

for (int j = 0; j < (strcmp(operation, "add") == 0 ? colsA : colsB); j++) {

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

}

printf("\n");

}

return 0;

}