Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

// Function declarations
bool isSafeState(int processes, int resources, int available[], int max[][MAX_RESOURCES], int
allocation[][MAX_RESOURCES], int need[][MAX_RESOURCES], int safeSequence[]);
void calculateNeed(int processes, int resources, int max[][MAX_RESOURCES], int
allocation[][MAX_RESOURCES], int need[][MAX_RESOURCES]);

int main() {
    int processes, resources;
    int available[MAX_RESOURCES];
    int max[MAX_PROCESSES][MAX_RESOURCES];
    int allocation[MAX_PROCESSES][MAX_RESOURCES];
    int need[MAX_PROCESSES][MAX_RESOURCES];
    int safeSequence[MAX_PROCESSES];

    printf("Enter the number of processes: ");
    scanf("%d", &processes);

    printf("Enter the number of resources: ");
    scanf("%d", &resources);

    printf("Enter the available resources:\n");
    for (int i = 0; i < resources; i++) {
        scanf("%d", &available[i]);
    }

    printf("Enter the maximum resource matrix:\n");
    for (int i = 0; i < processes; i++) {
        printf("Enter details for P%d\n", i);
        for (int j = 0; j < resources; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter the allocation resource matrix:\n");
    for (int i = 0; i < processes; i++) {
```

```
        for (int j = 0; j < resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    // Calculate the need matrix
    calculateNeed(processes, resources, max, allocation, need);

    if (isSafeState(processes, resources, available, max, allocation, need, safeSequence)) {
        printf("SYSTEM IS IN SAFE STATE\n");
        printf("The Safe Sequence is -- (");
        for (int i = 0; i < processes; i++) {
            printf("P%d ", safeSequence[i]);
        }
        printf(")\n");
    } else {
        printf("SYSTEM IS NOT IN SAFE STATE\n");
    }

    printf("\nProcess\tAllocation\t\tMax\t\tNeed\n");
    for (int i = 0; i < processes; i++) {
        printf("P%d\t", i);
        for (int j = 0; j < resources; j++) {
            printf("%d ", allocation[i][j]);
        }
        printf("\t");
        for (int j = 0; j < resources; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\t");
        for (int j = 0; j < resources; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }

    return 0;
}

void calculateNeed(int processes, int resources, int max[][MAX_RESOURCES], int
allocation[][MAX_RESOURCES], int need[][MAX_RESOURCES]) {
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
```

```
        }
    }
}

bool isSafeState(int processes, int resources, int available[], int max[][MAX_RESOURCES], int
allocation[][MAX_RESOURCES], int need[][MAX_RESOURCES], int safeSequence[]) {
    int work[MAX_RESOURCES];
    bool finish[MAX_PROCESSES] = {false};
    int count = 0;

    // Initialize work to available resources
    for (int i = 0; i < resources; i++) {
        work[i] = available[i];
    }

    // Finding an index i such that finish[i] is false and need[i] <= work
    while (count < processes) {
        bool found = false;
        for (int i = 0; i < processes; i++) {
            if (!finish[i]) {
                bool canAllocate = true;
                for (int j = 0; j < resources; j++) {
                    if (need[i][j] > work[j]) {
                        canAllocate = false;
                        break;
                    }
                }

                if (canAllocate) {
                    // Simulate allocation to process i
                    for (int j = 0; j < resources; j++) {
                        work[j] += allocation[i][j];
                    }
                    finish[i] = true;
                    safeSequence[count++] = i;
                    found = true;
                    printf("P%d is visited (", i);
                    for (int j = 0; j < resources; j++) {
                        printf("%d ", work[j]);
                    }
                    printf(")\n");
                }
            }
        }
```

```
        if (!found) {
            // If no process could be allocated, check if all processes are finished
            for (int i = 0; i < processes; i++) {
                if (!finish[i]) {
                    return false;
                }
            }
            return true;
        }
    }
    return true;
}
```

OUTPUT:

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the available resources:
3 3 2
Enter the maximum resource matrix:
Enter details for P0
7 5 3
Enter details for P1
3 2 2
Enter details for P2
9 0 2
Enter details for P3
2 2 2
Enter details for P4
4 3 3
Enter the allocation resource matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
P1 is visited (5 3 2 )
P3 is visited (7 4 3 )
P4 is visited (7 4 5 )
P0 is visited (7 5 5 )
P2 is visited (10 5 7 )
SYSTEM IS IN SAFE STATE
The Safe Sequence is -- (P1 P3 P4 P0 P2 )

Process Allocation              Max             Need
P0      0 1 0   7 5 3   7 4 3
P1      2 0 0   3 2 2   1 2 2
P2      3 0 2   9 0 2   6 0 0
P3      2 1 1   2 2 2   0 1 1
P4      0 0 2   4 3 3   4 3 1

Process returned 0 (0x0)   execution time : 51.391 s
Press any key to continue.
```