```
#include <stdio.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int main()
{
    int n, m, i, j;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

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

    int max_claim[MAX_PROCESSES][MAX_RESOURCES];
    int allocation[MAX_PROCESSES][MAX_RESOURCES];
    int need[MAX_PROCESSES][MAX_RESOURCES];
    int available[MAX_RESOURCES];

    // Input the maximum claim for each process
    printf("Enter the maximum claim matrix:\n");
    for (i = 0; i < n; i++)
    {
        printf("Process %d: ", i);
        for (j = 0; j < m; j++)
        {
            scanf("%d", &max_claim[i][j]);
        }
    }

    // Input the allocation matrix
    printf("Enter the allocation matrix:\n");
    for (i = 0; i < n; i++)
    {
        printf("Process %d: ", i);
        for (j = 0; j < m; j++)
        {
```

```
        scanf("%d", &allocation[i][j]);
        need[i][j] = max_claim[i][j] - allocation[i][j];
    }
}

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

// Initialize the finish array to false
int finish[MAX_PROCESSES];
for (i = 0; i < n; i++)
{
    finish[i] = 0;
}

// Safety algorithm
int work[MAX_RESOURCES];
for (i = 0; i < m; i++)
{
    work[i] = available[i];
}

int safe_sequence[MAX_PROCESSES];
int safe_count = 0;
while (safe_count < n)
{
    int found = 0;
    for (i = 0; i < n; i++)
    {
        if (finish[i] == 0)
        {
            int can_allocate = 1;
            for (j = 0; j < m; j++)
            {
                if (need[i][j] > work[j])
                {
                    can_allocate = 0;
                    break;
                }
            }

            if (can_allocate)
            {
                for (j = 0; j < m; j++)
                {
                    work[j] += allocation[i][j];
                }
                safe_sequence[safe_count++] = i;
                finish[i] = 1;
                found = 1;
            }
        }
    }

    if (!found)
    {
        printf("System is in an unsafe state. Deadlock detected.\n");
        break;
    }
}

if (safe_count == n)
{
    printf("System is in a safe state. Safe sequence: ");
    for (i = 0; i < n; i++)
    {
        printf("P%d ", safe_sequence[i]);
    }
    printf("\n");
```

```
        // Print the process detail table
        printf("\nProcess Detail Table:\n");
        printf("Process\tMax\tAllocation\tNeed\n");
        for (i = 0; i < n; i++)
        {
            printf("P%d\t", i);
            for (j = 0; j < m; j++)
            {
                printf("%d ", max_claim[i][j]);
            }
            printf("\t");
            for (j = 0; j < m; j++)
            {
                printf("%d ", allocation[i][j]);
            }
            printf("\t");
            for (j = 0; j < m; j++)
            {
                printf("%d ", need[i][j]);
            }
            printf("\n");
        }
    }

    return 0;
}
```

OUTPUT:-

```
saba00@ubuntu:~/TE-OSY/OS-Banker's Algorithm$ ./a.out
Enter the number of processes: 5
Enter the number of resources: 3
Enter the maximum claim matrix:
Process 0: 5 4 4
Process 1: 4 3 3
Process 2: 9 1 3
Process 3: 8 6 4
Process 4: 2 2 3
```

```
Enter the allocation matrix:
Process 0: 1 1 2
Process 1: 2 1 2
Process 2: 3 0 1
Process 3: 0 2 0
Process 4: 1 1 2
Enter the available resources: 3 2 1
System is in a safe state. Safe sequence: P1 P4 P0 P2 P3

Process Detail Table:
Process     Max   Allocation    Need
P0      5 4 4   1 1 2   4 3 2
P1      4 3 3   2 1 2   2 2 1
P2      9 1 3   3 0 1   6 1 2
P3      8 6 4   0 2 0   8 4 4
P4      2 2 3   1 1 2   1 1 1
```