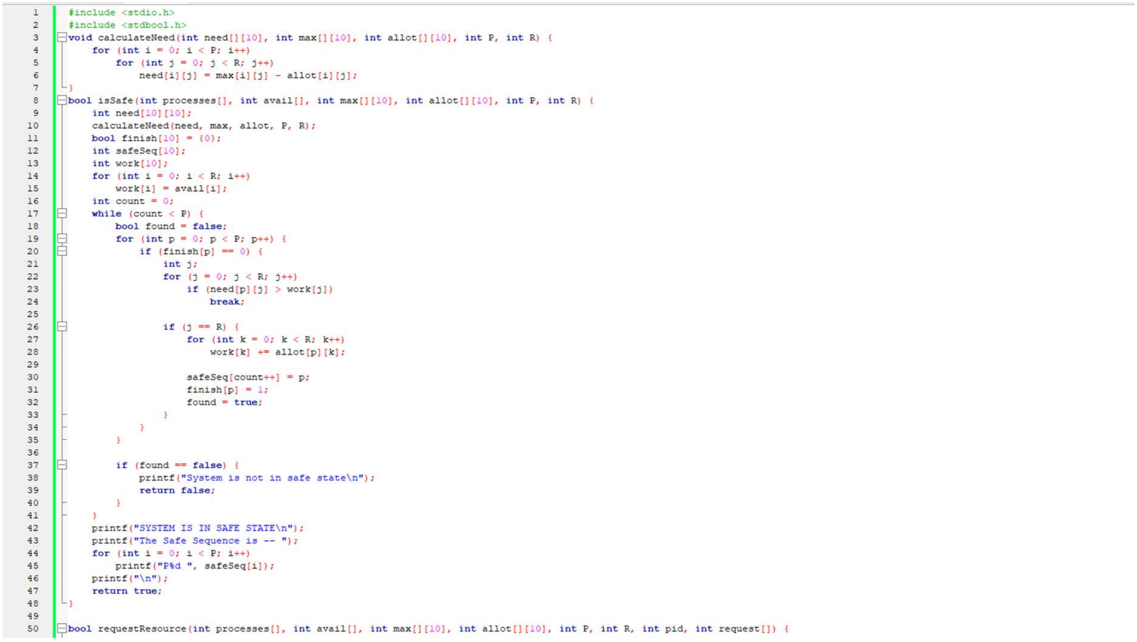
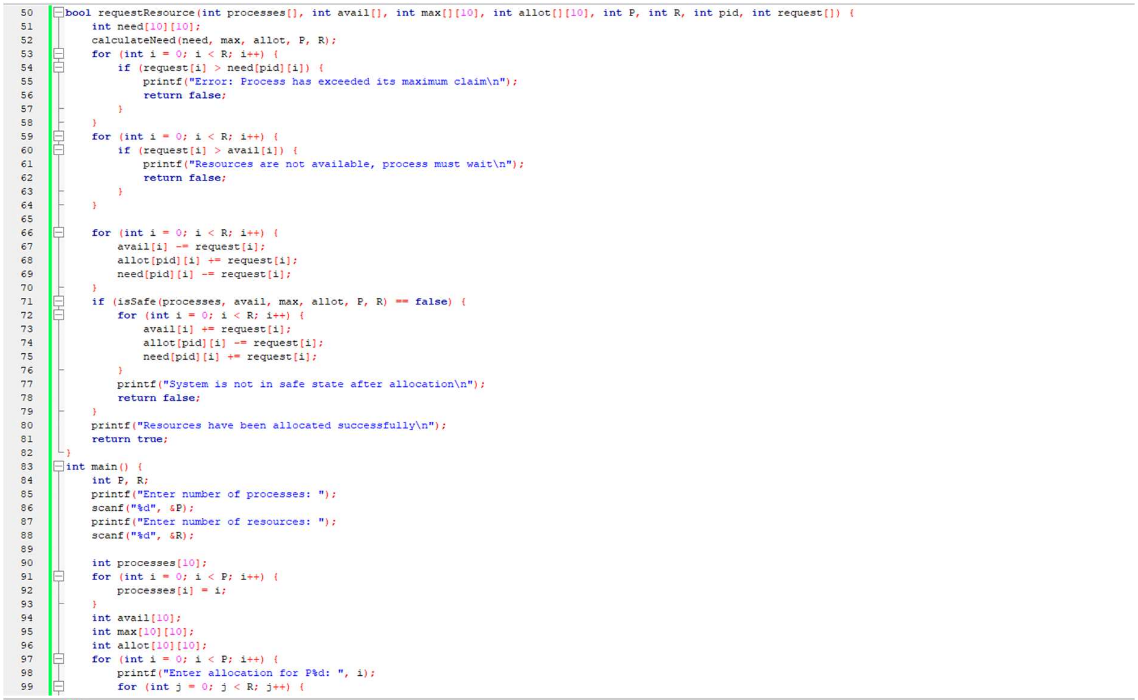
Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

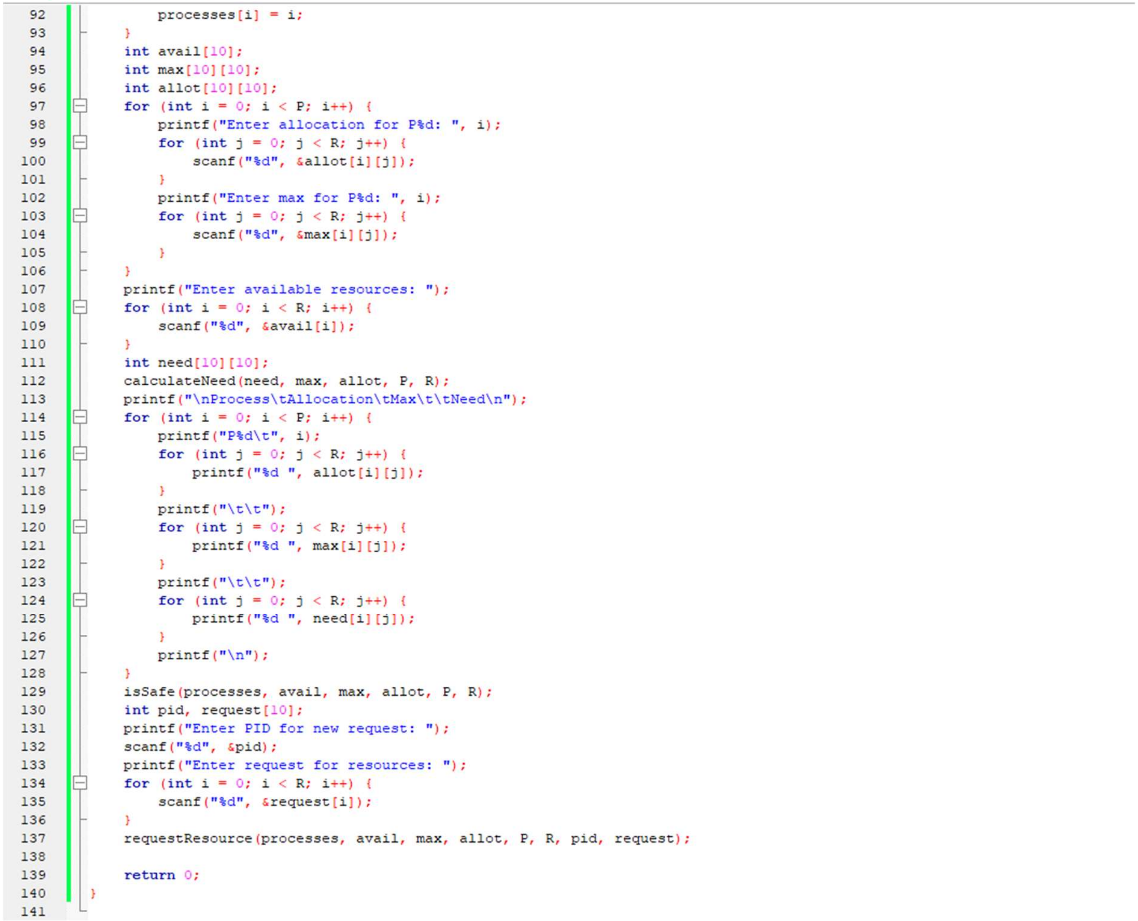
INPUT:



```c
#include <stdio.h>
#include <stdbool.h>
void calculateNeed(int need[][10], int max[][10], int allot[][10], int P, int R) {
    for (int i = 0; i < P; i++)
        for (int j = 0; j < R; j++)
            need[i][j] = max[i][j] - allot[i][j];
}
bool isSafe(int processes[], int avail[], int max[][10], int allot[][10], int P, int R) {
    int need[10][10];
    calculateNeed(need, max, allot, P, R);
    bool finish[10] = {0};
    int safeSeq[10];
    int work[10];
    for (int i = 0; i < R; i++)
        work[i] = avail[i];
    int count = 0;
    while (count < P) {
        bool found = false;
        for (int p = 0; p < P; p++) {
            if (finish[p] == 0) {
                int j;
                for (j = 0; j < R; j++)
                    if (need[p][j] > work[j])
                        break;

                if (j == R) {
                    for (int k = 0; k < R; k++)
                        work[k] += allot[p][k];

                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = true;
                }
            }
        }

        if (found == false) {
            printf("System is not in safe state\n");
            return false;
        }
    }
    printf("SYSTEM IS IN SAFE STATE\n");
    printf("The Safe Sequence is -- ");
    for (int i = 0; i < P; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");
    return true;
}

bool requestResource(int processes[], int avail[], int max[][10], int allot[][10], int P, int R, int pid, int request[]) {
```



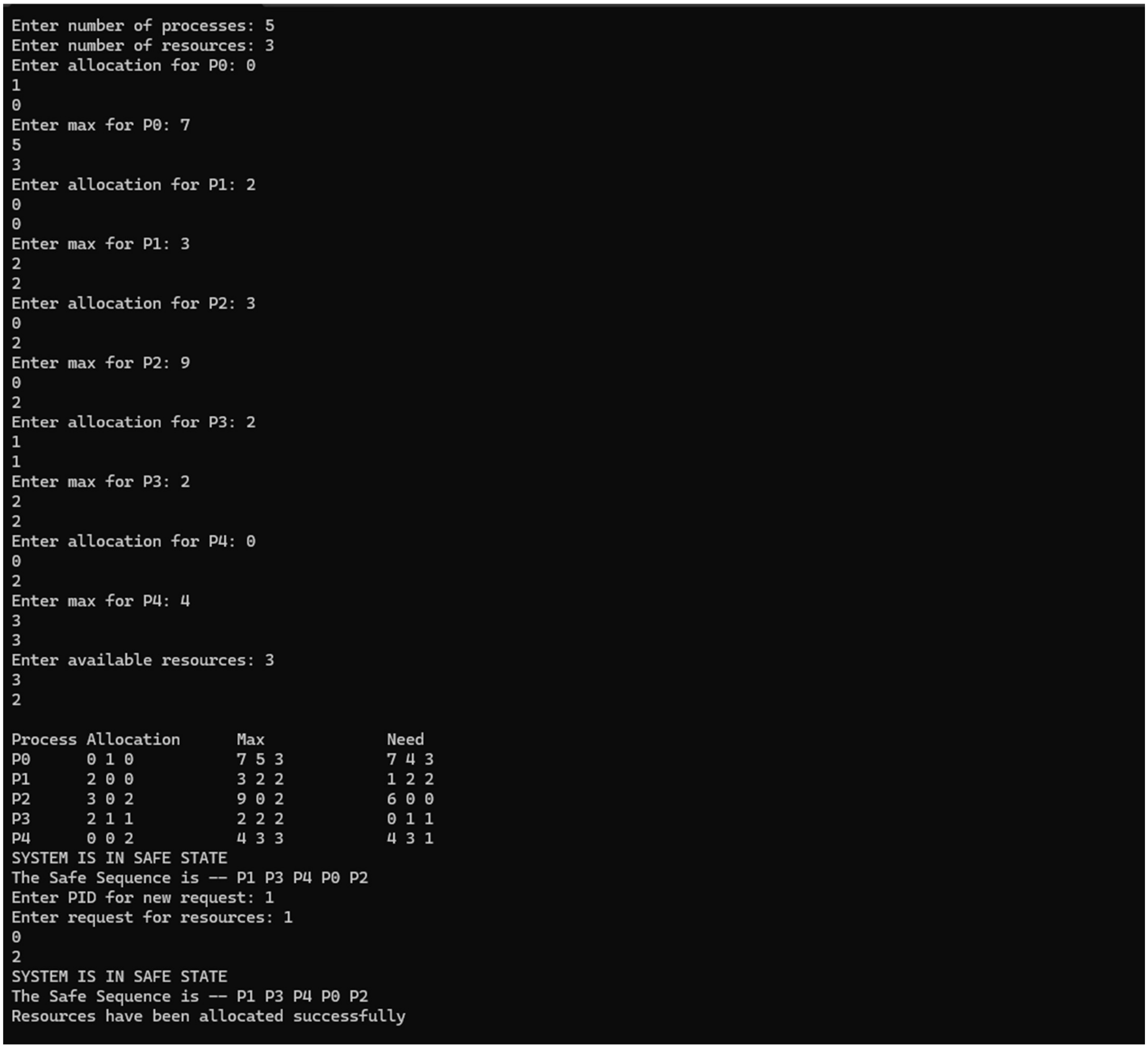
```c
bool requestResource(int processes[], int avail[], int max[][10], int allot[][10], int P, int R, int pid, int request[]) {
    int need[10][10];
    calculateNeed(need, max, allot, P, R);
    for (int i = 0; i < R; i++) {
        if (request[i] > need[pid][i]) {
            printf("Error: Process has exceeded its maximum claim\n");
            return false;
        }
    }
    for (int i = 0; i < R; i++) {
        if (request[i] > avail[i]) {
            printf("Resources are not available, process must wait\n");
            return false;
        }
    }

    for (int i = 0; i < R; i++) {
        avail[i] -= request[i];
        allot[pid][i] += request[i];
        need[pid][i] -= request[i];
    }
    if (isSafe(processes, avail, max, allot, P, R) == false) {
        for (int i = 0; i < R; i++) {
            avail[i] += request[i];
            allot[pid][i] -= request[i];
            need[pid][i] += request[i];
        }
        printf("System is not in safe state after allocation\n");
        return false;
    }
    printf("Resources have been allocated successfully\n");
    return true;
}
int main() {
    int P, R;
    printf("Enter number of processes: ");
    scanf("%d", &P);
    printf("Enter number of resources: ");
    scanf("%d", &R);

    int processes[10];
    for (int i = 0; i < P; i++) {
        processes[i] = i;
    }
    int avail[10];
    int max[10][10];
    int allot[10][10];
    for (int i = 0; i < P; i++) {
        printf("Enter allocation for P%d: ", i);
        for (int j = 0; j < R; j++) {
```



```c
        processes[i] = i;
    }
    int avail[10];
    int max[10][10];
    int allot[10][10];
    for (int i = 0; i < P; i++) {
        printf("Enter allocation for P%d: ", i);
        for (int j = 0; j < R; j++) {
```

```c
            scanf("%d", &allot[i][j]);
        }
        printf("Enter max for P%d: ", i);
        for (int j = 0; j < R; j++) {
            scanf("%d", &max[i][j]);
        }
    }
    printf("Enter available resources: ");
    for (int i = 0; i < R; i++) {
        scanf("%d", &avail[i]);
    }
    int need[10][10];
    calculateNeed(need, max, allot, P, R);
    printf("\nProcess\tAllocation\tMax\t\tNeed\n");
    for (int i = 0; i < P; i++) {
        printf("P%d\t", i);
        for (int j = 0; j < R; j++) {
            printf("%d ", allot[i][j]);
        }
        printf("\t\t");
        for (int j = 0; j < R; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\t\t");
        for (int j = 0; j < R; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
    isSafe(processes, avail, max, allot, P, R);
    int pid, request[10];
    printf("Enter PID for new request: ");
    scanf("%d", &pid);
    printf("Enter request for resources: ");
    for (int i = 0; i < R; i++) {
        scanf("%d", &request[i]);
    }
    requestResource(processes, avail, max, allot, P, R, pid, request);

    return 0;
}
```

OUTPUT:



```
Enter number of processes: 5
Enter number of resources: 3
Enter allocation for P0: 0
1
0
Enter max for P0: 7
5
3
Enter allocation for P1: 2
0
0
Enter max for P1: 3
2
2
Enter allocation for P2: 3
0
2
Enter max for P2: 9
0
2
Enter allocation for P3: 2
1
1
Enter max for P3: 2
2
2
Enter allocation for P4: 0
0
2
Enter max for P4: 4
3
3
Enter available resources: 3
3
2

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
SYSTEM IS IN SAFE STATE
The Safe Sequence is -- P1 P3 P4 P0 P2
Enter PID for new request: 1
Enter request for resources: 1
0
2
SYSTEM IS IN SAFE STATE
The Safe Sequence is -- P1 P3 P4 P0 P2
Resources have been allocated successfully
```