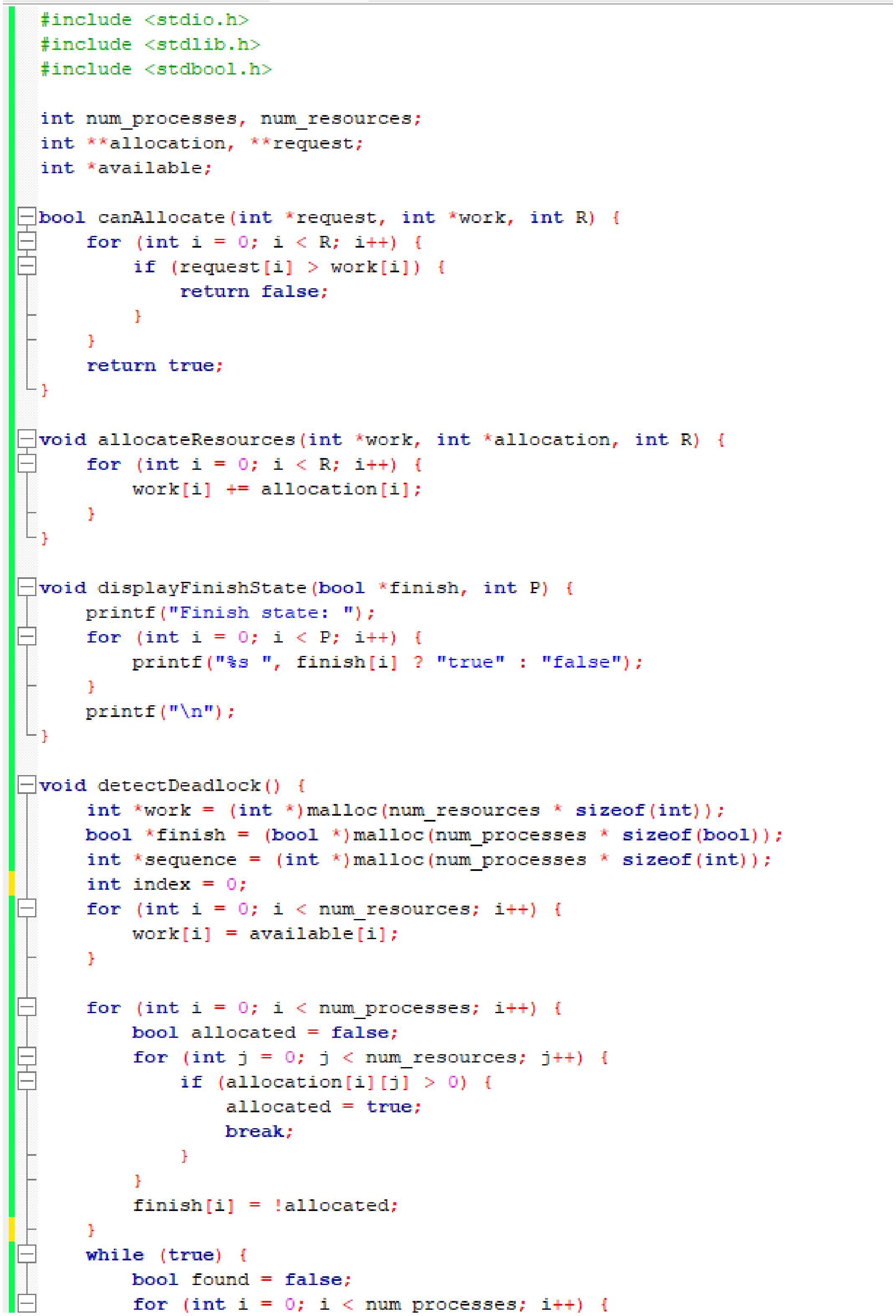
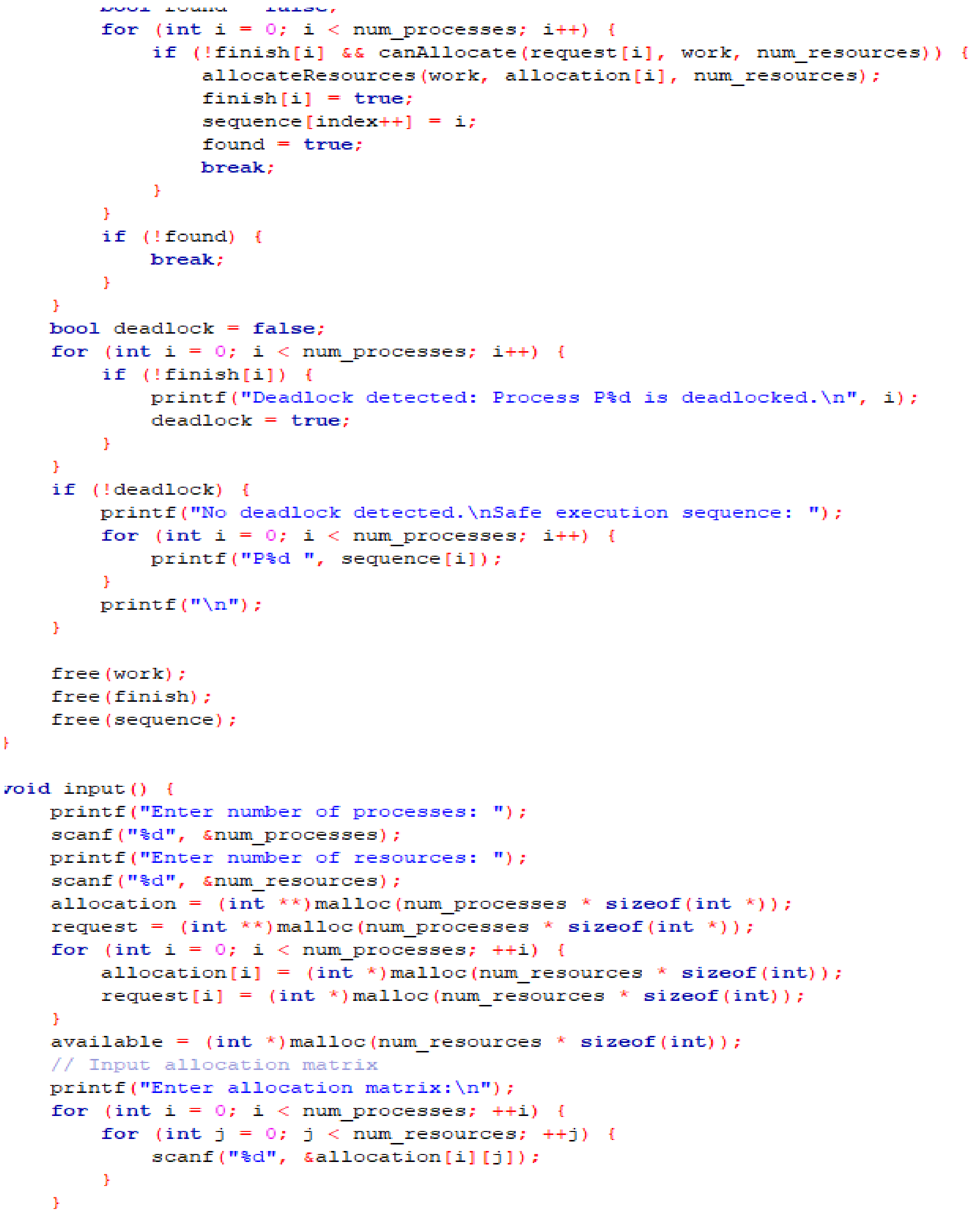
LAB PROGRAM 8

Write a program to simulate deadlock detection.

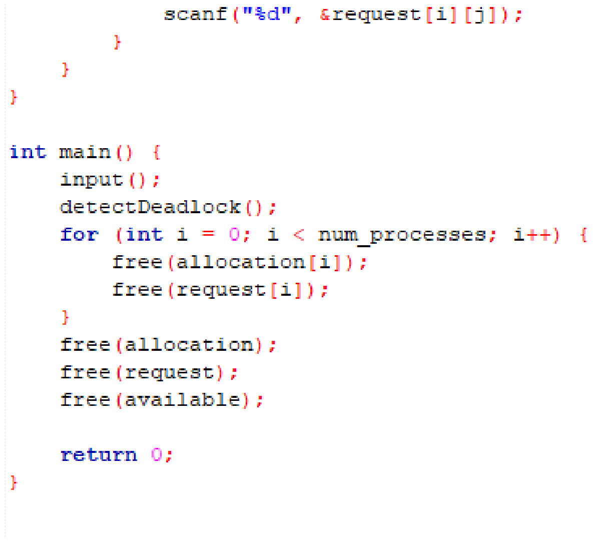
INPUT



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

int num_processes, num_resources;
int **allocation, **request;
int *available;

bool canAllocate(int *request, int *work, int R) {
    for (int i = 0; i < R; i++) {
        if (request[i] > work[i]) {
            return false;
        }
    }
    return true;
}

void allocateResources(int *work, int *allocation, int R) {
    for (int i = 0; i < R; i++) {
        work[i] += allocation[i];
    }
}

void displayFinishState(bool *finish, int P) {
    printf("Finish state: ");
    for (int i = 0; i < P; i++) {
        printf("%s ", finish[i] ? "true" : "false");
    }
    printf("\n");
}

void detectDeadlock() {
    int *work = (int *)malloc(num_resources * sizeof(int));
    bool *finish = (bool *)malloc(num_processes * sizeof(bool));
    int *sequence = (int *)malloc(num_processes * sizeof(int));
    int index = 0;
    for (int i = 0; i < num_resources; i++) {
        work[i] = available[i];
    }

    for (int i = 0; i < num_processes; i++) {
        bool allocated = false;
        for (int j = 0; j < num_resources; j++) {
            if (allocation[i][j] > 0) {
                allocated = true;
                break;
            }
        }
        finish[i] = !allocated;
    }
    while (true) {
        bool found = false;
        for (int i = 0; i < num_processes; i++) {
```

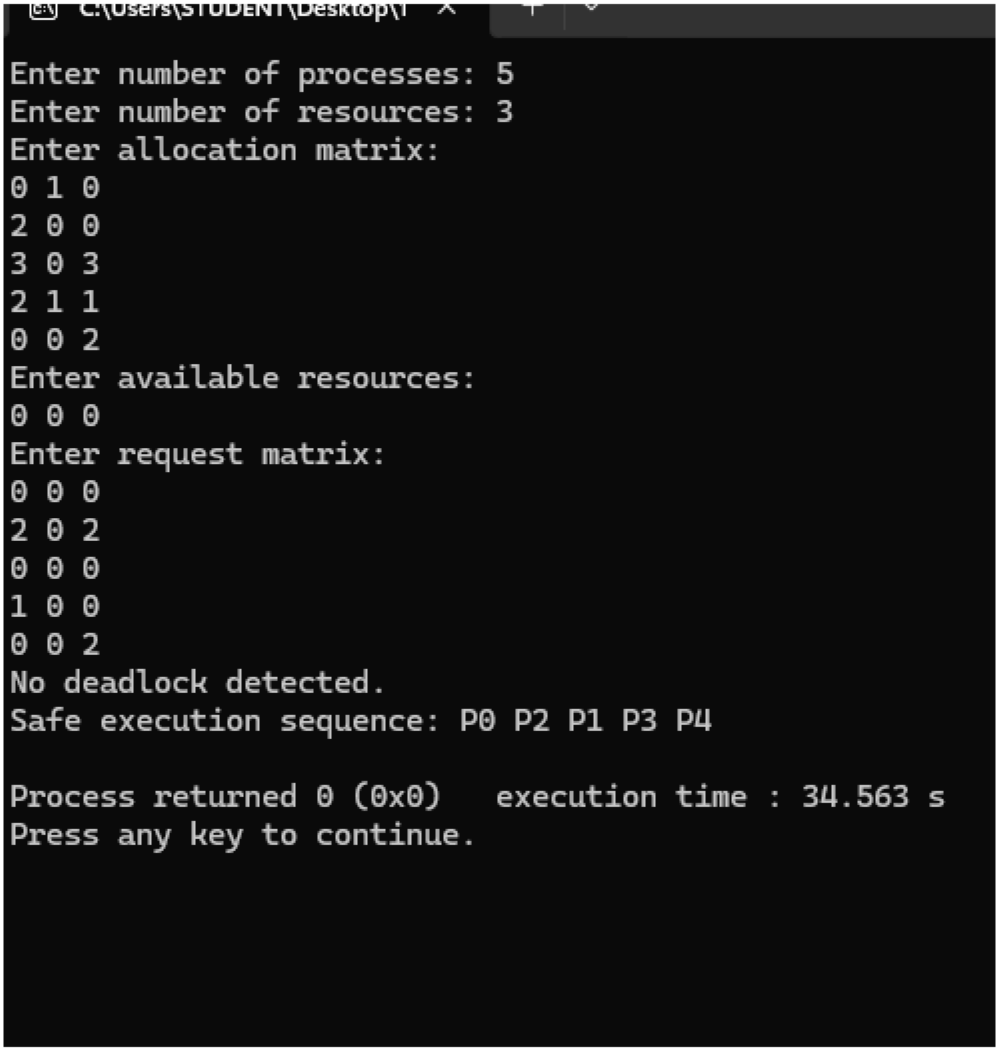


```
        bool found = false;
        for (int i = 0; i < num_processes; i++) {
            if (!finish[i] && canAllocate(request[i], work, num_resources)) {
                allocateResources(work, allocation[i], num_resources);
                finish[i] = true;
                sequence[index++] = i;
                found = true;
                break;
            }
        }
        if (!found) {
            break;
        }
    }
    bool deadlock = false;
    for (int i = 0; i < num_processes; i++) {
        if (!finish[i]) {
            printf("Deadlock detected: Process P%d is deadlocked.\n", i);
            deadlock = true;
        }
    }
    if (!deadlock) {
        printf("No deadlock detected.\nSafe execution sequence: ");
        for (int i = 0; i < num_processes; i++) {
            printf("P%d ", sequence[i]);
        }
        printf("\n");
    }

    free(work);
    free(finish);
    free(sequence);
}

void input() {
    printf("Enter number of processes: ");
    scanf("%d", &num_processes);
    printf("Enter number of resources: ");
    scanf("%d", &num_resources);
    allocation = (int **)malloc(num_processes * sizeof(int *));
    request = (int **)malloc(num_processes * sizeof(int *));
    for (int i = 0; i < num_processes; ++i) {
        allocation[i] = (int *)malloc(num_resources * sizeof(int));
        request[i] = (int *)malloc(num_resources * sizeof(int));
    }
    available = (int *)malloc(num_resources * sizeof(int));
    // Input allocation matrix
    printf("Enter allocation matrix:\n");
    for (int i = 0; i < num_processes; ++i) {
        for (int j = 0; j < num_resources; ++j) {
            scanf("%d", &allocation[i][j]);
        }
    }
```



```
            scanf("%d", &request[i][j]);
        }
    }
}

int main() {
    input();
    detectDeadlock();
    for (int i = 0; i < num_processes; i++) {
        free(allocation[i]);
        free(request[i]);
    }
    free(allocation);
    free(request);
    free(available);

    return 0;
}
```

OUTPUT



```
Enter number of processes: 5
Enter number of resources: 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter available resources:
0 0 0
Enter request matrix:
0 0 0
2 0 2
0 0 0
1 0 0
0 0 2
No deadlock detected.
Safe execution sequence: P0 P2 P1 P3 P4

Process returned 0 (0x0)   execution time : 34.563 s
Press any key to continue.
```