Bankers algorithm

Code

```c
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define N 10 // max number of resources and max number of processes are 10



int total_resources[N];

int available_resources[N];



struct process {

    int pid;

    int max[N];

    int alloc[N];

    int need[N];

};



bool can_execute(struct process *p, int num_resources) {

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

        if (p->need[i] > available_resources[i]) {

            return false;

        }

    }

    return true;

}
```

```c
void is_safe_sequence(struct process total_process[], int num_processes, int num_resources) {
    int safe_sequence[N];
    int completed_count = 0;
    bool process_executed[N] = {0};

    while (completed_count < num_processes) {
        bool progress_made = false;

        for (int i = 0; i < num_processes; i++) {
            if (!process_executed[i] && can_execute(&total_process[i], num_resources)) {
                for (int j = 0; j < num_resources; j++) {
                    available_resources[j] += total_process[i].alloc[j];
                }
                safe_sequence[completed_count++] = total_process[i].pid;
                process_executed[i] = true;
                progress_made = true;
            }
        }

        if (!progress_made) {
            printf("System is in an unsafe state. No safe sequence found.\n");
            return;
        }
    }

    printf("System is in a safe state.\nSafe sequence: ");
    for (int i = 0; i < completed_count; i++) {
```

```
            printf("P%d ", safe_sequence[i]);
        }
        printf("\n");
    }

int main() {
    struct process total_process[N];
    int num_processes, num_resources;

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

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

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

    printf("Start entering details of processes: \n");
    for (int i = 0; i < num_processes; i++) {
        printf("Enter process ID of process %d: ", i + 1);
        scanf("%d", &total_process[i].pid);

        printf("Enter maximum resources required by process %d: ", i + 1);
        for (int j = 0; j < num_resources; j++) {
            scanf("%d", &total_process[i].max[j]);
```

```
        }

        printf("Enter allocated resources of process %d: ", i + 1);
        for (int j = 0; j < num_resources; j++) {
            scanf("%d", &total_process[i].alloc[j]);
            available_resources[j] -= total_process[i].alloc[j];
        }
    }

    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            total_process[i].need[j] = total_process[i].max[j] - total_process[i].alloc[j];
        }
    }

    is_safe_sequence(total_process, num_processes, num_resources);

    return 0;
}
```


Input sequence

```
5 3 10 5 7 0 7 5 3 0 1 0 1 3 2 2 2 0 0 2 9 0 2 3 0 2 3 2 2 2 2 1 1 4 4 4 3 0 0 2
```

Output

```
Safe sequence: P1 P3 P4 P0 P2
```