Below is a C program that simulates the Banker's Algorithm. The program includes two main parts: the safety check algorithm and the resource allocation algorithm. It takes as input the number of processes, number of resources, the allocation matrix, the maximum demand matrix, and the available resources. The program then checks if the system is in a safe state and outputs the safe sequence of processes, if one exists.

C Program Implementation:

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

void calculateNeed(int need[][10], int max[][10], int alloc[][10], int np, int nr) {
    // Calculate the need matrix
    for (int i = 0; i < np; i++) {
        for (int j = 0; j < nr; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
}

bool isSafe(int processes[], int avail[], int max[][10], int alloc[][10], int np, int nr) {
    int need[10][10];
    calculateNeed(need, max, alloc, np, nr);
```

```
bool finish[np];
for (int i = 0; i < np; i++) {
    finish[i] = false;
}

int safeSeq[np];
int work[nr];
for (int i = 0; i < nr; i++) {
    work[i] = avail[i];
}

int count = 0;
while (count < np) {
    bool found = false;
    for (int p = 0; p < np; p++) {
        if (!finish[p]) {
            int j;
            for (j = 0; j < nr; j++) {
                if (need[p][j] > work[j])
                    break;
            }
            if (j == nr) {
                for (int k = 0; k < nr; k++)
                    work[k] += alloc[p][k];

                safeSeq[count++] = processes[p];
                finish[p] = true;
```

```
                    found = true;
                }
            }
        }
        if (!found) {
            printf("System is not in safe state.\n");
            return false;
        }
    }

    printf("System is in safe state.\nSafe sequence is: ");
    for (int i = 0; i < np; i++) {
        printf("%d ", safeSeq[i]);
    }
    printf("\n");
    return true;
}

void requestResources(int processes[], int avail[], int max[][10], int alloc[][10],
int np, int nr) {
    int process_num;
    int request[nr];

    printf("Enter the process number making the request (0-%d): ", np-1);
    scanf("%d", &process_num);

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

    for (int i = 0; i < nr; i++) {
        if (request[i] > avail[i]) {
            printf("Requested resources are greater than available resources.
Request cannot be granted.\n");
            return;
        }
    }

    for (int i = 0; i < nr; i++) {
        if (request[i] > max[process_num][i] - alloc[process_num][i]) {
            printf("Requested resources exceed maximum claim for process.
Request cannot be granted.\n");
            return;
        }
    }

    for (int i = 0; i < nr; i++) {
        avail[i] -= request[i];
        alloc[process_num][i] += request[i];
        max[process_num][i] -= request[i];
    }

    if (isSafe(processes, avail, max, alloc, np, nr)) {
        printf("Request can be granted. Resources have been allocated.\n");
    } else {
        printf("Request cannot be granted as it leads to unsafe state. Rolling back
the request.\n");
        for (int i = 0; i < nr; i++) {
            avail[i] += request[i];
            alloc[process_num][i] -= request[i];
            max[process_num][i] += request[i];
        }
    }
}

int main() {
    int np, nr;
    int processes[10], avail[10];
    int max[10][10], alloc[10][10];

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

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

    printf("Enter the allocation matrix:\n");
    for (int i = 0; i < np; i++) {
        processes[i] = i;
        for (int j = 0; j < nr; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
```

```
printf("Enter the maximum demand matrix:\n");
for (int i = 0; i < np; i++) {
    for (int j = 0; j < nr; j++) {
        scanf("%d", &max[i][j]);
    }
}

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

if (isSafe(processes, avail, max, alloc, np, nr)) {
    printf("System is in a safe state.\n");
} else {
    printf("System is not in a safe state.\n");
}

char ch;
printf("Do you want to request resources for a process? (y/n): ");
scanf(" %c", &ch);

if (ch == 'y' || ch == 'Y') {
    requestResources(processes, avail, max, alloc, np, nr);
}

return 0;
```

}
```

### How the Program Works:

1. **Process Structure**:
   - The program first defines the necessary matrices and variables: `processes` (to hold process IDs), `avail` (to store the number of available resources), `max` (to store the maximum demand matrix), and `alloc` (to store the allocation matrix).
   - The `need` matrix is calculated using the `calculateNeed` function.

2. **Safety Check Algorithm**:
   - The `isSafe()` function checks if the system is in a safe state by simulating the allocation of resources to processes and determining if all processes can finish without entering a deadlock. It returns true if the system is in a safe state and prints the safe sequence of processes.

3. **Resource Allocation Algorithm**:
   - The `requestResources()` function handles resource requests from processes. It first checks if the requested resources can be allocated without exceeding the available resources or the maximum demand for that process. If the request can be granted without leading to an unsafe state, the resources are allocated. Otherwise, the request is rolled back.

4. **Main Function**:
   - The program first takes input for the number of processes, number of resources, the allocation matrix, the maximum demand matrix, and the
```

available resources.
   - It then checks if the system is initially in a safe state.
   - Finally, the user is given the option to make a resource request for a process.

Example Scenario:

If you input the following:

- Number of processes: 5
- Number of resources: 3
- Allocation Matrix:
  ```
  0 1 0
  2 0 0
  3 0 2
  2 1 1
  0 0 2
  ```
- Maximum Demand Matrix:
  ```
  7 5 3
  3 2 2
  9 0 2
  2 2 2
  4 3 3
  ```
- Available Resources:
  ```
  ```

```
  3 3 2
```

The program will check if the system is in a safe state and output the safe sequence, if one exists. It will then allow you to request resources for any process and will check if the request can be granted.

Compilation and Execution:

To compile and run this program, save it in a file (e.g., `bankers_algorithm.c`), and then use the following commands in a terminal:

```bash
gcc bankers_algorithm.c -o bankers_algorithm
./bankers_algorithm
```

This program effectively simulates the Banker's Algorithm, showing whether the system is in a safe state and preventing unsafe resource allocations.