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

#define MAX 10

int n, m; // n = number of processes, m = number of resources
int allocation[MAX][MAX], max[MAX][MAX], need[MAX][MAX], available[MAX];

void input() {
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter the number of resources: ");
   scanf("%d", &m);

   printf("Enter the Allocation Matrix:\n");
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         scanf("%d", &allocation[i][j]);
      }
   }

   printf("Enter the Max Matrix:\n");
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         scanf("%d", &max[i][j]);
      }
   }
```

```
    printf("Enter the Available Resources:\n");
    for (int i = 0; i < m; i++) {
        scanf("%d", &available[i]);
    }

    // Calculate the Need Matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

// Function to check if the system is in a safe state using the Safety Algorithm
int is_safe() {
    int work[m], finish[n];
    for (int i = 0; i < m; i++) {
        work[i] = available[i];
    }

    for (int i = 0; i < n; i++) {
        finish[i] = 0; // mark all processes as not finished
    }

    int safe_sequence[n], count = 0;

    while (count < n) {
        int found = 0;
```

```
    for (int p = 0; p < n; p++) {
        if (finish[p] == 0) {
            int j;
            for (j = 0; j < m; j++) {
                if (need[p][j] > work[j]) {
                    break;
                }
            }

            if (j == m) { // If all the needs of process p can be satisfied
                for (int k = 0; k < m; k++) {
                    work[k] += allocation[p][k];
                }
                finish[p] = 1; // Process p is finished
                safe_sequence[count++] = p;
                found = 1;
                break;
            }
        }
    }

    if (found == 0) {
        return 0; // The system is not in a safe state
    }
}

printf("Safe sequence: ");
for (int i = 0; i < n; i++) {
```

```
        printf("P%d ", safe_sequence[i]);
    }
    printf("\n");
    return 1; // The system is in a safe state
}

// Function to check if a resource request can be granted using the Resource Request Algorithm
int request_resources(int process_num, int request[]) {
    for (int i = 0; i < m; i++) {
        if (request[i] > need[process_num][i]) {
            printf("Error: Process has exceeded its maximum claim.\n");
            return 0;
        }
    }

    for (int i = 0; i < m; i++) {
        if (request[i] > available[i]) {
            printf("Error: Resources are not available.\n");
            return 0;
        }
    }

    // Temporarily allocate resources and check if the system is in a safe state
    for (int i = 0; i < m; i++) {
        available[i] -= request[i];
        allocation[process_num][i] += request[i];
        need[process_num][i] -= request[i];
    }
```

```
    if (is_safe()) {
        printf("Resources allocated successfully.\n");
        return 1;
    } else {
        // Rollback if not safe
        for (int i = 0; i < m; i++) {
            available[i] += request[i];
            allocation[process_num][i] -= request[i];
            need[process_num][i] += request[i];
        }
        printf("Resources cannot be allocated as it leads to an unsafe state.\n");
        return 0;
    }
}

int main() {
    int choice, process_num, request[m];

    input();

    do {
        printf("\nBanker's Algorithm Menu:\n");
        printf("1. Safety Algorithm\n");
        printf("2. Resource Request Algorithm\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
        case 1:
            if (is_safe()) {
                printf("The system is in a safe state.\n");
            } else {
                printf("The system is not in a safe state.\n");
            }
            break;
        case 2:
            printf("Enter the process number (0 to %d): ", n - 1);
            scanf("%d", &process_num);

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

            if (request_resources(process_num, request)) {
                printf("Request granted.\n");
            } else {
                printf("Request denied.\n");
            }
            break;
        case 3:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
} while (choice != 3);
```

```
    return 0;
}
```