PROGRAM CODE

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

struct proc {
	int* max;
	int* alloc;
	int* need;
	int finish;
};

void display(struct proc p[], int avail[], int m, int n) {
	int i, j;

	printf("SNAPSHOT\n\nPROCESS ID");

	printf("\tMAX (");
	for(j=0; j<m; j++)
		printf(" %c ", j+65);
	printf(")");

	printf("\t\tALLOCATION (");
	for(j=0; j<m; j++)
		printf(" %c ", j+65);
	printf(")");

	printf("\t\tNEED (");
	for(j=0; j<m; j++)
		printf(" %c ", j+65);
	printf(")\n");

	for(i=0; i<n; i++) {
		printf("%d\t\t", i+1);

		for(j=0; j<m; j++)
			printf("%d ",p[i].max[j]);

		printf("\t\t\t");
		for(j=0; j<m; j++)
			printf("%d ",p[i].alloc[j]);

		printf("\t\t\t\t");
		for(j=0; j<m; j++)
			printf("%d ",p[i].need[j]);

		printf("\n");
	}

	printf("\nAVAILABLE (");
	for(j=0; j<m; j++)
		printf(" %c ", j+65);
	printf(")\n");
	for(j=0; j<m; j++)
		printf("%d ", avail[j]);
	printf("\n");
}

int safety(struct proc p[], int avail[], int m, int n) {
	int i, j, safe[n], index = 0, count = 0;

	printf("\n");
```

```
    display(p, avail, m, n);

    int temp[m];
    for(j=0; j<m; j++)
        temp[j] = avail[j];

    for(i=0; ; i = (i+1)%n) {
        if(p[i].finish == 0) {
            for(j=0; j<m; j++) {
                if(p[i].need[j] > avail[j]) {
                    count++;
                    break;
                }
            }

            if(j == m){
                count = 0;

                for(j=0; j<m; j++)
                    avail[j] += p[i].alloc[j];

                p[i].finish = 1;

                safe[index] = i+1;
                index++;

                if(index == n) {
                    printf("\nThe system is safe and the safe sequence is <");
                    for(i=0; i<n; i++)
                        printf(" P%d ", safe[i]);
                    printf(">\n");

                    for(j=0; j<m; j++)
                        avail[j] = temp[j];
                    for(i=0; i<n; i++)
                        p[i].finish = 0;

                    return 1;
                }
            }
        } else {
            count++;
        }

        if(count == n) {
            printf("\nDEADLOCK! The system is unsafe!\n");

            for(j=0; j<m; j++)
                avail[j] = temp[j];
            for(i=0; i<n; i++)
                p[i].finish = 0;

            return 0;
        }
    }
}

void request(struct proc p[], int avail[], int m, int n) {
    int j, num, req[m];

    printf("\nEnter process ID of request: ");
    scanf("%d", &num);
    num--;
    printf("\nEnter the REQUEST (");
    for(j=0; j<m; j++)
        printf(" %c ", j+65);
    printf(") : ");

    for(j=0; j<m; j++)
        scanf("%d", &req[j]);

    for(j=0; j<m; j++) {
        if(req[j] > p[num].need[j]) {
            printf("\nP%d's request has exceeded its maximum claim!\nHence, the request cannot be granted immediately!\n", num+1);
            return;
        }
    }

    for(j=0; j<m; j++) {
        if(req[j] > avail[j]) {
            printf("\nResources requested by P%d are not available!\nHence, the request cannot be granted immediately!\n", num+1);
            return;
        }
    }

    int temp_max[m];
    int temp_alloc[m];
    int temp_need[m];
    int temp[m];

    for(j=0; j<m; j++) {
        temp[j] = avail[j];
        temp_max[j] = p[num].max[j];
        temp_alloc[j] = p[num].alloc[j];
        temp_need[j] = p[num].need[j];
    }

    for(j=0; j<m; j++) {
        avail[j] -= req[j];
        p[num].alloc[j] += req[j];
        p[num].need[j] -= req[j];
    }

    if(safety(p, avail, m, n))
        printf("Hence, the request can be granted immediately.\n");
    else
        printf("Hence, the request cannot be granted immediately!\n");

    for(j=0; j<m; j++) {
        avail[j] = temp[j];
        p[num].max[j] = temp_max[j];
        p[num].alloc[j] = temp_alloc[j];
        p[num].need[j] = temp_need[j];
    }
}

void main() {
    int m, n, i, j, in;

    printf("\nBANKER'S ALGORITHM\n\nEnter number of resources: ");
    scanf("%d", &m);
    printf("\n");
    int avail[m];

    printf("Enter number of AVAILABLE instances of resources: ");
    for(j=0; j<m; j++)
        scanf("%d", &avail[j]);
    printf("\n");

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

    struct proc p[n];

    for(i=0; i<n; i++) {
        p[i].max = (int*) malloc(sizeof(int) * m);
        p[i].alloc = (int*) malloc(sizeof(int) * m);
        p[i].need = (int*) malloc(sizeof(int) * m);
        p[i].finish = 0;
    }

    for(i=0; i<n; i++) {
        printf("\nMAX of process %d: ", i+1);
        for(j=0; j<m; j++) {
            scanf("%d", &p[i].max[j]);
        }

        printf("ALLOCATION of process %d: ", i+1);
        for(j=0; j<m; j++) {
            scanf("%d", &p[i].alloc[j]);
        }

        for(j=0; j<m; j++)
            p[i].need[j] = p[i].max[j] - p[i].alloc[j];
    }

    while(1) {
        printf("\n1. Safety Algorithm\n2. Resource Request Algorithm\n3. Exit\nEnter your input: ");
        scanf("%d", &in);

        switch(in) {
            case 1:
                safety(p, avail, m, n);
                break;
            case 2:
                request(p, avail, m, n);
                break;
            case 3:
                printf("\nExit.\n\n");
                exit(0);
            default:
                printf("\nInvalid option!\n");
                break;
        }
    }
}
```

SAMPLE OUTPUT

```
BANKER'S ALGORITHM

Enter number of resources: 4

Enter number of AVAILABLE instances of resources: 3 3 2 1

Enter number of processes: 5

MAX of process 1: 4 2 1 2
ALLOCATION of process 1: 2 0 0 1

MAX of process 2: 5 2 5 2
ALLOCATION of process 2: 3 1 2 1

MAX of process 3: 2 3 1 6
ALLOCATION of process 3: 2 1 0 3

MAX of process 4: 1 4 2 4
ALLOCATION of process 4: 1 3 1 2

MAX of process 5: 3 6 6 5
ALLOCATION of process 5: 1 4 3 2

1. Safety Algorithm
2. Resource Request Algorithm
3. Exit
Enter your input: 1

SNAPSHOT

PROCESS ID     MAX ( A  B  C  D )             ALLOCATION ( A  B  C  D )              NEED ( A  B  C  D )
1              4 2 1 2                2 0 0 1                        2 2 1 1
2              5 2 5 2                3 1 2 1                        2 1 3 1
3              2 3 1 6                2 1 0 3                        0 2 1 3
4              1 4 2 4                1 3 1 2                        0 1 1 2
5              3 6 6 5                1 4 3 2                        2 2 3 3

AVAILABLE ( A  B  C  D )
3 3 2 1

The system is safe and the safe sequence is < P1  P4  P5  P2  P3 >

1. Safety Algorithm
2. Resource Request Algorithm
3. Exit
Enter your input: 2

Enter process ID of request: 2

Enter the REQUEST ( A  B  C  D ) : 1 1 0 0

SNAPSHOT

PROCESS ID     MAX ( A  B  C  D )             ALLOCATION ( A  B  C  D )              NEED ( A  B  C  D )
1              4 2 1 2                2 0 0 1                        2 2 1 1
2              5 2 5 2                4 2 2 1                        1 0 3 1
3              2 3 1 6                2 1 0 3                        0 2 1 3
4              1 4 2 4                1 3 1 2                        0 1 1 2
5              3 6 6 5                1 4 3 2                        2 2 3 3
```

```
AVAILABLE ( A  B  C  D )
2 2 2 1

The system is safe and the safe sequence is < P1  P4  P5  P2  P3 >
Hence, the request can be granted immediately.

1. Safety Algorithm
2. Resource Request Algorithm
3. Exit
Enter your input: 2

Enter process ID of request: 5

Enter the REQUEST ( A  B  C  D ) : 0 0 2 0

SNAPSHOT

PROCESS ID         MAX ( A  B  C  D )                     ALLOCATION ( A  B  C  D )                       NEED ( A  B  C  D )
1                  4 2 1 2                      2 0 0 1                                2 2 1 1
2                  5 2 5 2                      3 1 2 1                                2 1 3 1
3                  2 3 1 6                      2 1 0 3                                0 2 1 3
4                  1 4 2 4                      1 3 1 2                                0 1 1 2
5                  3 6 6 5                      1 4 5 2                                2 2 1 3

AVAILABLE ( A  B  C  D )
3 3 0 1

DEADLOCK! The system is unsafe!
Hence, the request cannot be granted immediately!

1. Safety Algorithm
2. Resource Request Algorithm
3. Exit
Enter your input: 3

Exit.
```