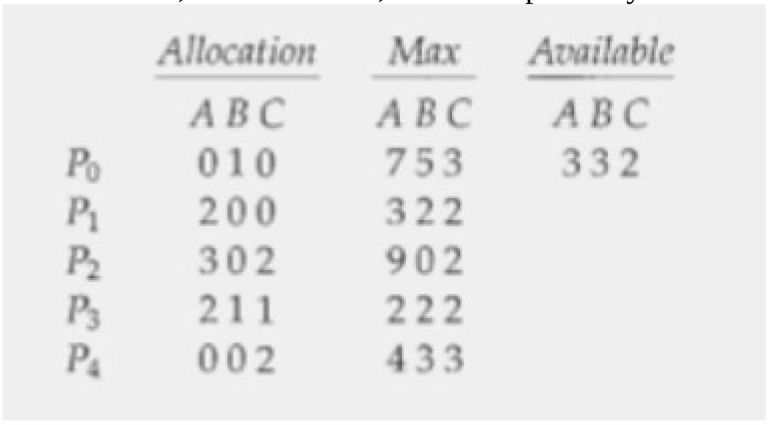
LAB 8

1.Consider the following snapshot of the system. Write C/C++ program to implement Banker's algorithm for deadlock avoidance. The program has to accept all inputs from the user. Assume the total number of instances of A,B and C are 10,5 and 7 respectively.



	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

(a) What is the content of the matrix Need?
(b) Is the system in a safe state?
(c) If a request from process P1 arrives for (1, 0, 2), can the request be granted immediately? Display the updated Allocation, Need and Available matrices.
(d) If a request from process P4 arrives for (3, 3, 0), can the request be granted immediately?
(e) If a request from process P0 arrives for (0, 2, 0), can the request be granted immediately?

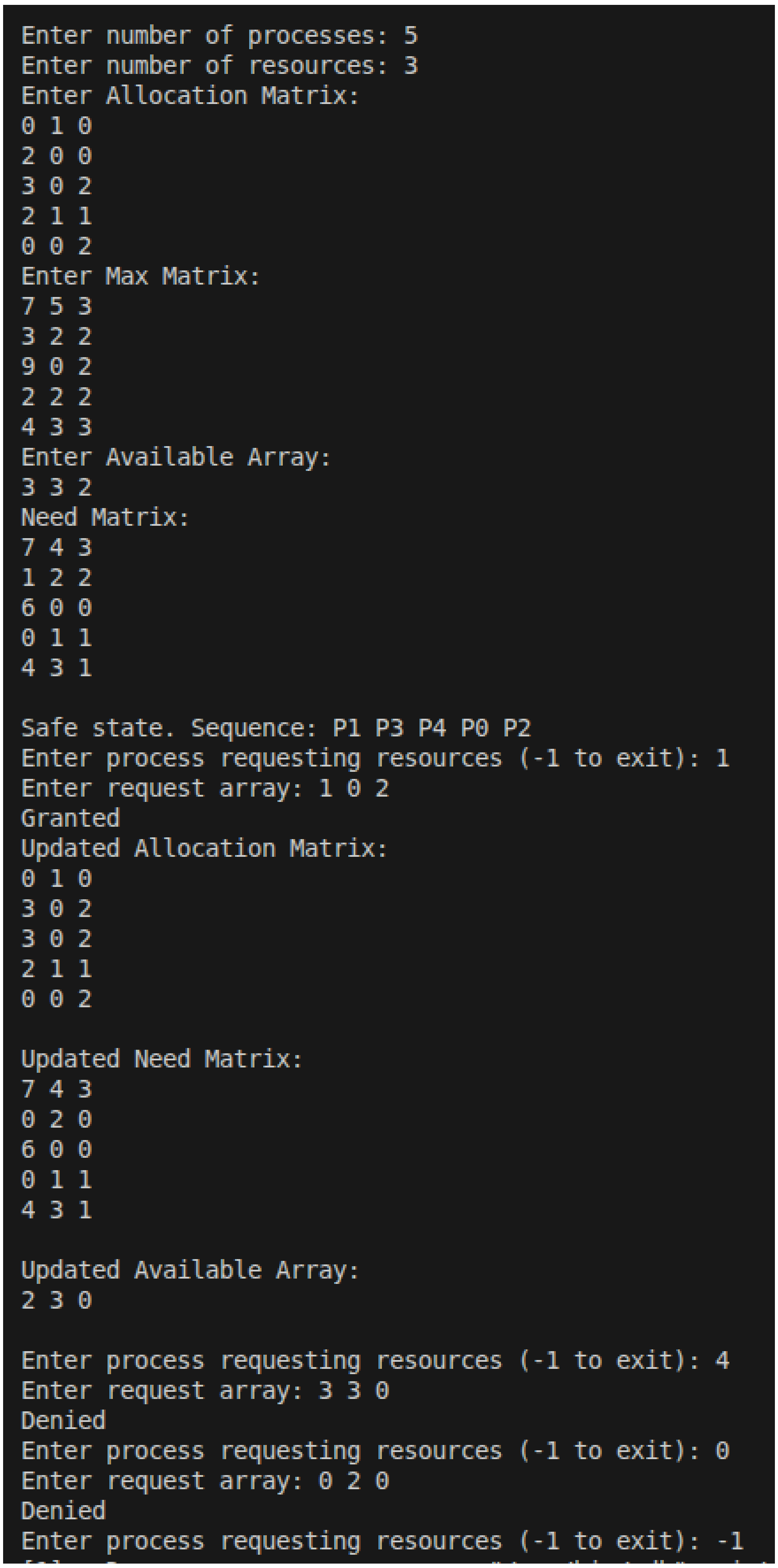
Code:

```
#include <stdio.h>

#define MAX 10

void calculateNeed(int p, int r, int need[MAX][MAX], int max[MAX][MAX], int alloc[MAX]
[MAX]) {
for (int i = 0; i < p; i++) {
for (int j = 0; j < r; j++) {
need[i][j] = max[i][j] - alloc[i][j];
}
}
}

#include <stdio.h>

#define MAX 10

void calculateNeed(int p, int r, int need[MAX][MAX], int max[MAX][MAX], int alloc[MAX]
[MAX]) {
for (int i = 0; i < p; i++) {
for (int j = 0; j < r; j++) {
void printMatrix(int p, int r, int matrix[MAX][MAX]) {
for (int i = 0; i < p; i++) {
for (int j = 0; j < r; j++) {
printf("%d ", matrix[i][j]);
```

```
}
printf("\n");
}
printf("\n");
}

int isSafe(int p, int r, int avail[MAX], int max[MAX][MAX], int alloc[MAX][MAX], int
need[MAX][MAX], int safeSeq[MAX]) {
int work[MAX];
int finish[MAX];
for (int i = 0; i < r; i++) work[i] = avail[i];
for (int i = 0; i < p; i++) finish[i] = 0;

int count = 0;
while (count < p) {
int found = 0;
for (int p_idx = 0; p_idx < p; p_idx++) {
if (finish[p_idx] == 0) {
int j;
for (j = 0; j < r; j++) {
if (need[p_idx][j] > work[j]) break;
}
if (j == r) {
for (int k = 0; k < r; k++) {
#include <stdio.h>

#define MAX 10

void calculateNeed(int p, int r, int need[MAX][MAX], int max[MAX][MAX], int alloc[MAX]
[MAX]) {
for (int i = 0; i < p; i++) {
for (int j = 0; j < r; j++) {
work[k] += alloc[p_idx][k];
}
safeSeq[count++] = p_idx;
finish[p_idx] = 1;
found = 1;
}
}
}
if (found == 0) return 0;
}
return 1;
}

int requestResource(int p, int r, int pid, int req[MAX], int avail[MAX], int max[MAX][MAX], int
alloc[MAX][MAX], int need[MAX][MAX]) {
for (int i = 0; i < r; i++) {
if (req[i] > need[pid][i]) {
printf("Denied\n");
return 0;
}
```

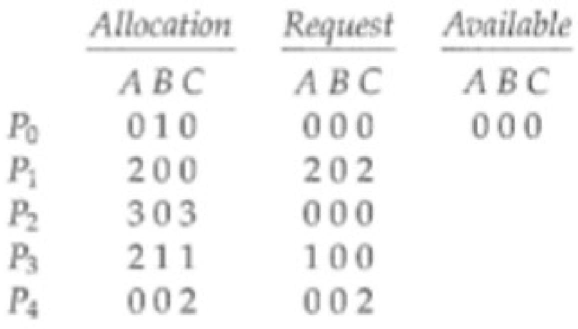
```
}
for (int i = 0; i < r; i++) {
if (req[i] > avail[i]) {
printf("Denied\n");
return 0;
}
}

for (int i = 0; i < r; i++) {
avail[i] -= req[i];
alloc[pid][i] += req[i];
need[pid][i] -= req[i];
}

int safeSeq[MAX];
if (isSafe(p, r, avail, max, alloc, need, safeSeq)) {
printf("Granted\n");
printf("Updated Allocation Matrix:\n");
printMatrix(p, r, alloc);
printf("Updated Need Matrix:\n");
printMatrix(p, r, need);
printf("Updated Available Array:\n");
for (int i = 0; i < r; i++) {
printf("%d ", avail[i]);
}
printf("\n\n");
return 1;
} else {
for (int i = 0; i < r; i++) {
avail[i] += req[i];
alloc[pid][i] -= req[i];
need[pid][i] += req[i];
}
printf("Denied\n");
return 0;
}
}

int main() {
int p, r;
int alloc[MAX][MAX], max[MAX][MAX], avail[MAX], need[MAX][MAX];
int safeSeq[MAX];

printf("Enter number of processes: ");
scanf("%d", &p);
printf("Enter number of resources: ");
scanf("%d", &r);

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

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

printf("Enter Available Array:\n");
for (int i = 0; i < r; i++) {
scanf("%d", &avail[i]);
}

calculateNeed(p, r, need, max, alloc);
printf("Need Matrix:\n");
printMatrix(p, r, need);

if (isSafe(p, r, avail, max, alloc, need, safeSeq)) {
printf("Safe state. Sequence: ");
for (int i = 0; i < p; i++) printf("P%d ", safeSeq[i]);
printf("\n");
} else {
printf("Unsafe state.\n");
}

int reqProc;
int req[MAX];
while (1) {
printf("Enter process requesting resources (-1 to exit): ");
scanf("%d", &reqProc);
if (reqProc == -1) break;
printf("Enter request array: ");
for (int i = 0; i < r; i++) {
scanf("%d", &req[i]);
}
requestResource(p, r, reqProc, req, avail, max, alloc, need);
}

return 0;
}
```

Output:



```
Enter number of processes: 5
Enter number of resources: 3
Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Available Array:
3 3 2
Need Matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

Safe state. Sequence: P1 P3 P4 P0 P2
Enter process requesting resources (-1 to exit): 1
Enter request array: 1 0 2
Granted
Updated Allocation Matrix:
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2

Updated Need Matrix:
7 4 3
0 2 0
6 0 0
0 1 1
4 3 1

Updated Available Array:
2 3 0

Enter process requesting resources (-1 to exit): 4
Enter request array: 3 3 0
Denied
Enter process requesting resources (-1 to exit): 0
Enter request array: 0 2 0
Denied
Enter process requesting resources (-1 to exit): -1
```

2. Consider the following snapshot of the system. Write C/C++ program to implement deadlock detection algorithm.
(a) Is the system in a safe state?
(b) Suppose that process P2 make one additional request for instance of type C, can the system still be in a safe state?



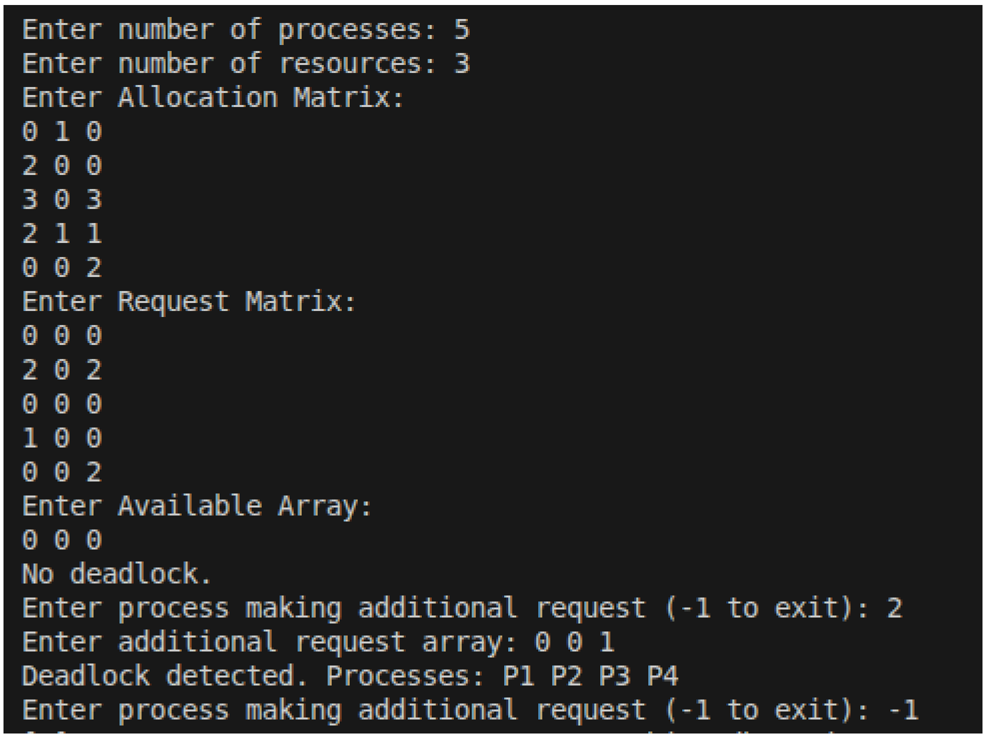
	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Code:

```
#include <stdio.h>

#define MAX 10

int detectDeadlock(int p, int r, int avail[MAX], int alloc[MAX][MAX], int req[MAX][MAX], int
deadlockedProcs[MAX]) {
int work[MAX];
int finish[MAX];
for (int i = 0; i < r; i++) work[i] = avail[i];

for (int i = 0; i < p; i++) {
int hasAllocation = 0;
for (int j = 0; j < r; j++) {
if (alloc[i][j] > 0) hasAllocation = 1;
}
if (hasAllocation == 0) finish[i] = 1;
else finish[i] = 0;
deadlockedProcs[i] = 0;
}

int found;
do {
found = 0;
for (int i = 0; i < p; i++) {
if (finish[i] == 0) {
int j;
for (j = 0; j < r; j++) {
if (req[i][j] > work[j]) break;
}
if (j == r) {
for (int k = 0; k < r; k++) {
work[k] += alloc[i][k];
}
finish[i] = 1;
found = 1;
}
}
}
} while (found != 0);

int isDeadlock = 0;
for (int i = 0; i < p; i++) {
if (finish[i] == 0) {
isDeadlock = 1;
deadlockedProcs[i] = 1;
}
}
return isDeadlock;
}

int main() {
int p, r;
int alloc[MAX][MAX], req[MAX][MAX], avail[MAX];
int deadlockedProcs[MAX];

printf("Enter number of processes: ");
scanf("%d", &p);
printf("Enter number of resources: ");
scanf("%d", &r);

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

printf("Enter Request Matrix:\n");
for (int i = 0; i < p; i++) {
for (int j = 0; j < r; j++) {
scanf("%d", &req[i][j]);
}
}

printf("Enter Available Array:\n");
for (int i = 0; i < r; i++) {
scanf("%d", &avail[i]);
}

if (detectDeadlock(p, r, avail, alloc, req, deadlockedProcs)) {
printf("Deadlock detected. Processes: ");
for (int i = 0; i < p; i++) {
if (deadlockedProcs[i]) printf("P%d ", i);
}
```

```
printf("\n");
} else {
printf("No deadlock.\n");
}

int reqProc;
int newReq[MAX];
while (1) {
printf("Enter process making additional request (-1 to exit): ");
scanf("%d", &reqProc);
if (reqProc == -1) break;
printf("Enter additional request array: ");
for (int i = 0; i < r; i++) {
scanf("%d", &newReq[i]);
req[reqProc][i] += newReq[i];
}
if (detectDeadlock(p, r, avail, alloc, req, deadlockedProcs)) {
printf("Deadlock detected. Processes: ");
for (int i = 0; i < p; i++) {
if (deadlockedProcs[i]) printf("P%d ", i);
}
printf("\n");
} else {
printf("No deadlock.\n");
}
}

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 Request Matrix:
0 0 0
2 0 2
0 0 0
1 0 0
0 0 2
Enter Available Array:
0 0 0
No deadlock.
Enter process making additional request (-1 to exit): 2
Enter additional request array: 0 0 1
Deadlock detected. Processes: P1 P2 P3 P4
Enter process making additional request (-1 to exit): -1
```