Write a C program to simulate deadlock detection.

1.1.1 Code:

```
#include<stdio.h>


int max[100][100];
int allocation[100][100];
int need[100][100];
int available[100];
int n,r;


int main()
{
        int i,j;
        printf("Deadlock Detection\n");
        input();
        show();
        cal();
        return 0;
}
```

```
void input()
{
        int i,j;
        printf("Enter the no of Processes: ");
        scanf("%d",&n);
        printf("Enter the no of resource instances: ");
        scanf("%d",&r);
        printf("Enter the Max Matrix:\n");
        for(i=0;i<n;i++)
        {
        for(j=0;j<r;j++)
        {
         scanf("%d",&max[i][j]);
        }
        }
        printf("Enter the Allocation Matrix:\n");
        for(i=0;i<n;i++)
        {
        for(j=0;j<r;j++)
        {
         scanf("%d",&allocation[i][j]);
```

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

void show()
{
        int i,j;
        printf("Process\t Allocation\t Max\t Available\t");
        for(i=0;i<n;i++)
        {
        printf("\nP%d\t   ",i+1);
        for(j=0;j<r;j++)
        {
        printf("%d ",allocation[i][j]);
        }
        printf("\t");
```

```
        for(j=0;j<r;j++)
        {
        printf("%d ",max[i][j]);
        }
        printf("\t");
        if(i==0)
        {
        for(j=0;j<r;j++)
        printf("%d ",available[j]);
        }
        }
}

void cal()
{
        int finish[100],temp,need[100][100],flag=1,k,c1=0;
        int dead[100];
        int safe[100];
        int i,j;
        for(i=0;i<n;i++)
        {
```

```
finish[i]=0;
}

for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
 need[i][j]=max[i][j]-allocation[i][j];
}
}
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
    if((finish[i]==0)&&(need[i][j]<=available[j]))
        {
        c++;
```

```
            if(c==r)
            {
               for(k=0;k<r;k++)
            {
                  available[k]+=allocation[i][j];
                  finish[i]=1;
                  flag=1;
            }
               if(finish[i]==1)
            {
                     i=n;
            }
            }
            }
      }
      }
}

      j=0;
      flag=0;
      for(i=0;i<n;i++)
```

```
        {
        if(finish[i]==0)
        {
        dead[j]=i;
        j++;
        flag=1;
        }
        }
        if(flag==1)
        {
        printf("\n\nSystem is in Deadlock and the Deadlock process are\n");
        for(i=0;i<n;i++)
        {
          printf("P%d\t",dead[i]);
        }
        }
        else
        {
        printf("\nNo Deadlock Occur");
        }
}
```

OUTPUT

```
Deadlock Detection
Enter the no of Processes: 3
Enter the no of resource instances: 3
Enter the Max Matrix:
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix:
3 3 3
2 0 4
1 2 4
Enter the available Resources:
1 2 0
Process  Allocation      Max     Available
P0         3 3 3        3 6 8   1 2 0
P1         2 0 4        4 3 3
P2         1 2 4        3 4 4

System is in Deadlock and the Deadlock process are
P0      P1      P2
```

```
Deadlock Detection
Enter the no of Processes: 5
Enter the no of resource instances: 3
Enter the Max Matrix:
0 0 0
2 0 2
0 0 0
1 0 0
0 0 2
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 3
3 1 1
0 0 2
Enter the available Resources:
0 0 0
Process  Allocation      Max     Available
P0         0 1 0        0 0 0   0 0 0
P1         2 0 0        2 0 2
P2         3 0 3        0 0 0
P3         3 1 1        1 0 0
P4         0 0 2        0 0 2
No Deadlock Occur
```

10/8/ Write a program to simulate deadlock detection

```
#include <stdio.h>
#define MaxP 10
#define MaxR 10
int num_processes, num_resources;
int max[MAX_R];
int need[MAX_P][MAX_R];
int allocation[MAX_P][MAX_R];
void main(){
int process_order[MAX_P];
 input data();
 int flag = check_safety(process_order);
 if(flag == 1){
printf("Deadlock is detected\n");
}
}
int check_safety(int process_order[]){
int work[MAX_R];
int finish[MAX_P] = {0};
for(int i=0; i<num_resources; i++){
  work[i] = available[i];
}
int completed = 0;
while(completed < num_processes){
 int found = 0;
 for(int i=0; i<num_processes; i++){
  if(!finish[i]){
  int j;
  for(j=0; j<num_resources; j++){
   if(need[i][j] > work[j]){
   break;
  }
```

```
if (j == num_resources) {
    for (int k = 0; k < num_resources; k++) {
        work[k] += allocation[i][j];
    }
    finish[i] = 1;
    process_order[completed] = i;
    completed++;
    found = 1;
}
}
}
if (!found) {
    return 0;
}
}
return 1;
}

void input_data() {
    printf("Enter no of processes: ");
    scanf("%d", &num_process);
    printf("Enter the no of resources");
    scanf("%d", &num_resources);
    printf("Enter the available resources: \n");
    for (int i=0; i<num_resources; i++) {
        scanf("%d", &available[i]);
    }
    printf("Enter the request matrix: \n");
    for (int i = 0; i < num_process; i++)
    for (int j = 0; j < num_resources; j++) {
        scanf("%d", &max[i][j]);
    }
}
```

```
printf("Enter allocation matrix:\n");
for (int i=0; i<num_process; i++){
 for (int j=0; j<num_resources; j++){
 scanf("%d", allocation[i][j])
 need[i][j] = max[i][j] allocation[i][j]
}
}
}
```

Output

Enter the no of processes: 5
enter the no of resources: 3
enter the available resources: 3 3 2
enter the request matrix:

7	5	3
3	2	2
9	0	2
2	2	2
4	3	3

enter the allocation matrix:

0	1	0
2	0	0
3	0	2
2	1	1
0	0	2

10 10
23 8 23

deadlock is present.