/*banker's algo*/

```c
#include<stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

int i,j;

printf("**** Banker's Algo ****\n");

input();

show();

cal();

return 0;

}

void input()

{

int i,j;

printf("Enter the no of Processes: ");

scanf("%d",&n);

printf("\nEnter the no of resources instances: ");

scanf("%d",&r);

printf("\nEnter the Max Matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{
```

```c
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",&alloc[i][j]);

}

}

printf("Enter the available Resources\n");

for(j=0;j<r;j++)

{

scanf("%d",&avail[j]);

}

}

void show()

{

int i,j;

printf("Process\tAllocation\tMax\tAvailable\tNeed\t");

for(i=0;i<n;i++)

{

printf("\nP%d\t ",i+1);

for(j=0;j<r;j++)

{

printf("%d ",alloc[i][j]);

}

printf("\t\t");

for(j=0;j<r;j++)

{
```

```c
printf("%d ",max[i][j]);

}

printf("\t");

if(i==0)

{

for(j=0;j<r;j++)

printf("%d ",avail[j]);

}

printf("\t\t");

for(j=0;j<r;j++)

{

printf("%d ",max[i][j]-alloc[i][j]);

}

}

}

void cal()

{

int finish[100],temp,need[100][100],flag=1,k,c1=0;

int safe[100]; //if the system wasn't at safe state then how much states were in safe state

int i,j;

for(i=0;i<n;i++) //This array is initialized to keep track of whether each process has finished (1 if
finished, 0 otherwise),it is set to all 0s, indicating that no process has finished at the start

{

finish[i]=0;

}

//find need matrix

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-alloc[i][j];
```

```
}

}

printf("\n");

while(flag) //while loop that runs until flag becomes 0(i.e the condition become false)(for each
process this while loop will execute)

{

flag=0;

for(i=0;i<n;i++)

{

int c=0;

for(j=0;j<r;j++)

{

if((finish[i]==0)&&(need[i][j]<=avail[j])) //(finish[i] == 0)=process is not finished

{

c++; //if loop cond. is satisfied, it increments a counter c. This counter keeps track of how many
resource types a process can be allocated.

if(c==r) //If c becomes equal to the total number of resource types r, it means that all the resources a
process needs are available, and it can be allocated all of them.

{

for(k=0;k<r;k++)

{

avail[k]+=alloc[i][j]; //it proceeds to allocate the resources, update the available resources, mark the
process as finished, and set the flag to 1 to continue checking other processes

finish[i]=1;

flag=1;

}

printf("P%d->",i+1); //the process number (P%d) to indicate that the process has been allocated
resources.

if(finish[i]==1)

{

i=n; //effectively breaking out of the outer loop, because there's no need to continue checking other
resources for this process.(the while loop exit cond.)

}
```

```c
}

}

}

}

}

for(i=0;i<n;i++)

{

if(finish[i]==1)

{

c1++; //A counter to keep track of the number of processes that have finished.

}

else

{

//printf("P%d->",i+1); // the finish array and checks which processes have finished (finish[i] == 1) and
increments c1 accordingly.

}

}

if(c1==n)

{printf("\n The system is in safe state");

}

else //If c1 is less than n, it indicates that not all processes have finished, and the system is in an
unsafe state with processes potentially deadlocked. It prints messages indicating a deadlock and an
unsafe state.

{

printf("\n Process are in dead lock");

printf("\n System is in unsafe state");

}

}
```

```
**** Banker's Algo ****
Enter the no of Processes: 5

Enter the no of resources instances: 3

Enter the Max 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
Enter the available Resources
3 3 2
Process Allocation    Max      Available    Need
P1       0 1 0        7 5 3    3 3 2        7 4 3
P2       2 0 0        3 2 2                 1 2 2
P3       3 0 2        9 0 2                 6 0 0
P4       2 1 1        2 2 2                 0 1 1
P5       0 0 2        4 3 3                 4 3 1
P2->P4->P5->P3->P1->
 The system is in safe state
------------------------------------
Process exited after 61.92 seconds with return value 0
Press any key to continue . . .
```