8. Design, develop and implement a C/C++/Java program to implement *Banker’s algorithm*. Assume suitable input required to demonstrate the results.

```
#include<stdio.h>
struct process
{
      int
i_all[6],i_max[6],i_need[6],i_finished;
} p[10];
int i_avail[6], i_sseq[10], i_ss=0, i_check1=0, i_check2=0, n,
i_work[6]; int i_nor;

void main()
{
      int safeseq(void);
      int tj,ch,i=0,j=0,k,ch1;

      printf("Enter number of processes : ");
      scanf("%d",&n);
      printf("Enter the Number of Resources : ");
      scanf("%d", &i_nor);
      printf("Enter the Available Resources : ");
      for(j=0;j<n;j++)
      {
            for(k=0; k<i_nor; k++)
            {
                  if(j==0)
                  {
                        printf("\n For Resource type %d : ",k);
                        scanf("%d", &i_avail[k]);
                  }
                  p[j].i_max[k]=0;
                  p[j].i_all[k]=0;
                  p[j].i_need[k]=0;
                  p[j].i_finished=0;
                  p[j].i_request[k]=0;
            }
      }
      printf("\n Enter Max resources for all processes\n");
      for(i=0;i<n;i++)
      {
            for(j=0; j< i_nor; j++)
            {
                  scanf("%d",&p[i].i_max[j]);
            }
      }

      printf("\n Enter Allocated resources for all processes\n");
      for(i=0;i<n;i++)
      {
            for(j=0;j<i_nor;j++)
            {
                  scanf("%d",&p[i].i_all[j]);
                  if(p[i].i_all[j]>p[i].i_max[j])
                  {
```

```
                printf("\n Allocation should be less < or
                == max"); j--;
            }
            else
            p[i].i_need[j]=p[i].i_max[j]-p[i].i_all[j];
        }
    }

    if(safeseq()==1)
    {
        printf("\n The System is in Safe state\n ");
    }
    else
    printf("\n The System is Not in safe state\n ");

    printf("\n Need\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<i_nor;j++)
        printf(" %d ",p[i].i_need[j]);
        printf("\n");
    }
}

int safeseq()
{
    int tk,tj,i,j,k;
    i_ss=0;
    for(j=0; j<i_nor; j++)
    i_work[j] = i_avail[j];
    for(j=0;j<n;j++)
    p[j].i_finished=0;
    for(tk=0; tk<i_nor; tk++)
    {
        for(j=0;j<n;j++)
        {
            if(p[j].i_finished==0)
            {
                i_check1=0;
                for(k=0; k<i_nor; k++)
                if(p[j].i_need[k]<=i_work[k])
                i_check1++;
                if(i_check1== i_nor)
                {
                    for(k=0;k< i_nor;k++)
                    {
                        i_work[k]= i_work[k]+p[j]. i_all[k];
                        p[j]. i_finished=1;
                    }
                    i_sseq[i_ss]=j;
                    i_ss++;
                 }
            }
         }
}
    i_check2=0;
    for(i=0;i<n;i++)
```

```
     if(p[i].i_finished==1)
     i_check2++;
     if(i_check2>=n)
     {
           printf("The Safe Sequence is\t :");
           for(tj=0;tj<n;tj++)
           printf("%d ", i_sseq[tj]);
           return 1;
     }
     return 0;
}
```

Output :

```
Enter number of processes: 5
Enter the number of Resources: 3
Enter the Available Resources:
For Resource type 0: 3
For Resource type 1: 3
For Resource type 2: 2

Enter Max resources for all processes
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter Allocated resources for all processes
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
The safe sequence is : 1,3,4,0,2
The System is in Safe state

Need
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
```