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

typedef struct
{
      char pid[10];
      int alloc[10];
      int need[10];
      int max[10];
}pro;

typedef struct
{
      pro *p;
      int n;
      int m;
      int avail[10];
      char r[10];
      int req[10];
}process;



process* initialize(process *p1,int n, int m)
{
      p1->p=(pro*)malloc(sizeof(pro)*n);
      p1->n=n;
      p1->m=m;
      return p1;
}

void input(process *p1)
{
      int i,j;
      printf("Enter Process\n");
      for(i=0;i<p1->n;i++)
      {
                  scanf("%s",p1->p[i].pid);
      }

      for(i=0;i<p1->n;i++)
      {
                  printf("Enter max for %s : ",p1->p[i].pid);
                  for(j=0;j<p1->m;j++)
                  {
                              scanf("%d",&p1->p[i].max[j]);
                  }
      }

      for(i=0;i<p1->n;i++)
      {
                  printf("Enter allocation for %s : ",p1->p[i].pid);
                  for(j=0;j<p1->m;j++)
```

```
                {
                        scanf("%d",&p1->p[i].alloc[j]);
                }
        }

        for(i=0;i<p1->n;i++)
        {
                for(j=0;j<p1->m;j++)
                {
                        p1->p[i].need[j]=p1->p[i].max[j]-p1-
>p[i].alloc[j];
                }
        }

        printf("Enter available :");
        for(j=0;j<p1->m;j++)
        {
                scanf("%d",&p1->avail[j]);
        }

        printf("Process Requesting : ");
        scanf("%s",p1->r);

        printf("%s's Request : ",p1->r);
        for(j=0;j<p1->m;j++)
        {
                scanf("%d",&p1->req[j]);
        }

}


void output(process *p1)
{
        printf("Proc\t Max\t    Alloc\t Need\n");
        int i,j;
        for(i=0;i<p1->n;i++)
        {
                printf("\n%s\t",p1->p[i].pid);
                for(j=0;j<p1->m;j++)
                {
                        printf("%d  ",p1->p[i].max[j]);
                }
                printf("   ");
                for(j=0;j<p1->m;j++)
                {
                        printf("%d  ",p1->p[i].alloc[j]);
                }
                printf("   ");
                for(j=0;j<p1->m;j++)
                {
                        printf("%d  ",p1->p[i].need[j]);
                }
        }
```

```
}

int banker(process *p1)
{
      int comp[p1->n];
      int no,count=0;
      int i,j,temp=0;
      for(i=0;i<p1->n;i++)
      {
                  comp[i]=0;
      }
      no=0;
      i=0;
      int flag;
      char a[p1->n][10];
      while(no!=p1->n && count<3*p1->n)
      {
                  flag=0;
                  for(j=0;j<p1->m;j++)
                  {
                              if(p1->p[i].need[j]>p1->avail[j])
                              {
                                    flag++;
                                    break;
                              }
                  }


                  if(flag==0  && comp[i]!=1)
                  {
                              for(j=0;j<p1->m;j++)
                                    p1->avail[j]+=p1->p[i].alloc[j];

                              comp[i]=1;
                              strcpy(a[temp],p1->p[i].pid);
                              temp++;
                              no++;
                  }

                  i=(i+1)%p1->n;
                  count++;
      }
      if(no==p1->n)
      {
                  printf("Safe sequence is : ");
                  for(i=0;i<p1->n;i++)
                              printf("%s ",a[i]);
                  printf("\n");
                  return 1;
      }
      else
                  printf("The system is not in safe state\n");
                  return 0;
}
```

```
void request(process *p1)
{
	int i,j,flag=0;;
	for(i=0;i<p1->n;i++)
	{
			if(strcmp(p1->r,p1->p[i].pid)==0)
				break;
	}
	int index=i;
	for(j=0;j<p1->m;j++)
	{
			if(p1->p[index].need[j]<p1->req[j])
			{
				printf("Process exceeds maximum claim\n");
				flag=1;
				break;
			}
	}

	for(j=0;j<p1->m;j++)
	{
			if(p1->avail[j]<p1->req[j] && flag==0)
			{
				printf("Resources are not available. %s
should wait\n",p1->p[index].pid);
				flag=1;
				break;
			}
	}
	if(flag==0)
	{
			for(j=0;j<p1->m;j++)
			{
				p1->avail[j]-=p1->req[j];
				p1->p[index].alloc[j]+=p1->req[j];
				p1->p[index].need[j]-=p1->req[j];

			}

			int res=banker(p1);
			if(res==1)
			{
				printf("The %s request can be granted
immediately\n",p1->p[index].pid);
			}
			else
				printf("The %s request can not be granted
immediately\n",p1->p[index].pid);
	}
}
```

```
void main()
{
      process *p1;
      int ch=1;
      while(ch!=4)
      {
                  printf("\n1.Read data\n2.Print Data\n3.Safety
Sequence\n4.Exit\n");
                  scanf("%d",&ch);

                  switch(ch)
                  {
                              case 1:
                              {
                                    int n,m;
                                    printf("Enter number of process : ");
                                    scanf("%d",&n);
                                    printf("Enter number of instances of
each resource : ");
                                    scanf("%d",&m);
                                    p1=initialize(p1,n,m);
                                    input(p1);
                                    break;
                              }
                              case 2:
                              {
                                    output(p1);
                                    break;
                              }
                              case 3:
                              {
                                    request(p1);
                                    break;
                              }
                              case 4:
                              {
                                    printf("Thank You\n");
                                    break;
                              }
                              default:
                                    printf("Invalid input\n");
                  }
      }
}



/*

Output:
1.  Read data
2.  Print Data
```

```
3.  Safety Sequence
4.  Exit
1
Enter number of process : 5
Enter number of instances of each resource : 3
Enter Process
p0 p1 p2 p3 p4
Enter max for p0 : 7 5 3
Enter max for p1 : 3 2 2
Enter max for p2 : 9 0 2
Enter max for p3 : 2 2 2
Enter max for p4 : 4 3 3
Enter allocation for p0 : 0 1 0
Enter allocation for p1 : 2 0 0
Enter allocation for p2 : 3 0 2
Enter allocation for p3 : 2 1 1
Enter allocation for p4 : 0 0 2
Enter available :3 3 2
Process Requesting : p1
p1's Request : 1 0 2

1.  Read data
2.  Print Data
3.  Safety Sequence
4.  Exit
2
Proc   Max      Alloc    Need

p0    7  5  3     0  1  0     7  4  3
p1    3  2  2     2  0  0     1  2  2
p2    9  0  2     3  0  2     6  0  0
p3    2  2  2     2  1  1     0  1  1
p4    4  3  3     0  0  2     4  3  1
1.  Read data
2.  Print Data
3.  Safety Sequence
4.  Exit
3

Safe sequence is : p1 p3 p4 p0 p2
The p1 request can be granted immediately

1.  Read data
2.  Print Data
3.  Safety Sequence
4.  Exit
4
Thank You
*/
```