Student Name: Bhavesh Mahala
Student ID: 11804138
Email Address: bhaveshmahala20@gmail.com
GitHub Link: https://github.com/bhaveshmahala

Question:24 Consider following and Generate a solution in C to find whether the system is in safe state or not?

Available				Processes	Allocation				Max			
A	B	C	D		A	B	C	D	A	B	C	D
1	5	2	0	P0	0	0	1	2	0	0	1	2
				P1	1	0	0	0	1	7	5	0
				P2	1	3	5	4	2	3	5	6
				P3	0	6	3	2	0	6	5	2
				P4	0	0	1	4	0	6	5	6

Description:-

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[i] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

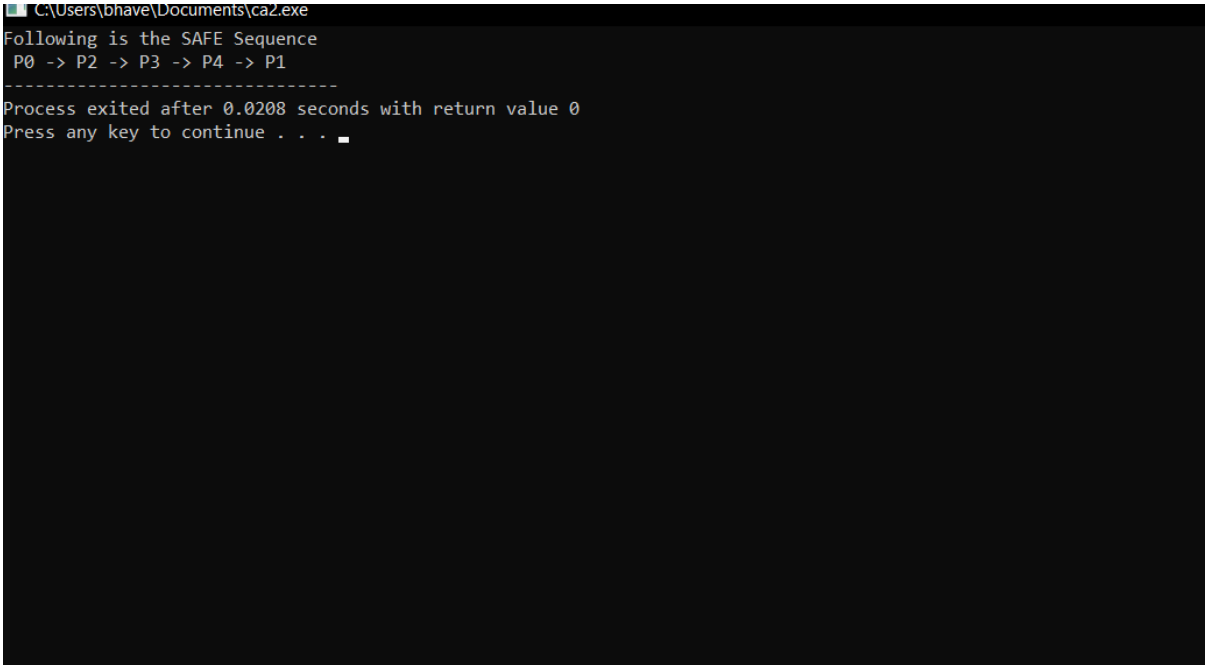
Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$ for its execution.
- Need [i, j] = Max [i, j] – Allocation [i, j]

Code:-(Predefined values in the code)

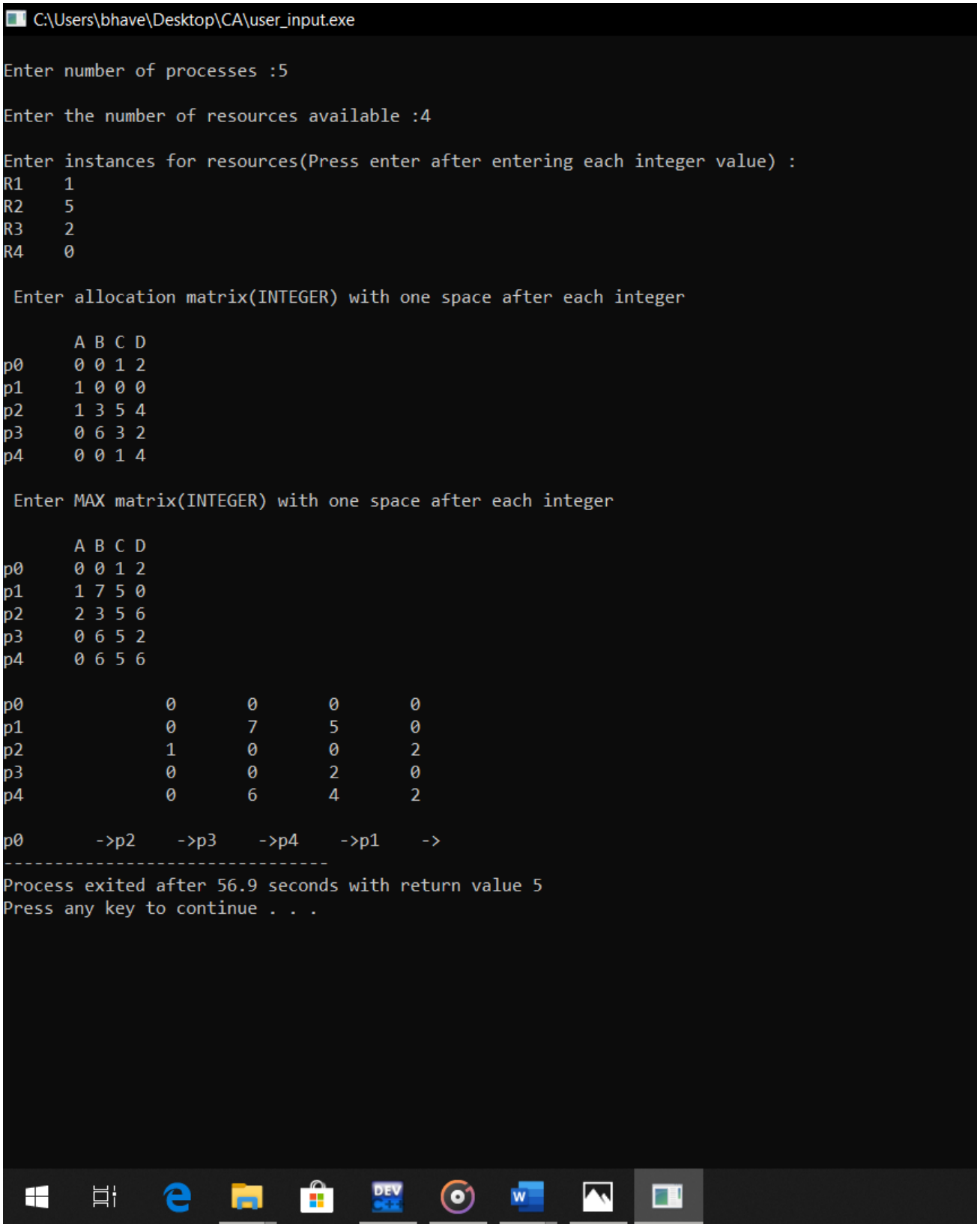
```c
/* Question 23-- Consider following and Generate a solution in C to find whether the system is in safe state or not?

    Available       Processes       Allocation              Max
 A   B   C   D                  A   B   C   D       A   B   C   D
 1   5   2   0       P0         0   0   1   2       0   0   1   2
                     P1         1   0   0   0       1   7   5   0
                     P2         1   3   5   4       2   3   5   6
                     P3         0   6   3   2       0   6   5   2
                     P4         0   0   1   4       0   6   5   6


#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;//declaration of variables
    n = 5; // Indicates the total number of processes of the system
    m = 4; // Indicates the total number of resources in the system
    int alloc[5][4] = { { 0, 0, 1, 2 }, // P0    // Allocation Matrix
                        { 1, 0, 0, 0 }, // P1    // Indicates where the process you have received a resource
                        { 1, 3, 5, 4 }, // P2
                        { 0, 6, 3, 2 }, // P3
                        { 0, 0, 1, 4 } }; // P4

    int max[5][4] = { { 0, 0, 1, 2 }, // P0    // MAX Matrix
                      { 1, 7, 5, 0 }, // P1
                      { 2, 3, 5, 6 }, // P2
                      { 0, 6, 5, 2 }, // P3
                      { 0, 6, 5, 6 } }; // P4

    int avail[4] = { 1, 5, 2, 0 }; // Available Resources

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {       //Sorting the process
        f[k] = 0;
    }
    int need[n][m];                 //Express how many more resources can be allocated in future
    for (i = 0; i < n; i++) {       //Sorting the process
        for (j = 0; j < m; j++)     //Sorting the process
            need[i][j] = max[i][j] - alloc[i][j]; //Need= maximum resources - currently allocated resources
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {

                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                        break;
                    }
                }

                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)             //Sorting the process for safe state.
        printf(" P%d ->", ans[i]);          //Printing all hte process in safe state order
        printf(" P%d", ans[n - 1]);

    return (0);

}
```

Output:-


C:\Users\bhave\Documents\ca2.exe
Following is the SAFE Sequence
P0 -> P2 -> P3 -> P4 -> P1
Process exited after 0.0208 seconds with return value 0
Press any key to continue . . .

Code:-(User is asked to enter the values)

```c
/* Question 24-- Consider following and Generate a solution in C to find whether the system is in safe state or not?

    Available       Processes       Allocation              Max
 A   B   C   D                    A   B   C   D       A   B   C   D
 1   5   2   0        P0          0   0   1   2       0   0   1   2
                      P1          1   0   0   0       1   7   5   0
                      P2          1   3   5   4       2   3   5   6
                      P3          0   6   3   2       0   6   5   2
                      P4          0   0   1   4       0   6   5   6


#include<stdio.h>
int main()
{
    int num;
    int n;
    int i,j,k,c,c1;
    int avail[20],arr[10];
    int need[20][20],alloc[20][20],max[20][20];

    printf("\nEnter number of processes :");
    scanf("%d",&num);

    printf("\nEnter the number of resources available :");
    scanf("%d",&n);

    printf("\nEnter instances for resources(Press enter after entering each integer value) :\n");
    for(i=0;i<n;i++)
    {
        printf("R%d    ",i+1);
        scanf("%d",&avail[i]);
    }
    printf("\n Enter allocation matrix(INTEGER) with one space after each integer  \n");   //Allocation Matrix
    printf("\n       A B C D \n");            //For pretty formatting output
    for(i=0;i<num;i++)
    {
        printf("p%d     ",    i);          arr[i]=0;       //to print the process number
        for(j=0;j<n;j++)
        {
            scanf("%d",    &alloc[i][j]);
        }
    }
    printf("\n Enter MAX matrix(INTEGER) with one space after each integer  \n");        //MAX Matrix
    printf("\n       A B C D \n");           //For pretty formatting output
    for(i=0;i<num;i++)                                  //Sorting the process
    {
        printf("p%d     ",    i);                       //to print the process number
        for(j=0;j<n;j++)                                //Sorting the process
        {
            scanf("%d",&max[i][j]);
        }
    }
    for(i=0;i<num;i++)                                  //Sorting the process
    {
        printf("\np%d\t",i) ;
        for(j=0;j<n;j++)                                //Sorting the process
        {
            need[i][j]=max[i][j]-alloc[i][j];               //Need= maximum resources - currently allocated resources
            printf("\t%d",need[i][j]);
        }
    }
    k=0;     c1=0;
    printf("\n\n");
    while(k<15)
    {
        for(i=0;i<num;i++)                              //Sorting the process
        {
            c=0;
            for(j=0;j<n;j++)                                //Sorting the process
            {
                if(arr[i]==1) break;
                if(need[i][j]<=avail[j])
                {
                    c++;
                }
                if(c==n)
                {
                    for(j=0;j<n;j++)
                    {
                        avail[j]+=alloc[i][j];
                    }
                    printf("p%d\t ->",i);  arr[i]=1;     c1++;
                }
            }
        }
        k++;
    }
}
```


C:\Users\bhave\Desktop\CA\user_input.exe
Enter number of processes :5
Enter the number of resources available :4
Enter instances for resources(Press enter after entering each integer value) :
R1 1
R2 5
R3 2
R4 0
Enter allocation matrix(INTEGER) with one space after each integer
A B C D
p0 0 0 1 2
p1 1 0 0 0
p2 1 3 5 4
p3 0 6 3 2
p4 0 0 1 4
Enter MAX matrix(INTEGER) with one space after each integer
A B C D
p0 0 0 1 2
p1 1 7 5 0
p2 2 3 5 6
p3 0 6 5 2
p4 0 6 5 6
p0 0 0 0 0
p1 0 7 5 0
p2 1 0 0 2
p3 0 0 2 0
p4 0 6 4 2
p0 ->p2 ->p3 ->p4 ->p1 ->
Process exited after 56.9 seconds with return value 5
Press any key to continue . . .