os1) a1)

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

#define MAX 10
int alloc[MAX][MAX];
int maxm[MAX][MAX];
int need[MAX][MAX];
int avail[MAX];
int work[MAX];
int seqn[MAX];
int finish[MAX];
int req[MAX];

int n,r,i,j;

void accept(){
        n = 5; r = 3;
        int ralloc[5][3] = {{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
        int rmaxm[5][3] = {{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
        int ravail[3] = {3,3,2};

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                alloc[i][j] = ralloc[i][j];
        }

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        maxm[i][j] = rmaxm[i][j];
        }

        for(i = 0; i < n; i++)
                for(j = 0; j < r; j++)
                        need[i][j] = maxm[i][j] - alloc[i][j];

        for(i = 0; i < r; i++)
                avail[i] = ravail[i];

        for(i = 0; i < r; i++)
                work[i] = avail[i];

}


void display(){
        printf("\nAllocation matrix\n");

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

```
        printf("\nMaximum resources matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",maxm[i][j]);
                printf("\n");
        }

        printf("\nNeed Matrix\n");
        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",need[i][j]);
                printf("\n");
        }

        printf("\nAvailable resources matrix\n");
        for(i = 0; i < r; i++)
                printf("%d\t",avail[i]);

        printf("\n");
}

int safestateCheck(){
        int ind = 0, fl = 1, cnt;

        for(i = 0; i < n; i++) finish[i] = 0;

        for(i = 0; i < n; i++)
        for(j = 0; j < r; j++){
                if(need[i][j] < 0){
                        printf("\nAllocated resources are more than maximum
available resources of P%d\n",i);
                        return 0;

                }
        }

        while (fl){
                fl = 0;
                for(i = 0; i < n; i++){
                        if(finish[i] == 0){
                                cnt = 0;
                                for(j = 0; j < r; j++){
                                        if(need[i][j] <= work[j])
                                                        cnt++;
                                        else
                                                break;
                                }

                                if(cnt == r){
                                        for(j = 0; j < r; j++)
                                                work[j]+=alloc[i][j];
                                        seqn[ind++] = i;
                                        finish[i] = 1;
```

```
                                        fl = 1;
                                }
                        }
                }


        }
        for(i = 0; i < n; i++)
                if(finish[i] == 0) return 0;

        return 1;
}


int main(){
        int choice;
        accept();
        do{
        printf("1.Display data\n2.Check Safestate\n3.Quit\nENTER CHOICE\n");
        scanf("%d",&choice);

        switch(choice){
        case 1: display(); break;
        case 2: {
                int res = safestateCheck();
                if(res == 1){
                        printf("\nSystem is in safe state.\n");
                        for(i = 0; i < n; i++){
                                printf("P%d\t",seqn[i]);
                        }
                        printf("\n");
                }
                else printf("\nSystem is not in safe state\n");
                }
                break;
        case 3: printf("\nBye!"); break;
        default : printf("\nWrong Choice, try again!");
        }}while(choice != 3);

        return 0;
}

/*
OUTPUT :

1.Display data
2.Check Safestate
3.Quit
ENTER CHOICE
1

Allocation matrix
0       1       0
2       0       0
```

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

Maximum resources matrix
7       5       3
3       2       2
9       0       2
2       2       2
4       3       3

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

Available resources matrix
3       3       2
1.Display data
2.Check Safestate
3.Quit
ENTER CHOICE
2

System is in safe state.
P1      P3      P4      P0      P2
1.Display data
2.Check Safestate
3.Quit
ENTER CHOICE
3

Bye!

*/
```

---

a2)

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

#define MAX 10
int alloc[MAX][MAX];
int maxm[MAX][MAX];
int need[MAX][MAX];
int avail[MAX];
int work[MAX];
int seqn[MAX];
int finish[MAX];
int req[MAX];

int n,r,i,j;
```

```
void accept(){
        n = 5; r = 4;
        int ralloc[5][4] = {{0,0,1,2},{1,0,0,0},{1,3,5,4},{0,6,3,2},{0,0,1,4}};
        int rmaxm[5][4] = {{0,0,1,2},{1,7,5,0},{2,3,5,6},{0,6,5,2},{0,6,5,6}};
        int ravail[4] = {1,5,2,0};

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                alloc[i][j] = ralloc[i][j];
        }

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        maxm[i][j] = rmaxm[i][j];
        }

        for(i = 0; i < n; i++)
                for(j = 0; j < r; j++)
                        need[i][j] = maxm[i][j] - alloc[i][j];

        for(i = 0; i < r; i++)
                avail[i] = ravail[i];

        for(i = 0; i < r; i++)
                work[i] = avail[i];

}


void display(){
        printf("\nAllocation matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",alloc[i][j]);
                printf("\n");
        }

        printf("\nMaximum resources matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",maxm[i][j]);
                printf("\n");
        }

        printf("\nNeed Matrix\n");
        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",need[i][j]);
                printf("\n");
        }

        printf("\nAvailable resources matrix\n");
```

```
        for(i = 0; i < r; i++)
                printf("%d\t",avail[i]);

        printf("\n");
}

int safestateCheck(){
        int ind = 0, fl = 1, cnt;

        for(i = 0; i < n; i++) finish[i] = 0;

        for(i = 0; i < n; i++)
        for(j = 0; j < r; j++){
                if(need[i][j] < 0){
                        printf("\nAllocated resources are more than maximum
available resources of P%d\n",i);
                        return 0;

                }
        }

        while (fl){
                fl = 0;
                for(i = 0; i < n; i++){
                        if(finish[i] == 0){
                                cnt = 0;
                                for(j = 0; j < r; j++){
                                        if(need[i][j] <= work[j])
                                                        cnt++;
                                        else
                                                break;
                                }

                                if(cnt == r){
                                        for(j = 0; j < r; j++)
                                                work[j]+=alloc[i][j];
                                        seqn[ind++] = i;
                                        finish[i] = 1;
                                        fl = 1;
                                }
                        }
                }


        }
        for(i = 0; i < n; i++)
                if(finish[i] == 0) return 0;

        return 1;
}


int main(){
        int choice;
```

```
        accept();
        do{
        printf("1.Display data\n2.Check Safestate\n3.Quit\nENTER CHOICE\n");
        scanf("%d",&choice);

        switch(choice){
        case 1: display(); break;
        case 2: {
                int res = safestateCheck();
                if(res == 1){
                        printf("\nSystem is in safe state.\n");
                        for(i = 0; i < n; i++){
                                printf("P%d\t",seqn[i]);
                        }
                        printf("\n");
                }
                else printf("\nSystem is not in safe state\n");
                }
                break;
        case 3: printf("\nBye!"); break;
        default : printf("\nWrong Choice, try again!");
        }}while(choice != 3);

        return 0;
}

/*
OUTPUT:
1.Display data
2.Check Safestate
3.Quit
ENTER CHOICE
1

Allocation matrix
0       0       1       2
1       0       0       0
1       3       5       4
0       6       3       2
0       0       1       4

Maximum resources matrix
0       0       1       2
1       7       5       0
2       3       5       6
0       6       5       2
0       6       5       6

Need Matrix
0       0       0       0
0       7       5       0
1       0       0       2
0       0       2       0
0       6       4       2
```

```
Available resources matrix
1       5       2       0
1.Display data
2.Check Safestate
3.Quit
ENTER CHOICE
2

System is in safe state.
P0      P2      P3      P4      P1
1.Display data
2.Check Safestate
3.Quit
ENTER CHOICE
3

Bye!
*/
-----------------------------------------------
b1)
#include<stdio.h>
#include<stdlib.h>

#define MAX 10
int alloc[MAX][MAX];
int maxm[MAX][MAX];
int need[MAX][MAX];
int avail[MAX];
int work[MAX];
int seqn[MAX];
int finish[MAX];
int req[] = {1,0,2};

int n,r,i,j;

void accept(){
        n = 5; r = 3;
        int ralloc[5][3] = {{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
        int rmaxm[5][3] = {{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
        int ravail[3] = {3,3,2};

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                alloc[i][j] = ralloc[i][j];
        }

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        maxm[i][j] = rmaxm[i][j];
        }

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

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

        for(i = 0; i < r; i++)
                avail[i] = ravail[i];

        for(i = 0; i < r; i++)
                work[i] = avail[i];


}


void display(){
        printf("\nAllocation matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",alloc[i][j]);
                printf("\n");
        }

        printf("\nMaximum resources matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",maxm[i][j]);
                printf("\n");
        }

        printf("\nNeed Matrix\n");
        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",need[i][j]);
                printf("\n");
        }

        printf("\nAvailable resources matrix\n");
        for(i = 0; i < r; i++)
                printf("%d\t",avail[i]);

        printf("\n");
}

int safestateCheck(){
        int ind = 0, fl = 1, cnt;

        for(i = 0; i < n; i++) finish[i] = 0;

        for(i = 0; i < n; i++)
        for(j = 0; j < r; j++){
                if(need[i][j] < 0){
                        printf("\nAllocated resources are more than maximum
available resources of P%d\n",i);
                        return 0;
```

```
                }
        }

        while (fl){
                fl = 0;
                for(i = 0; i < n; i++){
                        if(finish[i] == 0){
                                cnt = 0;
                                for(j = 0; j < r; j++){
                                        if(need[i][j] <= work[j])
                                                        cnt++;
                                        else
                                                break;
                                }

                                if(cnt == r){
                                        for(j = 0; j < r; j++)
                                                work[j]+=alloc[i][j];
                                        seqn[ind++] = i;
                                        finish[i] = 1;
                                        fl = 1;
                                }
                        }
                }


        }
        for(i = 0; i < n; i++)
                if(finish[i] == 0) return 0;

        return 1;
}

void requestCheck(){
        int cnd1 = 1, cnd2 = 1, p = 1;

        printf("\nAvailable resouces :\n");
        for(i = 0; i < r; i++) printf("%d\t",avail[i]);

        for(i = 0;i <r; i++){
                if(req[i] > avail[i]){
                printf("\nRequest cannot be greater than available
resources\n");
                cnd1 = 0;break;
                }
        }

        for(i = 0; i < r; i++){
                if(req[i] > need[p][i]){
                        cnd2 = 0;
                        printf("\nRequest cannot exceed need\n");
                        break;
                        }
```

```
        }

        if(cnd1 && cnd2){
                for(i = 0; i < r; i++){
                        avail[i]-=req[i];
                        alloc[p][i]+=req[i];
                        need[p][i] -= req[i];
                }

                if(!safestateCheck()){
                        printf("\nSystem is not in safe state\n");
                        printf("\nRequest cannot be granted!\n");

                        for(i = 0; i < r; i++){
                        avail[i]+=req[i];
                        alloc[p][i]-=req[i];
                        need[p][i] += req[i];
                        }
                }
                else printf("\nSystem is in safe state. Request can be
allocated.\n");
        }
        else printf("\nSystem is in unsafe state\n");
}


int main(){
        int choice;
        accept();
        do{
        printf("\n1.Display data\n2.Check Safestate\n3.Process
Request\n4.Quit\nENTER CHOICE\n");
        scanf("%d",&choice);

        switch(choice){
        case 1: display(); break;
        case 2: {
                int res = safestateCheck();
                if(res == 1){
                        printf("\nSystem is in safe state.\n");
                        for(i = 0; i < n; i++){
                                printf("P%d\t",seqn[i]);
                        }
                        printf("\n");
                }
                else printf("\nSystem is not in safe state\n");
                }
                break;
        case 3: requestCheck(); break;
        case 4: printf("\nBye!"); break;
        default : printf("\nWrong Choice, try again!");
        }}while(choice != 4);

        return 0;
```

```
}

/*
OUTPUT:

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
1

Allocation matrix
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2

Maximum resources matrix
7       5       3
3       2       2
9       0       2
2       2       2
4       3       3

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

Available resources matrix
3       3       2

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
2

System is in safe state.
P1      P3      P4      P0      P2

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
3

Available resouces :
```

```
3       3       2
System is in safe state. Request can be allocated.

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
4

Bye!
*/
-------------------------------------
b2)
#include<stdio.h>
#include<stdlib.h>

#define MAX 10
int alloc[MAX][MAX];
int maxm[MAX][MAX];
int need[MAX][MAX];
int avail[MAX];
int work[MAX];
int seqn[MAX];
int finish[MAX];
int req[] = {0,4,2,0};

int n,r,i,j;

void accept(){
        n = 5; r = 4;
        int ralloc[5][4] = {{0,0,1,2},{1,0,0,0},{1,3,5,4},{0,6,3,2},{0,0,1,4}};
        int rmaxm[5][4] = {{0,0,1,2},{1,7,5,0},{2,3,5,6},{0,6,5,2},{0,6,5,6}};
        int ravail[4] = {1,5,2,0};

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                alloc[i][j] = ralloc[i][j];
        }

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        maxm[i][j] = rmaxm[i][j];
        }

        for(i = 0; i < n; i++)
                for(j = 0; j < r; j++)
                        need[i][j] = maxm[i][j] - alloc[i][j];

        for(i = 0; i < r; i++)
                avail[i] = ravail[i];

        for(i = 0; i < r; i++)
                work[i] = avail[i];
```

```
}


void display(){
        printf("\nAllocation matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",alloc[i][j]);
                printf("\n");
        }

        printf("\nMaximum resources matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",maxm[i][j]);
                printf("\n");
        }

        printf("\nNeed Matrix\n");
        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",need[i][j]);
                printf("\n");
        }

        printf("\nAvailable resources matrix\n");
        for(i = 0; i < r; i++)
                printf("%d\t",avail[i]);

        printf("\n");
}

int safestateCheck(){
        int ind = 0, fl = 1, cnt;

        for(i = 0; i < n; i++) finish[i] = 0;

        for(i = 0; i < n; i++)
        for(j = 0; j < r; j++){
                if(need[i][j] < 0){
                        printf("\nAllocated resources are more than maximum
available resources of P%d\n",i);
                        return 0;

                }
        }

        while (fl){
                fl = 0;
                for(i = 0; i < n; i++){
```

```
                        if(finish[i] == 0){
                                cnt = 0;
                                for(j = 0; j < r; j++){
                                        if(need[i][j] <= work[j])
                                                        cnt++;
                                        else
                                                break;
                                }

                                if(cnt == r){
                                        for(j = 0; j < r; j++)
                                                work[j]+=alloc[i][j];
                                        seqn[ind++] = i;
                                        finish[i] = 1;
                                        fl = 1;
                                }
                        }
                }


        }
        for(i = 0; i < n; i++)
                if(finish[i] == 0) return 0;

        return 1;
}

void requestCheck(){
        int cnd1 = 1, cnd2 = 1, p = 1;

        printf("\nAvailable resouces :\n");
        for(i = 0; i < r; i++) printf("%d\t",avail[i]);

        for(i = 0;i <r; i++){
                if(req[i] > avail[i]){
                printf("\nRequest cannot be greater than available
resources\n");
                cnd1 = 0;break;
                }
        }

        for(i = 0; i < r; i++){
                if(req[i] > need[p][i]){
                        cnd2 = 0;
                        printf("\nRequested resources cannot exceed the
need\n");
                        break;
                        }
        }

        if(cnd1 && cnd2){
                for(i = 0; i < r; i++){
                        avail[i]-=req[i];
                        alloc[p][i]+=req[i];
```

```
                        need[p][i] -= req[i];
                }

                if(!safestateCheck()){
                        printf("\nSystem is not in safe state\n");
                        printf("\nRequest cannot be granted!\n");

                        for(i = 0; i < r; i++){
                        avail[i]+=req[i];
                        alloc[p][i]-=req[i];
                        need[p][i] += req[i];
                        }
                }
                else printf("\nSystem is in safe state. Request can be
allocated.\n");
        }
        else printf("System is in unsafe state\n");
}


int main(){
        int choice;
        accept();
        do{
        printf("\n1.Display data\n2.Check Safestate\n3.Process
Request\n4.Quit\nENTER CHOICE\n");
        scanf("%d",&choice);

        switch(choice){
        case 1: display(); break;
        case 2: {
                int res = safestateCheck();
                if(res == 1){
                        printf("\nSystem is in safe state.\n");
                        for(i = 0; i < n; i++){
                                printf("P%d\t",seqn[i]);
                        }
                        printf("\n");
                }
                else printf("\nSystem is not in safe state\n");
                }
                break;
        case 3: requestCheck(); break;
        case 4: printf("\nBye!"); break;
        default : printf("\nWrong Choice, try again!");
        }}while(choice != 4);

        return 0;
}

/*
OUTPUT:

1.Display data
```

```
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
1

Allocation matrix
0       0       1       2
1       0       0       0
1       3       5       4
0       6       3       2
0       0       1       4

Maximum resources matrix
0       0       1       2
1       7       5       0
2       3       5       6
0       6       5       2
0       6       5       6

Need Matrix
0       0       0       0
0       7       5       0
1       0       0       2
0       0       2       0
0       6       4       2

Available resources matrix
1       5       2       0

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
2

System is in safe state.
P0      P2      P3      P4      P1

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
3

Available resouces :
1       5       2       0
System is in safe state. Request can be allocated.

1.Display data
2.Check Safestate
3.Process Request
```

```
4.Quit
ENTER CHOICE
4

Bye!
*/
--------------------
b3)
#include<stdio.h>
#include<stdlib.h>

#define MAX 10
int alloc[MAX][MAX];
int maxm[MAX][MAX];
int need[MAX][MAX];
int avail[MAX];
int work[MAX];
int seqn[MAX];
int finish[MAX];
int req[] = {0,0,1};

int n,r,i,j;

void accept(){
        n = 5; r = 3;
        int ralloc[5][3] = {{0,1,0},{2,0,0},{3,0,3},{2,1,1},{0,0,2}};
        int rmaxm[5][3] = {{0,1,0},{4,0,2},{3,0,3},{3,1,1},{0,0,1}};
        int ravail[3] = {0,0,0};

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                alloc[i][j] = ralloc[i][j];
        }

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        maxm[i][j] = rmaxm[i][j];
        }

        for(i = 0; i < n; i++)
                for(j = 0; j < r; j++)
                        need[i][j] = maxm[i][j] - alloc[i][j];

        for(i = 0; i < r; i++)
                avail[i] = ravail[i];

        for(i = 0; i < r; i++)
                work[i] = avail[i];
}


void display(){
        printf("\nAllocation matrix\n");
```

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

        printf("\nMaximum resources matrix\n");

        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",maxm[i][j]);
                printf("\n");
        }

        printf("\nNeed Matrix\n");
        for(i = 0; i < n; i++){
                for(j = 0; j < r; j++)
                        printf("%d\t",need[i][j]);
                printf("\n");
        }

        printf("\nAvailable resources matrix\n");
        for(i = 0; i < r; i++)
                printf("%d\t",avail[i]);

        printf("\n");
}

int safestateCheck(){
        int ind = 0, fl = 1, cnt;

        for(i = 0; i < n; i++) finish[i] = 0;

        for(i = 0; i < n; i++)
        for(j = 0; j < r; j++){
                if(need[i][j] < 0){
                        printf("\nAllocated resources are more than maximum
available resources of P%d\n",i);
                        return 0;

                }
        }

        while (fl){
                fl = 0;
                for(i = 0; i < n; i++){
                        if(finish[i] == 0){
                                cnt = 0;
                                for(j = 0; j < r; j++){
                                        if(need[i][j] <= work[j])
                                                        cnt++;
                                        else
                                                break;
                                }
```

```
                                if(cnt == r){
                                        for(j = 0; j < r; j++)
                                                work[j]+=alloc[i][j];
                                        seqn[ind++] = i;
                                        finish[i] = 1;
                                        fl = 1;
                                }
                        }
                }


        }
        for(i = 0; i < n; i++)
                if(finish[i] == 0) return 0;

        return 1;
}

void requestCheck(){
        int cnd1 = 1, cnd2 = 1, p = 4;

        printf("\nAvailable resouces :\n");
        for(i = 0; i < r; i++) printf("%d\t",avail[i]);

        for(i = 0;i <r; i++){
                if(req[i] > avail[i]){
                printf("\nRequest cannot be greater than available
resources\n");
                cnd1 = 0;break;
                }
        }

        for(i = 0; i < r; i++){
                if(req[i] > need[p][i]){
                        cnd2 = 0;
                        printf("\nRequested resources cannot exceed the
need\n");
                        break;
                        }
        }

        if(cnd1 && cnd2){
                for(i = 0; i < r; i++){
                        avail[i]-=req[i];
                        alloc[p][i]+=req[i];
                        need[p][i] -= req[i];
                }

                if(!safestateCheck()){
                        printf("\nSystem is not in safe state\n");
                        printf("\nRequest cannot be granted!\n");

                        for(i = 0; i < r; i++){
```

```
                        avail[i]+=req[i];
                        alloc[p][i]-=req[i];
                        need[p][i] += req[i];
                        }
                }
                else printf("\nSystem is in safe state. Request can be
allocated.\n");
        }
        else printf("System is in unsafe state\n");
}


int main(){
        int choice;
        accept();
        do{
        printf("\n1.Display data\n2.Check Safestate\n3.Process
Request\n4.Quit\nENTER CHOICE\n");
        scanf("%d",&choice);

        switch(choice){
        case 1: display(); break;
        case 2: {
                int res = safestateCheck();
                if(res == 1){
                        printf("\nSystem is in safe state.\n");
                        for(i = 0; i < n; i++){
                                printf("P%d\t",seqn[i]);
                        }
                        printf("\n");
                }
                else printf("\nSystem is not in safe state\n");
                }
                break;
        case 3: requestCheck(); break;
        case 4: printf("\nBye!"); break;
        default : printf("\nWrong Choice, try again!");
        }}while(choice != 4);

        return 0;
}

/*
OUTPUT:

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
1

Allocation matrix
0       1       0
```

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

Maximum resources matrix
0       1       0
4       0       2
3       0       3
3       1       1
0       0       1

Need Matrix
0       0       0
2       0       2
0       0       0
1       0       0
0       0       -1

Available resources matrix
0       0       0

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
2

Allocated resources are more than maximum available resources of P4

System is not in safe state

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
3

Available resouces :
0       0       0
Request cannot be greater than available resources

Requested resources cannot exceed the need
System is in unsafe state

1.Display data
2.Check Safestate
3.Process Request
4.Quit
ENTER CHOICE
4
```

```
Bye!
*/
```