SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 6

Implementation of Producer/Consumer Problem using Semaphores

Submission Date:21-04-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

1. Assignment 1: Develop a C program to implement Banker's algorithm for deadlock avoidance with multiple instances of resource types

Algorithm:

```
1) Get choice from user
2) If choice is equal to 1, then ask for data input
3) If choice is equal to 2, then print the data
4) If choice is equal to 3, then call bankers algorithm to execute
5) If choice is equal to 4 then call for resource request
6) Else exit
```

Algorithm for data input:

```
1) Get no of process, no of resources, available instances, maximum required matrix,
   allotted instances from user
2) Calculate need matrix by subtracting allocation matrix from maximum required matrix
```

Algorithm for bankers algorithm:

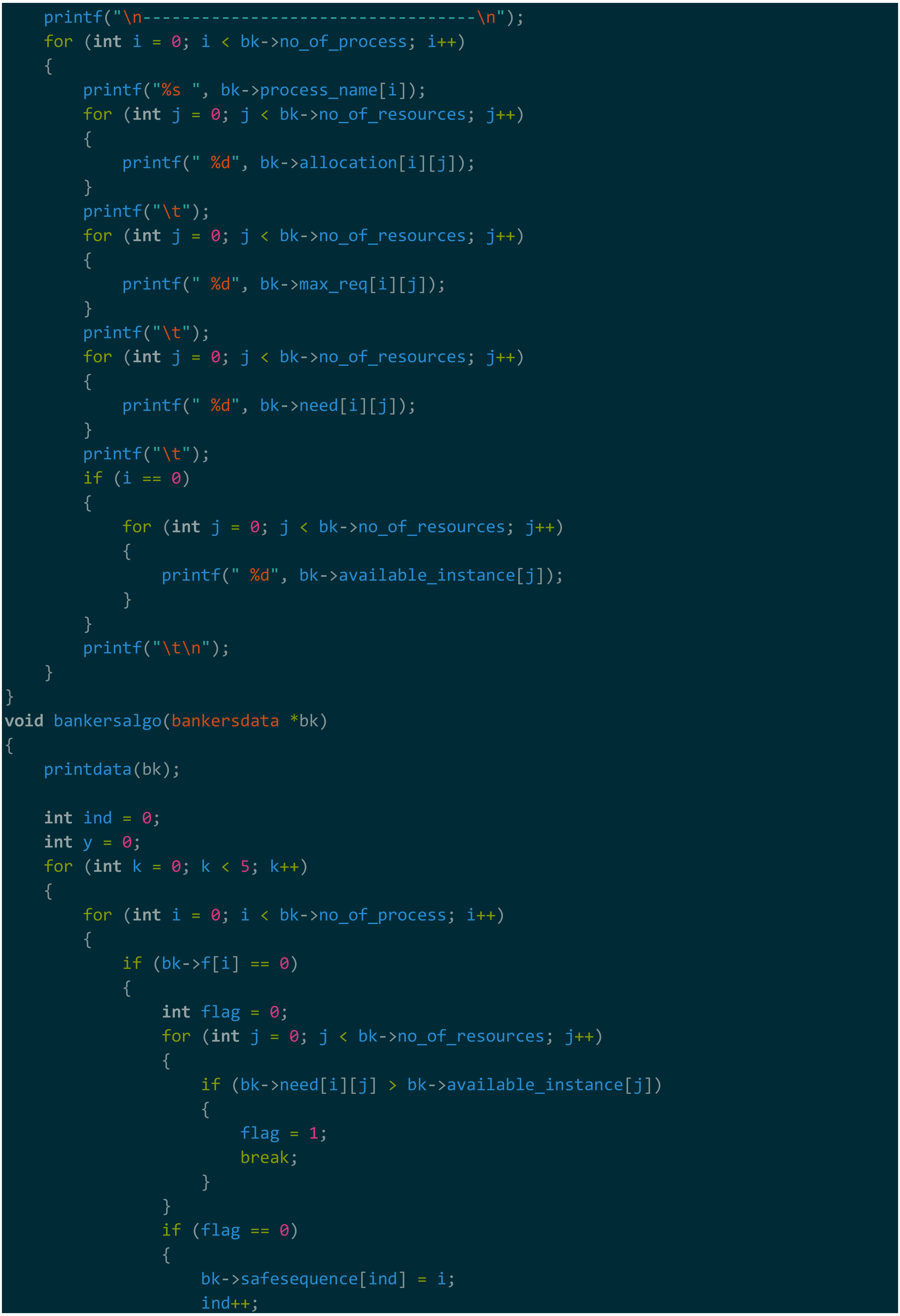
```
1) Set ind to 0
2) For i from 0 to no of process times
    a) If f[i] is equal to 0
        i. Set flag = 0
    b) For j no of resources times
        i. If need[I,j]greater than available[j] then set flag = 1
    c) If flag equal to 0 then
        i. Set safeseq[ind]=i
        ii. Increment ind
        iii. Set available [j] equal to sum of available[j] and allocstion[I,j]
        iv. Set f[i] is equal to 1
3) Set flag equal to 1
4) For i from 0 to no of process
    a) If f[i] is equal to 0 then print not a safe sequence and exit
    b) If flag is equal to 1 then print safe sequence
```

Algorithm for Resource Request:

1) Get process no from user to allocate
2) Get resource vector
3) Check if allocation[i] is greater than available[i] then exit
4) Else set available equal to available[i] minus allocationvector[i] and allocation[ind][i] equal to sum of allocation[ind][i] and allocationvector[i]
5) Now call bankers algorithm

Code:

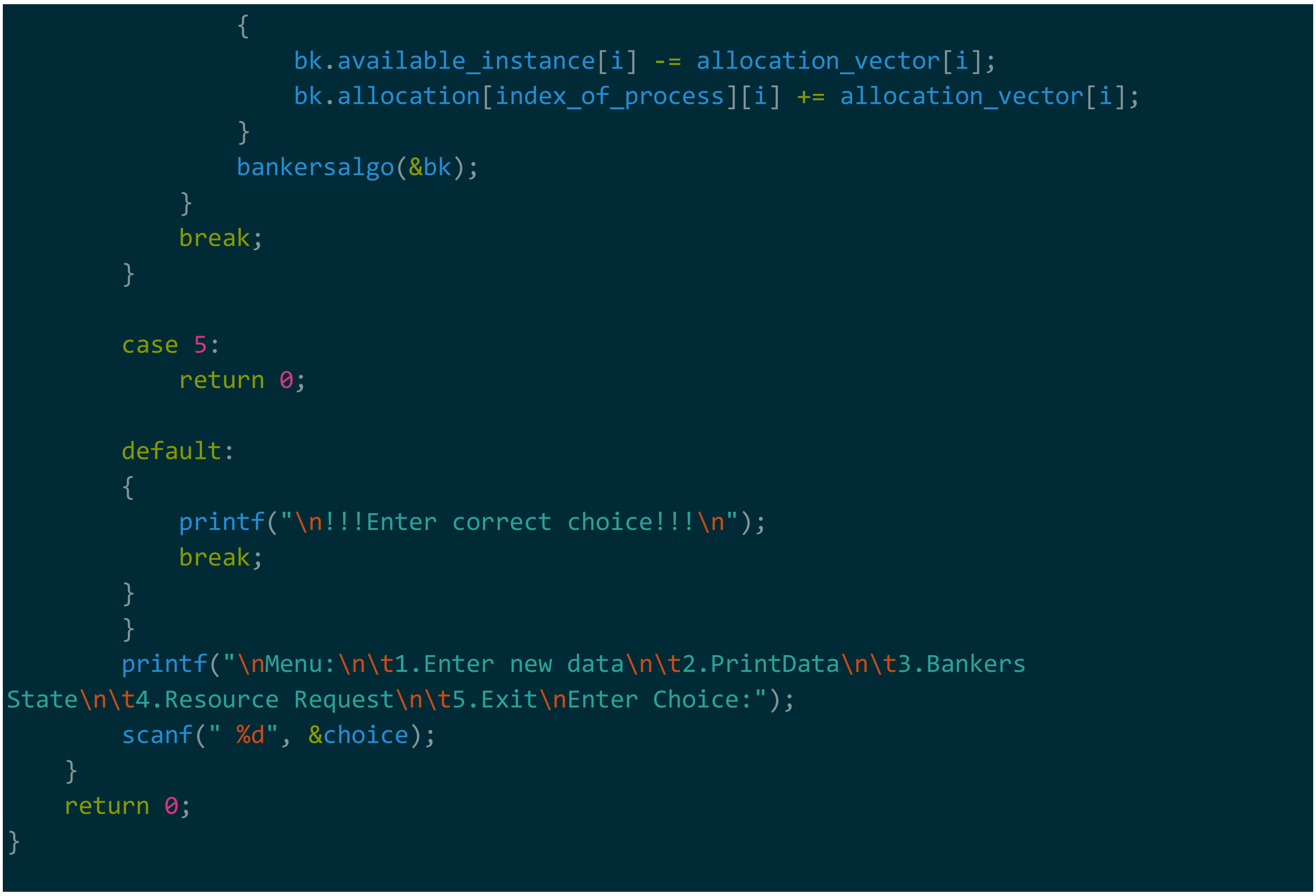
```c
// Assignment 1: Develop a C program to implement Banker’s algorithm for deadlock
avoidance with multiple instances of resource types

#include <stdio.h>
#include <string.h>
#define max 100

typedef struct bankersdata
{
    int no_of_process;
    int no_of_resources;
    char process_name[max][5];
    char resources_name[max][5];
    int available_instance[max];   // available_instance[no_of_resources]
    int max_req[max][max];         // max_req[no_of_process][no_of_resources];
    int allocation[max][max];      // allocation[no_of_process][no_of_resources];
    int f[max], safesequence[max]; // f[no_of_process],safesequence[no_of_process]
    int need[max][max];
} bankersdata;

void printdata(bankersdata *bk)
{
    printf("Pid\tAlloc\tMax  \tNeed \tAvail\t\n--\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    printf("\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    printf("\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    printf("\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    // printf("\n");
    printf("\t");
```



```
    printf("\n----------------------------------\n");
    for (int i = 0; i < bk->no_of_process; i++)
    {
        printf("%s ", bk->process_name[i]);
        for (int j = 0; j < bk->no_of_resources; j++)
        {
            printf(" %d", bk->allocation[i][j]);
        }
        printf("\t");
        for (int j = 0; j < bk->no_of_resources; j++)
        {
            printf(" %d", bk->max_req[i][j]);
        }
        printf("\t");
        for (int j = 0; j < bk->no_of_resources; j++)
        {
            printf(" %d", bk->need[i][j]);
        }
        printf("\t");
        if (i == 0)
        {
            for (int j = 0; j < bk->no_of_resources; j++)
            {
                printf(" %d", bk->available_instance[j]);
            }
        }
        printf("\t\n");
    }
}
void bankersalgo(bankersdata *bk)
{
    printdata(bk);

    int ind = 0;
    int y = 0;
    for (int k = 0; k < 5; k++)
    {
        for (int i = 0; i < bk->no_of_process; i++)
        {
            if (bk->f[i] == 0)
            {
                int flag = 0;
                for (int j = 0; j < bk->no_of_resources; j++)
                {
                    if (bk->need[i][j] > bk->available_instance[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0)
                {
                    bk->safesequence[ind] = i;
                    ind++;
                    for (int j = 0; j < bk->no_of_resources; j++)
                    {
                        bk->available_instance[j] += bk->allocation[i][j];
                    }
                    bk->f[i] = 1;
                }
            }
        }
    }

    int flag = 1;
    for (int i = 0; i < bk->no_of_process; i++)
    {
        if (bk->f[i] == 0)
        {
            flag = 0;
            printf("NOT A SAFE SYSTEM");
            break;
        }
    }
    if (flag == 1)
    {
        printf("SAFE SEQUENCE\n");
        for (int i = 0; i < bk->no_of_process - 1; i++)
        {
            printf(" %s ->", bk->process_name[bk->safesequence[i]]);
        }
        printf(" %s", bk->process_name[bk->safesequence[bk->no_of_process - 1]]);
    }
    printdata(bk);
}

bankersdata getdata()
{
    bankersdata bk;
    printf("\nEnter no of process:");
    scanf("%d", &bk.no_of_process);
    printf("\nEnter process ids:\n");
    for (int i = 0; i < bk.no_of_process; i++)
    {
        printf("process name of process %d:", i + 1);
        scanf(" %s", &bk.process_name[i]);
    }

    printf("\nEnter no of resources:");
    scanf("%d", &bk.no_of_resources);
    printf("\nEnter resource ids:\n");
    for (int i = 0; i < bk.no_of_resources; i++)
    {
        printf("resource name of resource %d:", i + 1);
        scanf(" %s", &bk.resources_name[i]);
    }

    printf("\nEnter available instances:\n");
```

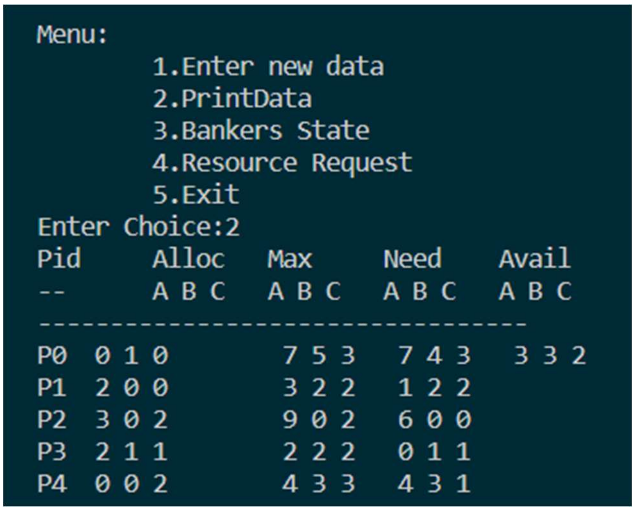
```c
    for (int i = 0; i < bk.no_of_resources; i++)
    {
        printf("available instances of resource %s:", bk.resources_name[i]);
        scanf(" %d", &bk.available_instance[i]);
    }

    printf("\nEnter Maximum requirement:\n");
    for (int i = 0; i < bk.no_of_process; i++)
    {
        printf("Maximum requirement for process %s:", bk.process_name[i]);
        for (int j = 0; j < bk.no_of_resources; j++)
        {
            scanf(" %d", &bk.max_req[i][j]);
        }
    }

    printf("\nEnter Allocated instances:\n");
    for (int i = 0; i < bk.no_of_process; i++)
    {
        printf("Allocated instances for process %s:", bk.process_name[i]);
        for (int j = 0; j < bk.no_of_resources; j++)
        {
            scanf(" %d", &bk.allocation[i][j]);
        }
    }

    for (int i = 0; i < bk.no_of_process; i++)
    {
        bk.f[i] = 0;
        for (int j = 0; j < bk.no_of_resources; j++)
        {
            bk.need[i][j] = bk.max_req[i][j] - bk.allocation[i][j];
        }
    }
    return bk;
}

int main(int argc, char const *argv[])
{
    bankersdata bk = getdata();
    int choice = 0;
    printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers State\n\t4.Resource
Request\n\t5.Exit\nEnter Choice:");
    scanf(" %d", &choice);
    while (choice)
    {
        switch (choice)
        {
        case 1:
        {
            bk = getdata();
            break;
        }
        case 2:
```

```
        {
            printdata(&bk);
            break;
        }

        case 3:
        {
            bankersalgo(&bk);
            break;
        }

        case 4:
        {
            char temp_process_name[5];
            printf("\nEnter process id for request:");
            scanf(" %s", &temp_process_name);
            int index_of_process = -1;
            for (int i = 0; i < bk.no_of_process; i++)
            {
                if (strcmp(temp_process_name, bk.process_name[i]) == 0)
                {
                    index_of_process = i;
                    break;
                }
            }
            if (index_of_process == -1)
            {
                printf("\nprocess name not correct!!!\n");
                break;
            }
            else
            {
                int allocation_vector[max];
                printf("\nEnter the request vector for %s:",
bk.process_name[index_of_process]);
                for (int i = 0; i < bk.no_of_resources; i++)
                {
                    scanf(" %d", &allocation_vector[i]);
                }
                int flag = 1;
                for (int i = 0; i < bk.no_of_resources; i++)
                {
                    if (allocation_vector[i] > bk.available_instance[i])
                    {
                        flag = 0;
                        break;
                    }
                }
                if (flag == 0)
                {
                    printf("\n!!!Resource cannot be allocated!!!");
                    break;
                }
                for (int i = 0; i < bk.no_of_resources; i++)
```

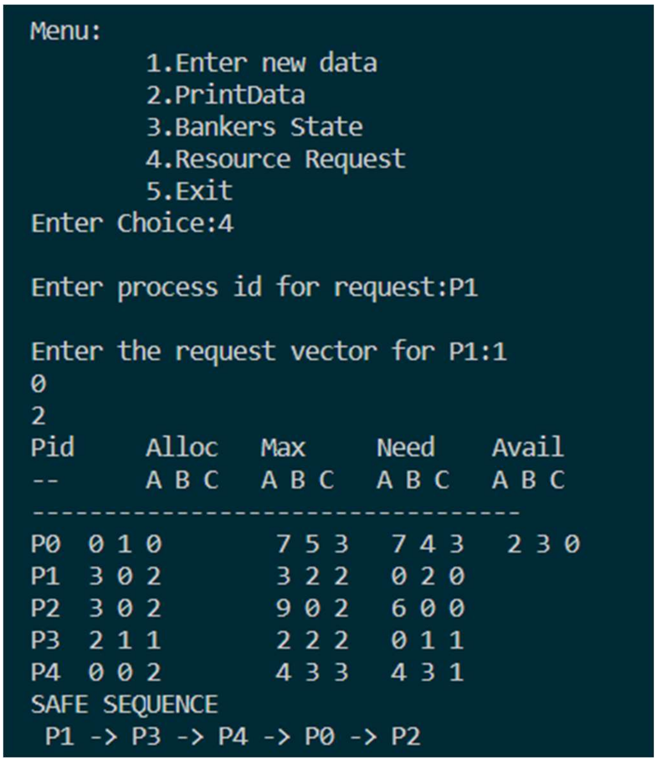


```
                {
                    bk.available_instance[i] -= allocation_vector[i];
                    bk.allocation[index_of_process][i] += allocation_vector[i];
                }
                bankersalgo(&bk);
            }
            break;
        }

        case 5:
            return 0;

        default:
        {
            printf("\n!!!Enter correct choice!!!\n");
            break;
        }
        }
        printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers 
State\n\t4.Resource Request\n\t5.Exit\nEnter Choice:");
        scanf(" %d", &choice);
    }
    return 0;
}
```

Output:



```
Menu:
        1.Enter new data
        2.PrintData
        3.Bankers State
        4.Resource Request
        5.Exit
Enter Choice:2
Pid     Alloc   Max     Need    Avail
--      A B C   A B C   A B C   A B C
----------------------------------
P0  0 1 0        7 5 3   7 4 3   3 3 2
P1  2 0 0        3 2 2   1 2 2
P2  3 0 2        9 0 2   6 0 0
P3  2 1 1        2 2 2   0 1 1
P4  0 0 2        4 3 3   4 3 1
```



```
Menu:
        1.Enter new data
        2.PrintData
        3.Bankers State
        4.Resource Request
        5.Exit
Enter Choice:4

Enter process id for request:P1

Enter the request vector for P1:1
0
2
Pid     Alloc   Max     Need    Avail
--      A B C   A B C   A B C   A B C
----------------------------------
P0  0 1 0        7 5 3   7 4 3   2 3 0
P1  3 0 2        3 2 2   0 2 0
P2  3 0 2        9 0 2   6 0 0
P3  2 1 1        2 2 2   0 1 1
P4  0 0 2        4 3 3   4 3 1
SAFE SEQUENCE
 P1 -> P3 -> P4 -> P0 -> P2
```

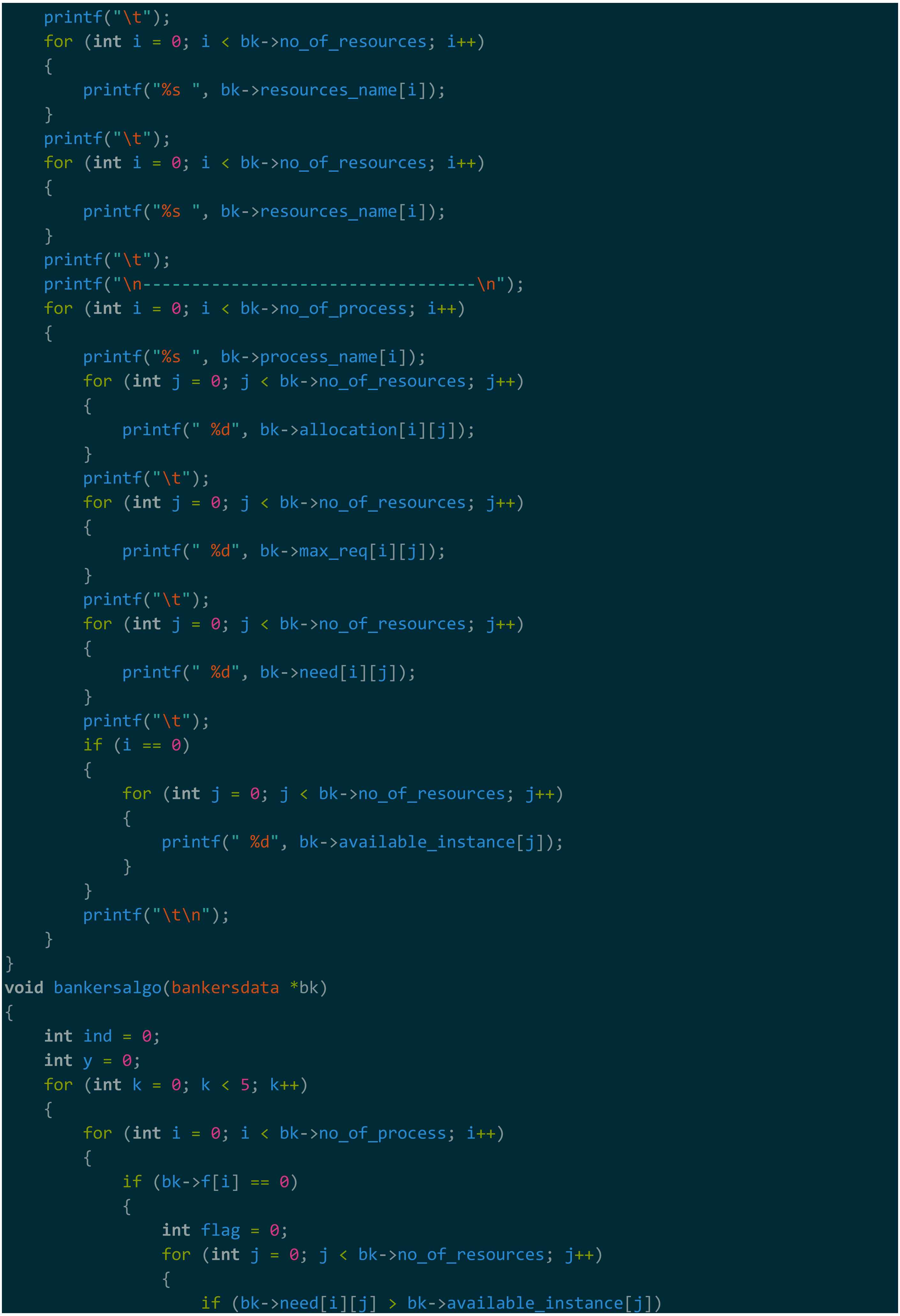
2. Assignment 2: Develop a C program to implement algorithm for deadlock detection with multiple instances of resource types and display the processes involved in deadlock

Algorithms:

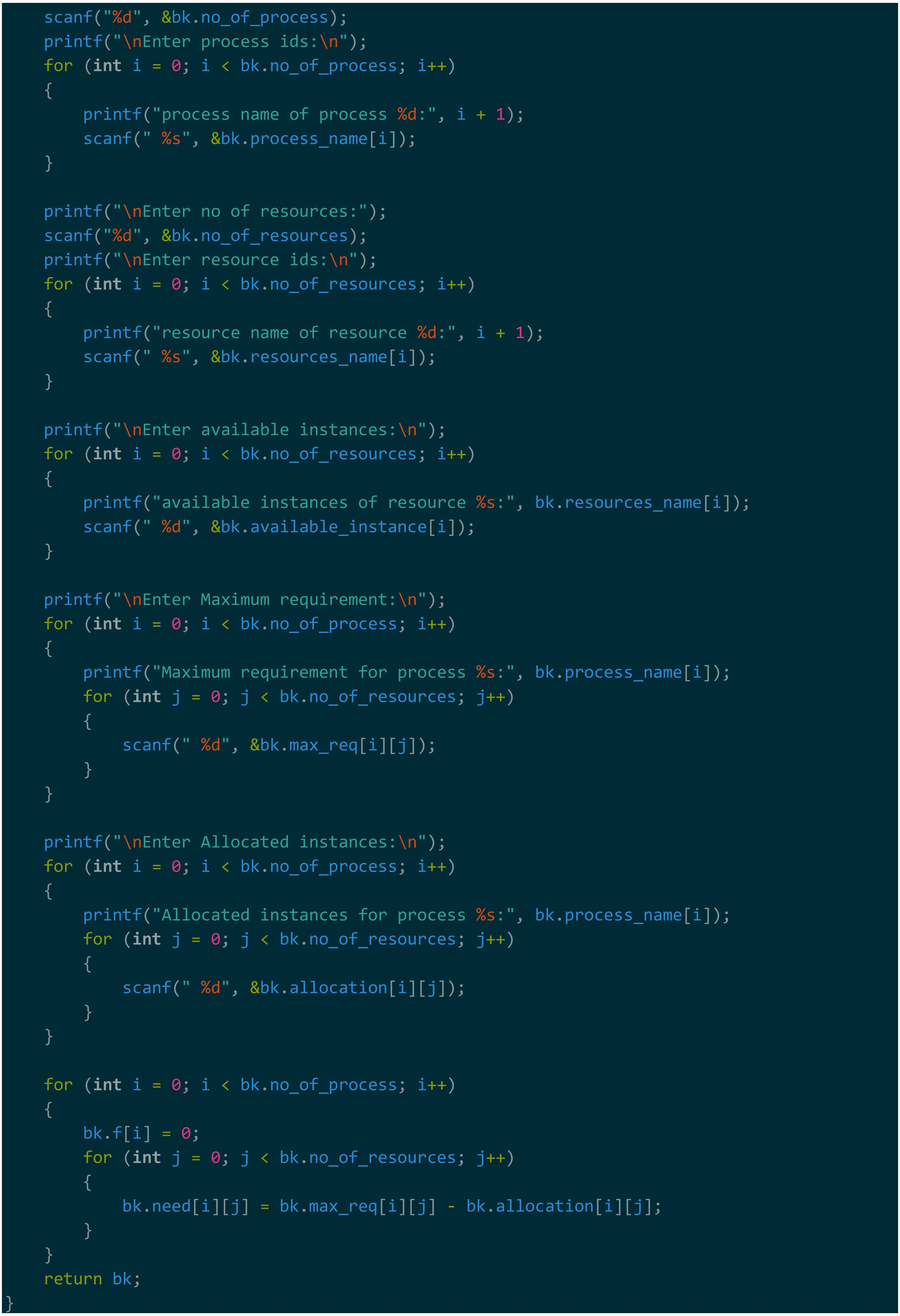
```
1) Get no of process, no of resources, available instances, maximum required matrix,
   allotted instances from user
2) Calculate need matrix by subtracting allocation matrix from maximum required matrix
3) Set ind to 0
4) For i from 0 to no of process times
      c) If f[i] is equal to 0
            i.   Set flag = 0
      d) For j no of resources times
            i.   If need[I,j]greater than available[j] then set flag = 1
      e) If flag equal to 0 then
            i.   Set safeseq[ind]=i
            ii.  Increment ind
            iii. Set available [j] equal to sum of available[j] and allocstion[I,j]
            iv.  Set f[i] is equal to 1
5) Set flag equal to 1
6) For i from 0 to no of process
      f) If f[i] is equal to 0 then print not a safe sequence and exit
      g) If flag is equal to 1 then print safe sequence
```

Code:

```
#include <stdio.h>
//#include <conio.h>
#include <string.h>
#define max 100

typedef struct bankersdata
{
    int no_of_process;
    int no_of_resources;
    char process_name[max][5];
    char resources_name[max][5];
    int available_instance[max];   // available_instance[no_of_resources]
    int max_req[max][max];         // max_req[no_of_process][no_of_resources];
    int allocation[max][max];      // allocation[no_of_process][no_of_resources];
    int f[max], safesequence[max]; // f[no_of_process],safesequence[no_of_process]
    int need[max][max];
} bankersdata;

void printdata(bankersdata *bk)
{
    printf("Pid\tAlloc\tMax  \tNeed \tAvail\t\n--\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    printf("\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
```



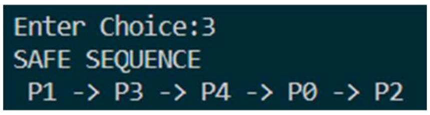
```c
    printf("\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    printf("\t");
    for (int i = 0; i < bk->no_of_resources; i++)
    {
        printf("%s ", bk->resources_name[i]);
    }
    printf("\t");
    printf("\n----------------------------------\n");
    for (int i = 0; i < bk->no_of_process; i++)
    {
        printf("%s ", bk->process_name[i]);
        for (int j = 0; j < bk->no_of_resources; j++)
        {
            printf(" %d", bk->allocation[i][j]);
        }
        printf("\t");
        for (int j = 0; j < bk->no_of_resources; j++)
        {
            printf(" %d", bk->max_req[i][j]);
        }
        printf("\t");
        for (int j = 0; j < bk->no_of_resources; j++)
        {
            printf(" %d", bk->need[i][j]);
        }
        printf("\t");
        if (i == 0)
        {
            for (int j = 0; j < bk->no_of_resources; j++)
            {
                printf(" %d", bk->available_instance[j]);
            }
        }
        printf("\t\n");
    }
}
void bankersalgo(bankersdata *bk)
{
    int ind = 0;
    int y = 0;
    for (int k = 0; k < 5; k++)
    {
        for (int i = 0; i < bk->no_of_process; i++)
        {
            if (bk->f[i] == 0)
            {
                int flag = 0;
                for (int j = 0; j < bk->no_of_resources; j++)
                {
                    if (bk->need[i][j] > bk->available_instance[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0)
                {
                    bk->safesequence[ind] = i;
                    ind++;
                    for (int j = 0; j < bk->no_of_resources; j++)
                    {
                        bk->available_instance[j] += bk->allocation[i][j];
                    }
                    bk->f[i] = 1;
                }
            }
        }
    }

    int flag = 1;
    for (int i = 0; i < bk->no_of_process; i++)
    {
        if (bk->f[i] == 0)
        {
            flag = 0;
            printf("\n\n!!!!NOT A SAFE SYSTEM!!!!\nDue to the following processes:");
            break;
        }
    }
    if (flag == 1)
    {
        printf("SAFE SEQUENCE\n");
        for (int i = 0; i < bk->no_of_process - 1; i++)
        {
            printf(" %s ->", bk->process_name[bk->safesequence[i]]);
        }
        printf(" %s", bk->process_name[bk->safesequence[bk->no_of_process - 1]]);
    }
    else
    {
        for (int i = 0; i < bk->no_of_process; i++)
        {
            if (bk->f[i] == 0)
            {
                printf(" %s ", bk->process_name[i]);
            }
        }
    }
}

bankersdata getdata()
{
    bankersdata bk;
    printf("\nEnter no of process:");
```



```
    scanf("%d", &bk.no_of_process);
    printf("\nEnter process ids:\n");
    for (int i = 0; i < bk.no_of_process; i++)
    {
        printf("process name of process %d:", i + 1);
        scanf(" %s", &bk.process_name[i]);
    }

    printf("\nEnter no of resources:");
    scanf("%d", &bk.no_of_resources);
    printf("\nEnter resource ids:\n");
    for (int i = 0; i < bk.no_of_resources; i++)
    {
        printf("resource name of resource %d:", i + 1);
        scanf(" %s", &bk.resources_name[i]);
    }

    printf("\nEnter available instances:\n");
    for (int i = 0; i < bk.no_of_resources; i++)
    {
        printf("available instances of resource %s:", bk.resources_name[i]);
        scanf(" %d", &bk.available_instance[i]);
    }

    printf("\nEnter Maximum requirement:\n");
    for (int i = 0; i < bk.no_of_process; i++)
    {
        printf("Maximum requirement for process %s:", bk.process_name[i]);
        for (int j = 0; j < bk.no_of_resources; j++)
        {
            scanf(" %d", &bk.max_req[i][j]);
        }
    }

    printf("\nEnter Allocated instances:\n");
    for (int i = 0; i < bk.no_of_process; i++)
    {
        printf("Allocated instances for process %s:", bk.process_name[i]);
        for (int j = 0; j < bk.no_of_resources; j++)
        {
            scanf(" %d", &bk.allocation[i][j]);
        }
    }

    for (int i = 0; i < bk.no_of_process; i++)
    {
        bk.f[i] = 0;
        for (int j = 0; j < bk.no_of_resources; j++)
        {
            bk.need[i][j] = bk.max_req[i][j] - bk.allocation[i][j];
        }
    }
    return bk;
}
```

```
int main(int argc, char const *argv[])
{
    bankersdata bk = getdata();
    int choice = 0;
    printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers State\n\t4.Exit\nEnter
Choice:");
    scanf(" %d", &choice);
    while (choice)
    {
        switch (choice)
        {
        case 1:
        {
            bk = getdata();
            break;
        }
        case 2:
        {
            printdata(&bk);
            break;
        }

        case 3:
        {
            bankersalgo(&bk);
            break;
        }

        case 4:
        {
            return 0;
            break;
        }

        default:
        {
            printf("\n!!!Enter correct choice!!!\n");
            break;
        }
        }
        printf("\nMenu:\n\t1.Enter new data\n\t2.PrintData\n\t3.Bankers
State\n\t4.Exit\nEnter Choice:");
        scanf(" %d", &choice);
    }
    return 0;
}
```

Output:



```
Enter Choice:3
SAFE SEQUENCE
 P1 -> P3 -> P4 -> P0 -> P2
```

Learning Outcome:

- Bankers algorithm implementation
- Importance of deadlock prevention
- Printing safe Sequence