B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,
Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Course – OBJECT ORIENTED PROGRAMMING USING C++
Course Code –18IS3PCOOP
AY 2020-21

Final report on mini Project work

Banker's Algorithm

Submitted to – Faculty Name: Shubha V Rao

Submitted by -

Moksh Jayanth GR 1BM19IS094

Mohan D 1BM19IS092

B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,
Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that the Project has been successfully presented at **B.M.S College Of Engineering** by **Moksh Jayanth G R** And **Mohan D** bearing USN:**1BM19IS094** And USN:**1BM19IS092** respectively in partial fulfilment of the requirements for the III Semester degree in **Bachelor of Engineering in Information Science & Engineering** of **Visvesvaraya Technological University, Belgaum** as a part the course **OBJECT ORIENTED PROGRAMMING USING C++*(*18IS3PCOOP)** during academic year 2020-2021.

Faculty Name – Shubha V Rao
Designation – Assistant Professor
Department of ISE, BMSCE

INDEX

Section 1	
1.1 Abstract	4
1.2 Introduction	5
1.3OOPs Concepts Used In The Project	6
1.4 Software Used	11
Section 2	
2.1 Implementation Of The Project's Code	12
2.2 Snapshots	19
Section 3	
3.1 Reference	21

ABSTRACT

The project that we have built using the concepts of object oriented programming in c++ aims to solve the problems of deadlock situations occurring in computer operating systems. The project makes of an famous algorithm called 'Banker's Algorithm' to solve the problems.

The project uses classes and objects to solve the problem. To make the program efficient, the project uses dynamic allocation of class's member variables. The project also makes use of modular programming, so along with the main(.cpp) file, there are two user defined header(.h) files.

The project's another subsidiary aim is to solve deadlock problems and write the complete solution of the problem into a '.txt' file using the 'files' concept of c++. The file can be used to store the problem's solution and can be referred anytime in future.

INTRODUCTION

Following points provides an quick introduction to our project.

Our Project :-

a) is designed to solve the deadlock problems arising in the operating systems using the concepts of c++.
b) makes use of many of OOPs concepts of c++ to solve the problem. Concepts like *classes and objects, dynamic allocation, operator overloading, Files, Exceptions, friend functions,* and many more.
c) consists of four files in total, one *main(.cpp)* file, two user defined header(.h) files and one text(.txt) file.
d) has a '*main(.cpp)*' file which contains main( ) function, all the class's objects and some significant variables essential for the program are declared in the main(.cpp) file.
e) has '*class(.h)*' file which contains definition of a single class called '*Matrix*'. The '*Matrix*' class has a single member variable, a dynamic parameterized constructor, couple of member functions and friend functions.
f) has *'functions(.h)'* file which contains all the functions declarations and definitions which are required for the program.
g) has *BankersFile(.txt)* file. The program writes the complete solution of the problem into this file.

OOPS CONCEPTS USED IN MAKING THIS PROJECT

a) Classes And Objects

A class is a blue-print of the object.
In the project we have a class named '*Matrix*' which has its own member variables and member functions. The member variables and functions could be accessed by creating entities what are known as *'Matrix'* objects.
An object is an instance of class.

Code snippet of *Matrix* Class definition :

```
//Matrix class
class Matrix{
    public:
    int** arrayPtr;
    int m,n;

    public:
    Matrix(int j,int k)  //Parametized Dynamic Constructor
    {
        m=j;n=k;
        arrayPtr= new int*[m];
        for(int i=0;i<m;i++)
        arrayPtr[i]=new int [n];
        for(int i=0;i<m;i++)
        {
        for(int j=0;j<n;j++)
        arrayPtr[i][j]=0;
        }
    }
    friend istream& operator>>(istream& ,Matrix);
    friend ostream& operator<<(ostream& ,Matrix);
    Matrix operator-(Matrix);
};
```

Code snippet of Object creation & implementation :

```
    Matrix allocationMatrix(pros, resources);     //Creating Matrix Objects
    Matrix maxMatrix(pros, resources);
    Matrix availableMatrix(1, resources);
    Matrix needMatrix(pros, resources);
```

```
needMatrix = maxMatrix - allocationMatrix;   //Calculating Need matrix
```

b) Parameterized Constructor

Constructors are the member functions used to initialize the objects of its class. It is invoked whenever an object of its Class is created.
In our project, the code snippet given below shows a parameterized constructor which takes two parameters which in turn hold the values of the dimensions of the 2-d matrix.(see the snippet below).

Code Snippet of Parameterized Constructor

```
Matrix(int j,int k)  //Parametized Constructor
    {
        m=j;n=k;
        arrayPtr= new int*[m];
        for(int i=0;i<m;i++)
        arrayPtr[i]=new int [n];
        for(int i=0;i<m;i++)
        {
        for(int j=0;j<n;j++)
        arrayPtr[i][j]=0;
        }
    }
```

c) Dynamic Initialization Of Variables

It is used to initialize the variables at the runtime, the memory is allocated at the runtime so no extra amount of memory is wasted.
Here the *arrayPtr* is the variable which is allocated with the memory dynamically.

Code Snippet of dynamically allocated '*arrayPtr*'

```
int** arrayPtr;
    int m,n;

    public:
    Matrix(int j,int k)  //Parametized Constructor
    {
        m=j;n=k;
        arrayPtr= new int*[m];
        for(int i=0;i<m;i++)
        arrayPtr[i]=new int [n];
        for(int i=0;i<m;i++)
        {
        for(int j=0;j<n;j++)
        arrayPtr[i][j]=0;
        }
    }
```

d) Files

This project uses concepts of files to write the problem's solution into a text file called '*BankersFile.txt*', the text file can be used to store the data for future use, the code Snippet of which is given below ;

Code snippet of writing data into *BankersFile*

```
fstream BankersFile;
   BankersFile.open("BankersFile.txt", ios::out);
   if (!BankersFile) {
      cout << "Error: Failed To Open BankersFile.\n ";
      return;
   }

   BankersFile<< "AVAILABLE MATRIX:\n"<< availableMatrix;
   BankersFile<<"ALLOCATION MATRIX:\n"<< allocationMatrix;
   BankersFile<<"MAX MATRIX:\n"<<maxMatrix;
   BankersFile<<"NEED MATRIX:\n"<<needMatrix;


   if(checkSafetyFlag){

      BankersFile << "SYSTEM IS IN SAFE STATE.\nSAFE"
       " SEQUENCE IS => ";
       int i;
       for (i = 0; i < pros-1 ; i++)
          BankersFile <<"P"<<"["<< safeSeq[i] << "]"<<" -> ";
      BankersFile <<"P"<<"["<< safeSeq[i] << "]"<<endl;

   }
   else BankersFile<<"THERE IS A DEADLOCK POSSIBILITY. THE SYSTEM IS NOT IN SAFE STATE.\n";


   BankersFile.close();
```

e) ostream & istream Operator Overloading Using Friend Function

Stream insertion operator << is used for output and stream extraction operator >> is used for input.
In our project, stream insertion and stream extraction operators are overloaded to perform input and output for '*Matrix*' objects.

Code snippet of 'friend' function used for operator overloading

```
    friend istream& operator>>(istream& ,Matrix);
    friend ostream& operator<<(ostream& ,Matrix);
```

```
//1.For overloading >> Operator
istream& operator>>(istream& in,Matrix obj){
    for(int i=0;i<obj.m;i++){
        cout<<'P'<<'['<<i<<']'<<':';
        for(int j=0;j<obj.n;j++)in>>obj.arrayPtr[i][j];
    }
    return(in);
}
//2.For overloading << Operator
```

```
ostream& operator<<(ostream& out,Matrix obj){
    for(int i=0;i<obj.m;i++){
        for(int i=0;i<obj.n;i++)out<<"-----------------
";out<<"\n"; // For table design
        out<<"|";
        for(int j=0;j<obj.n;j++){
            out<<"\t"<<obj.arrayPtr[i][j]<<"\t"<<"|";
            }
        out<<"\n";
    }
    for(int i=0;i<obj.n;i++)out<<"-----------------
";out<<"\n"; // For table design
    return(out);
}
```

```
    cin>>availableMatrix;
    cout<<availableMatrix;
```

f) Binary Operator Overloading(-)

In our project, we have also implemented binary operator overloading for *Matrix* objects. Code snippet is given below:

Code snippet for Subtraction(-) Operator overloading

```
    Matrix operator-(Matrix);
```

```
Matrix Matrix :: operator-(Matrix obj)
{
    Matrix temp(m, n);
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)temp.arrayPtr[i][j]=arrayPtr[i][j]-
obj.arrayPtr[i][j];
    }
```

```
    needMatrix = maxMatrix - allocationMatrix;
```

g) Exceptions

In our project, we included exceptions in few of the areas of the project. We have made use of Exceptions in the following scenarios:

1. When the user enters an negative value for the number of processes/resources, the exception is thrown.
2. When the user enters an negative value for any of the allocation, max or the available matrix, the exception is thrown.
3. When the System is unsafe or enters the state of **deadlock**, the exception is thrown.

Code snippets for Exception handling.

1.

```
try{
   cout<<"ENTER THE NUMBER OF PROCESSES:\t";                    //Accepting User input
   cin>>pros;
   if(pros<=0){throw "YOU HAVE TO ENTER POSITVE VALUE ONLY, PLEASE RUN THE PROGRAM AGAIN AND
GIVE A POSITIVE VALUE.\n";}  //Exception
   cout<<"ENTER THE NUMBER OF TYPES OF RESOURCES('A', 'B', 'C', ...) AVAILABLE:\t";
   cin>>resources;
   if(resources<=0){throw "YOU HAVE TO ENTER POSITVE VALUE ONLY, PLEASE RUN THE PROGRAM AGAIN
AND GIVE A POSTIVE VALUE.\n";} //Exception
   }catch(const char* inputMsg){
      cerr<<inputMsg<<endl;return 0;
   }
```

2.

```
try{                                    //Exception
   cout<<"ENTER THE ELEMENTS FOR ALLOCATION MATRIX:\n";
   cin>>allocationMatrix;
   cout<<"ENTER THE ELEMENTS FOR MAX MATRIX:\n";
   cin>>maxMatrix;
   cout<<"ENTER THE ELEMENTS FOR AVAILABLE MATRIX:\n";
   cin>>availableMatrix;
   }catch(const char* matrixMsg){
      cerr<<matrixMsg<<endl;return 0;
   }
```

3.

```
try{
   //Check whether the system is in DeadLock State, if not generate safe sequence.
   checkSafety(allocationMatrix, needMatrix, availableMatrix,pros,resources);
}catch (const char* msg) {
    cerr << msg << endl;
    return 0;
}
```

```
if (found == false)
      {
         throw "THERE IS A DEADLOCK POSSIBILITY. THE SYSTEM IS NOT IN SAFE STATE.\n";//Exception
         return;
      }
```

SOFTWARES USED

- **Microsoft visual studio code - as code editor**

  Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

- **GCC version 10.2 - compiler**

  The GNU Compiler Collection is an optimizing compiler produced by the GNU Project supporting various programming languages, hardware architectures and operating systems. The Free Software Foundation distributes GCC as free software under the GNU General Public License.

IMPLEMENTATION OF THE BANKER'S ALGORITM CODE

The entire code for our project is made into three different files as follows:

1) *main.cpp*
2) *class.h*
3) *functions.h*

The codes in these individual files are shown below:

1) *main.cpp*

```
#include "functions.h"
#include<unistd.h>

int main(){
system("clear"); //Clears the console

    int pros, resources, choice=1;
    header(cout);               //Adds a header to the console
    try{
    cout<<"ENTER THE NUMBER OF PROCESSES:\t";                   //Accepting User input
    cin>>pros;
    if(pros<=0){throw "YOU HAVE TO ENTER POSITVE VALUE ONLY, PLEASE RUN THE PROGRAM AGAIN AND
GIVE A POSITIVE VALUE.\n";}  //Exception
    cout<<"ENTER THE NUMBER OF TYPES OF RESOURCES('A', 'B', 'C', ...) AVAILABLE:\t";
    cin>>resources;
    if(resources<=0){throw "YOU HAVE TO ENTER POSITVE VALUE ONLY, PLEASE RUN THE PROGRAM AGAIN
AND GIVE A POSTIVE VALUE.\n";} //Exception
    }catch(const char* inputMsg){
        cerr<<inputMsg<<endl;return 0;
    }

    Matrix allocationMatrix(pros, resources);          //Creating Matrix Objects
    Matrix maxMatrix(pros, resources);
    Matrix availableMatrix(1, resources);
    Matrix needMatrix(pros, resources);

    try{                                    //Exception
    cout<<"ENTER THE ELEMENTS FOR ALLOCATION MATRIX:\n";
    cin>>allocationMatrix;
    cout<<"ENTER THE ELEMENTS FOR MAX MATRIX:\n";
    cin>>maxMatrix;
    cout<<"ENTER THE ELEMENTS FOR AVAILABLE MATRIX:\n";
    cin>>availableMatrix;
    }catch(const char* matrixMsg){
        cerr<<matrixMsg<<endl;return 0;
    }
    while(choice){                              //Bigger and outermost loop
    cout<<"DISPLAYING YOUR PROBLEM'S DATA:\n";
    cout<<"AVAILABLE MATRIX:\n";
```

```
    cout<<availableMatrix;
    cout<<"ALLOCATION MATRIX:\n";
    cout<<allocationMatrix;
    cout<<"MAX MATRIX:\n";
    cout<<maxMatrix;
    cout<<"NEED MATRIX:\n";
    needMatrix = maxMatrix - allocationMatrix;   //Calculating Need matrix
    cout<<needMatrix;
try{
    //Check whether the system is in DeadLock State, if not generate safe sequence.
    checkSafety(allocationMatrix, needMatrix, availableMatrix,pros,resources);
}catch (const char* msg) {
     cerr << msg << endl;
     return 0;
   }
    //Here the Resource Request Algorithm starts
    cout<<"DO YOU WISH TO MAKE CHANGES TO PROCESS'S REQUEST? ENTER 1 TO PROCEED OR ENTER
0 TO QUIT.\n";
    cin>>choice;
    if(choice){

        int val,proNum;
        cout<<"---------------------------------------\n";
        cout<<"\n\nSELECT THE BELOW OPERATIONS:\n\n";
        cout<<"1.CHANGE MAX OF THE PROCESS: \n";
        cout<<"2.CHANGE ALLOCATION OF THE PROCESS\n";
        cout<<"3.CHANGE THE INTIAL AVAILABLE RESOURCES\n";
        cout<<"4.EXIT\n\n";
        cout<<"---------------------------------------\n";

        do{
            cout<<"ENTER YOUR NUMBER: ";
            cin>>val;
        switch(val){
        case 1:
        {
            cout<<"\n\nEnter Process No: ";
            cin>>proNum;
            cout<<"\nEnter New Max for P["<<proNum<<"]: ";
            for(int i=0;i<resources;i++)
                cin>>maxMatrix.arrayPtr[proNum][i];
                break;
        }
        case 2:
        {
            cout<<"\n\nEnter Process No: ";
            cin>>proNum;
            cout<<"\nEnter New Allocation for P["<<proNum<<"]: ";
            for(int i=0;i<resources;i++)
                cin>>allocationMatrix.arrayPtr[proNum][i];
                break;
        }
```

```
        case 3:
        {
            cout<<"\nEnter Initial Available Resources: ";
            for(int i=0;i<resources;i++)
                cin>>availableMatrix.arrayPtr[0][i];
                break;
        }
        case 4: break;
        default: cout<<"ENTER A VALID CHOICE.\n";
        }
    }while(val!=4);
    }
    else if (!choice) break;
    else {cout<<"PLEASE ENTER A VALID CHOICE.\n";break;}
    }

    //writing all the contents and results to BankersFile
    writeToFile(availableMatrix, allocationMatrix, maxMatrix, needMatrix);

    footer(cout);          //Adds a Footer to the console.

    return 0; //END OF THE MAIN FUNTION
}
```

2) ***class.h***

```
#include<iostream>
using namespace std;

//Matrix class
class Matrix{
    public:
    int** arrayPtr;
    int m,n;

    public:
    Matrix(int j,int k)  //Parametized Dynamic Constructor
    {
        m=j;n=k;
        arrayPtr= new int*[m];
        for(int i=0;i<m;i++)
        arrayPtr[i]=new int [n];
        for(int i=0;i<m;i++)
        {
        for(int j=0;j<n;j++)
        arrayPtr[i][j]=0;
        }
    }
    friend istream& operator>>(istream& ,Matrix);
    friend ostream& operator<<(ostream& ,Matrix);
    Matrix operator-(Matrix);
};                                  //END OF CLASS
```

```
//1.For overloading >> Operator
istream& operator>>(istream& in,Matrix obj){
    for(int i=0;i<obj.m;i++){
        cout<<'P'<<'['<<(i+1)<<']'<<':';
        for(int j=0;j<obj.n;j++){
            in>>obj.arrayPtr[i][j];
            if(obj.arrayPtr[i][j]<0)throw "YOU HAVE TO ENTER POSITVE VALUE ONLY, PLEASE RUN THE PROGRAM
AGAIN AND GIVE A POSTIVE VALUE.\n";
            }
    }
    return(in);
}
//2.For overloading << Operator
ostream& operator<<(ostream& out,Matrix obj){
    for(int i=0;i<obj.m;i++){
        for(int i=0;i<obj.n;i++)out<<"----------";out<<"\n"; // For table design
        out<<"|";
        for(int j=0;j<obj.n;j++){
            out<<"\t"<<obj.arrayPtr[i][j]<<"\t"<<"|";
            }
        out<<"\n";
    }
    for(int i=0;i<obj.n;i++)out<<"----------";out<<"\n"; // For table design
    return(out);
}
//3. For overloading - operator
Matrix Matrix :: operator-(Matrix obj){
    Matrix temp(m, n);
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)temp.arrayPtr[i][j]=arrayPtr[i][j]-obj.arrayPtr[i][j];
    }
    return temp;
}
```

3) ***functions.h***

```
#include "class.h"
#include<fstream>

//Global Variables
int checkSafetyFlag, pros;
int safeSeq[15]; // To store safe sequence

//1. A function to display header in the console
ostream& header(ostream& out){
    out<<"--------------------*A MINI PROJECT ON IMPLEMENTATION OF BANKER'S ALGORITHM*--------------------
\n\n";
    out<<"/* QUICK NOTE: ROWS IN TABLE REPRESENT PROCESSES(P0, P1, P2, ...., Pm) AND THE COLUMNS
REPRESENT THE RESOURCES (R0, R1, R2, ....., Rn) */\n\n\n";
```

```
    out<<"/*THIS TABLE IS JUST FOR REFERENCE,THE TABLE IS FOR 3 PROCESSES AND 3
RESOURCES.*/\n\n";

    //Design Reference Table
    int i =0;
        out<<"\tR1\t\tR2\t\tR3\t\n";
      for(i=0;i<3;i++){
        out<<"   ----------------------------------------\nP"<<(i+1);
        out<<"|";
        for(int j=0;j<3; j++) {out<<"\t\t|";}
        out<<"\n";
    }
    out<<"   ----------------------------------------\n";
    out<<"\n\n";
    return out;
}

//2. A fuction to display footer in the console
ostream& footer(ostream& out){
    out<<"\n\n-------------------*THANK YOU AND HAVE A GREAT DAY*-------------------\n\n";
    return out;
}

// 3. Function to find the system is in safe state or not
void checkSafety(Matrix allocationMatrix, Matrix needMatrix, Matrix availableMatrix,int process,int resources)
{
    pros=process;
    //Initializing checkSafetyFlag to 0.
    checkSafetyFlag=0;

    // Mark all processes as infinish
    bool finish[pros] ;for(int i=0;i<pros;i++)finish[i]=0;


    // Make a copy of available resources i.e WORK = AVAILABLE
    int work[resources];
    for (int i = 0; i < resources ; i++)
        work[i] = availableMatrix.arrayPtr[0][i];

    // While all processes are not finished
    // or system is not in safe state.
    int count = 0;
    while (count < pros)
    {
        // Find a process which is not finish and
        // whose needs can be satisfied with current
        // work[] resources.
        bool found = false;
        for (int p = 0; p < pros; p++)
        {
            // First check if a process is finished,
            // if no, go for next condition
```

```
            if (finish[p] == 0)
            {
                // Check if for all resources of
                // current P need is less
                // than work
                int j;
                for (j = 0; j < resources; j++)
                    if (needMatrix.arrayPtr[p][j] > work[j])
                        break;

                // If all needs of p were satisfied.
                if (j == resources)
                {
                    // Add the allocated resources of
                    // current P to the available/work
                    // resources i.e.free the resources
                    for (int k = 0 ; k < resources ; k++)
                        work[k] += allocationMatrix.arrayPtr[p][k];

                    // Add this process to safe sequence.
                    safeSeq[count++] = p;

                    // Mark this p as finished
                    finish[p] = 1;

                    found = true;
                }
            }
        }

        // If we could not find a next process in safe
        // sequence.
        if (found == false)
        {
            throw "THERE IS A DEADLOCK POSSIBILITY. THE SYSTEM IS NOT IN SAFE STATE.\n";//Exception
            return;
        }
    }

    // If system is in safe state then
    // safe sequence will be as below
    cout << "SYSTEM IS IN SAFE STATE.\nSAFE"
         " SEQUENCE IS => ";
    int i;
    for (i = 0; i < pros-1 ; i++)
        cout <<"P"<<"["<< (safeSeq[i]+1)<< "]"<<" -> ";
     cout <<"P"<<"["<< (safeSeq[i]+1) << "]"<<endl;
    //Initailizing checkSafetyFlag to 1.
    checkSafetyFlag=1;
}

// 4. A Function to write data to a file.
```

```
void writeToFile(Matrix availableMatrix, Matrix allocationMatrix, Matrix maxMatrix, Matrix needMatrix){
    fstream BankersFile;
    BankersFile.open("BankersFile.txt", ios::out);
    if (!BankersFile) {
        cout << "Error: Failed To Open BankersFile.\n ";
        return;
    }

    header(BankersFile);                            //Adds HeaderTemplate to the file
    BankersFile<< "AVAILABLE MATRIX:\n"<< availableMatrix;
    BankersFile<<"ALLOCATION MATRIX:\n"<< allocationMatrix;
    BankersFile<<"MAX MATRIX:\n"<<maxMatrix;
    BankersFile<<"NEED MATRIX:\n"<<needMatrix;


    if(checkSafetyFlag){

        BankersFile << "SYSTEM IS IN SAFE STATE.\nSAFE"
         " SEQUENCE IS => ";
         int i;
         for (i = 0; i < pros-1 ; i++)
            BankersFile <<"P"<<"["<< (safeSeq[i]+1) << "]"<<" -> ";
        BankersFile <<"P"<<"["<< (safeSeq[i]+1) << "]"<<endl;

    }
    else BankersFile<<"THERE IS NO POSSIBLE SAFE SEQUENCE, HENCE THE SYSTEM WILL IN THE STATE
OF  DEADLOCK.\n";

    footer(BankersFile);              //Adds FooterTemplate to the file

    BankersFile.close();
}
```

SNAPSHOTS OF THE PROJECT

```
/* QUICK NOTE: ROWS IN TABLE REPRESENT PROCESSES(P0, P1, P2, ...., Pm) AND THE COLUMNS REPRESENT THE RESOURCES (R0, R1, R2, ....., Rn) */


/*THIS TABLE IS JUST FOR REFERENCE,THE TABLE IS FOR 3 PROCESSES AND 3 RESOURCES.*/

       R1             R2             R3
    ------------------------------------------
P1|             |              |              |
    ------------------------------------------
P2|             |              |              |
    ------------------------------------------
P3|             |              |              |
    ------------------------------------------


ENTER THE NUMBER OF PROCESSES:  5
ENTER THE NUMBER OF TYPES OF RESOURCES('A', 'B', 'C', ...) AVAILABLE:   3
ENTER THE ELEMENTS FOR ALLOCATION MATRIX:
P[1]:0 1 0
P[2]:2 0 0
P[3]:3 0 2
P[4]:2 1 1
P[5]:0 0 2
ENTER THE ELEMENTS FOR MAX MATRIX:
P[1]:7 5 3
P[2]:3 2 2
P[3]:9 0 2
P[4]:2 2 2
P[5]:4 3 3
ENTER THE ELEMENTS FOR AVAILABLE MATRIX:
P[1]:3 3 2
DISPLAYING YOUR PROBLEM'S DATA:
AVAILABLE MATRIX:
-------------------------------------------------
|       3       |       3       |       2       |
-------------------------------------------------
ALLOCATION MATRIX:
-------------------------------------------------
|       0       |       1       |       0       |
-------------------------------------------------
|       2       |       0       |       0       |
-------------------------------------------------
|       3       |       0       |       2       |
-------------------------------------------------
|       2       |       1       |       1       |
-------------------------------------------------
|       0       |       0       |       2       |
-------------------------------------------------
MAX 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       |
-------------------------------------------------
SYSTEM IS IN SAFE STATE.
SAFE SEQUENCE IS => P[2] -> P[4] -> P[5] -> P[1] -> P[3]
DO YOU WISH TO MAKE CHANGES TO PROCESS'S REQUEST? ENTER 1 TO PROCEED OR ENTER 0 TO QUIT.
0


--------------------*THANK YOU AND HAVE A GREAT DAY*--------------------
```

Fig 1.1 – Program's output displayed on the console.

(P.T.O)

BankersFile

```
----------*A MINI PROJECT ON IMPLEMENTATION OF BANKER'S ALGORITHM*----------

/* QUICK NOTE: ROWS IN TABLE REPRESENT PROCESSES(P0, P1, P2, ...., Pm) AND THE COLUMNS REPRESENT THE RESOURCES (R0, R1, R2, ....., Rn) */

/*THIS TABLE IS JUST FOR REFERENCE,THE TABLE IS FOR 3 PROCESSES AND 3 RESOURCES.*/

      R1            R2            R3
   ___________________________________
P1|            |             |            |
   ___________________________________
P2|            |             |            |
   ___________________________________
P3|            |             |            |
   ___________________________________

AVAILABLE MATRIX:
_____________________________________
|      3      |      3      |      2      |
_____________________________________
ALLOCATION MATRIX:
_____________________________________
|      0      |      1      |      0      |
_____________________________________
|      2      |      0      |      0      |
_____________________________________
|      3      |      0      |      2      |
_____________________________________
|      2      |      1      |      1      |
_____________________________________
|      0      |      0      |      2      |
_____________________________________
MAX 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      |
_____________________________________
SYSTEM IS IN SAFE STATE.
SAFE SEQUENCE IS => P[2] -> P[4] -> P[5] -> P[1] -> P[3]

----------*THANK YOU AND HAVE A GREAT DAY*----------
```

Fig 1.2: Solution to the problem written into a file named 'BankersFile.txt'.

(P.T.O)

REFERENCES

- Stack Overflow - https://stackoverflow.com/
- Geeks for Geeks - https://www.geeksforgeeks.org/
- Data structures, Algorithms and Applications in C++ Second edition by Sartaj Sahni
- Java T point - https://www.javatpoint.com/