Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

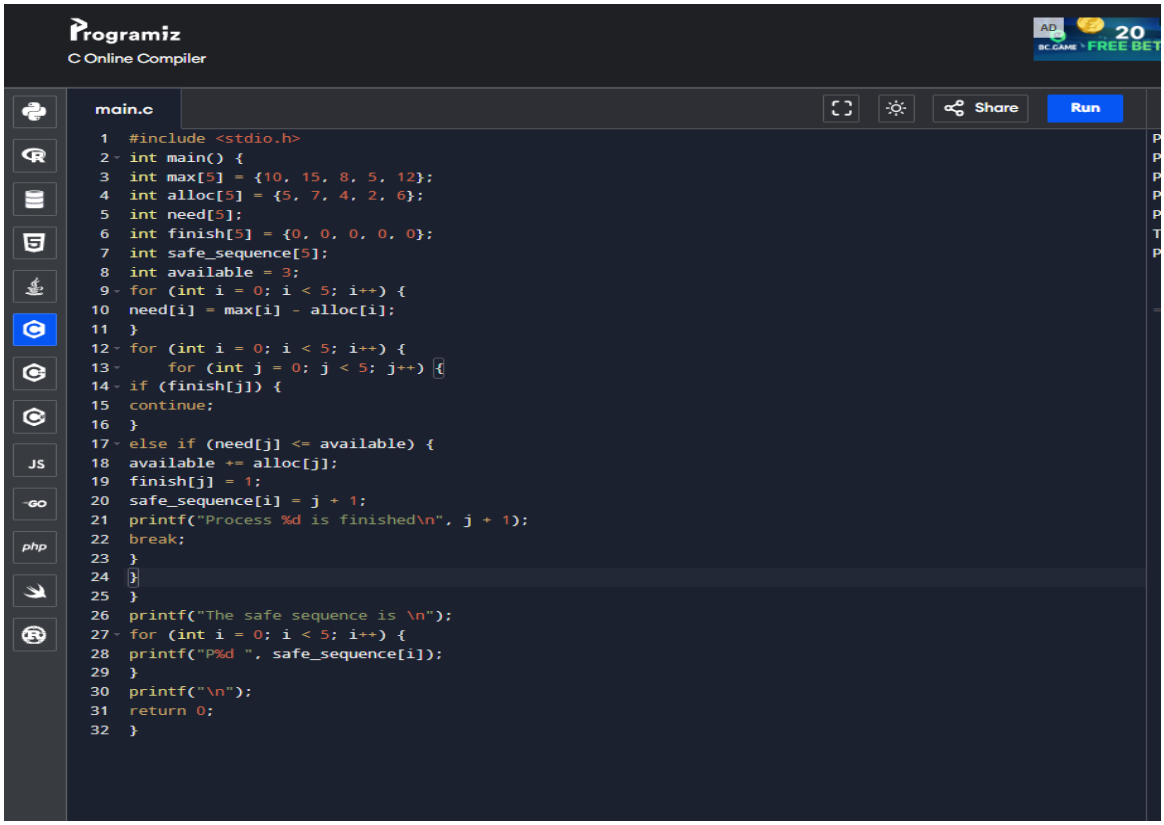
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

Academic Year: 2024-25 | **Name of Student: Pandey Kalash**
Semester: IV | **Student ID:23106049**
Class / Branch: SE AIML | **Date of Performance:28-02-25**
Subject: CSL403 Operating System Lab | **Date of Submission:28-02-25**
Name of Instructor: Prof. Poonam Tiware

Experiment No.6

Aim: Write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm.

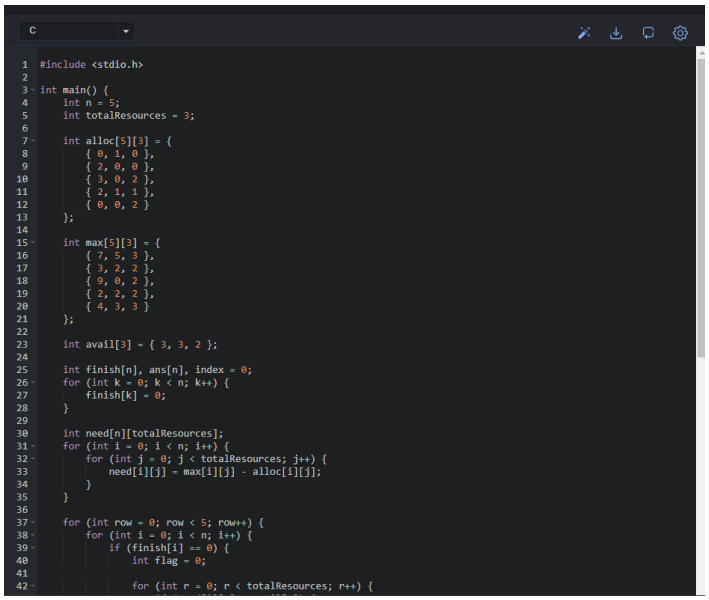
Program 1:




```c
#include <stdio.h>
int main() {
int max[5] = {10, 15, 8, 5, 12};
int alloc[5] = {5, 7, 4, 2, 6};
int need[5];
int finish[5] = {0, 0, 0, 0, 0};
int safe_sequence[5];
int available = 3;
for (int i = 0; i < 5; i++) {
need[i] = max[i] - alloc[i];
}
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
if (finish[j]) {
continue;
}
else if (need[j] <= available) {
available += alloc[j];
finish[j] = 1;
safe_sequence[i] = j + 1;
printf("Process %d is finished\n", j + 1);
break;
}
}
}
printf("The safe sequence is \n");
for (int i = 0; i < 5; i++) {
printf("P%d ", safe_sequence[i]);
}
printf("\n");
return 0;
}
```

Output 1:

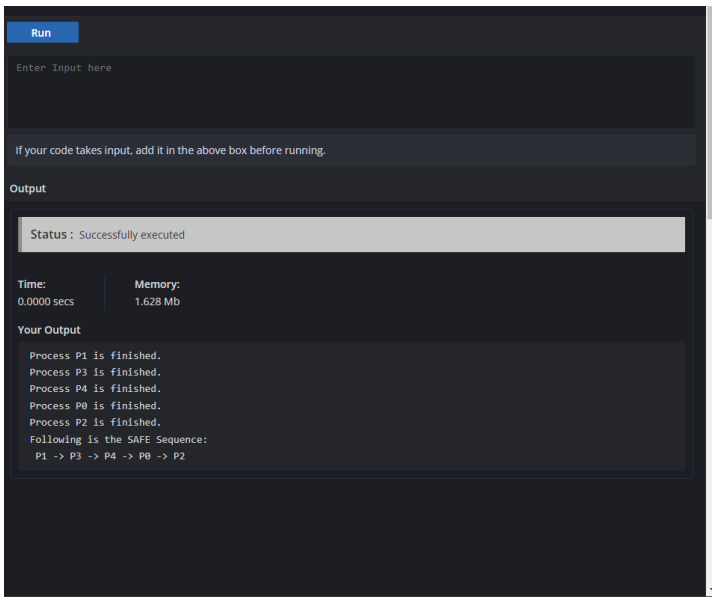
Program 2:



```c
#include <stdio.h>

int main() {
    int n = 5;
    int totalResources = 3;

    int alloc[5][3] = {
        { 0, 1, 0 },
        { 2, 0, 0 },
        { 3, 0, 2 },
        { 2, 1, 1 },
        { 0, 0, 2 }
    };

    int max[5][3] = {
        { 7, 5, 3 },
        { 3, 2, 2 },
        { 9, 0, 2 },
        { 2, 2, 2 },
        { 4, 3, 3 }
    };

    int avail[3] = { 3, 3, 2 };

    int finish[n], ans[n], index = 0;
    for (int k = 0; k < n; k++) {
        finish[k] = 0;
    }

    int need[n][totalResources];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < totalResources; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    for (int row = 0; row < 5; row++) {
        for (int i = 0; i < n; i++) {
            if (finish[i] == 0) {
                int flag = 0;

                for (int r = 0; r < totalResources; r++) {
```

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

                if (flag == 0) {
                    ans[index++] = i;
                    for (int y = 0; y < totalResources; y++) {
                        avail[y] += alloc[i][y];
                    }
                    finish[i] = 1;
                    printf("Process P%d is finished.\n", i);
                }
            }
        }
    }

    int flag = 1;
    for (int i = 0; i < n; i++) {
        if (finish[i] == 0) {
            flag = 0;
            printf("The system is not in a safe state!\n");
            break;
        }
    }

    if (flag == 1) {
        printf("Following is the SAFE Sequence:\n");
        for (int i = 0; i < n - 1; i++)
            printf(" P%d ->", ans[i]);
        printf(" P%d\n", ans[n - 1]);
    }

    return 0;
}
```

Output:



Conclusion: Hence, we have studied Deadlock avoidance through Bankers Algorithm.