LAB ASSESSMENT – 2

```
#include <stdio.h>
int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

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

    for (i = 0; i < processes; i++)
{
        running[i] = 1;
        counter++;
    }

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

    printf("\nEnter Claim Vector:");
    for (i = 0; i < resources; i++)
{
       scanf("%d", &maxres[i]);
    }

  printf("\nEnter Allocated Resource Table:\n");
    for (i = 0; i < processes; i++)
```

```
{
      for(j = 0; j < resources; j++)
{
  scanf("%d", &current[i][j]);
       }
    }

    printf("\nEnter Maximum Claim Table:\n");
    for (i = 0; i < processes; i++)
{
       for(j = 0; j < resources; j++)
{
          scanf("%d", &maximum_claim[i][j]);
       }
    }

printf("\nThe Claim Vector is: ");
    for (i = 0; i < resources; i++)
{
      printf("\t%d", maxres[i]);
}

    printf("\nThe Allocated Resource Table:\n");
    for (i = 0; i < processes; i++)
{
      for (j = 0; j < resources; j++)
{
          printf("\t%d", current[i][j]);
       }
printf("\n");
    }
```

```
    printf("\nThe Maximum Claim Table:\n");
    for (i = 0; i < processes; i++)
{
        for (j = 0; j < resources; j++)
{
       printf("\t%d", maximum_claim[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < processes; i++)
{
        for (j = 0; j < resources; j++)
{
           allocation[j] += current[i][j];
        }
    }

    printf("\nAllocated resources:");
    for (i = 0; i < resources; i++)
{
        printf("\t%d", allocation[i]);
    }

    for (i = 0; i < resources; i++)
{
       available[i] = maxres[i] - allocation[i];
}

    printf("\nAvailable resources:");
```

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

    while (counter != 0)
{
        safe = 0;
        for (i = 0; i < processes; i++)
{
            if (running[i])
{
                exec = 1;
                for (j = 0; j < resources; j++)
{
                    if (maximum_claim[i][j] - current[i][j] > available[j])
{
                        exec = 0;
                        break;
                    }
                }
                if (exec)
{
                    printf("\nProcess%d is executing\n", i + 1);
                    running[i] = 0;
                    counter--;
                    safe = 1;

                    for (j = 0; j < resources; j++)
{
```

```
                    available[j] += current[i][j];
                }
              break;
              }
            }
        }
        if (!safe)
{
            printf("\nThe processes are in unsafe state.\n");
            break;
        }
else
{
            printf("\nThe process is in safe state");
            printf("\nAvailable vector:");

            for (i = 0; i < resources; i++)
{
                printf("\t%d", available[i]);
            }

       printf("\n");
        }
    }
    return 0;
}
```

OUTPUT SCREEN - SAFE

```
Enter number of processes: 3

Enter number of resources: 3

Enter Claim Vector:5

5
5

Enter Allocated Resource Table:
1
2
1
2
0
1
2
2
1

Enter Maximum Claim Table:
2
2
4
2
1
3
3
4
1
```

```
The Claim Vector is:    5       5       5
The Allocated Resource Table:
        1       2       1
        2       0       1
        2       2       1

The Maximum Claim Table:
        2       2       4
        2       1       3
        3       4       1

Allocated resources:    5       4       3
Available resources:    0       1       2

Process2 is executing

The process is in safe state
Available vector:       2       1       3

Process1 is executing

The process is in safe state
Available vector:       3       3       4

Process3 is executing

The process is in safe state
Available vector:       5       5       5
```

OUTPUT SCREEN – UNSAFE

```
Enter number of processes: 3

Enter number of resources: 3

Enter Claim Vector:0
1
2

Enter Allocated Resource Table:
1
2
1
2
0
1
2
3
1

Enter Maximum Claim Table:
1
0
3
0
1
2
1
1
0
```

```
The Claim Vector is:    0       1       2
The Allocated Resource Table:
        1       2       1
        2       0       1
        2       3       1

The Maximum Claim Table:
        1       0       3
        0       1       2
        1       1       0

Allocated resources:    5       5       3
Available resources:    -5      -4      -1

The processes are in unsafe state.


...Program finished with exit code 0
Press ENTER to exit console.
```

Name: Ananya Prasad

Reg No: 20BCE10093

LAB ASSESSMENT-2

Let us assume that there are m resources and n processes

Available: Array of length resources (m). It is the number of available resources
If available[j] = k, then are k resources available of type R_j.

Need: n x m gives the remaining resource needs of each process. If need[i][j] = k
then process Pi may need k instances of resource type R_j to complete a task.

Max: n x m matrix representing the maximum number of instances of each resource
that a process can request. If max[i][j] = k, then process Pi can request
almost k instances of resource type R_j.

Allocation: n x m matrix representing the number of resources of each type currently
allocated to each process. If allocation[i][j] = k, then process Pi is
currently allocated k instances of resource type R_j.

Need[i][j] = Max[i][j] - Allocation[i][j]

In the code discussed above,

- First we ask for the number of processes. For i less than processes, the resources run.
- Now we ask for the number of resources and for the claim vector for the number of resources entered.
- Now the program asks for the allocated resource table and maximum claim table.
- Program outputs the matrix form of input values
- Allocated resources are found by available[i] = maxres[i] - allocation[i] (NEED)
- Available resources are found by available[i].
- Allocation[i] is updated by allocation[i] = allocation[i] + current[i][j]
- Now processes are checked if they are running or not and then checked if safe or unsafe.