17. Illustrate the deadlock avoidance concept by simulating Banker's algorithm with C.

```
#include <stdio.h>
int main() {
   int n, m, i, j, k;
   n = 5; // Number of processes
   m = 3; // Number of resources
   int alloc[5][3] = { {0, 1, 0},  // P0
                       {2, 0, 0},  // P1
                       {3, 0, 2},  // P2
                       {2, 1, 1},  // P3
                       {0, 0, 2} }; // P4
   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}; // Available resources
   int f[n], ans[n], ind = 0;
   for (k = 0; k < n; k++)
      f[k] = 0;
   int need[n][m];
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];
   }
   printf("\nNeed Matrix:\n");
```

```
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }
    int y = 0;
    for (k = 0; k < 5; k++) { // Repeat to ensure all processes checked
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1;
                        break;
                    }   }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                } } } }
    printf("\nSafe Sequence: ");
    for (i = 0; i < n - 1; i++)
        printf("P%d -> ", ans[i]);
    printf("P%d\n", ans[n - 1]);
    return 0;
}
```

OUTPUT-

Need Matrix:

7 4 3

1 2 2

6 0 0

0 1 1

4 3 1

Safe Sequence: P1 -> P3 -> P4 -> P0 -> P2