Task 7

Task: Implement an Algorithm for Dead Lock Detection in C

Program:

```
#include <stdio.h>

static int mark[20]; // Marks processes that can finish (safe processes)

int i, j, np, nr; // np: number of processes, nr: number of resources


int main() {

   int alloc[10][10], request[10][10], avail[10], r[10], w[10];


   // Input number of processes and resources

   printf("\nEnter the number of processes: ");

   scanf("%d", &np);

   printf("\nEnter the number of resources: ");

   scanf("%d", &nr);


   // Input total number of resources for each resource type

   for(i = 0; i < nr; i++) {

      printf("\nTotal amount of Resource R%d: ", i + 1);

      scanf("%d", &r[i]);

   }


   // Input the request matrix (how many resources each process is requesting)

   printf("\nEnter the request matrix:\n");

   for(i = 0; i < np; i++) {

      for(j = 0; j < nr; j++) {

         scanf("%d", &request[i][j]);

      }
```

```
}

// Input the allocation matrix (how many resources each process is allocated)
printf("\nEnter the allocation matrix:\n");
for(i = 0; i < np; i++) {
   for(j = 0; j < nr; j++) {
      scanf("%d", &alloc[i][j]);
   }
}

// Available resources calculation
for(j = 0; j < nr; j++) {
   avail[j] = r[j];  // Start with total resources
   for(i = 0; i < np; i++) {
      avail[j] -= alloc[i][j];  // Subtract the allocated resources from the total
   }
}

// Marking processes with zero allocation (safe processes)
for(i = 0; i < np; i++) {
   int count = 0;
   for(j = 0; j < nr; j++) {
      if(alloc[i][j] == 0)
         count++;
      else
        break;
   }
   if(count == nr)
```

```
mark[i] = 1;  // If process has no allocation, it's safe
  }

  // Initialize working vector (W) as available resources
  for(j = 0; j < nr; j++) {
    w[j] = avail[j];
  }

  // Check for processes that can be executed (requests <= available resources)
  for(i = 0; i < np; i++) {
    if(mark[i] != 1) {  // If the process is not marked safe yet
      int canBeProcessed = 1;
      for(j = 0; j < nr; j++) {
        if(request[i][j] > w[j]) {  // If the request exceeds available resources, can't process
          canBeProcessed = 0;
          break;
        }
      }
      if(canBeProcessed) {  // If the request can be satisfied
        mark[i] = 1;  // Mark process as safe
        for(j = 0; j < nr; j++) {
          w[j] += alloc[i][j];  // Add allocated resources back to available pool
        }
      }
    }
  }

  // Check for unmarked (unsafe) processes
```

```
    int deadlock = 0;

    for(i = 0; i < np; i++) {

        if(mark[i] != 1) {  // If process is not marked as safe

            deadlock = 1;  // Deadlock detected

            break;

        }

    }


    // Output the result

    if(deadlock) {

        printf("\nDeadlock detected\n");

    } else {

        printf("\nNo Deadlock possible\n");

    }


    return 0;

}
```

OUTPUT: