```
/*
Given an array of integers, write an algorithm and a
program to left rotate the array by a specific number
of elements.

INPUT FORMAT:
  - the first line contains the number of test cases, T.
     - for each test cases there will be three input lines.
     - first line contains the size of array, N.
     - second line contains N space-seperated integers
     describing array.
     - third line will take number of rotations, K.

OUTPUT FORMAT:
  - the output will contain T number of lines.
     - for each line output will be the rotated array for
     that test case.

CONSTRAINTS:
  - T > 0
  - N > 0
  - K > 0
*/


// LIBS
#include <stdio.h>
#include <stdlib.h>
#include "header1.h" // custom header file

// FUNCTION DECLARATION
int leftRotateArray(int *, int *, int, int);

// MAIN FUNCTION
int main() {
  // Number of test cases
  int T;
  scanf("%d", &T);

  // Test cases begin
  for (int case_index = 0; case_index < T; case_index++) {
     // Length of array, Rotations
     int N, K;
     scanf("%d", &N);
```

```
    // Initializing arrays
    int *given_array;
    int *rotated_array;
    given_array = (int*) malloc(sizeof(int) * N);
    rotated_array = (int*) malloc(sizeof(int) * N);

    // Input rotations
    scanf("%d", &K);

    // Error handlings
    if ((given_array == NULL) || (rotated_array == NULL) ||
(K <= 0) || (T <= 0)) {
    return -1;
    }

    // Input Array, Left Rotate Array, Print Array, while
checking for errors
    if (((inputArray(given_array, N)) != 0) ||
(leftRotateArray(given_array, rotated_array, N, K) != 0) ||
(printArray(rotated_array, N) != 0)) {
    return -1;
    }

    // Clean up memory
    free(given_array);
    free(rotated_array);
  }

  return 0;
}

// FUNCTION DEFINITION
int leftRotateArray(int *a1, int *a2, int length, int
rotations) {
  if ((a1 == NULL) || (a2 == NULL) || (length <= 0) ||
(rotations <= 0)) {
    return -1;
  }

  for (int index = 0; index < length; index++) {
    int new_index;
    new_index = (index - rotations);
    if (new_index < 0) {
```

```
      new_index = (length + new_index);
      }
      a2[new_index] = a1[index];
  }
}
```