DD2452 - Homework 1
Author: Mert Alp Taytak

```
#include <stdlib.h>

// Definitions

#define SIZE_QUEUE 100

#define INDEX_FIRST_NAME 0
#define INDEX_LAST_NAME 1
#define INDEX_SSN 2

#define SIZE_FIRST_NAME 15
#define SIZE_LAST_NAME 25
#define SIZE_SSN 10

struct message {
    char first_name[SIZE_FIRST_NAME];
    char last_name[SIZE_LAST_NAME];
    char ssn[SIZE_SSN];
};

struct message_queue {
    struct message messages[SIZE_QUEUE];
    int count;
};

// Globals

struct message_queue queue;
```

```
// Functions

/*@
    predicate is_copied(char *ptr1, char *ptr2, integer bytes) =
        \forall integer i; 0 <= i < bytes ==> *(ptr1+i) == *(ptr2+i);
*/


/*@
    requires n1 >= 0 && n2 >= 0;

    behavior copy_zero_characters:
        requires n1 == 0 || n2 == 0;
        assigns \nothing;

    behavior copy_normal:
        requires \valid(s1+(0..n1-1));
        requires \valid(s2+(0..n2-1));
        requires \separated(s1, s2);
        ensures is_copied(s1, s2, \min(n1, n2));
        assigns s1[0..\min(n1,n2)-1];
*/
void copy(char s1[], int n1, char s2[], int n2)
{
    if (n1 == 0 || n2 == 0) {
        //@ assert n1 == 0 || n2 == 0;
        return;
    }

    //@ assert n1 != 0 && n2 != 0;
    int min;
    int i = 0;

    if (n1 < n2) {
        //@ assert n1 < n2;
        min = n1;
        //@ assert min == n1;
    }
    else {
        //@ assert n2 <= n1;
        min = n2;
        //@ assert min == n2;
    }

    //@ assert min == \min(n1, n2);
    //@ assert i == 0;

    /*@
        loop invariant 0 <= i < min;
        loop assigns i, s1[0..min-1];
    */
    while (i < min) {
        s1[i] = s2[i];
        //@ assert s1[i] == s2[i];
        i++;
        //@ assert i == \at(i, LoopCurrent) + 1;
    }

    //@ assert i == min;
    //@ assert \forall integer k; 0 <= k < min ==> s1[k] == s2[k];
}
```

```
/*@
    requires n1 > 0 && \valid(first_name+(0..n1-1));
    requires n2 > 0 && \valid(last_name+(0..n2-1));
    requires n3 > 0 && \valid(social_security_number+(0..n3-1));
    requires \separated(first_name, last_name, social_security_number);

    behavior has_no_space:
        assumes queue.count >= SIZE_QUEUE;
        ensures \result == -1;
        assigns \nothing;

    behavior has_space:
        assumes queue.count < SIZE_QUEUE;
        ensures
            \let oldc = \old(queue.count);
            \result == 0 &&
            queue.count == oldc + 1 &&
            is_copied((char*)(queue.messages[oldc].first_name),first_name,\min(n1, SIZE_FIRST_NAME))&&
            is_copied((char*)(queue.messages[oldc].last_name), last_name, \min(n2, SIZE_LAST_NAME)) &&
            is_copied((char*)(queue.messages[oldc].ssn), social_security_number, \min(n3, SIZE_SSN));
        assigns
            queue.count,
            queue.messages[queue.count],
            queue.messages[queue.count].first_name[0..\min(n1,SIZE_FIRST_NAME)-1],
            queue.messages[queue.count].last_name[0..\min(n2,SIZE_LAST_NAME)-1],
            queue.messages[queue.count].ssn[0..\min(n3,SIZE_SSN)-1];
*/
```

```c
int leave_message(
    char first_name[], int n1,
    char last_name[], int n2,
    char social_security_number[], int n3)
{
    if (queue.count >= SIZE_QUEUE) {
        //@ assert queue.count >= SIZE_QUEUE;
        return -1;
    }

    //@ assert queue.count < SIZE_QUEUE;

    int index = queue.count;
    struct message message = queue.messages[index];

    copy(message.first_name, SIZE_FIRST_NAME, first_name, n1);
    //@ assert is_copied((char*)(message.first_name), first_name, \min(n1,SIZE_FIRST_NAME));

    copy(message.last_name, SIZE_LAST_NAME, last_name, n2);
    //@ assert is_copied((char*)(message.last_name), last_name, \min(n2,SIZE_LAST_NAME));

    copy(message.ssn, SIZE_SSN, social_security_number, n3);
    //@ assert is_copied((char*)(message.ssn), social_security_number, \min(n3,SIZE_SSN));

    queue.count = index + 1;
    //@ assert queue.count == \at(queue.count, Pre) + 1;

    return 0;
}
```