```c
#include <stdlib.h>

typedef struct node_t {
    int x;
    struct node_t *next;
} *Node;


typedef enum {
 SUCCESS=0,
 MEMORY_ERROR,
 UNSORTED_LIST,
 NULL_ARGUMENT,
} ErrorCode;


int getListLength(Node list);
bool isListSorted(Node list);
Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code);

// copy a node and return a pointer to it
// Node node - the node to copy
// return NULL if allocation failed
Node copyNode(Node node){
    if (node == NULL){
        return NULL;
    }
    Node node_copy = malloc(sizeof(node_t));
    if (node_copy == NULL)
    {
        return NULL;
    }

    node_copy->x = node->x;
    node->next = NULL;
    return node_copy;
}
// free nodes after a memory error
// Node head - pointer to the first node
// int count - number of nodes to free

void freeNodesAfterMemoryError(Node head, int count){
    Node previuos_node = head;
    Node current_node = previuos_node->next;
    while (count > 1)
    {
        free(previuos_node);
        previuos_node = current_node;
        current_node = previuos_node->next;
        count--;
    }
    free(previuos_node);
}

// take two sorted Node lists, create a new sorted Node list that is the union of the two.
// returns a pointer to the new list or NULL if an error accoured
//Node list1 - pointer to the first list
//Node list2 - pointer to the second list
// ErrorCode* error_code - pointer to an ErrorCode enum which will be updated with ann according value
Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code){
    if (list1 == NULL || list2 == NULL){
        *error_code = NULL_ARGUMENT;
        return NULL;
    }
    if (!isListSorted(list1) || !isListSorted(list2))
    {
        *error_code = UNSORTED_LIST;
        return NULL;
    }
    int list1_len = getListLength(list1);
    int list2_len = getListLength(list2);
    int merged_len = list1_len + list2_len;

    Node out_merged;
    Node current_node = out_merged;
    int counter = 0;
    while (counter < merged_len)
    {
        if (list1_len == 0){
            current_node = copyNode(list2);
            list2_len--;
        }
        else if(list2_len == 0){
            current_node = copyNode(list1);
            list1_len--;
        }
        else{
            if(list1->x <= list2->x){
                current_node = copyNode(list1);
                list1_len--;
            }
            else{
                current_node = copyNode(list2);
                list2_len--;
            }
        }

        if (current_node == NULL){
            freeNodesAfterMemoryError(out_merged, counter);
            *error_code MEMORY_ERROR;
            return NULL;
        }
        current_node = current_node->next;
        counter++;
    }

    *error_code = SUCCESS;
    return out_merged;
}
```