```c
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>

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

} *Node;

typedef enum {
    SUCCESS=0,
    MEMORY_ERROR,
    UNSORTED_LIST,
    NULL_ARGUMENT,

} ErrorCode;

//Function declarations:

/* Receives sorted linked lists; returns a merged list and stores error
code in error_code. */
Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code);

/* Performs the actual merging. */
ErrorCode mergeSortedLists_aux(Node list1, Node list2, Node mergedList);

/* Frees a null-terminated linked list. */
void freeList(Node list);

/* Creates a node and returns an error code (or SUCCESS if no error had
occurred.) */
ErrorCode createNode(Node mergedList);

Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code)
{
    assert(error_code != NULL);

    *error_code = SUCCESS;


    if (list1 == NULL || list2 == NULL)
    {
        *error_code = NULL_ARGUMENT;
    }
    else if (!isListSorted(list1) || !isListSorted(list2))
    {
        *error_code = UNSORTED_LIST;
    }

    if (*error_code != SUCCESS)
    {
        return NULL;
    }

    Node mergedList = malloc(sizeof(*mergedList));
    *error_code = mergeSortedLists_aux(list1, list2, mergedList);
    if (*error_code != SUCCESS)
    {
        freeList(mergedList);
        return NULL;
    }

    return mergedList;
}


ErrorCode mergeSortedLists_aux(Node list1, Node list2, Node mergedList)
{
    ErrorCode memory_error = SUCCESS;
    Node prev = mergedList;
    while (list1 != NULL && list2 != NULL && memory_error == SUCCESS)
    {
        if(list1->x > list2->x)
        {
            mergedList->x = list2->x;
            list2 = list2->next;
        }
        else
        {
            mergedList->x = list1->x;
            list1 = list1->next;
        }

        memory_error = createNode(mergedList);
        prev = mergedList;
        mergedList = mergedList->next;
    }

    while (list1 != NULL && memory_error == SUCCESS)
    {
        mergedList->x = list1->x;
        list1 = list1->next;

        memory_error = createNode(mergedList);
        prev = mergedList;
        mergedList = mergedList->next;
    }

    while (list2 != NULL && memory_error == SUCCESS)
    {
        mergedList->x = list2->x;
        list2 = list2->next;

        memory_error = createNode(mergedList);
        prev = mergedList;
        mergedList = mergedList->next;
    }

    if (memory_error == SUCCESS)
    {
        prev->next = NULL;
        free(mergedList); //Freeing the excess element.
    }
    return memory_error;
}

void freeList(Node list)
{
    if (list == NULL)
    {
        return;
    }

    Node temp;
    while (list != NULL)
    {
        temp = list->next;
        free(list);
        list = temp;
    }
}

ErrorCode createNode(Node mergedList)
{
    if(mergedList == NULL)
    {
        return NULL_ARGUMENT;
    }
    mergedList->next = malloc(sizeof(*mergedList));
    if (mergedList->next == NULL)
    {
        return MEMORY_ERROR;
    }
    return SUCCESS;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
errors list:

Implementation errors:

1) x has to be dereferenced to have a value placed in it. ("x = strlen(...")
2) x % 2 == 0 needs to be x % 2 != 0.
3) malloc size should be + 1. (This is to account for the null terminator.)
4) Should be [*x - 1 - i]. (Both to capture the first character, and to avoid read violation.)
5) Check if x is NULL. (Instructions imply that it could be.)


Proper programming errors:

1) Need to check malloc's return value.
2) Need to add a null terminator.
3) Make the second "if" a simple "else".
4) Need to implement a value to a pointer ASAP (use malloc the next line or at the same line).
5) Since we are using c99 "int i" can go inside the for loop.
6) Curly brackets around loop & condition blocks. (Convention.)
7) malloc(*x) -> malloc(sizeof(*str) * (*x)) though it should work either way.

*/

//THE CODE BEFORE CHANGES
char* foo(char* str, int* x) {
   char* str2;
   int i;
   x = strlen(str);
   str2 = malloc(*x);
   for (i = 0; i < *x; i++)
        str2[i] = str[*x - i];
   if (*x % 2 == 0) {
      printf("%s", str);
   }
   if (*x % 2 != 0)
   {
      printf("%s", str2);
   }
   return str2;
 }

//THE FIXED CODE
char* foo(char* str, int* x)
{
   if (str == NULL)
   {
      return NULL;
   }
```

```
    int length = strlen(str);
    if (x != NULL)
    {
        *x = length;
    }

    char* str2 = malloc(sizeof(char) * (*(x) + 1));
    if(str2 == NULL)
    {
        return NULL;
    }

    for (int i = 0; i < length; i++)
    {
        str2[i] = str[length - i - 1];
    }
    str2[length] = '\0';

    if (length % 2 != 0)
    {
        printf("%s", str);
    }
    else
    {
        printf("%s", str2);
    }
    return str2;
}
```