Question 2 Part 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.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);


// Static functions declarations:
static Node createNode(int data);
static void destroyList(Node ptr);
static bool validateListAttributes(int list_length, bool is_list_sorted, ErrorCode*
error_code);


static Node createNode(int data)
{
    Node ptr = malloc(sizeof(*ptr));
    if (ptr == NULL)
    {
        return NULL;
    }
    ptr->x = data;
    ptr->next = NULL;
    return ptr;
}

static void destroyList(Node ptr)
{
    while (ptr != NULL)
    {
        Node toDelete = ptr;
        ptr = ptr->next;
        free(toDelete);
    }
}

static bool validateListAttributes(int list_length, bool is_list_sorted, ErrorCode*
error_code)
{
    if (list_length == 0)
    {
        *error_code = NULL_ARGUMENT;
        return false;
    }
    if (is_list_sorted == false)
    {
        *error_code = UNSORTED_LIST;
```

```
        return false;
    }
    return true;
}

Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code)
{
    if (!(validateListAttributes(getListLength(list1), isListSorted(list1),
error_code)))
    {
        return NULL;
    }
    if (!(validateListAttributes(getListLength(list2), isListSorted(list2),
error_code)))
    {
        return NULL;
    }
    Node head_pointer = NULL, previous_pointer = NULL;
    while ((list1 != NULL) || (list2 != NULL) )
    {
        Node current_pointer = createNode(0);
        if (current_pointer == NULL)
        {
            destroyList(head_pointer);
            *error_code = MEMORY_ERROR;
            return NULL;
        }

        if ((list1 == NULL) || ((list2 != NULL) && (list1->x >= list2->x)))
        {
            current_pointer->x = list2->x;
            list2 = list2->next;
        }
        else
        {
            current_pointer->x = list1->x;
            list1 = list1->next;
        }

        if (head_pointer == NULL)
        {
            head_pointer = current_pointer;
        }
        else
        {
            previous_pointer->next = current_pointer;
        }

        previous_pointer = current_pointer;
    }
    *error_code = SUCCESS;
    return head_pointer;
}
```

Question 2 Part 2

errors summary

a. code conventions errors:

1. bad name choices for variables:
   - str_len instead of x
   - reversed_str instead str2
   - foo is a bad name choice.
2. allocation of variable in the beginning of the function instead of in place we use them:
   - int i should be written inside the loop and not in the beginning of the function.
3. missing use of Defines:
   - number '2' appears in the code instead of defining it.
   - in addition, we can define the NULL_BYTE but it is not actually needed.
4. bad understanding of conditions:
   - should use if - else instead of if - if at the end of the function because there are only two options the covers all the Scenarios.

b. wrong implementation of the function:

1. allocation process of str2 was wrong:
   - missing use of sizeof(*reversed_str) to detect the size of the variable in memory.
   - missing allocation for null byte at the end of the string.
   - not checking if allocation succeed or not.
2. bad use of a pointers:
   - x is a pointer and therefore in order the change the value it's point to we should write: *x = strlen(str) instead of x = strlen(str).
3. parameters might point to NULL:
   - we should check both str and str_len do not point to NULL before we use them.
4. array indexes are out of bounds:
   - in the first iteration when i = 0, str[*x - i] = str2[n] and it's not part of the allocated area of the array.
5. we should switch the print circumstances according to the demand:
   - we should print the reversed string if the number of chars is even and the string if it's odd.

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

#define EVEN_NUM_DIVIDER 2
#define NULL_BYTE 1

char* foo(char* str, size_t* str_len)
{
    if (str == NULL || str_len == NULL)
    {
        return NULL;
    }
    *str_len = strlen(str);
    char *reversed_str = malloc(sizeof(*reversed_str) * (*str_len + NULL_BYTE));
    if (reversed_str == NULL)
    {
        return NULL;
    }
    for (int i = 0; i < *str_len; i++)
```

```
    {
        reversed_str[i] = str[*str_len-1-i];
    }
    reversed_str[*str_len] = '\0';
    if (*str_len % EVEN_NUM_DIVIDER == 0)
    {
        printf("%s", reversed_str);
    }
    else
    {
        printf("%s", str);
    }
    return reversed_str;
}
```