חלק 2 – 2.1:

mergeSortedLists.c

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

#include "mergeSortedLists.h"

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



/*the function will merge two sorted lists to one sorted list.
error codes :
if one of the lists provided is null - NULL.
if there is any failure with malloc - MEMORY_ERROR.
if the list is not sorted - UNSORTED_LIST.

success - returned if the list is merged and it sorted right .
*/
Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code)
{
    if(!list1 || !list2)
    {
        *error_code = NULL_ARGUMENT;
        return NULL;
    }
    Node list = malloc(sizeof(*list)); //creating the list for the first t
ime
    if(!list)
    {
        *error_code = MEMORY_ERROR;
        return NULL;
    }
    Node ptr_to_list1 = list1 , ptr_to_list2 = list2;
    Node first_node_to_copy = compareNodes(ptr_to_list1,ptr_to_list2);
    list->x = first_node_to_copy->x;
    list->next = NULL;
    if(compareNodes(ptr_to_list1,ptr_to_list2)==ptr_to_list1)
    {
        ptr_to_list1 = ptr_to_list1->next;
    }
    else
    {
        ptr_to_list2 = ptr_to_list2->next;
    }
    int length_of_list = getListLength(list1) + getListLength(list2) , i=1
;
    Node ptr_to_list = list;
    for(; i<length_of_list ; i++) //creates all other nodes in the right o
rder
    {
        Node node_to_copy = compareNodes(ptr_to_list1,ptr_to_list2);
        if(node_to_copy == ptr_to_list1)
        {
            ptr_to_list-
>next = createNodeAndCopy(ptr_to_list1 , error_code);
            ptr_to_list1 = ptr_to_list1->next;
        }
        else
        {
            ptr_to_list-
>next = createNodeAndCopy(ptr_to_list2 , error_code);
            ptr_to_list2 = ptr_to_list2->next;
        }
        ptr_to_list = ptr_to_list->next;
        if(*error_code == MEMORY_ERROR)
        {
            listDestroy(list);
            return NULL;
        }
    }
    if(*error_code != SUCCESS)
    {
        return NULL;
    }
    isListSorted(list);
    *error_code = SUCCESS;
    return list;
}
```

```c
/*the function create a new Node and copying from the choosen Node all his
 details
error codes :
if one of the lists provided is null - NULL.
if there is any failure with malloc - MEMORY_ERROR.
*/
Node createNodeAndCopy(Node list, ErrorCode* error_code)
{
    Node new_node = malloc(sizeof(*new_node));
    if(!new_node)
    {
        *error_code = MEMORY_ERROR;
        listDestroy(list);
        return NULL;
    }
    new_node->x = list->x;
    new_node->next = NULL;

return new_node;
}

//The function choose which list need to provide the next node to be in th
e merge list.
Node compareNodes(Node list1 , Node list2)
{
    if(!list1)
    {
        return list2;
    }
    if(!list2)
    {
        return list1;
    }
    if(list1->x <= list2->x)
    {
        return list1;
    }
    return list2;
}

//In case of any malloc failure in any point of the program , list will be
 free to prevent any memory leak.
void listDestroy(Node list)
{
     while(list)
     {
        Node to_delete = list;
        list = list->next;

        free(to_delete);
     }
}
```

mergeSortedLists.h

```
#ifndef MERGESORTEDLISTS_H_
#define MERGESORTEDLISTS_H_


#include <stdbool.h>

typedef struct node_t *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);
Node createNodeAndCopy(Node list, ErrorCode* error_code);
Node compareNodes(Node list1 , Node list2);
void insertDetailsToList(Node ptr_to_list ,Node list1 , Node list2 ,ErrorC
ode* error_code);
void listDestroy(Node list);
Node createNode(int a);

#endif /* MERGESORTEDLISTS_H */
```

חלק 2.2:

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

#define EVEN 2
#define ENDSTR '\0'
char* foo(char* str, int* x)
{
    char* str2;
    int i,lenght= strlen(str);
    if(x!=NULL)
    {
        *x = lenght;
    }
    str2 = malloc(sizeof(char)*(lenght+1));
    if(!str2)
    {
        return NULL;
    }
    for (i = 0; i < lenght; i++)
    {
        str2[i] = str[lenght - i -1];
    }
    str2[i]=ENDSTR;
    if (lenght % EVEN == 0)
    {
        printf("%s", str2);
    }
    else // not even
    {
         printf("%s", str);
    }
    return str2;
}
```