תרגיל בית 1

מגישים:
איה ספירה 206713935

אורי מינץ 314616897

סעיף א

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

char *stringduplicator(char *s, int times) {
    assert(!s);
    assert(times > 0);
    int LEN = strlen(*s);
    char *out = malloc(LEN * times);
    assert(out);
    for (int i = 0; i < times; i++) {
    out = out + LEN;
    strcpy(out, s);
    }
    return out;
}
```

שגיאות תכנות

1. שני ה-assert בודקים פרמטרים מהמשתמש, בניגוד לשימוש המתאים של assert שבו אנחנו מנחים שהתוכן בפנים נכון בנוסף עושים assert ל- out במקום לעשות if כדי לבדוק אם ה-malloc עבד.
2. מחזירים out פוינטר לסוף המערך ולא לתחילת הסטרינג.
3. הפונקציה strlen מקבלת char** במקום char*.
4. הסטרינג out מקבל את אורך הסטריג ולא משאירה מקום ל-'\0' במאלוק שלה.
5. בתוך לולאת ה-for השורות בסדר הפוך וזה יוצר מצב שבו ה-string הראשון מועתק למקום של השני וכך הלאה, עד לאחרון שמועתק למקום שלא קיים.

שגיאות קונבציה

1. שם הפונקציה לא נכתב עם אות גדולה בתחילת המילה השנייה.
2. השם s לפרמטר לא מסביר את המשמעות שלו.
3. המשתנה LEN נכתב באותיות גדולות למרות שהוא משתנה ולא define.
4. אין הזחה ל-for

סעיף ב

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

char *stringDuplicator(char *string , int times){
    if(string == NULL)
    {
        return NULL;
    }
    if(times <= 0)
    {
        return NULL;
    }
    int string_len = strlen(string);
    char *string_list = malloc((string_len+1) * times);
    if(string_list == NULL)
    {
        return NULL;
    }
    char* string_element = string_list;
    for (int i = 0; i < times ; i++){
        strcpy(string_element, string);
        string_element += string_len;
    }
    string_element[0] = '\0';
    return string_list;
}
```

תרגיל בית 2

```c
#include "dry_question.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
/**
 * @brief creat new node
 *
 * @return Node
 */
static Node creatNode();

/**
 * @brief destroy the cerrent list
 *
 */
static void destroyList(Node list);

/**
 * @brief copy the list form "from_list" to "to_list"
 *
 * @param from_list
 * @param to_list
 * @return ErrorCode
 */
static ErrorCode finishList(Node from_list, Node to_list);

/**
 * @brief checks if the list are valid
 *
 * @param list1
 * @param list2
```

```
 * @return ErrorCode
 */
static ErrorCode chackLists(Node list1, Node list2);

ErrorCode megeSortedLists(Node list1, Node list2, Node *mergedOut)
{
    ErrorCode list_valid= chackLists(list1,list2);
    if(list_valid != SUCCESS)
    {
        return list_valid;
    }
    Node iterator1 = list1;
    Node iterator2 = list2;
    *mergedOut = creatNode();
    Node new_list = *mergedOut;

    if(new_list == NULL)
    {
        return MEMORY_ERROR;
    }
    assert(iterator1 != NULL && iterator2 != NULL);
    if(iterator1->x <= iterator2->x)
    {
        new_list->x = iterator1->x;
        iterator1 = iterator1->next;
    }
    else
    {
        new_list->x = iterator2->x;
        iterator1 = iterator2->next;
    }
```

```
    else
    {
        new_list->x = iterator2->x;
        iterator1 = iterator2->next;
    }
    Node cerent_node = new_list;
    while(iterator1 != NULL && iterator2 != NULL)
    {
        Node new_node = creatNode();
        if(new_node == NULL)
        {
            destroyList(new_list);
            return MEMORY_ERROR;
        }
        if(iterator1->x <= iterator2->x)
        {
            new_node->x = iterator1->x;
            iterator1 = iterator1->next;
        }
        else
        {
            new_node->x = iterator2->x;
            iterator2 = iterator2->next;
        }
        cerent_node->next = new_node;
        cerent_node = new_node;
    }

```

```
    ErrorCode list_copy_status = finishList(iterator1, cerent_no
    if(list_copy_status != SUCCESS)
    {
        destroyList(new_list);
        return MEMORY_ERROR;
    }

    list_copy_status = finishList(iterator2, cerent_node);
    if(list_copy_status != SUCCESS)
    {
        destroyList(new_list);
        return MEMORY_ERROR;
    }

    return SUCCESS;
}

static ErrorCode finishList(Node from_list, Node to_list)
{
    if(from_list == NULL)
    {
        return SUCCESS;
    }
    while(from_list != NULL)
    {
        Node new_node = creatNode();
        if(new_node == NULL)
        {
            return MEMORY_ERROR;
        }
        new_node->x = from_list->x;
        from_list = from_list->next;
```

```
        to_list->next = new_node;
        to_list = to_list->next;
    }
    to_list->next = NULL;
    return SUCCESS;
}


static Node creatNode()
{
    Node new_node = malloc(sizeof(*new_node));
    if(new_node == NULL)
    {
        return NULL;
    }
    return new_node;
}


static void destroyList(Node list)
{
    if(list == NULL)
    {
        return;
    }
    destroyList(list->next);
    free(list);
    return;
}
static ErrorCode chackLists(Node list1, Node list2)
{
    if(list1 == NULL || list2 == NULL)
```

```
    {
        return EMPTY_LIST;
    }
    if(getListLength(list1) <= 0 || getListLength(list2) <= 0)
    {
        return EMPTY_LIST;
    }
    if(!isListSorted(list1) || !isListSorted(list2))
    {
        return EMPTY_LIST;
    }
    return SUCCESS;
}
```