תרגיל בית 1 – חלק יבש

מגישות: דורין שטיימן ושני גורן

שאלה 1: מימוש mergeSortedLists

קובץ ה-c:

```c
#include "q1.h"
#include <assert.h>

Node nodeCreate(int x)
{
    Node ptr = malloc(sizeof(*ptr));
    if (!ptr) {
        return NULL;
    }
    ptr->x = x;
    ptr->next = NULL;
    return ptr;
}

Node compareLists(Node first_node, Node second_node)
{
    assert(first_node && second_node);
    Node new_node = NULL;
    if (first_node->x < second_node->x) {
        new_node = nodeCreate(first_node->x);
        if (new_node == NULL) {
            return NULL;
        }
        first_node = first_node->next;
    }
    else {
        new_node = nodeCreate(second_node->x);
        if (new_node == NULL) {
            return NULL;
        }
        second_node = second_node->next;
    }
    return new_node;
}
```

```c

void listDestroy(Node ptr)
{
    while (ptr) {
        Node toDelete = ptr;
        ptr = ptr->next;
        free(toDelete);
    }
}

Node connectRest(Node ptr1, Node ptr2)
{
    if (ptr1 == NULL && ptr2 == NULL) {
        return NULL;
    }
    return ptr1 == NULL ? ptr2 : ptr1;
}

```

```c
ErrorCode mergeSortedLists(Node list1, Node list2, Node* merged_out)
    if (!list1 || !list2){
        return NULL_ARGUMENT;
    }
    if (!isListSorted(list1) || !isListSorted(list2)) {
        return UNSORTED_LIST;
    }
    if (!getListLength(list1) || !getListLength(list2)) {
        return EMPTY_LIST;
    }
    Node ptr1 = list1, ptr2 = list2;
    Node head = compareLists(ptr1, ptr2);
    if (head == NULL) {
        return MEMORY_ERROR;
    }
    Node ptr_new = head;
    while (ptr1 != NULL || ptr2 != NULL) {
        ptr_new->next = compareLists(ptr1, ptr2);
        if (ptr_new->next == NULL) {
            listDestroy(head);
            *merged_out = NULL;
            return MEMORY_ERROR;
        }
        ptr_new = ptr_new->next;
    }
    ptr_new->next = connectRest(ptr1, ptr2);
    assert(merged_out);
    *merged_out = head;
    return SUCCESS;
}
```

שאלה 2 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. בהקצאת out צריך להוסיף 1+ בסוף, אחרת לא יהיה מקום ל-'\0'.
2. Assert על out – צריך לבדוק באמצעות if אם הקצאת הזיכרון הצליחה, במידה ולא יש להחזיר NULL.
3. צריך להחליף בין שתי השורות שבתוך לולאת ה-for, מכיוון שההוספה הראשונית גורמת לכך שלא נעתיק אף תו ל-len התווים הראשונים והם ישארו ריקים. כמו כן, זה יגרום לחריגת זיכרון כי ננסה לגשת למקום במחרוזת מעבר לזיכרון שהוקצה.
4. בהחזרת out ביציאה מן הפונקציה, נרצה להחזיר פוינטר לתחילת המחרוזת. בפועל, מכיוון שקידמנו את out בתוך לולאת ה-for, הפונקציה מחזירה פוינטר לתו האחרון במחרוזת.
5. ה-assert על s בודק את התנאי ההפוך מזה שנרצה לבדוק – אם קיבלנו מצביע ל-s ששונה מ - NULL- התוכנית תקרוס למרות שבפועל נרצה להעתיק אותה times פעמים.

שגיאות קונבנציה:

6. הגדרת המשתנה LEN באותיות גדולות -> len
7. השם של המחרוזת s – שם לא מקובל למשתנה.
8. בלולאת ה-for אין אידנטציה כנדרש – הבלוק צריך להיות מוזח פנימה.
9. שם הפונקציה – stringDuplicator לא מתאים לקונבנציה לשמות פונקציות.

שאלה 2 סעיף ב:

נכתוב כעת את הקוד מחדש:

```
char* duplicateStr(char* str, int times) {
    assert(str);
    assert(times > 0);
    int len = strlen(str);
    printf("len: %d\n", len);
    char* out = malloc(len * times + 1);
    if (!out) {
        return NULL;
    }
    char* ptr = out;
    for (int i = 0; i < times; i++) {
        strcpy(out, str);
        out = out + len;
    }
    return ptr;
}
```