234124- מבוא לתכנות מערכות

סמסטר חורף תשפ"ב

תרגיל בית 1 – חלק יבש

שם סטודנט	ת.ז
אריג' בויראת	324122845
האיא חיחי	322475146

2.1 זיהוי שגיאות

סעיף א- זיהוי שגיאות

השגיאה	***סוג השגיאה \ הבעיה***
שם הפונקציה(stringduplicator*)	שגיאת קונבנציה
הסוגר הפותח והסוגר הסוגר של בלוק פונקציה תמיד יופיעו כל אחד בשורה נפרדת.	שגיאת קונבנציה
שם הפרמטר שמקבל הפונקציה אינו תקין (char *s)	שגיאת קונבנציה
לא מחזירים מצביע לתחילת המחרוזת הנדרשת(return out)	שגיאת תוכנה
שם המשתנה(int LEN)	שגיאת קונבנציה
שולחים ל strlen ((strlen(*s) Charבמקום מצביע לchar	שגיאת תוכנה
חייבים להקצות מקום ל{\0} char *out = malloc(LEN * times)	שגיאת תוכנה
שימוש ב assertבמקום if assert(out)	שגיאת תוכנה
הזזה של כתובת לאתחול char* strcpy(out, s)	שגיאת תוכנה

סעיף ב- קוד מתוקן

```c
///2.1.2

#include <stdlib.h>
#include <string.h>
#include <assert.h>
char *duplicteString(char *str, int times)
{
    if (str == NULL)
    {
        return NULL;
    }
    assert(times > 0);
    int len = strlen(str);
    char *out = malloc(len * times + 1);
    if (out == NULL)
    {
        return NULL;
    }
    char *original_pointer = out;
    for (int i = 0; i < times; i++)
    {
        strcpy(out, str);
        out = out + len;
    }
    return original_pointer;
}
```

2.2 מיזוג רשימה מקושרת

קובץ .h

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

/** explination about our sulotion:
 *  creates an empty list called head with size len(list1)+len(list2)
 * and merges the two lists (list1 and list2) into head which is in increasing order.
 * we compare the numbers in each node in both lists and insert the smallest number first
 * and then comare with next number.
 * in case one of the lists is not empty we inserting the rest of the numbers in head
 */

typedef struct node_t *Node;

typedef enum
{
    SUCCESS = 0,
    MEMORY_ERROR,
    EMPTY_LIST,
    UNSORTED_LIST,
    NULL_ARGUMENT,
} ErrorCode;

int getListLength(Node list);
bool isListSorted(Node list);
ErrorCode mergeSortedLists(Node list1, Node list2, Node *mergedOut);
```

קובץ .c

1

```c
#include "merged_list.h"

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

//_________static function_________
static Node createList(int len);
static Node createNode();
static void freeMergedList(Node head);
static void completeList(Node list_not_complete, Node merge_list_current);

ErrorCode mergeSortedLists(Node list1, Node list2, Node *mergedOut)
{
    mergedOut = NULL;
    if (list1 == NULL || list2 == NULL)
    {
        return EMPTY_LIST;
    }

    if (!isListSorted(list1) && !isListSorted(list2))
    {
        return UNSORTED_LIST;
    }

    int len = getListLength(list1) + getListLength(list2);
    Node head = createList(len);
```

2

```c
    Node head = createList(len);
    if (head == NULL)
    {
        return MEMORY_ERROR;
    }
    Node iterator1 = list1, iterator2 = list2, current = head;
    while (iterator1 != NULL || iterator2 != NULL)
    {
        if (iterator1->x <= iterator2->x)
        {
            current->x = iterator1->x;
            iterator1 = iterator1->next;
        }
        else
        {
            current->x = iterator2->x;
            iterator2 = iterator2->next;
        }
        current = current->next;
    }
    if (iterator1 != NULL)
    {
        completeList(iterator1, current);
    }
    else
    {
        completeList(iterator2, current);
    }
```

3

```c
    }
    *mergedOut = head;
    return SUCCESS;
}

static Node createList(int len)
{
    Node head = NULL;
    for (int i = 0; i < len; i++)
    {
        Node new_node = createNode();
        if (new_node == NULL)
        {
            freeMergedList(head);
            return NULL;
        }

        new_node->next = head;
        head = new_node;
    }
    return head;
}

static Node createNode()
{
    Node new_node = (Node)malloc(sizeof(*new_node));
    if (new_node == NULL)
    {
        return NULL;
```

4

```c
        return NULL;
    }
    return new_node;
}

static void freeMergedList(Node head)
{
    if (head == NULL)
    {
        return;
    }

    while (head != NULL)
    {
        Node to_delete = head;
        head = head->next;
        free(to_delete);
    }
}

static void completeList(Node list_not_complete, Node merge_list_current)
{
    while (list_not_complete != NULL)
    {
        merge_list_current->x = list_not_complete->x;
        merge_list_current = merge_list_current->next;
        list_not_complete = list_not_complete->next;
    }
```