תרגיל בית 1-חלק יבש:

2.1-חלק א':

נדרשנו לממש את הפונקציה $mergeSortedLists$ עבור קטע הקוד הבא:

להלן טיפוס Node שמכיל מספר שלם, ערכי שגיאה/הצלחה אפשריים, ושלוש פונקציות:

```
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);
```

רעיון המימוש: ליצור מערך שיכיל את ערכי שני הרשימות, את המערך נמיין ואז נהפוך שוב לרשימה ממוינת כנדרש.

המימוש (בהתחלה יש את הגדרת ה$struct$, ההצהרות שניתנו לנו ופונקציות עזר ובסוף את הפונקציה המבוקשת):

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

```c
/*swap between two elements*/
void swap(int* first, int* second)
{
    int temp=*first;
    *first=*second;
    *second=temp;
    return;
}

/*return the index of the max number in the array*/
int indexOfMax(int array[], int len)
{
    assert( (array!=NULL) && (len>0))
    int i_max=0;
    for(int i=1;i<len;i++)
    {
        if(array[i]>array[i_max])
        {
            i_max=i;
        }
    }
    return i_max;
}
```

```c
/*recursive max sort for sorting array*/
void maxSort(int array[],int len)
{
    if(len==1)
    {
        return;
    }
    int i_max=indexOfMax(array,len);
    swap(&array[len-1],&array[i_max]);
    maxSort(array,len-1);
}
/*free the whole list*/
void destroy(Node list)
{
    if(list==NULL)
    {
        return;
    }
    Node current=list;
    while(current!=NULL)
    {
        Node to_free=current;
        current=current->next;
        free(to_free);
    }
    return;
}
```

```c
/*transform an array to a list
error_code values:MEMORY_ERROR if allocation failed
otherwise SUCCESS
function return value: a list (Node) in case of success, otherwise NULL*/
Node transformToList(int array[],int len,ErrorCode*error_code)
{
    assert((array!=NULL) && (len>0));
    Node new_element=malloc(sizeof(*new_element));
    if(new_element==NULL)
    {
        *error_code=MEMORY_ERROR;
        return NULL;
    }
    Node list=new_element; //a pointer to the beginning of the list
    new_element->x=array[0];
    new_element->next=NULL;
    for(int i=1;i<len;i++)
    {
        new_element->next=malloc(sizeof(*new_element));
        if(new_element==NULL)
        {
            destroy(list);
            *error_code=MEMORY_ERROR;
            return NULL;
        }
        new_element->x=array[i];
        new_element->next=NULL;
        new_element=new_element->next;
    }
    new_element->next=NULL;
    *error_code=SUCCESS;
    return list;
}
```

ההמשך בדף הבא:

```c
Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code)
{
    if(!isListSorted(list1) || (!isListSorted(list2)))
    {
        *error_code=UNSORTED_LIST;
        return NULL;

    }
    if((list1==NULL)&&(list2==NULL)) //there are no elements to copy
    {
        *error_code=NULL_ARGUMENT;
        return NULL;
    }
    int list1_len=getListLength(list1);
    int list2_len=getListLength(list2);
    int total_len=list1_len+list2_len; //>=1 because at least one list is not NULL
    int array=malloc(sizeof(int)*total_len);
    if(array==NULL)
    {
        *error_code=MEMORY_ERROR;
        return NULL;
    }
    Node current1=list1;
    for(int i=0;i<list1_len;i++)
    {
        array[i]=current1->x;
        current1=current1->next
    }
    Node current2=list2;
    for(int i=list1_len;i<total_len;i++)
    {
        array[i]=current2->x;
        current2=current2->next
    }
    maxSort(array,total_len);
    return transformToList(array,total_len,*error_code);
}
```

2.2- חלק ב':

בחלק זה יש לזהות שגיאות בקוד ולתקן אותם.

הקוד המקורי (כולל פירוט מצומצם של הטעויות שמצאנו בו) הינו:

```
//the original code:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* foo(char* str, int* x) { //error- str is not changeable, so a const is needed
                               //convention error-x is not an informative name
                               //convention error- the name of the function isn't a verb
                              //error- didn't check if str is valid-str!=NULL
    char* str2;
    int i;
    x = strlen(str);    //error- missing *: *x=strlen(str)
    str2 = malloc(*x); //error- missing 1 for '/0' : malloc(*x + 1)
                      //error- didn't check allocation- may fail
    for (i = 0; i < *x; i++)
        str2[i] = str[*x - i]; //error- need *x-i-1 in order to not exceed from the array
                                // convention error- missing {} after for
    if (*x % 2 == 0) {        //error-str_len in even-need to print str2;
        printf("%s", str);
    }
    if (*x % 2 != 0)      //error-str_len is odd-need to print str;
    {
        printf("%s", str2);
    }
    return str2;
}
```

כעת נסביר את השגיאות:

שגיאות קונבנציה:

1. השם של המשתנה x בפרמטר של הפונקציה אינו מספק שום מידע על מהות המשתנה (בלי צורך לקרוא את הקוד) ולכן הופך את הקוד לפחות קריא. כלומר על שמות המשתנים להעיד על תפקידם.

2. לפי קונבנציה, שמות של פונקציות צריכות להיות מתוארות כפעלים, אבל שם הפונקציה foo לא מעיד על פעולה כלשהי (מה שגם מקשה להבין מהי מטרת הפונקציה ללא קריאת הקוד בה). בקוד המתוקן שיוצג בהמשך השם שונה.
3. אחרי לולאת ה- for יש לעשות סוגריים מסולסלות ולכתוב בהן את הפעולה שהלולאה עושה, כי כל בלוק צריך להיות מוקף בסוגריים מסולסלות, בפרט בלוק של תנאי.

שגיאות תכנות:

1. אי בדיקה שהערך של str שנשלח לפונקציה הוא תקין: מכיוון שבהמשך הקוד יש פעולות על המחרוזת, יש לבדוק ש: $str! = NULL$ (ולהחזיר שגיאה אם היא כן).
2. בשורה $x = strlen(str)$ צריך להיות כתוב $x^* = strlen(str)$ כי הפונקציה $str(len)$ מחזירה int ואילו x הוא מטיפוס $int *$. יש לעשות את השינוי כי המטרה של x זה לשמור את אורך המחרוזת.
3. בפקודה $str2 = malloc(*x)$ יש לכתוב $str2 = malloc(*x + 1)$ כיוון ש$strlen$ מחזירה את אורך המחרוזת ללא התו $'0/'$ שגם עבורו יש להקצות מקום כדי שבהמשך נוכל להעתיק נכון את המחרוזת כולה.
4. אי בדיקת כישלון של $malloc$ אחרי מה שכתוב בטעות 3- מכיוון ש $malloc$ עלולה להיכשל יש לבדוק תמיד האם הקצאת הזיכרון נכשלה, ולהחזיר שגיאה במקרה שאכן נכשלה.
5. יש לתקן את הפקודה שמבוצעת בלולאת הfor: אנחנו מבצעים $str2[i] = str[*x - i]$ אבל ככה עלולות להיות חריגות מגבולות המערך, למשל עבור $i = 0$ ננסה להעתיק מ$str[*x]$ אבל מכיוון שאורך המחרוזת הוא $*x$ והאינדקסים מתחילים ב- 0, התו האחרון בסוף המחרוזת זה עבור $str[*x - 1]$, לכן ניגש למקום לא חוקי בזיכרון. לפיכך יש לתקן את הקוד ל-$str2[i] = str[*x - i - 1]$
6. לפי ההוראות, אם מספר התווים במחרוזת זוגי יש להדפיס את המחרוזת ההפוכה, אבל אנחנו מדפיסים את המחרוזת (תיקון הקוד בהמשך)
7. עבור מספר אי זוגי של תווים יש להדפיס את המחרוזת אבל אנחנו מדפיסים את המחרוזת ההפוכה (תיקון בהמשך)

נעיר שיש עוד תיקון שהוא יותר בשביל תכנות נכון אבל אינו בהכרח טעות וזה הוספת $const$ לארגומנט של המחרוזת, כי איננו מעוניינים לשנות אותה במהלך הקוד. כעת נציג את הקוד המתוקן עבור השגיאות בהן הבחנו:

```c
//the fixed code: (according to the errors we found)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* reserveStrAndPrintAccordingToLen(const char* str, int* str_len) {
    if(str==NULL)  //The instructions don't state the value of the return value
    {                    //in case of an error, so we returned NULL
        return NULL;
    }
    char* str2=NULL;
    *str_len=strlen(str);
    str2=malloc(strlen(str)+1);
    if(str2==NULL)
    {
        return NULL;  //The instructions don't state the value of the return value
                          //in case of an error, so we returned NULL
    }
    for (int i = 0; i < *str_len; i++)
    {
        str2[i] = str[*str_len - i-1];
    }
    if (*str_len % 2 == 0)
    {
        printf("%s", str2);
    }
    if (*str_len % 2 != 0)
    {
        printf("%s", str);
    }
    return str2;
}
```