**שאלה 2.1:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.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);

Node createNode(int value);

void destroyList(Node list);

bool memoryAllocationError (Node node, Node list, ErrorCode\* error\_code);

// creates new node with x equals value given

Node createNode(int value){

    Node node = malloc(sizeof(\*node));

    if (!node){ //memory allocation error

        return NULL;

    }

    node->x = value;

    node->next = NULL;

    return node;

}

// destroys the list completely.

void destroyList(Node list){

    Node to\_destroy = NULL;

    while(list)

    {

        to\_destroy = list;

        list = list->next;

        free(to\_destroy);

    }

}

//checks if there is memory allocation error in the given node.

//in case of memory allocation error, return true, destroy the list completely, and puts in variable error\_code MEMORY\_ERROR.

bool memoryAllocationError (Node node, Node list, ErrorCode\* error\_code){

    if (!node){

        if(error\_code){

            \*error\_code = MEMORY\_ERROR;

        }

        destroyList(list);

        return true;

    }

    return false;

}

Node mergeSortedLists(Node list1, Node list2, ErrorCode\* error\_code){

    if (getListLength(list1) == 0 || getListLength(list2) == 0){ //unsorted list

        if(error\_code){

            \*error\_code = NULL\_ARGUMENT;

        }

        return NULL;

    }

    if (isListSorted(list1) == 0 || isListSorted(list2) == 0){ //empty list

        if(error\_code){

            \*error\_code = UNSORTED\_LIST;

        }

        return NULL;

    }

    Node ptr\_first\_list = list1;

    Node ptr\_second\_list = list2;

    Node merged\_list = NULL;

    if (ptr\_first\_list->x < ptr\_second\_list->x) {

        merged\_list = createNode(ptr\_first\_list->x);

        ptr\_first\_list = ptr\_first\_list->next;

    } else {

        merged\_list = createNode(ptr\_second\_list->x);

        ptr\_second\_list = ptr\_second\_list->next;

    }

    if (!merged\_list){ //memory allocation error

        return NULL;

    }

    Node iterator = merged\_list;

    while(ptr\_first\_list && ptr\_second\_list){ //there are more nodes on both lists - list 1 and list 2

        if (ptr\_first\_list->x < ptr\_second\_list->x){

            iterator->next = createNode(ptr\_first\_list->x);

            if (memoryAllocationError(iterator->next, merged\_list, error\_code)){ //memory allocation error

                return NULL;

            }

            ptr\_first\_list = ptr\_first\_list->next;

            iterator = iterator->next;

        } else{

            iterator->next = createNode(ptr\_second\_list->x);

            if (memoryAllocationError(iterator->next, merged\_list, error\_code)){ //memory allocation error

                return NULL;

            }

            ptr\_second\_list = ptr\_second\_list->next;

            iterator = iterator->next;

        }

    }

    while (ptr\_first\_list){ //there are more nodes on list 1

        iterator->next = createNode(ptr\_first\_list->x);

        if (memoryAllocationError(iterator->next, merged\_list, error\_code)){ //memory allocation error

            return NULL;

        }

        ptr\_first\_list = ptr\_first\_list->next;

        iterator = iterator->next;

    }

    while (ptr\_second\_list){ //there are more nodes on list 2

        iterator->next = createNode(ptr\_second\_list->x);

        if (memoryAllocationError(iterator->next, merged\_list, error\_code)){ //memory allocation error

            return NULL;

        }

        ptr\_second\_list = ptr\_second\_list->next;

        iterator = iterator->next;

    }

    if(error\_code){

        \*error\_code = SUCCESS;

    }

    return merged\_list;

}

**שאלה 2.2:**

**הקוד המקורי**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* foo(char\* str, int\* x) {

    char\* str2;

    int i;

    x = strlen(str);

    str2 = malloc(\*x);

    for (i = 0; i < \*x; i++)

        str2[i] = str[\*x - i];

    if (\*x % 2 == 0) {

        printf("%s", str);

    }

    if (\*x % 2 != 0)

    {

        printf("%s", str2);

    }

    return str2;

}

**הקוד המתוקן**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// $ - Correctness Error

// @ - Conventions Error

char\* returnReversedStringAndPrint(char\* str,

    int\* length) {                        // @ - 1

    int len = strlen(str);

     if(!length) {                        // $ - 1

        \*length = len;

    }

    char\* str2;

    int i;

    str2 = malloc(sizeof(char)\*(len+1));  // @ - 2, $ - 2

    if(!str2)                             // @ - 3

    {

        return NULL;

    }

    for (i = 0; i < len; i++)

    {                                     // @ - 4

        str2[i] = str[len - i - 1];       // $ - 3

    }

    if (len % 2 != 0) {                   // $ - 4

        printf("%s", str);

    }

    if (len % 2  == 0)

    {

        printf("%s", str2);

    }                                     // $ - 5

    return str2;

}

/\*\*

 \*

 \*  $ - Correctness Error

 \*      1 Check if received valid length pointer,

 \*        if not returns reversed string.

 \*      2 Didn't malloc enough space, no space for "\0"

 \*      3 Add -1 to str[] to not put the '\0' in the end of str in str2.

 \*      4 Fixed wrong logic print

 \*      5 Changed pointer address insted of content, added \*.

 \*

 \*  @ - Conventions Error

 \*      1 Irrelevent function name and irrelevent int name.

 \*      2 Did not use sizeof function.

 \*      3 Did not checked if malloc was succesfull.

 \*      4 Added curly brackets.

 \*

 \*/