מבוא לתכנות מערכות – 234124

אביב תש"ף

תרגיל בית מס. 1-יבש

שם: שלי פרנסיס ת"ז: 316577725

שם: שי יחזקאל ת"ז: 205917883

תאריך: 04/04/2020

2. תרגיל יבשים

2.1 מיזוג רשימות

```
#include <stdbool.h>
#include <stdlib.h> // used for NULL and malloc

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

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

void freeList(Node node);
int getListLength(Node list);
bool isListSorted(Node list);
ErrorCode mergeSortedLists(Node list1, Node list2, Node *merged_out);
ErrorCode checkInputArguements(Node list1, Node list2, int *length1, int *length2,Node *merged_out);
int getNextValueForList(Node *node_in_list1, Node *node_in_list2);
ErrorCode allocateNextNode(Node *current_node, bool first_iteration);

/*  Function checks given arguement for mergeSortedlist functions, and calculates length on the way.
	Error Codes:  MEMORY_ERROR - Allocation error
				NULL_ARGUEMENT,UNSORTED_LIST,EMPTY_LIST - See
"checkInputArguements" function
	Returns    :  ErrorCode indicating successfull/unsuccessfull merging  */
ErrorCode mergeSortedLists(Node list1, Node list2, Node *merged_out)
{
	int length1, length2;
	ErrorCode input_check_code;

	// Check if arguements are valid
	if ((input_check_code = checkInputArguements(list1, list2, &length1, &length2,merged_out)) !=
SUCCESS)
	{
		return input_check_code;
	}

	Node node_in_list1, node_in_list2, node_merged_head;
	node_in_list1 = list1; // Iterate on dummy nodes so original list won't be overriden
	node_in_list2 = list2;

	bool first_iteration = true; //In first iteration we will allocate merged_out, without
cycling to next node
	while ((node_in_list1 || node_in_list2))
	{

		if (allocateNextNode(&node_merged_head, first_iteration) == MEMORY_ERROR)
		{
			freeList(*merged_out); //Will also free node_merged_head
			*merged_out = NULL;
			return MEMORY_ERROR;
		}
```

```
            else if (first_iteration)
            { //Set merged_out node to be linked to the first node
                    first_iteration = false;
                    *merged_out = node_merged_head;
            }

            node_merged_head->x = getNextValueForList(&node_in_list1, &node_in_list2);
        }

        return SUCCESS;
}
/*  Function checks which next value in ascending order value-wise should be added to the merged
list.
            Also iterates the node from which the value was taken
        Error Codes:  None. Assumes both list are valid as they were validated already
        Returns     :  Value to add to the linked list  */
int getNextValueForList(Node *node_in_list1, Node *node_in_list2)
{
        Node *chosen_node;

        if (!(*node_in_list1))
        {
            chosen_node = node_in_list2;
        }
        else if (!(*node_in_list2))
        {
            chosen_node = node_in_list1;
        }

        else if ((*node_in_list1)->x < (*node_in_list2)->x)
        {
            chosen_node = node_in_list1;
        }
        else
        {
            chosen_node = node_in_list2;
        }

        int next_val;
        next_val = (*chosen_node)->x;

        (*chosen_node) = (*chosen_node)->next;
        return next_val;
}

/*  Function frees given node ad all linked nodes, thus freeing the list
        Error Codes:  Allocation error - MEMORY_ERROR
        Returns     :  ErrorCode indicating successfull/unsuccessfull allocations  */
void freeList(Node node)
{
        while (node)
        {
            Node next_node = node->next;
            free(node);
            node = next_node;
        }
}

/*  Function allocates next node to given node, and cycles to the allocated node.
            If it is the first iteration - allocate current node.
        Error Codes:  Allocation error - MEMORY_ERROR
        Returns     :  ErrorCode indicating successfull/unsuccessfull allocations  */
ErrorCode allocateNextNode(Node *current_node, bool first_iteration)
```

```
{
	if (first_iteration)
	{ // Allocate without cycling to next node
		(*current_node) = malloc(sizeof(*(*current_node)));
	}
	else
	{
		(*current_node)->next = malloc(sizeof(*(*current_node)));
		(*current_node) = (*current_node)->next;
	}
	if (!(*current_node)) // If *current_node == NULL
	{
		return MEMORY_ERROR;
	}
	(*current_node)->next = NULL;
	return SUCCESS;
}
/*  Function checks given arguement for mergeSortedlist functions,
                                        and calculates length on the way. According to insctructions
in 2.1.
	Error Codes:  At least one of the lists are empty - NULL_ARGUEMENT
                              At least one of the lists are unsorted - UNSORTED_LIST
                              At least one of the lists are empty - EMPTY_LIST
	Returns    :  ErrorCode indicating valid/invalid arguements  */
ErrorCode checkInputArguements(Node list1, Node list2, int *length1, int *length2,Node *merged_out)
{
	if (!merged_out)
	{
		return NULL_ARGUMENT;
	}
	if(!list1||!list2)
	{
		return EMPTY_LIST;
	}
	if (!isListSorted(list1) || !isListSorted(list2))
	{
		return UNSORTED_LIST;
	}

	*length1 = getListLength(list1);
	*length2 = getListLength(list2);
	if (*length1 == 0 || *length2 == 0)
	{
		return EMPTY_LIST;
	}
	return SUCCESS;
}
}
```

String Duplicator 2.2

2.2.1 מציאת שגיאות

להלן השגיאות שמצאנו בתכנית:

שגיאות תכנות

פקודה	שגיאה
assert(!s);	הפקודה הנוכחית תכשיל כל מחזורת תקינה, ולא תחזיר שגיאה עבור מחרוזות שהן NULL כמצופה.
char *out = malloc(LEN * times);	הקצאת הזכרון הנוכחית אינה מתחשבת בתו ה"\0" שנמצא בסוף כל מחרוזת תקינה, שכן פונקציית strlen מחזירה את מספר התווים ללא "\0".
out = out + LEN; strcpy(out, s);	סדר הפקודות הנ"ל קודם מקדם את המצביע של out ורק לאחר מכן מעתיק את תוכן מחרוזת S לתוכה, כך שבתחילת הריצה נקבל מחרוזת ש"LEN" התווים הראשונים אינם מוגדרים, ואילו באיטרציה האחרונה נגיע לזליגה.
return out;	אנו מחזירים בפקודה זו את המצביע לאחר שקודם בלולאת הfor המקדימה לו, ולא את המצביע לתו הראשון של המחרוזת כמצופה.

**מצאנו עוד שגיאות תכנות אשר תיקנו בפתרון המצורף ולא הוספנו אותם לסעיף זה לאור הדרישה של כתיבת 4 שגיאות בלבד.

שגיאות קונבנציה

מיקום שגיאה	שגיאה
*stringDuplicator	שם הפונקציה אינו תואם את קונבנציית השמות שמגדירה כי הפונקציה צריכה להתחיל בפועל
char *s	שם המשתנה אינו אנפורמטיבי למשתמש כפי שדורשת הקונבנציה.
int LEN	שם המשתנה אינו תואם לקונבנציה שדורשת ששמות משנים יהיו באותיות קטנות

for (int i = 0; i < times; i++) {
out = out + LEN;
strcpy(out, s);
}
	מבנה לולאת הfor אינו כולל הזחות בתוך הblock הפנימי שמוקף בסוגריים מסולסלים

**מצאנו עוד שגיאות קונבנציה אשר תיקנו בפתרון המצורף ולא הוספנו אותם לסעיף זה לאור הדרישה של כתיבת 4 שגיאות בלבד (שם המשתנה out בפתרון שלנו ל-str_out).

2.2.1 תיקון

להלן התיקון המוצא:

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

/*  function duplicates a given string the given amount of times
 Error Codes:  Allocation error in malloc - returns NULL
 Returns    :  Pointer to string duplicated the amount of times requested*/
char *duplicateString(char *str, int times)
{
	assert(str);
	assert(times > 0);
	int len = strlen(str);
	char *str_out = malloc((len * times) + 1); // +1 for the \0
	assert(str_out);
	if (!str_out)
	{
		return NULL;
	}
	for (int i = 0; i < times; i++)
	{// Last character \0 is being overriden except for the last iteration
		strcpy(str_out + i * len, str);
	}
	return str_out;
}
```