

תרגיל בית 1

מגשים:

איה ספירה 206713935

אורי מינץ 314616897

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. שני ה-assert בודקים פרמטרים מהמשתמש, בניגוד לשימוש המתאים של assert שבו אנחנו מנחים שהתוכן בפנים נכון בנוסף עושים assert ל-out במקום לעשות if כדי לבדוק אם ה-malloc עבד.
2. מחזירים out פוינטר לסוף המערך ולא לתחילת הסטרינג.
3. הפונקציה strlen מקבלת char** במקום char*.
4. הסטרינג out מקבל את אורך הסטרינג ולא משאירה מקום ל-'0' במאלוק שלה.
5. בתוך לולאת ה-for השורות בסדר הפוך וזה יוצר מצב שבו ה-string הראשון מועתק למקום של השני וכך הלאה, עד לאחרון שמועתק למקום שלא קיים.

שגיאות קונבנציה

1. שם הפונקציה לא נכתב עם אות גדולה בתחילת המילה השנייה.
2. השם s לפרמטר לא מסביר את המשמעות שלו.
3. המשתנה LEN נכתב באותיות גדולות למרות שהוא משתנה ולא define.
4. אין הזחה ל-for.

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4
5  char *stringDuplicator(char *string , int times){
6      if(string == NULL)
7      {
8          return NULL;
9      }
10     if(times <= 0)
11     {
12         return NULL;
13     }
14     int string_len = strlen(string);
15     char *string_list = malloc((string_len+1) * times);
16     if(string_list == NULL)
17     {
18         return NULL;
19     }
20     char* string_element = string_list;
21     for (int i = 0; i < times ; i++){
22         strcpy(string_element, string);
23         string_element += string_len;
24     }
25     string_element[0] = '\0';
26     return string_list;
27 }
```

תרגיל בית 2

```
1  #include "dry_question.h"
2  #include <stdbool.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <assert.h>
6  /**
7   * @brief creat new node
8   *
9   * @return Node
10  */
11  static Node creatNode();
12
13  /**
14   * @brief destroy the cerrent list
15   *
16   */
17  static void destroyList(Node list);
18
19  /**
20   * @brief copy the list form "from_list" to "to_list"
21   *
22   * @param from_list
23   * @param to_list
24   * @return ErrorCode
25   */
26  static ErrorCode finishList(Node from_list, Node to_list);
27
28  /**
29   * @brief checks if the list are valid
30   *
31   * @param list1
32   * @param list2
```

```

33     * @return ErrorCode
34     */
35     static ErrorCode chackLists(Node list1, Node list2);
36
37     ErrorCode megeSortedLists(Node list1, Node list2, Node *mergedOut)
38     {
39         ErrorCode list_valid= chackLists(list1,list2);
40         if(list_valid != SUCCESS)
41         {
42             return list_valid;
43         }
44         Node iterator1 = list1;
45         Node iterator2 = list2;
46         *mergedOut = creatNode();
47         Node new_list = *mergedOut;
48
49         if(new_list == NULL)
50         {
51             return MEMORY_ERROR;
52         }
53         assert(iterator1 != NULL && iterator2 != NULL);
54         if(iterator1->x <= iterator2->x)
55         {
56             new_list->x = iterator1->x;
57             iterator1 = iterator1->next;
58         }
59         else
60         {
61             new_list->x = iterator2->x;
62             iterator1 = iterator2->next;
63         }

```

```

59     else
60     {
61         new_list->x = iterator2->x;
62         iterator1 = iterator2->next;
63     }
64     Node cerent_node = new_list;
65     while(iterator1 != NULL && iterator2 != NULL)
66     {
67         Node new_node = creatNode();
68         if(new_node == NULL)
69         {
70             destroyList(new_list);
71             return MEMORY_ERROR;
72         }
73         if(iterator1->x <= iterator2->x)
74         {
75             new_node->x = iterator1->x;
76             iterator1 = iterator1->next;
77         }
78         else
79         {
80             new_node->x = iterator2->x;
81             iterator2 = iterator2->next;
82         }
83         cerent_node->next = new_node;
84         cerent_node = new_node;
85     }
86

```

```

87     ErrorCode list_copy_status = finishList(iterator1, cerent_node);
88     if(list_copy_status != SUCCESS)
89     {
90         destroyList(new_list);
91         return MEMORY_ERROR;
92     }
93
94     list_copy_status = finishList(iterator2, cerent_node);
95     if(list_copy_status != SUCCESS)
96     {
97         destroyList(new_list);
98         return MEMORY_ERROR;
99     }
100
101     return SUCCESS;
102 }
103
104 static ErrorCode finishList(Node from_list, Node to_list)
105 {
106     if(from_list == NULL)
107     {
108         return SUCCESS;
109     }
110     while(from_list != NULL)
111     {
112         Node new_node = creatNode();
113         if(new_node == NULL)
114         {
115             return MEMORY_ERROR;
116         }
117         new_node->x = from_list->x;
118         from_list = from_list->next;

```

```

119         to_list->next = new_node;
120         to_list = to_list->next;
121     }
122     to_list->next = NULL;
123     return SUCCESS;
124 }
125
126
127 static Node creatNode()
128 {
129     Node new_node = malloc(sizeof(*new_node));
130     if(new_node == NULL)
131     {
132         return NULL;
133     }
134     return new_node;
135 }
136
137
138 static void destroyList(Node list)
139 {
140     if(list == NULL)
141     {
142         return;
143     }
144     destroyList(list->next);
145     free(list);
146     return;
147 }
148 static ErrorCode chackLists(Node list1, Node list2)
149 {
150     if(list1 == NULL || list2 == NULL)
151     {
152         return EMPTY_LIST;
153     }
154     if(getListLength(list1) <= 0 || getListLength(list2) <= 0)
155     {
156         return EMPTY_LIST;
157     }
158     if(!isListSorted(list1) || !isListSorted(list2))
159     {
160         return EMPTY_LIST;
161     }
162     return SUCCESS;
163 }

```