

סעיף א

בצאו 5 שגיאות תכנתה ו-4 שגיאות קונבנצייה¹ (code convention) בפונקציה הבאה. מטרת הפונקציה היא לשכפל מספר פעמים את המחרוזת המקבלת לתוכה מרוחות וושה. למשל, הקריאה stringDuplicator("Hello", 3) יחזיר את המחרוזת "HelloHelloHello". במקרה של שגיאה בריצת הפונקציה, הפונקציה תחזיר NULL. ניתן להניח שהערך של-times" ציבוי.

```
#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;
}
```

סעיף ב

כתבו גרסה מותוקנת של הפונקציה.

שגיאות קונבנצייה

השגיאות הידועות בפונקציה זו הן:

- 1. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה).
- 2. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה).

השגיאות הידועות בפונקציה זו הן:

- 1. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה ("camelCase").
- 2. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה ("CamelCase").

השגיאות הידועות בפונקציה זו הן:

- 1. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה LEN).
- 2. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה len).

השגיאות הידועות בפונקציה זו הן:

- 1. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה OUT).
- 2. כפולה של סימן נספחה בפונקציה (בזינית גנייה גנייה נספחה strOut).

↙ הגדלת מערך

1 - כ. מגדלת מערך טיכון עוגף אחד, יוציאו מערך כוכב
malloc(LEN * times + 1) בזין גארט, Out מערך

2 - מגדלת מערך שאלות בולס בולס כאות assert נמנע ל-
ההנפקה null בפונקציית assert נמנע בולס בולס כאות assert
assert(time > 0), assert(!s) • null בולס בולס כאות assert(out) •

3 - מגדלת מערך בולס בולס כאות assert(!s) •

4 - מגדלת מערך null בולס בולס כאות assert(!s) •

5 - מגדלת מערך strlent(*s); *str * מילא גיבוב נס, א' כאות assert(!s) •

6 - מגדלת מערך מילא גיבוב נס, א' כאות assert(!s) •

2.1.2 סעיף ב

כתבו גרסה מותוקנת של הפקנציה.

```
2
3 #include <stdio.h>
4 #include <string.h>
5 #include <assert.h>
6 #include <malloc.h>
7
8 char *stringduplicator(char *s, int times , char *(*strcpy)(char * , char *) , int (*strlen) (char *)) {
9     if ( s==null)
10         return null
11     is (times <=0 )
12         return null ;
13     int len = strlen(s);
14     char * str_out_first = malloc((len*times)+1);
15     if (str_out == null )
16         return null;
17     char * str_out=str_out_first ;
18     for (int i=0 ; i<times ; i++)
19     {
20         strcpy(str_out_first,s)
21         str_out_first+=len ;
22     }
23
24     return str_out;
25
26 }
27 }
```

2.2 מיזוג רשימות מקושורות ממוגנות

```
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

```
enum mergeSortedLists( Node left, Node right, Node *merged)
{
    if ( getListLength (left)==0 || getListLength (right)==0)
    {
        mergedOut=NULL_ARGUMENT;
        return EMPTY_LIST;
    }

    if ( !isListSorted(left )== false || !isListSorted(right)== false )
    {
        mergedOut=NULL_ARGUMENT;
        return UNSORTED_LIST;
    }

    Node* ptr_left=left;
    Node* ptr_right=right;

    if((ptr_left->x)>=(ptr_right->x)){
        mergedOut->x = ptr_left->x;
        mergedout->next = NULL;
        ptr_left = ptr_left->next;}
    else{
        mergedOut->x = ptr_right->x;
        mergedout->next = NULL;
        ptr_right = ptr_right->next)}
}
```

כבר אסב נאך ולבינן בז' גנדייה
השורה, ואחר מכן חזרה לאחיה כוכביה.
הה, בז' אונט כטביה.

```

38     node head=mergedOut;
39     while (ptr_left!=NULL && ptr_right!=NULL){
40         if((ptr_left->x)>=(ptr_right->x))
41         {
42             Node newNode = createNode(ptr_left->x);
43             if (newNode = NULL){
44                 return MEMORY_ERROR;
45             }
46             newNode->next=NULL;
47             head->next=newNode;
48             head = head->next;
49         }
50         else
51         {
52             Node newNode = createNode(ptr_right->x);
53             if (newNode = NULL){
54                 return MEMORY_ERROR;
55             }
56             newNode->next=NULL;
57             head->next=newNode;
58             head = head->next;
59         }

```

createNode(ptr_right->x);

```
60
61     while (ptr_left!=NULL)
62     {
63         Node newNode = createNode(ptr_left->x);
64         if (newNode == NULL){
65             ↪ return MEMORY_ERROR;
66         }
67         newNode->next=NULL;
68         head->next=newNode;
69         head = head->next;
70     }
71
72     while (ptr_right!=NULL){
73         Node newNode = createNode(ptr_right->x);
74         if (newNode == NULL){
75             ↪ return MEMORY_ERROR;
76         }
77         newNode->next = NULL;
78         head->next = newNode;
79         head = head->next;
80     }
81     ↪ return SUCCESS;
82 }
83
```

Java 语言的插入操作如何实现
mergesort 和归并排序的实现

```
1  ↗ Node createNode(int x) {  
2      ↗     Node ptr = malloc(sizeof(*ptr));  
3      ↗     if (!ptr) {  
4          ↗         return NULL;  
5      ↗     }  
6  
7      ↗     ptr->x = x;  
8      ↗     ptr->next = NULL;  
9      ↗     return ptr;  
10     ↗ }
```

• UpNext in a doubly ended Queue

```

Node createNode(int x) {
    Node ptr = malloc(sizeof(*ptr));
    if (!ptr) {
        return NULL;
    }

    ptr->x = x;
    ptr->next = NULL;
    return ptr;
}

enum mergeSortedLists( Node left, Node right, Node *merged)
{
    if ( getListLenght (left)==0 || getListLenght (right)==0)
    {
        mergedOut=NULL_ARGUMENT;
        return EMPTY_LIST;
    }

    if ( isListSorted(left )== false || isListSorted(right)== false )
    {
        mergedOut=NULL_ARGUMENT;
        return UNSORTED_LIST;
    }

    Node* ptr_left=left;
    Node* ptr_right=right;

    if((ptr_left->x)>=(ptr_right->x)){
        mergedOut->x = ptr_left->x;
        mergedout->next = NULL;
        ptr_left = ptr_left->next;}
}

```

```
else{  
    mergedOut->x = ptr_right->x;  
    mergedout->next = NULL;  
    ptr_right = ptr_right->next)  
}  
  
node head=mergedOut;  
while (ptr_left!=NULL && ptr_right!=NULL){  
    if((ptr_left->x)>=(ptr_right->x))  
    {  
        Node newNode = createNode(ptr_left->x);  
        if (newNode = NULL){  
            return MEMORY_ERROR;  
        }  
        newNode->next=NULL;  
        head->next=newNode;  
        head = head->next;  
    }  
    else  
    {  
        Node newNode = createNode(ptr_right->x);  
        if (newNode = NULL){  
            return MEMORY_ERROR;  
        }  
        newNode->next=NULL;  
        head->next=newNode;  
        head = head->next;  
    }  
}
```

```
while (ptr_left!=NULL)
{
Node newNode = createNode(ptr_left->x);
if (newNode = NULL){
return MEMORY_ERROR;
}
newNode->next=NULL;
head->next=newNode;
head = head->next;
}

while (ptr_right!=NULL){
Node newNode = createNode(ptr_right->x);
if (newNode = NULL){
return MEMORY_ERROR;
}
newNode->next = NULL;
head->next = newNode;
head = head->next;
}
return SUCCESS;
}
```