מבוא לתכנות מערכות

תרגיל בית 1

סמסטר אביב 2019

# חלק יבש

# מגישים:

ברק ביבר- 308224989

סבטלנה לדיגין - 319649778

# Exercise 1:

Original version:

// name should be a verb (convention mistake 1)  
// char \*s is an unapproved abbreviation (convention mistake 2)  
char \*stringDuplicator(char \*s, int times) {  
 assert(!s); // assert that s is NULL instead of not NULL (coding error 1)  
 assert(times > 0);  
 int LEN = strlen(s); // variable name should be lower case (convention mistake 3)  
 char \*out = malloc(LEN \* times); // Should be +1 for the NULL terminator (coding error 2)  
 assert(out);  
 // indent the for body (convention mistake 4)   
 for (int i = 0; i < times; i++) {  
 out = out + LEN; // Skips the first LEN bytes of out (coding error 3)   
 strcpy(out, s);   
 }  
 return out;  
 // Should use a pointer to out in the for loopinstead of using out since we are returning   
 // the out+LEN\*times (coding error 4)  
}

All in all 4 convention mistakes and 4 coding errors

Fixed version:

char \*duplicateString(char \*str, int times) {  
 assert(str);   
 assert(times > 0);  
 int len = strlen(str);  
 char \*out = malloc(len \* times + 1);  
 assert(!out);  
 char \*out\_ptr = out;  
   
 for (int i = 0; i < times; i++) {  
 strcpy(out\_ptr, str);  
 out\_ptr = out\_ptr + len;  
 }  
  
 return out;  
}

# Exercise 2:

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <stdbool.h>*

typedef struct node\_t {

int x;

struct node\_t \*next;

} \*Node;

typedef enum {

SUCCESS=0,

MEMORY\_ERROR,

EMPTY\_LIST,

UNSORTED\_LIST,

} ErrorCode;

int getListLength(Node list);

bool isListSorted(Node list);

ErrorCode mergeSortedLists(Node list1, Node list2, Node \*mergedOut);

Node createNode(int x);

void freeList(Node listHead);

ErrorCode concatenateList(Node listSource, Node listOut);

Node createNode(int x) {

Node new\_node = malloc(sizeof(struct node\_t));

*if*(!new\_node) {

*return* NULL;

}

new\_node->x = x;

new\_node->next = NULL;

*return* new\_node;

}

void freeList(Node listHead) {

*while*(listHead) {

Node next = listHead->next;

free(listHead);

listHead = next;

}

}

ErrorCode concatenateList(Node listSource, Node listOut) {

*while* (listSource) {

Node new\_node = createNode(listSource->x);

*if*(!new\_node) {

*return* MEMORY\_ERROR;

}

listOut->next = new\_node;

listOut = new\_node;

listSource = listSource->next;

}

*return* SUCCESS;

}

ErrorCode mergeSortedLists(Node list1, Node list2, Node \*mergedOut) {

if(!mergedOut){

return MEMORY\_ERROR;

}

*if*((!list1) || (!list2)) {

*return* EMPTY\_LIST;

}

*if*((!isListSorted(list1)) || (!isListSorted(list2))) {

*return* UNSORTED\_LIST;

}

Node merged\_list;

*if*(list1->x >= list2->x) {

merged\_list = createNode(list1->x);

list1 = list1->next;

} *else* {

merged\_list = createNode(list2->x);

list2 = list2->next;

}

*if*(!merged\_list) {

*return* MEMORY\_ERROR;

}

Node merged\_list\_current\_node = merged\_list;

*while*(list1 && list2) {

Node new\_node = NULL;

*if*(list1->x >= list2->x) {

new\_node = createNode(list1->x);

list1 = list1->next;

} *else* {

new\_node = createNode(list2->x);

list2 = list2->next;

}

*if*(!new\_node) {

freeList(merged\_list);

*return* MEMORY\_ERROR;

}

merged\_list\_current\_node->next = new\_node;

merged\_list\_current\_node = new\_node;

}

// Copy the remaining nodes *from* list1 or list 2

ErrorCode error;

error = concatenateList(list1, merged\_list\_current\_node);

*if*(error != SUCCESS) {

freeList(merged\_list);

*return* MEMORY\_ERROR;

}

error = concatenateList(list2, merged\_list\_current\_node);

*if*(error != SUCCESS) {

freeList(merged\_list);

*return* MEMORY\_ERROR;

}

\*mergedOut = merged\_list;

*return* SUCCESS;

}