מת"מ תרגיל 2- חלק יבש

שאלה 1:

שגיאות נכונות:

1) הפונקציה מתעלמת מערך הפוינטר out_len הנתון וכותבת את הערך למקום חדש (שורה 4). בנוסף, מיקום האזור החדש בזכרון לא ידוע מחוץ לפונקציה ולכן לא נגיש וגורם לדליפת זכרון.
2) הפונקציה לא לוקחת בחשבון את תו ה-NULL שאמור להיות בסוף המשתנה OUT. היא גם לא מקצה לו מקום בזכרון (שורות 6 ו-15) וגם לא דואגת לשים תו כזה בסוף המחרוזת.
3) הפונקציה לא בודקת האם הקצאת הזכרון הצליחה (שורות 4 ו-6).
4) הפונקציה לא מבצעת בדיקת קלט. אין בדיקה שהפוינטרים ל-out_len ו-str תקינים.

שגיאות תכנות נכון:

1) כפילות קוד - קטע הקוד בתוך ה-if (שורות 4-9) כמעט זהה לקוד בתוך ה-else (שורות 13-19).
2) המשתנה OUT באותיות גדולות למרות שהוא לא קבוע.

הפונקציה המתוקנת:

```
char* partialCopyString(char* str, bool copy_even, int* out_len) {
    char* out_str;
    if (NULL == str || NULL == out_len) return NULL;
    *out_len = strlen(str) / 2;
    if (!copy_even) {
        *out_len += strlen(str) % 2;
    }
    out_str = malloc(*out_len + 1);
    if (NULL == out_str) return NULL;
    for (int i = 0; i < strlen(str); i += 2) {
        out_str[i / 2] = str[i + (int)copy_even];
    }
    out_str[*out_len] = '\0';
    return out_str;
}
```

שאלה 2:

```
/**
* Creates a newly allocated copy of a given node.
*
* @return
*   A newly allocated copy of the original nodes.
*   NULL if the node is NULL or in any case of memory allocation failure.
*/
static Node copyNode(Node src) {
    if (NULL == src) return NULL;
    Node clone = malloc(sizeof(struct node_t));
    if (NULL == clone) return NULL;
    clone->data = src->data;
    clone->next = src->next;
    return clone;
}

/**
* Frees all memory allocated for the given node list.
* The function is provided with the first node in the list and iterates
* through all the nodes linked and frees them.
* This function can receive NULL. In this case, no action will be taken.
*/
static void freeList(Node first_node) {
    Node next_node;
    while (first_node != NULL) {
        next_node = first_node->next;
        free(first_node);
        first_node = next_node;
    }
}

/**
* Prints all list's node's data into the screen.
* The function is provided with the first node in the list, and how
* many steps to do when walking through the list. for example:
* if steps=1 it will print all nodes, and if steps=2 it will print
* only the un-even ones.
* Each data is printed with a space immediately after it.
* This function can receive NULL. In this case, no action will be taken.
*/
```

```
static void printList(Node first_node, int steps) {
    Node node_to_print = first_node;
    int current_index = 0;
    while (node_to_print != NULL) {
        if (0 == current_index % steps) {
            printf("%d ", node_to_print->data);
            current_index = 0;
        }
        node_to_print = node_to_print->next;
        current_index++;
    }
}

bool printCheckmarkOrdered(Node first_node) {
    if (NULL == first_node) return true;
    if (NULL == first_node->next) {
        printList(first_node, 1);
        return true;
    }
    Node original_list_iterator = first_node->next;
    Node reversed_list_head = copyNode(first_node->next);
    Node previous_reversed_list_head = NULL;
    reversed_list_head->next = NULL;
/*      Iterate through all the nodes in the original list,
*       and make a copied list containing all the even nodes, with reversed order */
    while (original_list_iterator->next != NULL &&
        original_list_iterator->next->next != NULL) {
        previous_reversed_list_head = reversed_list_head;
        reversed_list_head = copyNode(original_list_iterator->next->next);
        if (NULL == reversed_list_head) {
            freeList(previous_reversed_list_head);
            return false;
        }
        reversed_list_head->next = previous_reversed_list_head;
        original_list_iterator = original_list_iterator->next->next;
    }
/*  Print all the un-even nodes in the original list, and then
*   print all the nodes in the reversed list, which contains only the even nodes */
    printList(first_node, 2);
    printList(reversed_list_head, 1);
    freeList(reversed_list_head);
    return true;
}
```