מת"מ – תרגיל 1 – חלק יבש

<u>מגישים:</u>

שם : אורי זוהר   ת"ז: 205960750

שם : אביב אור-אל   ת"ז: 316114073

Question 1:

Code Errors:

1. `int LEN = strlen(*s);` - strlen requires type Char* argument.

(s) is type char* but the actual contents of s variable were sent.

2. `char *out = malloc(LEN * times);` - malloc returns type (*void) pointer, need to put (char*) before.

3. in addition to the above, malloc requires the amount of bytes to use, therefore the correct way is (sizeof(*char)*len*times).

So a "(char*)" was required before the malloc.

4. `out = out + LEN;` - wrong advancing. The return value of out will be the last place in the char* array.

5. `return out;` - returns the last pointer instead of the pointer to the first character.

Convention Errors:

1. Using shortcuts as names – LEN instead of length.
2. Variables should be lower cases only. (LEN isn't).
3. Name of function should start with lower case, and every other word should start with upper case. (stringduplicator).
4. Function name should be phrased as verb. (correct way is possibly duplicateStrings).

Question 2:

```c
Node createNode(){
    Node node = malloc(sizeof(*node));
    if(node == NULL) return NULL;
    node->next = NULL;
    return node;
}
ErrorCode insertToNode(Node main, Node list1){
    while(main->next != NULL){
        main = main->next;
    }
    Node to_add = createNode();
    if(to_add == NULL) return MEMORY_ERROR;
    to_add->x = list1->x;
    main->next = to_add;
    return SUCCESS;
}
void freeNodes(Node node){
    while(node!=NULL) {
        Node to_free = node;
        node = node->next;
        free(to_free);
    }
}
ErrorCode mergeSortedLists(Node list1, Node list2, Node *mergedOut){
    if(list1 == NULL || list2 == NULL){
        mergedOut = NULL;
        return EMPTY_LIST;
    }
    *mergedOut = createNode();
    if(*mergedOut == NULL) return MEMORY_ERROR;
    while(list1!=NULL || list2!=NULL){
        if(list1 != NULL && list2 != NULL){
            if(list1->x <= list2->x){
                if(insertToNode(*mergedOut,list1) != SUCCESS){
                    freeNodes(*mergedOut);
                    return MEMORY_ERROR;
                }
                list1 = list1->next;
                continue;
            }
            else{
                if(insertToNode(*mergedOut,list2) != SUCCESS){
                    freeNodes(*mergedOut);
                    return MEMORY_ERROR;
                }
                list2 = list2->next;
                continue;
            }
        }
        else if(list1 == NULL && list2 != NULL){
            if(insertToNode(*mergedOut,list2) != SUCCESS){
                freeNodes(*mergedOut);
                return MEMORY_ERROR;
            }
            list2 = list2->next;
            continue;
        }
        else if(list1 != NULL && list2 == NULL){
            if(insertToNode(*mergedOut,list1) != SUCCESS){
                freeNodes(*mergedOut);
                return MEMORY_ERROR;
            }
            list1 = list1->next;
            continue;
        }
        break;
    }
    Node to_free = (*mergedOut);
    (*mergedOut) = (*mergedOut)->next;
    free(to_free);
    return SUCCESS;
}
```