Week 2

Problem :-

https://www.hackerrank.com/challenges/merge-two-sorted-linked-lists/problem

Code :-

```
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();

typedef struct SinglyLinkedListNode SinglyLinkedListNode;
typedef struct SinglyLinkedList SinglyLinkedList;

struct SinglyLinkedListNode {
   int data;
   SinglyLinkedListNode* next;
};

struct SinglyLinkedList {
   SinglyLinkedListNode* head;
   SinglyLinkedListNode* tail;
};

SinglyLinkedListNode* create_singly_linked_list_node(int node_data) {
   SinglyLinkedListNode* node = malloc(sizeof(SinglyLinkedListNode));

   node->data = node_data;
   node->next = NULL;
```

```
    return node;
}

void insert_node_into_singly_linked_list(SinglyLinkedList**
singly_linked_list, int node_data) {
    SinglyLinkedListNode* node =
create_singly_linked_list_node(node_data);

    if (!(*singly_linked_list)->head) {
        (*singly_linked_list)->head = node;
    } else {
        (*singly_linked_list)->tail->next = node;
    }

    (*singly_linked_list)->tail = node;
}

void print_singly_linked_list(SinglyLinkedListNode* node, char* sep,
FILE* fptr) {
    while (node) {
        fprintf(fptr, "%d", node->data);

        node = node->next;

        if (node) {
            fprintf(fptr, "%s", sep);
        }
    }
}

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;
```

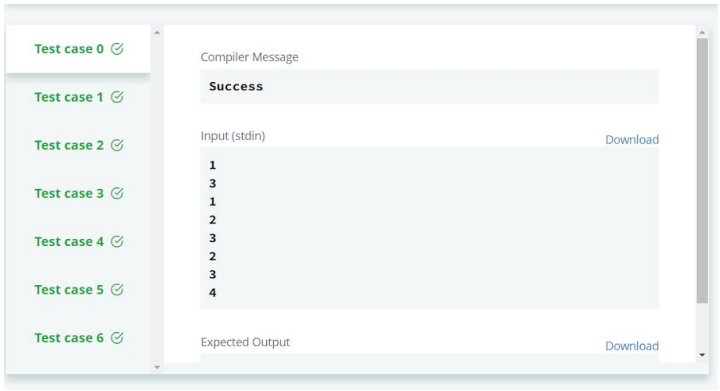
```
        free(temp);
    }
}

SinglyLinkedListNode* mergeLists(SinglyLinkedListNode* headA,
SinglyLinkedListNode* headB) {

    if (headA == NULL && headB == NULL)
      return NULL;
    else if (headA == NULL)
      return headB;
    else if (headB == NULL)
      return headA;

    if (headA->data <= headB->data)
      headA->next = mergeLists(headA->next, headB);
    else {
      SinglyLinkedListNode *temp = headB;
      headB = headB->next;
      temp->next = headA;
      headA = temp;
      headA->next = mergeLists(headA->next, headB);
    }
    return headA;

}


int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* tests_endptr;
    char* tests_str = readline();
    int tests = strtol(tests_str, &tests_endptr, 10);
```

```
    if (tests_endptr == tests_str || *tests_endptr != '\0') {
exit(EXIT_FAILURE); }

    for (int tests_itr = 0; tests_itr < tests; tests_itr++) {
        SinglyLinkedList* llist1 = malloc(sizeof(SinglyLinkedList));
        llist1->head = NULL;
        llist1->tail = NULL;

        char* llist1_count_endptr;
        char* llist1_count_str = readline();
        int llist1_count = strtol(llist1_count_str,
&llist1_count_endptr, 10);

        if (llist1_count_endptr == llist1_count_str ||
*llist1_count_endptr != '\0') { exit(EXIT_FAILURE); }

        for (int i = 0; i < llist1_count; i++) {
            char* llist1_item_endptr;
            char* llist1_item_str = readline();
            int llist1_item = strtol(llist1_item_str,
&llist1_item_endptr, 10);

            if (llist1_item_endptr == llist1_item_str ||
*llist1_item_endptr != '\0') { exit(EXIT_FAILURE); }

            insert_node_into_singly_linked_list(&llist1, llist1_item);
        }

        SinglyLinkedList* llist2 = malloc(sizeof(SinglyLinkedList));
        llist2->head = NULL;
        llist2->tail = NULL;

        char* llist2_count_endptr;
        char* llist2_count_str = readline();
        int llist2_count = strtol(llist2_count_str,
&llist2_count_endptr, 10);
```

```
        if (llist2_count_endptr == llist2_count_str ||
*llist2_count_endptr != '\0') { exit(EXIT_FAILURE); }

        for (int i = 0; i < llist2_count; i++) {
            char* llist2_item_endptr;
            char* llist2_item_str = readline();
            int llist2_item = strtol(llist2_item_str,
&llist2_item_endptr, 10);

            if (llist2_item_endptr == llist2_item_str ||
*llist2_item_endptr != '\0') { exit(EXIT_FAILURE); }

            insert_node_into_singly_linked_list(&llist2, llist2_item);
        }

        SinglyLinkedListNode* llist3 = mergeLists(llist1->head,
llist2->head);

        char *sep = " ";

        print_singly_linked_list(llist3, sep, fptr);
        fprintf(fptr, "\n");

        free_singly_linked_list(llist3);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);
```

```
    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) { break; }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] ==
'\n') { break; }

        size_t new_length = alloc_length << 1;
        data = realloc(data, new_length);

        if (!data) { break; }

        alloc_length = new_length;
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
    }

    data = realloc(data, data_length);

    return data;
}
```

Screenshot :-


Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
Compiler Message
Success
Input (stdin)
Download
1
3
1
2
3
2
3
4
Expected Output
Download