**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 :-

