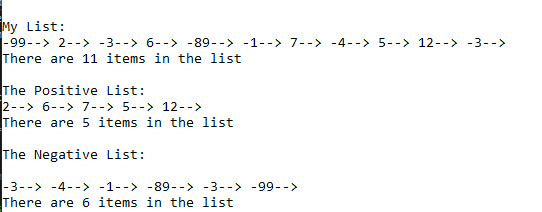
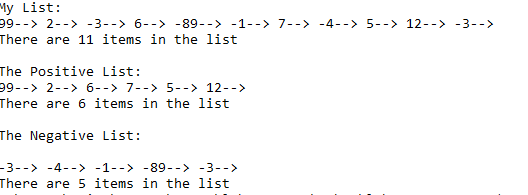
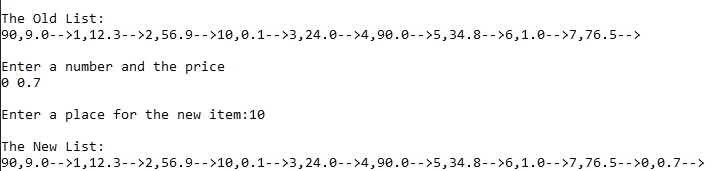
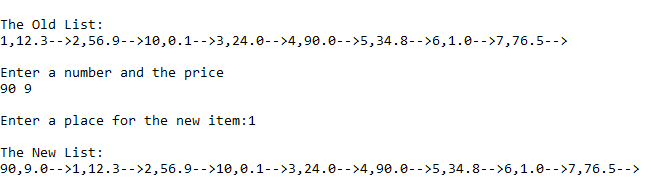
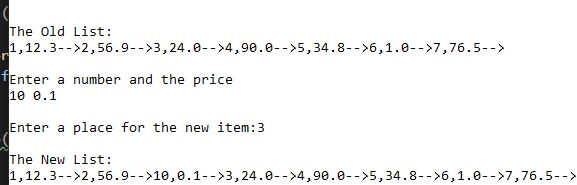
**מעבדה 7 – מת"מ -   
מגישים:  
אורי מלכא – 314862996  
אלן ציפין - 313206062  
  
שאלה 1 קוד:  
שאלה 1 פלטים:**  **שאלה 2 קוד:  
שאלה 3 קוד:  
  
  
  
שאלה 3 פלטים**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

typedef struct Item

{

int num;

struct Item\* next;

}Item;

typedef struct List

{

Item\* head;

Item\* tail;

int count;

}List;

void Error\_Msg(char\*);

/\*

Function name:Error\_Msg

Input: char\*

Output:void

Function Algorithm:prints a string and exits

\*/

void AddAsFirst(Item\*, List\*);

/\*

Function name: AddAsFirst

Input: item\*, list\*

Output: void

Function Algorithm: adds a node to the top of a linked list

\*/

void AddAsLast(Item\*, List\*);

/\*

Function name: AddAsLast

Input: item\*, list\*

Output: void

Function Algorithm: adds a node to the end of a linked list

\*/

void MoveToAnotherList(List\*, List\*, List\*);

/\*

Function name: MoveToAnotherList

Input: 3 list\*

Output: void

Function Algorithm: sorts between nodes

\*/

void CreateList(List\*, FILE\*);

/\*

Function name: CreateList

Input:file\*, list\*

Output: void

Function Algorithm: create a list

\*/

void PrintItem(Item\*);

/\*

Function name: PrintItem

Input:item\*

Output: void

Function Algorithm: print an item struct fields.

\*/

void PrintList(List\*, char\*);

/\*

Function name: PrintList

Input:List\*, char\*

Output: void

Function Algorithm: print an List of item struct

\*/

void DeleteList(List\*);

/\*

Function name: DeleteList

Input:List\*

Output: void

Function Algorithm: delete a linked list

\*/

int main()

{

List L, Posit, Negat;

FILE\* in = fopen("ThreeLists.txt", "rt");

if (in == NULL)

Error\_Msg("input file is wrong");

L.head = NULL;

L.tail = NULL;

L.count = 0;

Posit.head = NULL;

Posit.tail = NULL;

Posit.count = 0;

Negat.head = NULL;

Negat.tail = NULL;

Negat.count = 0;

CreateList(&L, in);

PrintList(&L, "\nMy List:\n");

MoveToAnotherList(&L, &Posit, &Negat);

PrintList(&Posit, "\n\nThe Positive List:\n");

PrintList(&Negat, "\n\nThe Negative List:\n\n");

fclose(in);

DeleteList(&Posit);

DeleteList(&Negat);

return 0;

}

void Error\_Msg(char\* msg)

{

printf("\n%s", msg);

exit(1);

}

void CreateList(List\* L, FILE\* f)

{

int value;

Item\* temp;

while (fscanf(f, "%d", &value) == 1)

{

temp = (Item\*)malloc(sizeof(struct Item));

if (temp == NULL)

{

DeleteList(L);

Error\_Msg("Memory!");

}

temp->num = value;

temp->next = NULL;

if (L->head == NULL)

L->head = temp;

else

L->tail->next = temp;

L->tail = temp;

L->count++;

}

}

void PrintItem(Item\* node)

{

printf("%d--> ", node->num);

}

void PrintList(List\* L, char\* title)

{

Item\* temp = L->head;

printf("%s", title);

while (temp)

{

PrintItem(temp);

temp = temp->next;

}

printf("\nThere are %d items in the list", L->count);

}

void AddAsFirst(Item\* x, List\* n) {

if (n->head == NULL) //case for empty list

n->tail = x;

x->next = n->head; //add a node to a linklied list from the head(top)

n->head = x;

n->count++;

}

void AddAsLast(Item\* x, List\* n) {

x->next = NULL;

//we use tail to add x to the linked list end

if (n->tail == NULL) //case for an empty list.

n->head = x;

else

n->tail->next = x;

n->tail = x;

n->count++; //we increased the size of the linked list so we update the counter.

}

void DeleteList(List\* l) {

//delete list by removing node from the head and link him to the other node and free the head. also updating the new head

while (l->head!= NULL)

{

Item\* temp;

temp = l-> head;

l->head = l->head->next;

free(temp);

l->count--;

}

}

void MoveToAnotherList(List\* main, List\* Posit, List\* Negat) {

int i;

Item\* temp=NULL;

while (main->head) {

//we use temp item to isolate the node that we want to move.

temp = main->head;

main->head = main->head->next; //we save in temp the node and we overwrite the node.

temp->next = NULL; //to send it as a node and not a list.

if (temp->num >= 0)

{

AddAsLast(temp, Posit);

}

else

AddAsFirst(temp, Negat);

main->count--;

}

}

#include<stdio.h>

#include<stdlib.h>

#define N 5

typedef struct Item

{

int num;

struct Item\* next;

}\*PItem;

void Error\_Msg(char\*);

void CreateListFromArray(PItem\*, PItem\*, int\*);

void DeleteList(PItem\*);

void ListDisplay(PItem);

int ListLength(PItem);

int main()

{

int Arr[N] = { 3,4,1,0,8 };

PItem list = NULL, tail = NULL;

CreateListFromArray(&list, &tail, Arr);

printf("The length of the list is %d members\n", ListLength(list));

printf("\nThe list is:\n");

ListDisplay(list);

DeleteList(&list);

tail = NULL;

return 0;

}

void Error\_Msg(char\* msg)

{

printf("\n%s", msg);

exit(1);

}

void CreateListFromArray(PItem\* head, PItem\* tail, int\* Arr)

{

int i;

PItem temp;

char msg[10] = "Memo!";

for (i = 0; i < N; i++)

{

temp = (PItem)malloc(sizeof(struct Item));

if (temp == NULL)

{

DeleteList(head);

Error\_Msg(msg);

}

temp->num = Arr[i];

temp->next = NULL;

if (\*head == NULL)

\*head = temp;

else

(\*tail)->next = temp;

\*tail = temp;

}

}

void DeleteList(PItem\* head)

{

PItem tmp = \*head;

while (\*head)

{

tmp = \*head;

\*head = (\*head)->next;

free(tmp);

}

}

int ListLength(PItem q) {

if (q == NULL) return 0;

return 1 + ListLength(q->next);

}

void ListDisplay(PItem q) {

if (q == NULL) return;

printf("%d->", q->num);

ListDisplay(q->next);

}

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

typedef struct Item

{

int num;

float price;

struct Item\* next;

}\*PItem;

void Error\_Msg(char\*);

/\*

Function name:Error\_Msg

Input: char\*

Output:void

Function Algorithm:prints a string and exits

\*/

void AddNewItem(PItem\*, PItem\*, int, int, float);

/\*

Function name:AddNewItem

Input: PItem\*, PItem\*, int, int, float

Output:void

Function Algorithm:ADD A NEW ITEM

\*/

void CreateList(PItem\*, PItem\*, FILE\*);

/\*

Function name : CreateList

Input : file\*, list\*

Output : void

Function Algorithm : create a list

\*/

void PrintItem(PItem);

/\*

Function name : PrintItem

Input : PItem

Output : void

Function Algorithm : print an item struct fields.

\*/

void PrintList(PItem, char\*);

/\*

Function name: PrintList

Input:PItem, char\*

Output: void

Function Algorithm: print an List of item struct

\*/

void WriteListToFile(PItem, FILE\*);

/\*

Function name: PrintList

Input:PItem, char\*

Output: void

Function Algorithm: copy the linked list to a file.

\*/

void DeleteList(PItem\*);

/\*

Function name: DeleteList

Input:PItem\*

Output: void

Function Algorithm: delete a linked list

\*/

int ListLength(PItem q);

/\*

Function name: ListLength

Input:PItem

Output: int

Function Algorithm: find the size of a linked list.

\*/

int main()

{

int a, n;

float b;

PItem head = NULL, tail = NULL;

FILE\* in = fopen("input\_price.txt", "rt");

if (in == NULL)

Error\_Msg("input file is wrong");

CreateList(&head, &tail, in);

PrintList(head, "\nThe Old List:\n");

printf("\n\nEnter a number and the price\n");

scanf("%d%f", &a, &b);

printf("\nEnter a place for the new item:");

scanf("%d", &n);

AddNewItem(&head, &tail, n, a, b);

PrintList(head, "\nThe New List:\n");

fclose(in);

WriteListToFile(head, in);

DeleteList(&head);

tail = NULL;

return 0;

}

void Error\_Msg(char\* msg)

{

printf("\n%s", msg);

exit(1);

}

void CreateList(PItem\* head, PItem\* tail, FILE\* f)

{

int a;

float b;

PItem temp;

while (fscanf(f, "%d %f", &a, &b) == 2)

{

temp = (PItem)malloc(sizeof(struct Item));

if (temp == NULL)

{

DeleteList(head);

Error\_Msg("Memmory!");

}

temp->num = a;

temp->price = b;

temp->next = NULL;

if (\*head == NULL)

\*head = temp;

else

(\*tail)->next = temp;

\*tail = temp;

}

}

void PrintItem(PItem node)

{

printf("%d,%.1f-->", node->num, node->price);

}

void PrintList(PItem head, char\* title)

{

printf("%s", title);

while (head)

{

PrintItem(head);

head = head->next;

}

}

void DeleteList(PItem\* head)

{

PItem tmp = \*head;

while (\*head)

{

tmp = \*head;

\*head = (\*head)->next;

free(tmp);

}

}

void AddNewItem(PItem\* head, PItem\* tail, int n, int a, float b) {

int i, size;

PItem new, buffer = \*head, temp;

size = ListLength(\*head); //get the size of the linked list

//create a new node to store the information.

new = (PItem)malloc(sizeof(struct Item));

if (new == NULL)

{

DeleteList(head);

Error\_Msg("Memmory!");

}

// initialize the new item to add.

new->num = a;

new->price = b;

new->next = NULL;

if (n != 1 &&n<size+1) { //cases for the linked list expect the ends.

for (i = 1; i < n - 1; i++) {

buffer = buffer->next; //bufffer = the node in the n-1 place

}

temp = buffer->next;

buffer->next = new;

new->next = temp;

}

if (n == 1) { //if we want to add as first

new->next = buffer;

\*head = new;

}

if (n == size+1) { //add as last

(\*tail)->next = new;

new->next = NULL;

}

if (n > size+1) //incase for out of range number we just print the old one agian and as the first and print error.

printf("\nError! the size is %d not in range!!\n", size);

}

int ListLength(PItem q) {

if (q == NULL) return 0;

return 1 + ListLength(q->next);

}

void WriteListToFile(PItem head, FILE\* out) {

PItem buffer;

buffer = head; //to save the pointer

out = fopen("input\_price.txt", "w");

if (out == NULL)

Error\_Msg("input file is wrong");

while (buffer)

{

fprintf(out, "%d %.2f ", buffer->num, buffer->price);

buffer = buffer->next; //buffer ++;

}

fclose(out);

}