**רשימה מקושרת חד כיוונית:**

**ADD TO LIST**

void add (List\_ptr \*hptr,int data){

List\_ptr p,new;

p = \*hptr;

new = (List\_ptr) malloc(sizeof(List));

if (!new){

printf("\n cannot allocate memory \n");

exit(0);}

new->data=data;

new->next = p;

\*hptr = new;

}

**IS EXIST IN LIST**

int isExist (List\_ptr hptr, char name[]){

int res;

while(hptr){

res = strcmp(hptr->name,name1);

if(!res)

return 0;

hptr = hptr->next;}

return 1;

}

**PRINT LIST**

void print (List\_ptr h)

{

while(h){

printf("%s\n", h->name);

h = h->next;}

}

**FREE LIST**

void freelist (List\_ptr \*head)

{

List\_ptr p;

while (\*head){

p=\*head;

\*head = p->next;

free(p);}

}

**DELETE FROM LIST**

void deleteItem (List\_ptr\* head\_ref, char key[])

{

List\_ptr temp=\*head\_ref,prev;

if (temp != NULL && !strcmp(temp->name,key))

{

\*head\_ref = temp->next;

free(temp);

return;}

while (temp != NULL && strcmp(temp->name,key))

{

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

**REVERSE LIST**

List\_ptr reverse\_List (List\_ptr rev, List\_ptr rest)

{

List\_ptr temp;

if(!rest){

return rev;}

temp = rest->next;

rest->next = rev;

return reverse\_List(rest,temp);

}

List\_ptr reverse(List\_ptr\*h)

{

return \*h = reverse\_List(NULL,\*h);

}

**רשימה מקושרת דו-כיוונית:**

**CREATE DOUBLY-LINED LIST**

void add\_List(struct node \*\*head,int data)

{

struct node\*new\_node = (struct node\*) malloc(sizeof(members));

struct node\*p1= \*head;

if (!new\_node){

printf("\nCouldn't allocate memory\n");

exit(0);}

new\_node->data = data;

/\*add to empty list\*/

if(!(\*head))

{

new\_node->next = NULL;

new\_node->prev = NULL;

\*head=new\_node;

return;

}

/\*add to end of list\*/

while(p1->next){

p1=p1->next;

}

p1->next = new\_node;

new\_node->prev = p1;

new\_node->next = NULL;

}

**PRINT DOUBLY-LINKED LIST**

void print(members \* list){

struct node \* ptr = list;

while(ptr) {

printf("%d - %d\n", ptr->data);

ptr = ptr->next;

}

}

**RETURN NODE OF DOUBLY LINKED LIST**

struct node\* search(struct node\* head, int key)

{

struct node \*curr = head;

if (head == NULL)

return NULL;

/\* Find the required node\*/

while (curr->ID != key) {

if (curr->next == NULL) {

printf("\nGiven node is not found"

" in the list!!!");

break;

}

curr = curr->next;

}

return curr;

}

**DELETE NODE FROM DOUBLY LINKED LIST**

void deleteNode(struct node\*\* head\_ref, struct node\* del)

{

/\* base case \*/

if (\*head\_ref == NULL || del == NULL)

return;

/\* If node to be deleted is head node \*/

if (\*head\_ref == del)

\*head\_ref = del->next;

/\* Change next only if node to be deleted is NOT the last node \*/

if (del->next != NULL)

del->next->prev = del->prev;

/\* Change prev only if node to be deleted is NOT the first node \*/

if (del->prev != NULL)

del->prev->next = del->next;

/\* Finally, free the memory occupied by del\*/

free(del);

return;

}

**רשימה מקושרת מעגלית:**

**ADD TO CIRCULAR LIST**

void add2list (ptr \*hptr, int n)

{

ptr p1, p2, t;

t = (ptr)malloc(sizeof(item));

if(!t){

printf("\n cannot allocate memory \n");

exit(0);

}

t->data = n;

if(!(\*hptr)){

t->next = t;

\*hptr=t;

return;

}

p1 = \*hptr;

p2=p1;

p1=p1->next;

while(p1!=\*hptr)/\*stop when p1 is head of list and p1 is last of circular list\*/

{

p2=p1;

p1=p1->next;

}

p2->next=t;/\*update the new last item in the list\*/

t->next=p1;/\*compliting to circular list\*/

\*hptr = t;

}

**PRINT CIRCULAR LIST**

void printlist(ptr head)

{

ptr temp=head;

do{

printf("\n%d", temp->data);

temp= temp->next;}

while (head!=temp);

}

**FREE CIRCULAR LIST**

void freelist(ptr \*hptr)

{

ptr p1=\*hptr;/\*save the head of list\*/

ptr p2;

p1=p1->next;/\*free from second item untill the last\*/

while(p1!=\*hptr)

{

p2 = p1->next;

free(p1);

p1=p2;

}

free(p1);/\*free the first item\*/

}

**DELETE FROM CIRCULAR LINKED LIST**

void deleteNode(struct node\* head, int key)

{

struct node \*curr = head, \*prev;

if (head == NULL)

return;

/\* Find the required node\*/

while (curr->data != key) {

if (curr->next == head) {

printf("\nGiven node is not found"

" in the list!!!");

break;

}

prev = curr;

curr = curr->next;

}

/\* Check if node is only node \*/

if (curr->next == head) {

head = NULL;

free(curr);

return;

}

/\* If more than one node, check if \*/

/\* it is first node \*/

if (curr == head) {

prev = head;

while (prev->next != head)

prev = prev->next;

head = curr->next;

prev->next = head;

free(curr);

}

/\* check if node is last node\*/

else if (curr->next == head) {

prev->next = head;

free(curr);

}

else {

prev->next = curr->next;

free(curr);

}

}

**RETURN NODE FROM CIRCULAR LINKED LIST**

struct node\* searchNode(struct node\* head, int key)

{

if (head == NULL)

return;

/\* Find the required node\*/

struct node \*curr = head,\*found;

while (curr->data != key) {

if (curr->next == head) {

printf("\nGiven node is not found"

" in the list!!!");

break;

}

curr = curr->next;

}

return curr;

}

**הכנסה מקובץ לרשימה:**

FILE \*file;

char line[200];

ptr head=NULL;

int STnum;

char \*STname;

file = fopen("klt1","r");

if(!file)

exit(0);

/\*---read from file and build the list---\*/

fgets(line,200,file);

printf("line: %s \n",line);

while(!feof(file))

{

STnum = atoi(strtok(line, " "));

STname = strtok(NULL, " ");

add2list (&head,STnum,STname);

fgets(line,200,file);

printf("line: %s \n",line);

}/\*---end---\*/