***// ORDERED (SINGLY) LINKED LIST***

#include<stdio.h>

#include<stdlib.h>

*// Node of the linked list.*

typedef struct Node

{

int data;

struct Node \*link;

} Node;

*// Handle of the list. Its head points to the first node in the list.*

typedef struct List

{

Node \*head;

int number\_of\_nodes;

} List;

*// Initializes a linked list.*

List\* list\_initialize();

*// Creates a node and stores the data.*

Node\* create\_node(int);

*// Deletes the node at the given position.*

*// Does not delete anything if position is out of range.*

void del\_at\_pos(int, List\*);

*// Displays the contents of the linked list in space separated manner.*

void display(List\*);

*// Inserts an element into an ordered linked list.*

void insert\_in\_order(int, List\*);

*// Returns the index of the key in the linked list.*

*// Returns -1 if key not found.*

int find\_key(int, List\*);

*// Destroy the list by freeing all the nodes.*

void destroy\_list(List\*);

int main()

{

int choice,element,position,index;

*// Initialize an empty list.*

List\* sll = list\_initialize();

do

{

scanf("%d", &choice);

switch(choice)

{

*// Delete at given position*

case 1:

{

scanf("%d", &position);

del\_at\_pos(position, sll);

}

break;

*// Insert element into the ordered list.*

case 2:

{

scanf("%d", &element);

insert\_in\_order(element, sll);

}

break;

*// Display the list.*

case 3:

{

display(sll);

}

break;

*// Find the given key.*

case 4:

{

scanf("%d", &element);

index = find\_key(element, sll);

printf("%d\n",index);

}

break;

default:

break;

}

}

while(choice != 0);

destroy\_list(sll);

return 0;

}

List\* list\_initialize()

{

List\* list = (List\*)malloc(sizeof(List));

list->head=NULL;

list->number\_of\_nodes=0;

return list;

}

Node\* create\_node(int data)

{

Node \*node = (Node\*)malloc(sizeof(Node));

node->data=data;

node->link=NULL;

return node;

}

void display(List\* sll)

{

Node \*p = sll->head;

if(p==NULL)

printf("\n");

else

{

while(p!=NULL)

{

printf("%d ",p->data);

p=p->link;

}

}

printf("\n");

}

void del\_at\_pos(int position, List\* sll)

{

Node\* curr=sll->head;

int n=(sll->number\_of\_nodes);

int i;

if(position<=0||position>n)

return;

else

{

Node\* p=sll->head;

Node\* q;

if(position==1)

{ sll->head=p->link;

free(p);

}

else

{

for(i=1;i<position;++i)

{

q=p;

p=p->link;

}

q->link=p->link;

free(p);

}

(sll->number\_of\_nodes)-=1;

}

}

void insert\_in\_order(int element, List\* sll) *// DESCENDING ORDER*

{

Node\* temp=create\_node(element);

Node\* p=sll->head;

(sll->number\_of\_nodes)+=1;

if((p==NULL)||(p->data<temp->data))

{

temp->link=sll->head;

sll->head=temp;

}

else

{

Node\* q = NULL;

while(p!=NULL&&(temp->data<=p->data))

{

q=p;

p=p->link;

}

temp->link=p;

q->link=temp;

}

}

int find\_key(int key, List\* sll)

{

int index=0;

Node\* q=sll->head;

while(q!=NULL)

{

if(q->data==key)

return index;

++index;

q=q->link;

}

if(q==NULL)

return -1;

}

void destroy\_list(List\* sll)

{

Node \*q=sll->head;

Node \*r;

while(q!=NULL)

{

r=q;

q=q->link;

free(r);

}

*//free(sll);*

}