***// SINGLY LINKED LIST***

#include<stdio.h>

#include<stdlib.h>

typedef struct Node

{

int data;

struct Node \*link;

} Node;

typedef struct List

{

Node\* head;

int number\_of\_nodes;

} List;

*/\* initializes a linked list \*/*

List\* list\_initialize();

*/\* Inserts a new node with data at the beginning \*/*

void list\_insert\_beginning(List\* list, int data);

*/\* Removes the last node of the linked list \*/*

void list\_delete\_rear(List\* list);

*/\* Prints the contents of a list \*/*

void list\_print(List\* list);

*/\* Deallocates resources held by the list \*/*

void list\_destroy(List\* list);

int main() {

int choice;

List \*list = list\_initialize();

do {

scanf("%d", &choice);

switch(choice) {

int element;

case 1:

/\* Insert an element at the beginning of the list \*/

scanf("%d", &element);

list\_insert\_beginning(list, element);

break;

case 2:

/\* Remove rear elements \*/

list\_delete\_rear(list);

break;

case 3:

/\* Print list contents \*/

list\_print(list);

break;

default:

list\_destroy(list);

break;

}

} while(choice >= 1 && choice < 4);

return 0;

}

List\* list\_initialize()

{

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

list->head=NULL;

list->number\_of\_nodes=0;

return list;

}

void list\_insert\_beginning(List\* list, int data)

{

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

temp->link = list->head;

temp->data = data;

list->head = temp;

}

void list\_delete\_rear(List\* list)

{

if(list->head == NULL)

return;

else

{

Node \*temp = list->head,\*prev;

if(temp->link==NULL)

{

list->head=NULL;

}

else

{while(temp->link!=NULL)

{

prev=temp;

temp=temp->link;

}

prev->link=NULL;

}

free(temp);

}

}

void list\_print(List\* list)

{

if(list->head == NULL)

printf("EMPTY");

else

{

Node \*temp = list->head;

while(temp!=NULL)

{

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

temp = temp->link;

}

}

printf("\n");

}

void list\_destroy(List\* list)

{

if(list->head==NULL);

else

{

Node \*temp = list->head, \*ptr;

while(temp!=NULL)

{

ptr=temp;

temp=temp->link;

free(ptr);

}

}

}

#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();

*/\* Inserts data at specified position 0 <= position <= length(list). \*/*

void list\_insert\_at(List\* list, int data, int position);

*/\*Inserts data at the end of the List\*/*

void insert\_at\_end(List \*list, int data);

*/\* Removes the First node of the linked list. No operation if the list was empty \*/*

void list\_delete\_front(List\* list);

*/\* Prints the entire List as "X1, X2, ..., Xn" where X1 to Xn are the elements of the list,*

*if list is empty prints "EMPTY" \*/*

void list\_print(List\* list);

*/\* Returns the position of the key if present else returns -1\*/*

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

*/\* deallocates resources held by the list \*/*

void list\_destroy(List\* list);

int main()

{

int choice;

List\* list = list\_initialize();

do {

scanf("%d", &choice);

switch(choice) {

int element, index;

case 1:

/\*Insert element at the End of the list\*/

scanf("%d", &element);

insert\_at\_end(list, element);

break;

case 2:

/\* Print list contents \*/

list\_print(list);

break;

case 3:

/\* Remove front element \*/

list\_delete\_front(list);

break;

case 4:

/\* Insert elements at specified positions \*/

scanf("%d%d", &element, &index);

list\_insert\_at(list, element, index);

break;

case 5:

/\*Find position of an element\*/

scanf("%d", &element);

index = find\_key(list, element);

if (index != -1) printf("%d\n", index);

break;

default:

break;

}

} while(choice != 0);

list\_destroy(list);

return 0;

}

List\* list\_initialize()

{

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

list->head = NULL;

list->number\_of\_nodes = 0;

return list;

}

void insert\_at\_end(List \*list, int data)

{

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

n->data=data;

n->link=NULL;

Node \*temp=list->head;

if(list->head==NULL){

list->head=n;

}

else{

while(temp->link!=NULL){

temp=temp->link;

}

temp->link=n;

}

list->number\_of\_nodes++;

}

void list\_print(List\* list)

{

Node \*temp=list->head;

if(list->head==NULL){

printf("EMPTY\n");

return;

}

while(temp!=NULL){

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

temp=temp->link;

}

printf("\n");

}

void list\_delete\_front(List\* list)

{

if(list->head==NULL){

Return;

}

Node \*temp=list->head;

list->head=list->head->link;

free(temp);

list->number\_of\_nodes--;

}

void list\_insert\_at (List \*list, int data, int position)

{

if(position<0 || position>list->number\_of\_nodes){

return;

}

Node \*p=list->head;

Node \*q=NULL;

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

temp->data=data;

int i=0;

while(p!=NULL && i!=position){

q=p;

p=p->link;

i+=1;

}

if(q==NULL){

temp->link=list->head;

list->head=temp;

}

else{

temp->link=p;

q->link=temp;

}

list->number\_of\_nodes++;

}

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

{

Node \*temp=list->head;

int i=0;

while(temp!=NULL){

if(key==temp->data){

return i;

}

i+=1;

temp=temp->link;

}

return -1;

}

void list\_destroy (List \*list)

{

Node \*temp=list->head;

Node \*q;

while(temp!=NULL){

q=temp;

temp=temp->link;

free(q);

}

}