***// DOUBLY LINKED LIST***

#include <stdio.h>

#include <stdlib.h>

*// Node of the linked list*

typedef struct Node {

int data;

struct Node \*prev;

struct Node \*next;

} 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\* initialize\_list();

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

Node\* create\_node(int data);

*// Inserts data at the beginning of the List*

void insert\_front(List\* dll, int data);

*// Deletes the node at position pos. No operation if pos is out of range.*

void position\_delete(List\* dll, int pos);

*// Return index of key in the list(0-based). Return -1 if not found*

int search\_list(List\* dll, int key);

*// Prints the entire list. Prints "EMPTY" if the list is empty.*

void display(List\* dll);

*// Deallocates resources held by the list*

void destroy\_list(List\* dll);

int main()

{

List\* dll = initialize\_list();

int ele, choice, pos;

do {

scanf("%d", &choice);

switch(choice) {

// Insert at beginning

case 1:

scanf("%d", &ele);

insert\_front(dll, ele);

break;

// Delete at position

case 2:

scanf("%d", &pos);

position\_delete(dll, pos);

break;

// Search for element

case 3:

scanf("%d", &ele);

pos = search\_list(dll, ele);

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

break;

// Print entire list

case 4:

display(dll);

break;

}

}

while (choice != 0);

destroy\_list(dll);

return 0;

}

List\* initialize\_list()

{

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->prev=NULL;

node->next=NULL;

return node;

}

void insert\_front(List\* dll, int data)

{

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

dll->number\_of\_nodes+=1;

if(dll->head==NULL)

{

dll->head=temp;

return;

}

temp->next=dll->head;

dll->head->prev=temp;

dll->head=temp;

}

void position\_delete(List\* dll, int pos)

{

int n=(dll->number\_of\_nodes-1),i;

if(pos<0||pos>n)

return;

dll->number\_of\_nodes-=1;

Node\* curr=dll->head;

if(pos==0)

dll->head=curr->next;

else if(pos<n)

{

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

curr=curr->next;

curr->prev->next=curr->next;

curr->next->prev=curr->prev;

}

else

{

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

curr=curr->next;

curr->prev->next=NULL;

}

free(curr);

}

int search\_list(List\* dll, int key)

{

int index=0;

Node\* q=dll->head;

while(q!=NULL)

{

if(q->data==key)

return index;

++index;

q=q->next;

}

if(q==NULL)

return -1;

}

void display(List\* dll)

{

Node\* curr=dll->head;

if(curr==NULL)

printf("EMPTY");

else

{

while(curr!=NULL)

{

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

curr=curr->next;

}

}

printf("\n");

}

void destroy\_list(List\* dll)

{

Node\* q=dll->head;

Node\* r;

while(q!=NULL)

{

r=q;

q=q->next;

free(r);

}

free(dll);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_