#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*next;

struct node\*prev;

};

void display();

struct node\*head=NULL;

int isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

void insertionAtBeginning()

{

int d;

struct node\*newnode=(struct node\*)malloc(sizeof(struct node));

printf("Enter the data to be inserted:");

scanf("%d",&d);

newnode->data=d;

newnode->prev=NULL;

newnode->next=head;

head=newnode;

}

void insertionAtEnd()

{

if (isempty())

{

insertionAtBeginning();

}

else

{

int data;

struct node\*newnode=(struct node\*)malloc(sizeof(struct node));

printf("Enter the data to be inserted:");

scanf("%d",&data);

newnode->data=data;

struct node\*temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp->next=newnode;

newnode->prev=temp;

newnode->next=NULL;

}

}

void insertionAtSpecificPosition()

{

if (isempty())

insertionAtBeginning();

else

{

int data,p;

struct node\*newnode=(struct node\*)malloc(sizeof(struct node));

printf("Enter the position where the element to be inserted:");

scanf("%d",&p);

if(p<1)

printf("Invalid Position\n");

else if(p==1)

insertionAtBeginning();

else

{

struct node\*temp=head;

for(int i=1;i<p-1 && temp!=NULL;i++)

{

temp=temp->next;

}

if(temp==NULL)

{

printf("Position out of bound.Node not inserted\n");

free(newnode);

}

else

{

printf("Enter the element to be inserted:");

scanf("%d",&data);

newnode->data=data;

struct node\*t=temp->next;

newnode->next=t;

newnode->prev=temp;

t->prev=newnode;

temp->next=newnode;

printf("Successfully inserted\n");

}

}

}

}

void deletionFromBeginning()

{

if(isempty())

printf("List is Empty!!\n");

else

{

struct node\*temp=head;

head=temp->next;

free(temp);

printf("Successfully deleted the element from beginning\n");

display();

}

}

void deletionFromEnd()

{

if(isempty())

printf("List is Empty!!\n");

else

{

struct node\*temp=head;

if(temp->next==NULL)

deletionFromBeginning();

else

{

struct node\*t=NULL;

while(temp->next!=NULL)

{

t=temp;

temp=temp->next;

}

t->next=NULL;

free(temp);

printf("Successfully deleted the element from the end\n");

}

display();

}

}

void deletionFromSpecificPosition()

{

if(isempty())

printf("List is Empty!!\n");

else

{

int p;

printf("Enter the position from where the element to be deleted:");

scanf("%d",&p);

if(p<1)

printf("Invalid position\n");

else if(p==1)

deletionFromBeginning();

else

{

struct node\*temp=head;

struct node\*ptr=NULL;

for(int i=1;i<p && temp!=NULL;i++)

{

ptr=temp;

temp=temp->next;

}

if(temp==NULL)

printf("Position out of bounds.Not not deleted.\n");

else

{

ptr->next=temp->next;

struct node\*n=temp->next;

n->prev=ptr;

free(temp);

printf("Successfully deleted the element from the specified position\n");

}

}

display();

}

}

void display()

{

if(isempty())

printf("List is Empty!!\n");

else

{

printf("Elements are:\n");

struct node\*temp=head;

while(temp->next!=NULL)

{

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

temp=temp->next;

}

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

}

}

void search()

{

if(isempty())

printf("List is Empty!!\n");

else

{

int s;

printf("Enter the element to be searched:");

scanf("%d",&s);

int found=0;

struct node\*temp=head;

while(temp->next!=NULL)

{

if(temp->data==s)

found=1;

temp=temp->next;

}

if (found==1)

printf("Element Found\n");

else

printf("Element not found\n");

}

}

int main()

{

int c;

printf("OPERATIONS ON DOUBLY LINKED LIST\n");

while(1)

{

printf("\n1.Insertion At Beginnning");

printf("\n2.Insertion At End");

printf("\n3.Insertion At Specific Position");

printf("\n4.Deletion From Beginning");

printf("\n5.Deletion From End");

printf("\n6.Deletion from a Specific Position");

printf("\n7.Display");

printf("\n8.Search");

printf("\n9.Exit");

printf("\nEnter your choice (1-9) : ");

scanf("%d",&c);

switch(c)

{

case 1:

insertionAtBeginning();

printf("Successfully inserted the element to the beginning\n");

display();

break;

case 2:

insertionAtEnd();

printf("Successfully inserted the element to the end\n");

display();

break;

case 3:

insertionAtSpecificPosition();

display();

break;

case 4:

deletionFromBeginning();

break;

case 5:

deletionFromEnd();

break;

case 6:

deletionFromSpecificPosition();

break;

case 7:

display();

break;

case 8:

search();

break;

case 9:

exit(0);

break;

default:

printf("Invalid Choice\n");

break;

}

}

}