/\*

DOUBLY-LINKED-LIST...

Assumptions...

1. None of the functions except create() will work if the List is Empty.

2. Many of the functions will work assuming the fact that the elements inside

the Linked List are unique....

\*/

#include <stdio.h>

#include <stdlib.h>

struct node//It stores contents of each node

{

int data;

struct node \*prev;

struct node \*next;

};

struct sllptrs//It stores the reference to 1st and last node

{

struct node \*head;

struct node \*tail;

};

struct node \*createnode(int v)//It creates a new Node.

{

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

newnode->data=v;

newnode->prev=NULL;

newnode->next=NULL;

return newnode;

}

struct sllptrs create(struct sllptrs ptr)//It creates the Linked List.

{

if(ptr.head!=NULL)

{

printf("List is already created.\n");

return ptr;

}

int ele,flag=1;//flag=1 indicates that the current node is the 1st node.

struct node \*newnode;

printf("Enter the elements one by one and -9999 to stop entering....\n");

while(1)

{

scanf("%d",&ele);

if(ele==-9999)

break;

newnode=createnode(ele);

if(flag)

{

ptr.head=ptr.tail=newnode;

flag=0;

}

else

{

newnode->prev=ptr.tail;

ptr.tail->next=newnode;

ptr.tail=ptr.tail->next;

}

}

printf("Linked List has been successfully created.\n");

return ptr;

}

//Displaying the contents of the linked list.

void display(struct sllptrs ptr)

{

if(ptr.head==NULL)

printf("The Linked List is empty.\n");

else

{

struct node \*it=ptr.head;

while(it!=NULL)

{

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

it=it->next;

}

printf("X\n");

}

}

//Insertion operations.

//Inserting at the beginning.

struct sllptrs addFirst(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

int ele;

printf("Enter the no. to be inserted : ");

scanf("%d",&ele);

struct node \*newnode=createnode(ele);

ptr.head->prev=newnode;

newnode->next=ptr.head;

ptr.head=newnode;

printf("Element inserted.\n");

return ptr;

}

//Inserting at the end.

struct sllptrs addLast(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

int ele;

printf("Enter the no. to be inserted : ");

scanf("%d",&ele);

struct node \*newnode=createnode(ele);

newnode->prev=ptr.tail;

ptr.tail->next=newnode;

ptr.tail=ptr.tail->next;

printf("Element inserted.\n");

return ptr;

}

//Inserting after a value.

struct sllptrs addAfter(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked List is empty.\n");

return ptr;

}

struct node \*newnode;

struct node \*it;

int ele,v,flag=0,c=0;//flag=0 indicates insertion is not done.

printf("Enter the no. after which you want to insert a new node : ");

scanf("%d",&ele);

it=ptr.head;

while(it!=NULL)

{

c++;

if(it->data==ele)

{

if(it==ptr.tail)

return addLast(ptr);

printf("Enter the no. to be inserted : ");

scanf("%d",&v);

newnode=createnode(v);

newnode->prev=it;

newnode->next=it->next;

it->next->prev=newnode;

it->next=newnode;

flag=1;

break;

}

it=it->next;

}

if(flag)

printf("Element Inserted.\n");

else

printf("Insertion failed.\n");

return ptr;

}

//Inserting before a value.

struct sllptrs addBefore(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked List is empty.\n");

return ptr;

}

struct node \*newnode;

struct node \*it;

int ele,v,flag=0,c=0;//flag=0 indicates insertion is not done.

printf("Enter the no. before which you want to insert a new node : ");

scanf("%d",&ele);

if(ptr.head->data==ele)

return addFirst(ptr);

it=ptr.head;

while(it!=NULL)

{

c++;

if(it->next!=NULL&&it->next->data==ele)

{

printf("Enter the no. to be inserted : ");

scanf("%d",&v);

newnode=createnode(v);

newnode->prev=it;

newnode->next=it->next;

it->next->prev=newnode;

it->next=newnode;

flag=1;

break;

}

it=it->next;

}

if(flag)

printf("Element Inserted.\n");

else

printf("Insertion failed.\n");

return ptr;

}

//Inserting at a particular position.

struct sllptrs add\_pos(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

int pos,ele;

printf("Enter the position in which the element is to be inserted : ");

scanf("%d",&pos);

if(pos==1)

return addFirst(ptr);

if(pos<1)

{

printf("Invalid Input.\n");

return ptr;

}

struct node \*newnode;

int c=0;

struct node \*it=ptr.head;

while(it!=NULL)

{

c++;

if(pos==c+1&&it->next!=NULL)

{

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

scanf("%d",&ele);

newnode=createnode(ele);

newnode->prev=it;

newnode->next=it->next;

it->next->prev=newnode;

it->next=newnode;

printf("Element inserted.\n");

return ptr;

}

it=it->next;

}

if(pos==c+1)

return addLast(ptr);

printf("Invalid Input.\n");

return ptr;

}

//Deletion Operations.

//Deletes the First node.

struct sllptrs deleteFirst(struct sllptrs ptr)

{

struct node \*temp;

if(ptr.head==NULL)

printf("Deletion unsuccessful. List is empty.\n");

else

{

temp=ptr.head;

if(ptr.head->next==NULL)

ptr.head=ptr.tail=NULL;

else

{

ptr.head=ptr.head->next;

ptr.head->prev=NULL;

}

free(temp);

printf("Deletion successful.\n");

}

return ptr;

}

//Deletes the Last node.

struct sllptrs deleteLast(struct sllptrs ptr)

{

struct node \*temp;

struct node \*it;

if(ptr.head==NULL)

printf("Deletion unsuccessful. List is empty.\n");

else

{

struct node \*temp=ptr.tail;

if(ptr.head->next==NULL)

ptr.head=ptr.tail=NULL;

else

{

ptr.tail=ptr.tail->prev;

ptr.tail->next=NULL;

}

free(temp);

printf("Deletion successful.\n");

}

return ptr;

}

//Deletes a node with a value.

struct sllptrs delete\_ele(struct sllptrs ptr)

{

int ele,flag=1;//flag=1 indicates that deletion has not been done.

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

printf("Enter the no. : ");

scanf("%d",&ele);

if(ptr.head->data==ele)

return deleteFirst(ptr);

struct node \*it=ptr.head->next;

while(it!=NULL)

{

if(it->data==ele)

{

it->prev->next=it->next;

if(it==ptr.tail)

ptr.tail=it;

else

it->next->prev=it->prev;//Can access this only when ele is not at tail.

free(it);

flag=0;

break;

}

it=it->next;

}

if(flag)

printf("Deletion unsuccessful.\n");

else

printf("Deletion successful.\n");

return ptr;

}

//Deletes a node after a certain value.

struct sllptrs deleteAfter(struct sllptrs ptr)

{

int ele,flag=1;//flag=1 indicates that deletion has not been done.

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

if(ptr.head->next==NULL)

{

printf("Deletion unsuccessful.\n");

return ptr;

}

printf("Enter the no. : ");

scanf("%d",&ele);

struct node \*it=ptr.head->next;

while(it!=NULL)

{

if(it->prev->data==ele)

{

it->prev->next=it->next;

if(it==ptr.tail)

ptr.tail=it;

else

it->next->prev=it->prev;//Can access this only when ele is not at tail.

free(it);

flag=0;

break;

}

it=it->next;

}

if(flag)

printf("Deletion could not be done.\n");

else

printf("Deletion successful.\n");

return ptr;

}

//Deletes a node before a certain value.

struct sllptrs deleteBefore(struct sllptrs ptr)

{

int ele,flag=1;//flag=1 indicates that deletion has not been done.

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

if(ptr.head->next==NULL)

{

printf("Deletion unsuccessful1.\n");

return ptr;

}

printf("Enter the no. : ");

scanf("%d",&ele);

if(ptr.head->next->data==ele)

return deleteFirst(ptr);

struct node \*it=ptr.head->next;

while(it->next!=NULL)

{

if(it->next->data==ele)

{

it->prev->next=it->next;

it->next->prev=it->prev;

free(it);

flag=0;

break;

}

it=it->next;

}

if(flag)

printf("Deletion unsuccessful.\n");

else

printf("Deletion successful.\n");

return ptr;

}

//Deletes a node at a certain position.

struct sllptrs delete\_pos(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

int pos,flag=1;

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

scanf("%d",&pos);

if(pos==1)

return deleteFirst(ptr);

if(pos<1)

{

printf("Invalid Input.\n");

return ptr;

}

int c=1;//Skipping 1st node.

struct node \*it=ptr.head->next;

while(it!=NULL)

{

c++;

if(pos==c)

{

if(it==ptr.tail)

return deleteLast(ptr);

it->prev->next=it->next;

it->next->prev=it->prev;

free(it);

printf("Deletion successful.\n");

return ptr;

}

it=it->next;

}

printf("Invalid Input.\n");

return ptr;

}

//Deletes the entire linked list.

struct sllptrs deleteList(struct sllptrs ptr)

{

struct node \*temp;

if(ptr.head==NULL)

printf("List is already empty.\n");

else

{

while(ptr.head!=NULL)

{

temp=ptr.head;

ptr.head=ptr.head->next;

free(temp);

}

ptr.tail=NULL;

printf("List is deleted.\n");

}

return ptr;

}

//Sorting the linked list.

//Dividing a linked list into two smaller linked lists.

void divide(struct sllptrs ptr,struct sllptrs \*ptr1,struct sllptrs \*ptr2)

{

struct node \*tmp1;

struct node \*tmp2;

tmp1=ptr.head;

tmp2=ptr.head->next;

while(tmp2!=NULL)

{

tmp2=tmp2->next;

if(tmp2!=NULL)

{

tmp1=tmp1->next;

tmp2=tmp2->next;

}

}

ptr1->head=ptr.head;

ptr1->tail=tmp1;

ptr2->head=tmp1->next;

ptr2->tail=ptr.tail;

ptr1->tail->next=NULL;//Breaking link between the two divided lists.

}

//Merging two sorted lists into a new sorted list.

struct node \*merge(struct node \*ptr1,struct node \*ptr2)

{

struct node \*head=NULL;

if(ptr1==NULL)

return ptr2;

if(ptr2==NULL)

return ptr1;

if(ptr1->data<=ptr2->data)

{

head=ptr1;

head->next=merge(ptr1->next,ptr2);

}

else

{

head=ptr2;

head->next=merge(ptr1,ptr2->next);

}

return head;

}

//Dividing list into smaller lists and then sorting...O(nlogn)

struct sllptrs divide\_and\_sort(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

printf("Linked list is empty.\n");

return ptr;

}

if(ptr.head->next==NULL)

return ptr;

struct sllptrs ptr1,ptr2;

divide(ptr,&ptr1,&ptr2);

ptr1=divide\_and\_sort(ptr1);

ptr2=divide\_and\_sort(ptr2);

ptr.head=merge(ptr1.head,ptr2.head);

ptr.tail=ptr1.tail->data<=ptr2.tail->data ? ptr2.tail : ptr1.tail;

return ptr;

}

//Counting the no. of nodes.

int count(struct sllptrs ptr)

{

int c=0;

struct node \*it=ptr.head;

while(it!=NULL)

{

c++;

it=it->next;

}

return c;

}

//Searching...

//Element at a given position.

void eleAtPos(struct sllptrs ptr)

{

if(ptr.head==NULL)

printf("Linked list is empty.\n");

else

{

int pos;

printf("Enter the position : ");

scanf("%d",&pos);

int c=0;

struct node \*it=ptr.head;

while(it!=NULL)

{

c++;

if(c==pos)

break;

it=it->next;

}

if(c==pos)

printf("The element at Node %d is %d.\n",pos,it->data);

else

printf("Invalid position.\n");

}

}

//Index of a given element

void indexOfEle(struct sllptrs ptr)

{

if(ptr.head==NULL)

printf("Linked list is empty.\n");

else

{

int ele,pos=0,flag=0;//flag stores the Node no. at which element is present.

printf("Enter the element to get its position in the linked list : ");

scanf("%d",&ele);

struct node \*it=ptr.head;

while(it!=NULL)

{

pos++;

if(it->data==ele)

{

flag=pos;

break;

}

it=it->next;

}

if(flag)

printf("The element %d is present at Node %d.\n",ele,flag);

else

printf("Element is not found.\n");

}

}

//Finding the root nth node of the linked list.

//n being the no. of nodes in the linked list which is unknown.

void root\_nth\_node(struct sllptrs ptr)

{

if(ptr.head==NULL)

printf("Linked list is empty.\n");

else

{

int c=1,r=1;

struct node \*it;

struct node \*res;

it=res=ptr.head;

while(it!=NULL)

{

if((r+1)\*(r+1)<=c)

{

r++;

res=res->next;

}

it=it->next;

c++;

}

printf("Required node is Node %d with value %d.\n",r,res->data);

}

}

//MENU

void display\_menu()

{

printf("-------------------------------------------------------------------\n");

printf("MENU :-\n");

printf("1. Create the linked list.\n");

printf("2. Display the contents of the list.\n");

printf("3. Add a node to the beginning of the list.\n");

printf("4. Add a node to the end of the list.\n");

printf("5. Add a node after a given element in the list.\n");

printf("6. Add a node before a given element in the list.\n");

printf("7. Add a node to a given position in the list.\n");

printf("8. Delete a node from the beginning of the linked list.\n");

printf("9. Delete a node from the end of the linked list.\n");

printf("10. Delete a node with a given value from the list.\n");

printf("11. Delete a node after an element from the linked list.\n");

printf("12. Delete a node before an element from the linked list.\n");

printf("13. Delete a node from a position from the linked list.\n");

printf("14. Delete the entire linked list.\n");

printf("15. Sorting the list.\n");

printf("16. No. of elements in the list.\n");

printf("17. Display the element at a given position in the list.\n");

printf("18. Display the position of a given element in the list.\n");

printf("19. To find the root nth node of the linked list in one scan.\n");

printf("20. Exit.\n");

printf("-------------------------------------------------------------------\n");

printf("Enter your choice : ");

}

void main()

{

int ch,search;//con indicates continuing condition.

struct sllptrs ptr;

ptr.head=ptr.tail=NULL;

do

{

display\_menu();

scanf("%d",&ch);

switch(ch)

{

case 1 :

ptr=create(ptr);

break;

case 2 :

display(ptr);

break;

case 3 :

ptr=addFirst(ptr);

break;

case 4 :

ptr=addLast(ptr);

break;

case 5 :

ptr=addAfter(ptr);

break;

case 6 :

ptr=addBefore(ptr);

break;

case 7 :

ptr=add\_pos(ptr);

break;

case 8 :

ptr=deleteFirst(ptr);

break;

case 9 :

ptr=deleteLast(ptr);

break;

case 10 :

ptr=delete\_ele(ptr);

break;

case 11 :

ptr=deleteAfter(ptr);

break;

case 12 :

ptr=deleteBefore(ptr);

break;

case 13 :

ptr=delete\_pos(ptr);

break;

case 14:

ptr=deleteList(ptr);

break;

case 15 :

if(ptr.head==NULL)

printf("The list is empty.\n");

else

{

ptr=divide\_and\_sort(ptr);

printf("The List has been sorted.\n");

}

break;

case 16 :

printf("There are %d element(s) in the linked list.\n",count(ptr));

break;

case 17 :

eleAtPos(ptr);

break;

case 18 :

indexOfEle(ptr);

break;

case 19 :

root\_nth\_node(ptr);

break;

case 20 :

exit(0);

default :

printf("Wrong Choice!\n");

}

}while(1);

}

//end of main().