/\*

SINGLY-LINKED-LIST...

Assumptions...

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

2. Loop inside a linked list begins from the tail of the list.

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

the Linked List are unique....

\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct node//It stores contents of each node

{

int data;

struct node \*link;

};

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->link=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

{

ptr.tail->link=newnode;

ptr.tail=ptr.tail->link;

}

}

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("List is empty.\n");

else

{

struct node \*it=ptr.head;

while(it!=NULL)

{

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

it=it->link;

}

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);

if(ptr.head==NULL)

ptr.head=ptr.tail=newnode;

else

{

newnode->link=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);

if(ptr.head==NULL)

ptr.head=ptr.tail=newnode;

else

{

ptr.tail->link=newnode;

ptr.tail=ptr.tail->link;

}

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->link=it->link;

it->link=newnode;

flag=1;

break;

}

it=it->link;

}

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->link!=NULL&&it->link->data==ele)

{

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

scanf("%d",&v);

newnode=createnode(v);

newnode->link=it->link;

it->link=newnode;

flag=1;

break;

}

it=it->link;

}

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->link!=NULL)

{

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

scanf("%d",&ele);

newnode=createnode(ele);

newnode->link=it->link;

it->link=newnode;

printf("Element inserted.\n");

return ptr;

}

it=it->link;

}

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;

ptr.head=ptr.head->link;

free(temp);

if(ptr.head==NULL)

ptr.tail=NULL;

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

{

it=ptr.head;

if(it->link==NULL)

{

free(ptr.tail);

ptr.head=ptr.tail=NULL;

}

else

{

while(it!=NULL)

{

if(it->link==ptr.tail)

{

temp=ptr.tail;

ptr.tail=it;

it->link=NULL;

free(temp);

}

it=it->link;

}

}

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,\*temp;

while(it->link!=NULL)

{

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

{

temp=it->link;

it->link=temp->link;

if(temp==ptr.tail)

ptr.tail=it;

free(temp);

flag=0;

break;

}

it=it->link;

}

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->link==NULL)

{

printf("Deletion unsuccessful.\n");

return ptr;

}

printf("Enter the no. : ");

scanf("%d",&ele);

struct node \*it=ptr.head,\*temp;

while(it->link!=NULL)

{

if(it->data==ele)

{

temp=it->link;

it->link=temp->link;

if(temp==ptr.tail)

ptr.tail=it;

free(temp);

flag=0;

break;

}

it=it->link;

}

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->link==NULL)

{

printf("Deletion unsuccessful1.\n");

return ptr;

}

printf("Enter the no. : ");

scanf("%d",&ele);

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

return deleteFirst(ptr);

struct node \*it=ptr.head;

struct node \*temp=it->link;

while(temp->link!=NULL)

{

if(temp->link->data==ele)

{

it->link=temp->link;

free(temp);

flag=0;

break;

}

it=temp;

temp=temp->link;

}

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;

}

struct node \*temp;

int c=0;

struct node \*it=ptr.head;

while(it->link!=NULL)

{

c++;

if(pos==c+1)

{

temp=it->link;

it->link=temp->link;

if(temp==ptr.tail)

ptr.tail=it;

free(temp);

printf("Deletion successful.\n");

return ptr;

}

it=it->link;

}

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->link;

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->link;

while(tmp2!=NULL)

{

tmp2=tmp2->link;

if(tmp2!=NULL)

{

tmp1=tmp1->link;

tmp2=tmp2->link;

}

}

ptr1->head=ptr.head;

ptr1->tail=tmp1;

ptr2->head=tmp1->link;

ptr2->tail=ptr.tail;

ptr1->tail->link=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->link=merge(ptr1->link,ptr2);

}

else

{

head=ptr2;

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

}

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->link==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;

}

//Reversing a Linked List.

struct sllptrs reverse(struct sllptrs ptr)//reversing the links.

{

struct node \*cur,\*prev,\*temp;

if(ptr.head==NULL)

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

else

{

ptr.tail=ptr.head;

cur=ptr.head;

prev=temp=NULL;

while(cur!=NULL)

{

temp=cur->link;

cur->link=prev;

prev=cur;

cur=temp;

}

ptr.head=prev;

printf("The linked list has been reversed.\n");

}

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->link;

}

return c;

}

//Searching...

//Element at a given position.

void eleAtPos(struct sllptrs ptr)

{

if(ptr.head==NULL)

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

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->link;

}

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");

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->link;

}

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<(int)sqrt(c))

{

r++;

res=res->link;

}

it=it->link;

c++;

}

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

}

}

//Loops inside the linked list.

//Forcefully creating a loop inside the Linked List.

struct sllptrs create\_loop(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

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

return ptr;

}

int pos,c=1,flag=0;//flag=0 indicattes no loop.

printf("Enter the position of the Node to which you want to join the link of the last node : ");

scanf("%d",&pos);

if(pos<c)

{

printf("Invalid Input. Loop is not created.\n");

return ptr;

}

struct node \*it=ptr.head;

while(it!=NULL)

{

if(c==pos)

{

ptr.tail->link=it;//Link of last node is connected to desired node.

ptr.tail=NULL;//Now tail has no significance, so it is made NULL.

flag=1;

break;

}

it=it->link;

c++;

}

if(flag)

printf("Loop is successfully created.\n");

else

printf("Invalid Input. Loop is not created.\n");

return ptr;

}

//Detecting and removing a loop.

struct sllptrs detect\_and\_remove(struct sllptrs ptr)

{

if(ptr.head==NULL)

{

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

return ptr;

}

int flag=0;//flag=0 indicates no loop.

struct node \*ptr1=ptr.head;//Slow pointer.

struct node \*ptr2=ptr.head->link;//Fast pointer.

struct node \*ptr\_start;//It stores the node at which ptr1=ptr2.

struct node \*it;

while(ptr2!=NULL)

{

ptr2=ptr2->link;

if(ptr2!=NULL)//Checking NULL condition.

{

ptr1=ptr1->link;

ptr2=ptr2->link;

}

if(ptr1==ptr2)

{

ptr\_start=ptr1;

flag=1;

break;

}

}

if(flag==0)

printf("There are no loops in the Linked List.\n");

else

{

printf("Loop detected in the Linked List.\n");

printf("Removing the loop.......\n");

it=ptr.head;

while(ptr\_start->link!=it)

{

ptr\_start=ptr\_start->link;

it=it->link;

}

//Now ptr\_start becomes the last node.

ptr.tail=ptr\_start;//Updating tail as loop is removed.

ptr\_start->link=NULL;

printf("Loop Removed.\n");

}

return ptr;

}

//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. Reversing a List.\n");

printf("17. No. of elements in the list.\n");

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

printf("19. Display the position of a given element in the list.\n");

printf("20. To find the root nth node of the linked list in one scan.\n");

printf("21. Create a loop inside the Linked List.\n");

printf("22. Detect and Remove a loop from the Linked List.\n");

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

printf("Enter your choice : ");

}

void main()

{

int ch,con=1,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 :

ptr=divide\_and\_sort(ptr);

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

break;

case 16 :

ptr=reverse(ptr);

break;

case 17 :

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

break;

case 18 :

eleAtPos(ptr);

break;

case 19 :

indexOfEle(ptr);

break;

case 20 :

root\_nth\_node(ptr);

break;

case 21 :

ptr=create\_loop(ptr);

break;

case 22 :

ptr=detect\_and\_remove(ptr);

break;

default :

printf("Wrong Choice!\n");

}

printf("Enter 0 to exit and 1 to continue : ");

scanf("%d",&con);

}while(con);

}

//end of main().