Write a program to implement all functions of Doubly linked list

#include<stdio.h>

#include<alloc.h>

#include<stdlib.h>

#include<conio.h>

typedef struct lklist

{

int info;

struct lklist\*next;

struct lklist\*pre;

}node;

void main()

{

void createlist(node\*\*,node\*\*);

void traversefirst(node\*);

void traverselast(node\*);

void insertatbegin(node\*\*,node\*\*,int);

void insertatend(node\*\*,node\*\*,int);

void insertafter(node\*,node\*\*,int,int);

void deleteatbegin(node\*\*,node\*\*);

void deleteatend(node\*\*,node\*\*);

void deleteafterelement(node\*,node\*\*,int);

void deletebeforeelement(node\*\* ,int);

node\*head,\*tail;

int ch,data,item;

createlist(&head,&tail);

clrscr();

while(1)

{

printf("menu is\n");

printf("1:insert at begining\n");

printf("2:traversing from first element\n");

printf("3:insert at end\n");

printf("4:insert after element\n");

printf("5:delete at begining\n");

printf("6:delete at end\n");

printf("7:delete after element\n");

printf("8:delete before element\n");

printf("9:traversing from last element\n");

printf("10:exit\n");

printf("enter your choice \n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter data to be inserted\n");

scanf("%d",&data);

insertatbegin(&head,&tail,data);

printf("data is inserted at begining\n");

break;

case 2:

if(head==NULL)

{

printf("list is empty\n");

}

else

{

traversefirst(head);

}

break;

case 3:

printf("enter data to be inserted at end\n");

scanf("%d",&data);

insertatend(&head,&tail,data);

printf("data is inserted at the end of the list\n");

break;

case 4:

printf("enter data to be inserted at specifi position\n");

scanf("%d",&data);

printf("enter data after which you want to insert the node\n");

scanf("%d",&item);

insertafter(head,&tail,data,item);

printf("node is inserted after element\n");

break;

case 5:

if(head==NULL)

{

printf("list is empty\n");

}

else

{

deleteatbegin(&head,&tail);

printf("node at begining is deleted successfully\n");

}

break;

case 6:

if(head==NULL)

{

printf("list is empty\n");

}

else

{

deleteatend(&head,&tail);

printf("node at end is deleted successfully\n");

}

break;

case 7:

printf("enter the value item after which the node will be deleted\n");

scanf("%d",&data);

deleteafterelement(head,&tail,data);

printf("deletion is done successfully after the given element\n");

break;

case 8:

printf("enter data before you want to delete\n");

scanf("%d",&data);

deletebeforeelement(&head,data);

printf("deletion is done successfully before the given element\n");

break;

case 9:

if(head==NULL)

{

printf("list is empty\n");

}

else

{

traverselast(tail);

}

break;

case 10:

exit(1);

default:

printf("wrong choice\n");

}

getch();

}

}

void createlist(node\*\*head,node\*\*tail)

{

\*head=NULL;

\*tail=NULL;

}

void insertatbegin(node\*\*head,node\*\*tail,int data)

{

node\*temp;

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

temp->pre=NULL;

temp->info=data;

temp->next=\*head;

if(\*head==NULL)

{

\*tail=temp;

}

else

{

(\*head)->pre=temp;

}

\*head=temp;

}

void traverselast(node\*tail)

{

printf("data in reverse order is\n");

while(tail!=NULL)

{

printf("data is=%d\n",tail->info);

tail=tail->pre;

}

}

void traversefirst(node\*head)

{

while(head!=NULL)

{

printf("data is=%d\n",head->info);

head=head->next;

}

}

void insertatend(node\*\*head,node\*\*tail,int data)

{

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

temp->info=data;

temp->next=NULL;

if(\*head==NULL)

{

temp->pre=NULL;

\*head=temp;

}

else

{

(\*tail)->next=temp;

temp->pre=\*tail;

}

\*tail=temp;

}

void insertafter(node\*head,node\*\*tail,int data,int item)

{

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

temp->info=data;

while(head!=NULL)

{

if(head->info==item)

{

break;

}

head=head->next;

}

if(head==NULL)

{

return;

}

if(head->next==NULL)

{

\*tail=temp;

}

else

{

(head->next)->pre=temp;

}

temp->next=head->next;

temp->pre=head;

head->next=temp;

}

void deleteatbegin(node\*\*head,node\*\*tail)

{

node\*temp=\*head;

if(\*head==\*tail)

{

\*head=NULL;

\*tail=NULL;

}

else

{

\*head=(\*head)->next;

(\*head)->pre=NULL;

}

free(temp);

}

void deleteatend(node\*\*head,node\*\*tail)

{

node\*temp=\*tail;

if(\*head==\*tail)

{

\*head=NULL;

\*tail=NULL;

}

else

{

(\*tail)=(\*tail)->pre;

(\*tail)->next=NULL;

}

free(temp);

}

void deleteafterelement(node\*head,node\*\*tail,int item)

{

node\*temp=head;

while(temp!=NULL)

{

if(item==head->info)

{

break;

}

else

{

head=head->next;

}

}

if(head==NULL)

{

return;

}

if(head->next==NULL)

{

return;

}

temp=head->next;

if(temp->next==NULL)

{

head->next=NULL;

\*tail=head;

}

else

{

(temp->next)->pre=head;

head->next=temp->next;

}

free(temp);

}

void deletebeforeelement(node\*\*head,int data)

{

node\*loc=\*head;

node\*temp;

while(loc!=NULL)

{

if(loc->info==data)

{

break;

}

else

{

loc=loc->next;

}

}

if(loc==NULL)

{

return ;

}

if(loc->pre==NULL)

{

return;

}

temp=loc->pre;

if(temp->pre==NULL)

{

\*head=loc;

loc->pre=NULL;

}

else

{

(temp->pre)->next=loc;

loc->pre=temp->pre;

}

free(temp);

}