E5-linked list

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

}\*temp,\*temp\_new;

struct node \*first=NULL;

struct node \*ptr01,\*ptrn,\*ptr,\*pre\_ptr,\*next\_ptr,\*ptr1,\*ptr2,\*tempx,\*temp\_new1;

struct node \*list1,\*list2;

void create(struct node \*temp)

{

int i,n,x;

printf("Enter number of nodes \n");

scanf("%d",&n);

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

{

printf("enter data to be inserted : ");

scanf("%d",&x);

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

temp->data=x;

temp->next=NULL;

if(first==NULL)

{

first=temp;

first->next=NULL;

}

else

{

ptr=first;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

return;

}

void display(struct node \*first)

{ ptr=first;

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

printf("\n");

return;

}

int counter()

{

int count=1;

ptr=first;

while(ptr->next!=NULL)

{

count++;

ptr=ptr->next;

}

return count;

}

void insert\_from\_beg()

{int x;

printf("enter data to be inserted : ");

scanf("%d",&x);

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

temp\_new->data=x;

temp\_new->next=first;

first=temp\_new;

return;

}

void insert\_from\_end()

{int x;

printf("enter data to be inserted : ");

scanf("%d",&x);

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

temp\_new->data=x;

temp\_new->next=NULL;

ptr=first;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp\_new;

return;

}

void custom\_insert()

{ int x,pos;

printf("enter data to be inserted : ");

scanf("%d",&x);

printf("enter before which value to insert \n");

scanf("%d",&pos);

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

temp\_new->data=x;

ptr=first;

while(ptr->data!=pos)

{

pre\_ptr=ptr;

ptr=ptr->next;

}

pre\_ptr->next=temp\_new;

temp\_new->next=ptr;

return;

}

void delete\_from\_beg()

{ printf("deletion in progress....\n");

ptr=first;

first=first->next;

free(ptr);

return;

}

void delete\_from\_end()

{ printf("deletion in progress....\n");

ptr=first;

while(ptr->next!=NULL)

{ pre\_ptr=ptr;

ptr=ptr->next;

}

pre\_ptr->next=NULL;

free(ptr);

return;

}

void custom\_delete()

{ int pos;

printf("enter the value of node to delete \n");

scanf("%d",&pos);

ptr=first;

if(pos==ptr->data)

{

delete\_from\_beg();

return;

}

else{

while(ptr->data!=pos)

{

pre\_ptr=ptr;

ptr=ptr->next;

}

pre\_ptr->next=ptr->next;

free(ptr);

}

return;

}

void reverse\_list()

{

pre\_ptr=NULL;

ptr=first;

next\_ptr=NULL;

while(ptr!=NULL)

{

next\_ptr=ptr->next;

ptr->next=pre\_ptr;

pre\_ptr=ptr;

ptr=next\_ptr;

}

first=pre\_ptr;

}

int search\_sorted\_list(int value\_in)

{

ptr=first;

while(ptr!=NULL)

{

if(value\_in==ptr->data)

{

return 1;

}

ptr=ptr->next;

}

return 0;

}

void merging\_ll()

{

int n1,n2,i;

struct node \*temp1,\*temp2,\*temp3;

struct node \*first1=NULL,\*first2=NULL,\*first3;

struct node \*ptr1,\*ptr2,\*ptr3=NULL;

printf("enter number of node of list 1 : ");

scanf("%d",&n1);

//logic of create

for(i=0;i<n1;i++)

{

temp1=(struct node\*)malloc(sizeof(struct node));

printf("Enter the data\n");

scanf("%d",&temp1->data);

temp1->next=NULL;

if(first1==NULL)

{

first1=temp1;

}

else

{

ptr1=first1;

while(ptr1->next!=NULL)

{

ptr1=ptr1->next;

}

ptr1->next=temp1;

}

}

printf("enter number of nodes of list 2: ");

scanf("%d",&n2);

//logic of create

for(i=0;i<n2;i++)

{

temp2=(struct node\*)malloc(sizeof(struct node));

printf("Enter the data\n");

scanf("%d",&temp2->data);

temp2->next=NULL;

if(first2==NULL)

{

first2=temp2;

}

else

{

ptr2=first2;

while(ptr2->next!=NULL)

{

ptr2=ptr2->next;

}

ptr2->next=temp2;

}

}

//merge logic as obs

ptr1=first1;

ptr2=first2;

while(ptr1!=NULL && ptr2!=NULL)

{

temp3=(struct node\*)malloc(sizeof(struct node));

temp3->next=NULL;

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

{

temp3->data=ptr1->data;

ptr1=ptr1->next;

}

else if(ptr1->data > ptr2->data)

{

temp3->data=ptr2->data;

ptr2=ptr2->next;

}

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

{

temp3->data=ptr1->data;

ptr1=ptr1->next;

ptr2=ptr2->next;

}

if(ptr3==NULL)

{

ptr3=temp3;

first3=temp3;

}

else

{

ptr3->next=temp3;

ptr3=temp3;

}

}

while(ptr1!=NULL)

{

temp3=(struct node\*)malloc(sizeof(struct node));

temp3->next=NULL;

temp3->data=ptr1->data;

ptr3->next=temp3;

ptr3=temp3;

ptr1=ptr1->next;

}

while(ptr2!=NULL)

{

temp3=(struct node\*)malloc(sizeof(struct node));

temp3->next=NULL;

temp3->data=ptr2->data;

ptr3->next=temp3;

ptr3=temp3;

ptr2=ptr2->next;

}

display(first3);

return;

}

void sort() // bubble sort

{

int tempn;

struct node \*ptra,\*ptrb;

ptra=first;

while(ptra->next != NULL)

{

ptrb=ptra->next;

while(ptra->data > ptrb->data)

{

tempn=ptra->data;

ptra->data=ptrb->data;

ptrb->data=tempn;

}

ptra=ptra->next;

}

ptrb=ptrb->next;

}

int main()

{ int k,reciever,val,value\_search;

printf("you have entered a menu driven program\n");

do{

printf("enter ur choice 1-7 || 0 to exit loop \n");

printf("\n\t 1) to find length \n\t 2) create list \n\t 3) to insert node \n\t 4) to delete node \n\t 5) to reverse list \n\t 6) search for node \n\t 7) to merge \n\t 8) to sort linked list \n");

scanf("%d",&val);

switch(val)

{

case 1:reciever=counter();

printf("length of linked list is : %d \n",reciever);

break;

case 2:create(list1);

display(first);

break;

case 3:

printf("where you wish to enter the node (choose):\n\t 1) front \n\t 2) end \n\t 3) or Specify the position\n");

scanf("%d",&k);

switch(k)

{

case 1:

insert\_from\_beg();

display(first);

break;

case 2:

insert\_from\_end();

display(first);

break;

case 3:

custom\_insert();

display(first);

break;

}

break;

case 4:

printf("where you wish to delete the node (choose):\n\t 1) front \n\t 2) end \n\t 3) or Specify the position\n");

scanf("%d",&k);

switch(k)

{

case 1:

delete\_from\_beg();

display(first);

break;

case 2:

delete\_from\_end();

display(first);

break;

case 3:

custom\_delete();

display(first);

break;

}

break;

case 5:

reverse\_list();

display(first);

break;

case 6:

printf("enter element to be searched : \n");

scanf("%d",&value\_search);

k=search\_sorted\_list(value\_search);

if(k==1)

{

printf("found\n");

}

else{

printf("not available\n");

}

break;

case 7:

merging\_ll();

break;

case 8:

sort();

display(first);

break;

}

}while(val!=0);

return 0;

}