Linked List Experiment to concatenate , reverse or find intersection of linked lists.

Name : Aditi Paretkar

Roll :1911034

Batch : A3

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node\*AddEnd(struct node \*a,int k);

void Display(struct node\*b);

struct node\*concatenate(struct node\*a,struct node\*b);

struct node\*intersect(struct node\*a,struct node \*b, struct node\*c);

struct node\*reverse(struct node\*a);

int main(void) {

struct node\*p= NULL;

struct node\*q= NULL;

struct node\*r= NULL;

int ch,n1,n2,i;

printf("Hello World\n");

printf("Enter 1 to concatenate or 2 to show the intersection or 3 to Reverse the list");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

printf("Enter the number of elements in the first list\n");

scanf("%d",&n1);

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

{

p = AddEnd(p,-1);

}

printf("The elements of the first list are \n");

Display(p);

printf("Enter the number of elements in the second list\n");

scanf("%d",&n2);

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

{

q = AddEnd(q,-1);

}

printf("The elements of the second list are\n");

Display(q);

p=concatenate(p,q);

printf("the concatenated list is\n");

Display(p);

}

break;

case 2:

{

printf("Enter the number of elements in the first list\n");

scanf("%d",&n1);

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

{

p = AddEnd(p,-1);

}

printf("The elements of the first list are \n");

Display(p);

printf("Enter the number of elements in the second list\n");

scanf("%d",&n2);

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

{

q = AddEnd(q,-1);

}

printf("The elements of the second list are\n");

Display(q);

r= intersect(p,q,r);

if(r==NULL)

{

printf("no common elements\n");

}

else

{printf("the linked list with common elements is :\n");

Display(r);

}

}

break;

case 3:

{

printf("Enter the number of elements elements of the list");

scanf("%d",&n1);

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

{

p=AddEnd(p,-1);

}

printf("the original list is\n");

Display(p);

p= reverse(p);

printf("The reversed list is\n");

Display(p);

}

}

return 0;

}

struct node\* AddEnd(struct node\*q,int k)

{

struct node \*p1=q;

int num;

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

if(k==-1)

{//this condition is executed when the AddEnd function is called from Main()

printf("enter the value of the element you want to insert\n");

scanf("%d",&num);

new->data = num;

}

else{

new->data=k;//this condition is executed when the AddEnd Function is called from the Intersect function

}

if(q==NULL)

{

new->next=NULL;

q=new;

}

else

{

while(p1->next!=NULL)

{

p1= p1->next;

}

p1->next = new;

new->next = NULL;

}

return q;

}

void Display(struct node \*q)

{

while (q!=NULL)

{

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

q=q->next;

printf("->");

}

printf("null\n");

}

struct node\*concatenate(struct node\*a, struct node\*b)

{

if(a==NULL)

{

return b;

}

else if (b==NULL)

{

return a;

}

else

{

struct node\*p1= a;

while(p1->next!=NULL)

{

p1=p1->next;

}

p1->next=b;

return a;

}

}

struct node\*intersect(struct node\*p,struct node\*q,struct node\*r)

{int count =0;

struct node\*p1=p;

struct node\*p2=q;

if(p==NULL)

{

return NULL;

}

else if(q==NULL)

{

return NULL;

}

else

{

while(p1!=NULL)

{p2=q;//to bring the pointer of the second list back to the start of the second list on each iteration of the first list.

while(p2!=NULL)

{

if(p1->data==p2->data)

{

count++;

r= AddEnd(r,p1->data);

}

p2=p2->next;

}

p1=p1->next;

}

if(count==0)

{//in case there are no common elements

return NULL;

}

else

{

return r;

}

}

}

struct node\*reverse(struct node\*p)

{

if(p==NULL)

{

return NULL;

}

struct node\*p1=p;//p will point to the head of the new list, p1 will be the current node and p2 will be the node before the current node.

struct node\*p2 = p->next;

p=p2;

p1->next=NULL;

while(p!=NULL)

{

p=p->next;

p2->next=p1;

p1=p2;

p2=p;

}

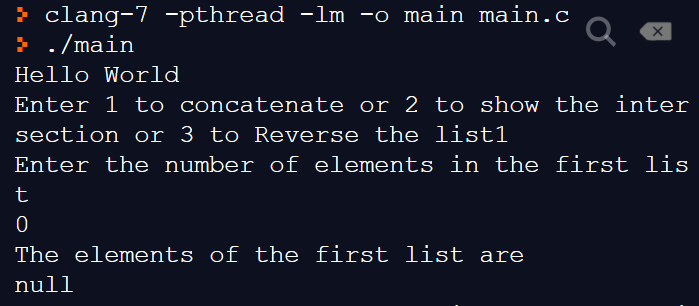
p=p1;

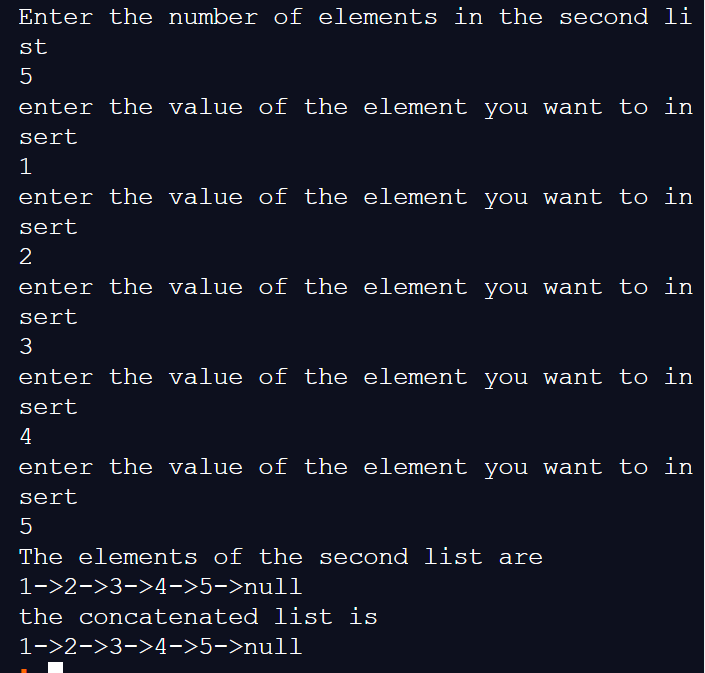
return p;

}

1. Concatenating two linked lists:

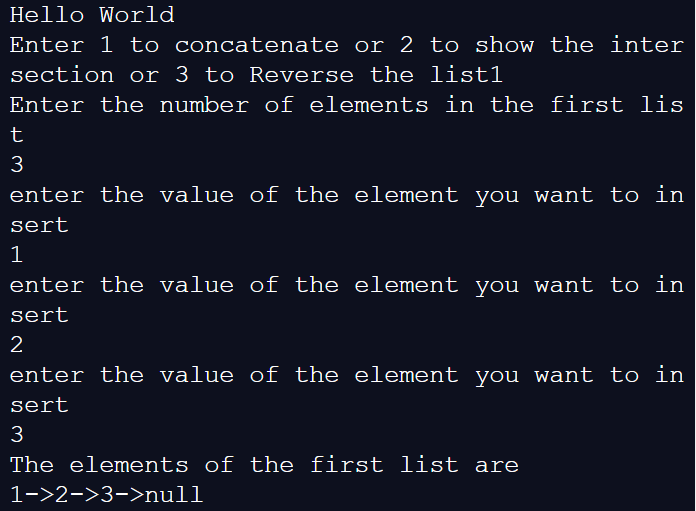
Case 1: when any of the linked lists are null , it will return the other linked list:

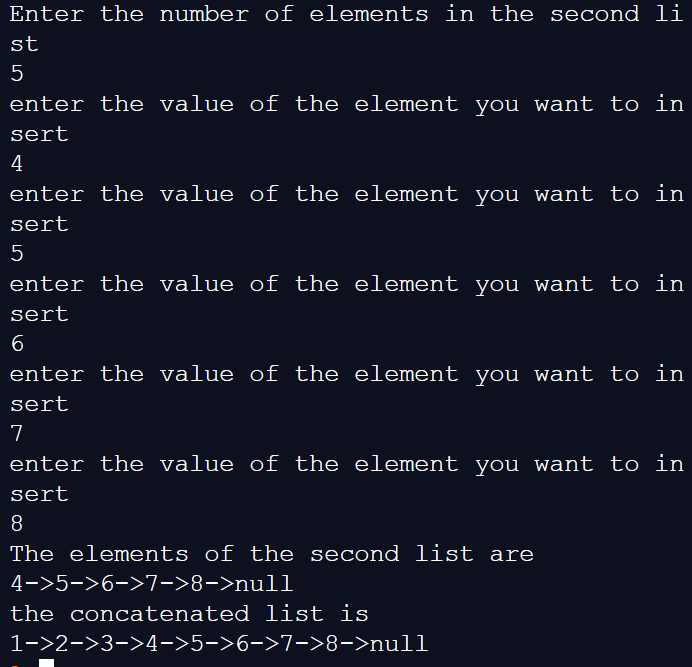




Case 2:

Concatenating when none of the lists are empty :

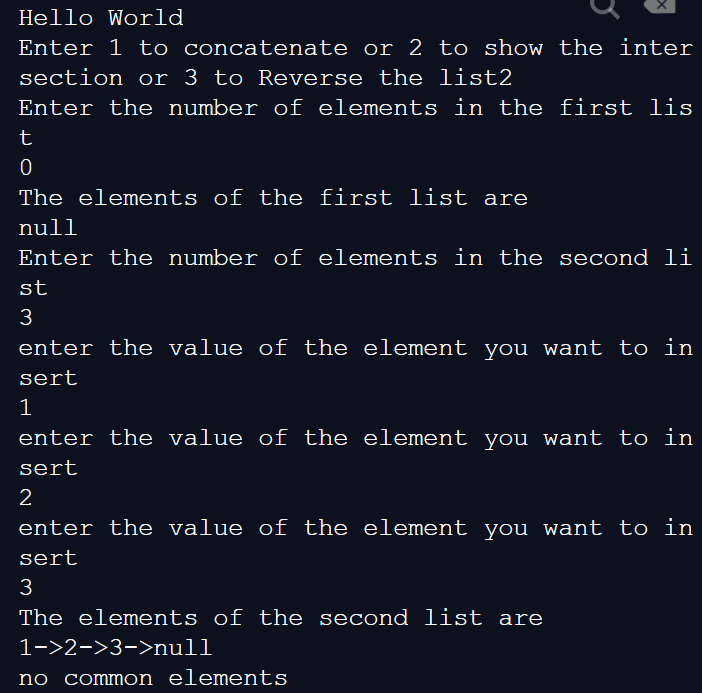




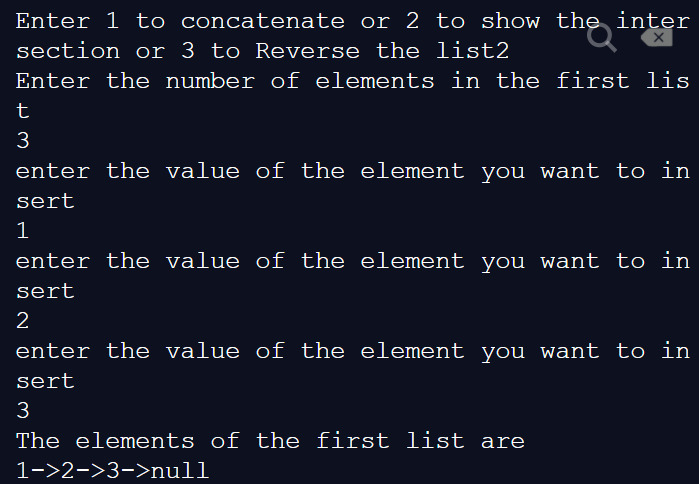
1. Finding the intersection of two linked lists :

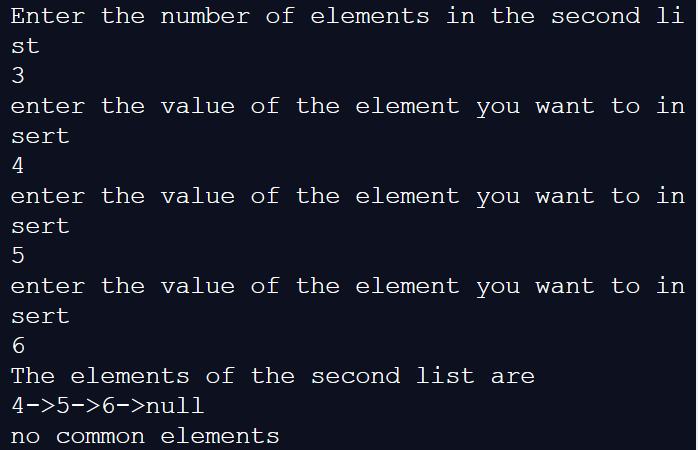
Case 1:

When one of the list is empty :

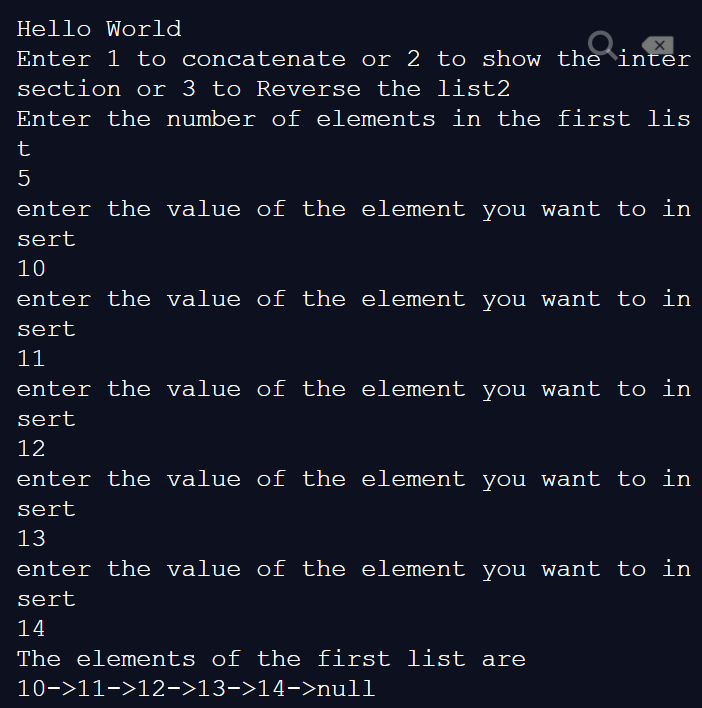


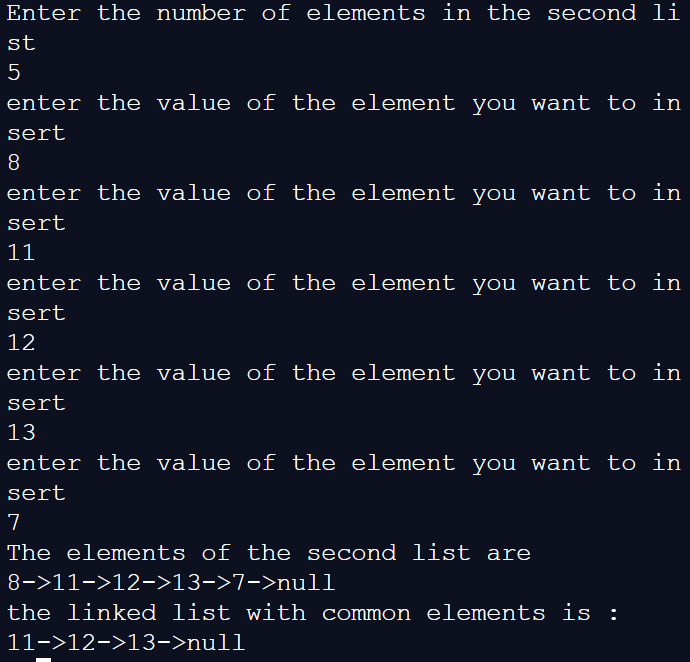
Case 2: when there are no common elements :





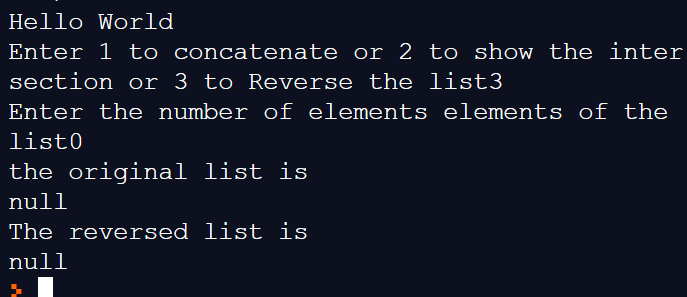
Case 3: creating a linked list with common elements from both :





3: Reversing the linked list :

Case 1 : when there are no elements in the list :



Case 2: reversing a normal linked list :

