**Batch: A3 Roll No.: 1911034**

**Experiment / assignment / tutorial No. 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of Advanced operations on linked list-Concatenate two lists, Reverse a list, Intersection of two lists. |

**Objective:** To Understand and implement Advanced operations on linked list-Concatenate two lists, Reverse a list, Intersection of two lists.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. <https://www.geeksforgeeks.org/data-structures/linked-list/>
3. <https://www.thecrazyprogrammer.com/2014/02/concatenation-of-two-linked-lists.html>

**Abstract**:

Linked lists are special lists of data elements linked to one another. The logical ordering is represented by having each element pointing to the next element. Each element is called a node, which has two parts, INFO part which stores the information and pointer which points to the next element.

The Last node’s Address field contains Null rather than a valid address. It’s a NULL Pointer and indicates the end of the list.

The linear order in Linked Lists is given by means of pointers

**Related Theory: -**

**Concatenate two lists-**

In this, we are asked to accept two linked lists from the user, and pointers to the start of each linked list. The result will give us another linked list that contains the two lists concatenated.

It is done as follows:

1. Traverse over the linked list ‘a’ until the element next to the node is not NULL.
2. If the element next to the current element is NULL (a->next == NULL) then change the element next to it to ‘b’ (a->next = b).

**Reverse a list-**

1. In this, we are asked to accept one linked list from a user, and we accept it using the AddEnd() function .

2. The output displayed to the user, the original linked list in the reversed form.

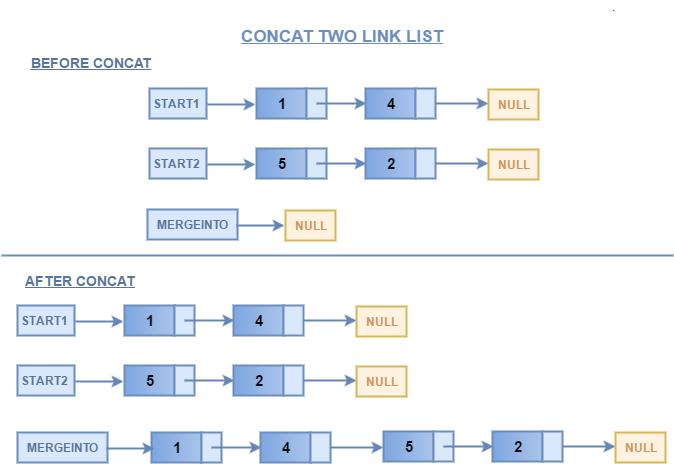
3. This is done by changing the links between nodes in the original list.

**Intersection of two lists-**

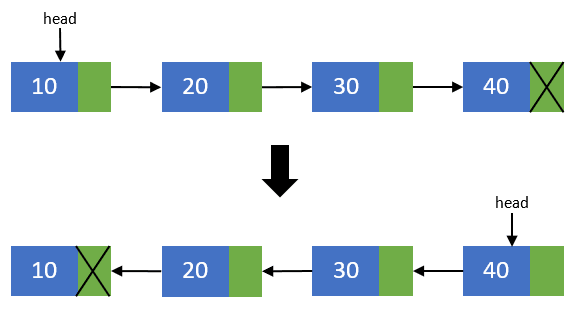
1. In this, the user is asked to provide us two linked lists, and we add each node using the AddEnd() function.
2. The result provided to the user is a new linked list , that contains all the elements which are common to both the lists(intersection of the lists)
3. It will return a pointer to the start of the new list
4. We also have to consider the special cases where there are no intersections in a list, there are two such cases :
5. When either of the list is empty
6. When there are no common elements
7. In the above two cases, the new linked list will not be made.

**Diagram for each Advanced Operation:**

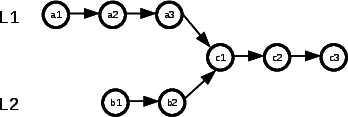
**Diagram for Concatenating Two Linked Lists:**



**Diagram to reverse a linked list**



**Diagram to find Intersection of two Linked Lists:**



**Algorithm for Concatenation of two lists, Reversing a list, Intersection of two lists:**

1. **Algorithm to concatenate two linked lists**

STEP1- Create two pointers p1, p2 to point to the heads of the two linked lists, and initially point them to NULL

STEP2- Accept the two linked lists by making use of the AddEnd() function and make p1 and p2 point to the heads of the two lists.

STEP3- If p1 = NULL return p2 , or if p2 = NULL , return p1

STEP4- If p1!=NULL and p2!=NULL, set ptr = p1.

STEP 5- Repeat STEP6 while ptr->next !=NULL

STEP6- Set ptr= ptr->next

[END OF LOOP]

STEP7- Set ptr->next = p2

STEP 7- Return p1 .

(p1 will now point to the head of the concatenated linked lists).

1. **Algorithm to find the Intersection of two linked lists**

STEP1- Create two pointers p1, p2 to point to the heads of the two linked lists, and initially point them to NULL

STEP2- Accept the two linked lists by making use of the AddEnd() function and make p1 and p2 point to the heads of the two lists.

STEP3 – create a new pointer p3 to point to the head of the new linked list and set it to NULL

STEP4- if p1=NULL or if p2=NULL, return p3=NULL(there will be no intersection).

STEP5- Else if p1!=NULL and p2!=NULL, set pa = p1, pb =p2, count=0 and repeat STEP 6 to 9 while pa!=NULL

STEP 6- Repeat steps 7 to 10 while pb!=NULL

STEP7- If pa -->data == pb --> data, create a new\_node with new\_node-->data = pa-->data and new\_node-->next =NULL

STEP8- If p3==NULL , set p3=new\_node , else add new\_node to the end of the list using AddEnd() function.

STEP9- Set count = count+1

STEP10- Set pb=pb -->next

[END OF INNER LOOP]

Step 11- Set pa= pa-->next

[END OF OUTER LOOP]

Step12- If count=0, return NULL (No common elements)

Step13 – Else return p3.

1. **Algorithm to reverse a Linked List (in this we are reversing the original linked list by changing the links between the nodes).**

STEP1- Accept the Linked List from the user using the AddEnd() function.

STEP2- Create 3 pointers , cnode- to point to the current node, pnode- to point to the one before cnode and head to point to the one after cnode, set count=0.

STEP3- Initially , let pnode =first node, cnode= second node , and head=cnode

STEP4- Repeat steps 5 to 8 while head!=NULL

STEP5- If count=0, set pnode-->next=NULL.

[end of if]

STEP6- Set head = head --> next

STEP7- set cnode--> next = pnode

STEP8 Set pnode= cnode , then set cnode= head.

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored.**

**Steps followed and options explored :**

1. The program is menu driven, i.e. the user is given a choice two choose between concatenating or finding intersection or reversing two lists.
2. On entering the choice , the required linked list/s is entered by calling the AddEnd Function.
3. The control is then passed to the appropriate function concatenate() , reverse() or intersect() to perform the required operation, according to the algorithms described above.
4. After performing the operation , a pointer to the start of the resultant linked list is returned and displayed to the user.

**Options Explored:**

1. The program creates the linked list using a struct node that contains a data field and a pointer
2. The program takes into consideration all possible options for the input , such as when a linked list is empty in the input , to return a null pointer.
3. It also considers other cases such as to return a NULL intersection of two lists when they have no common elements etc.
4. In each function , separate pointers have been created to point to the heads of the two lists to iterate through the lists and perform other operations on them.

**Assumptions made for Input:**

1. It is assumed that the data entered by the user will be an integer.
2. It is assumed that user will enter a finite number of element for each list at runtime.

**Built-In Functions Used:**

Malloc() – for dynamically allocating memory to ‘n’ number of nodes at runtime whose value will be given by the user.

**Program source code for Concatenation of two lists, Reversing a list, Intersection of two lists. :**

#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;

}

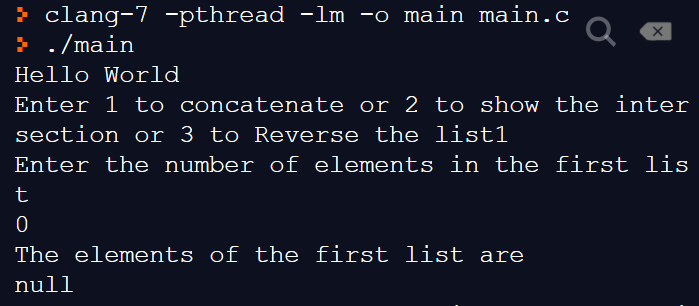
p=p1;

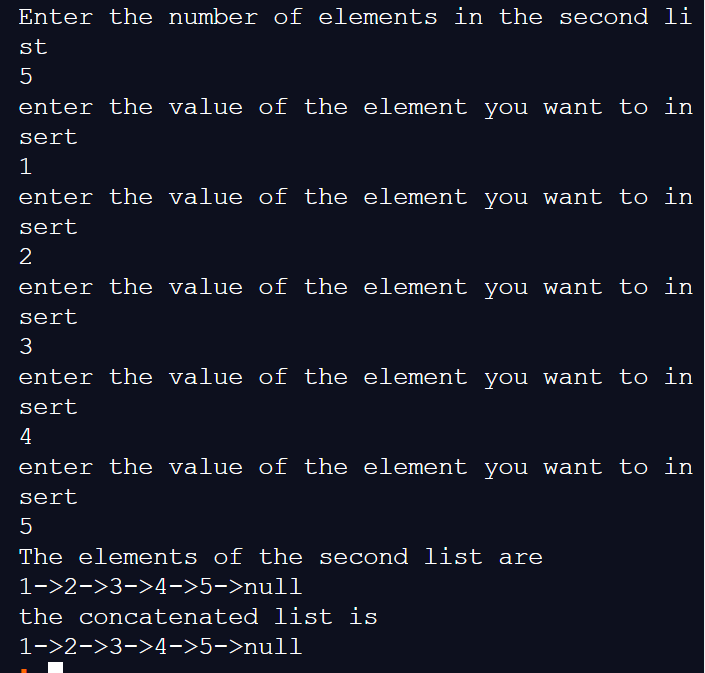
return p;

}

**Output Screenshots for each Advanced Operation :**Concatenating two linked lists:

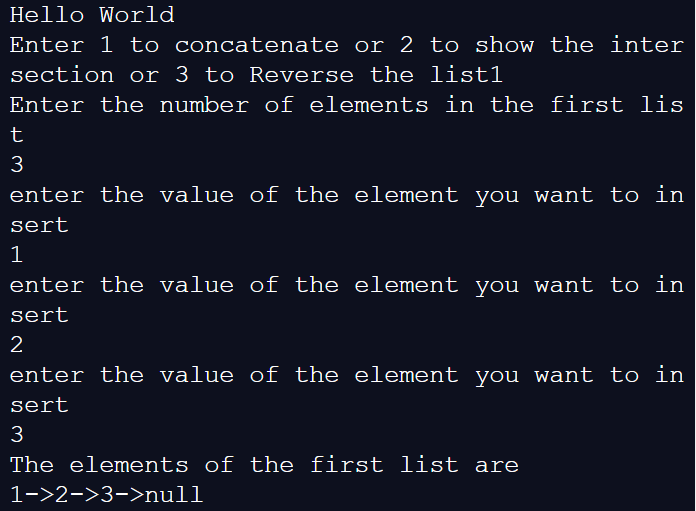
Case 1: when any of the linked lists are null , it will return the other linked list:

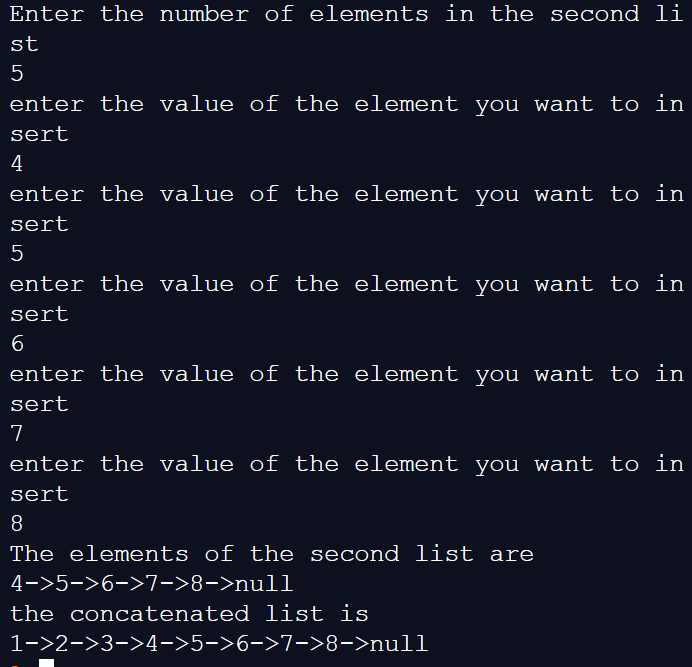




Case 2:

Concatenating when none of the lists are empty :

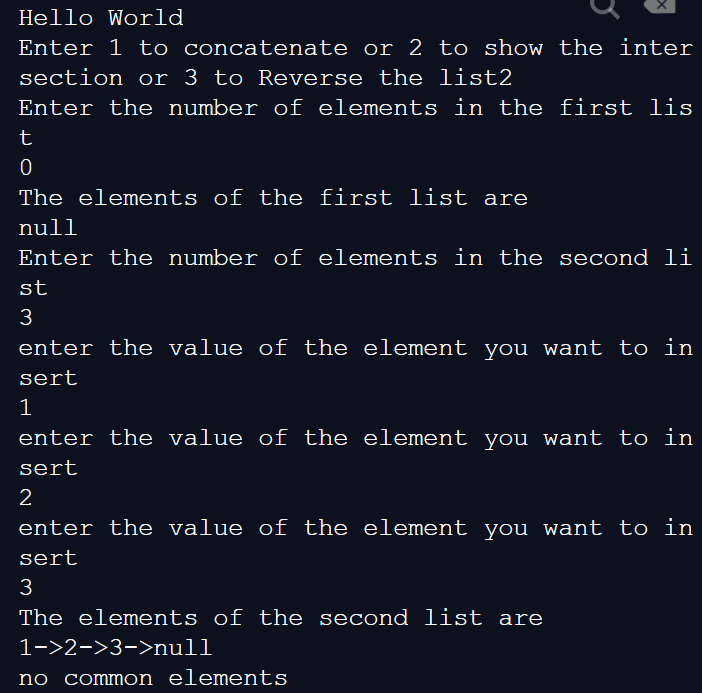




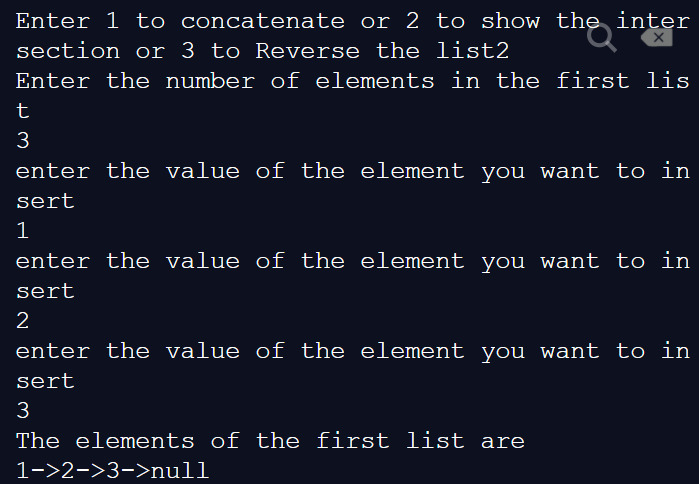
1. Finding the intersection of two linked lists :

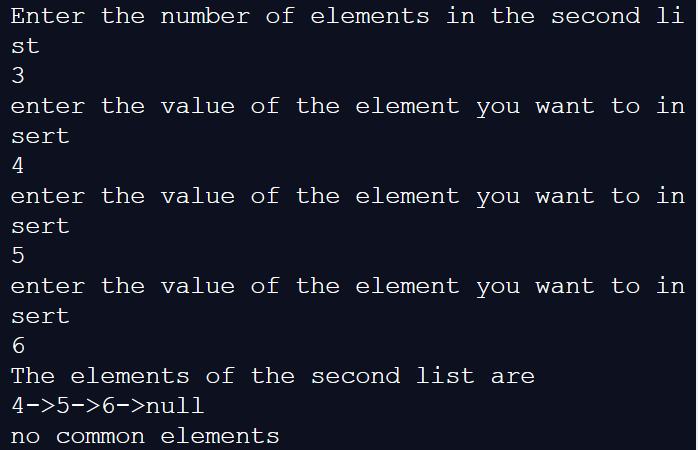
Case 1:

When one of the list is empty :

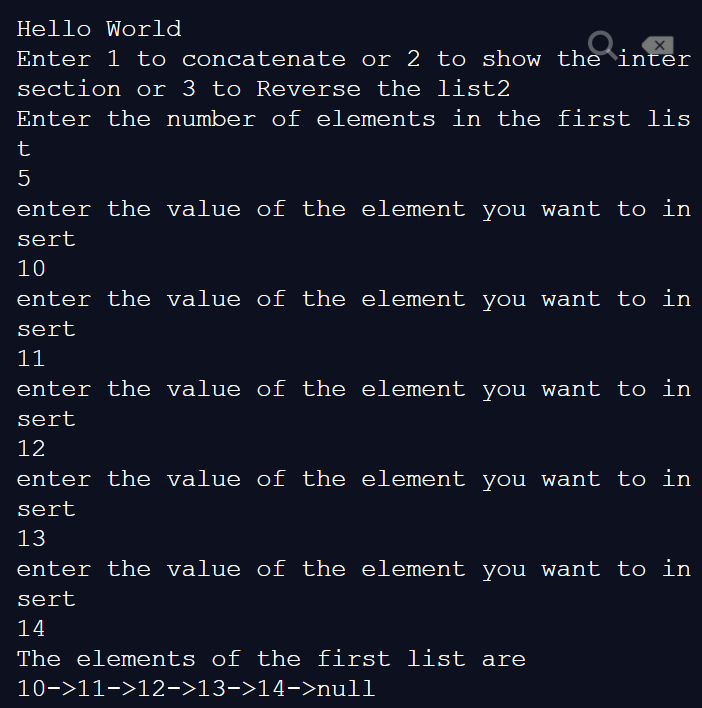


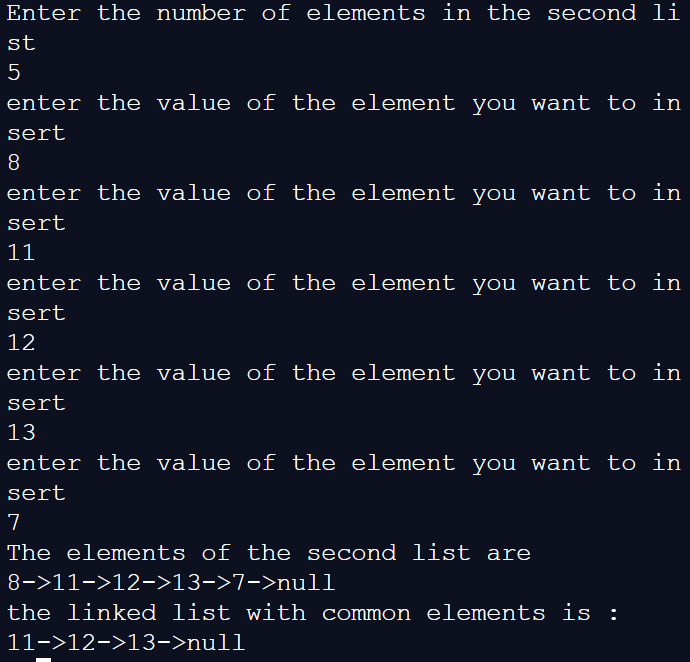
Case 2: when there are no common elements :





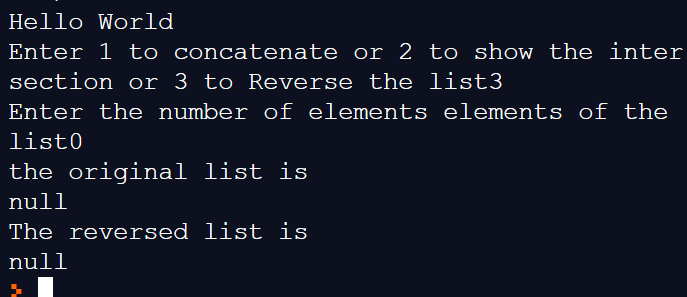
Case 3: creating a linked list with common elements from both :



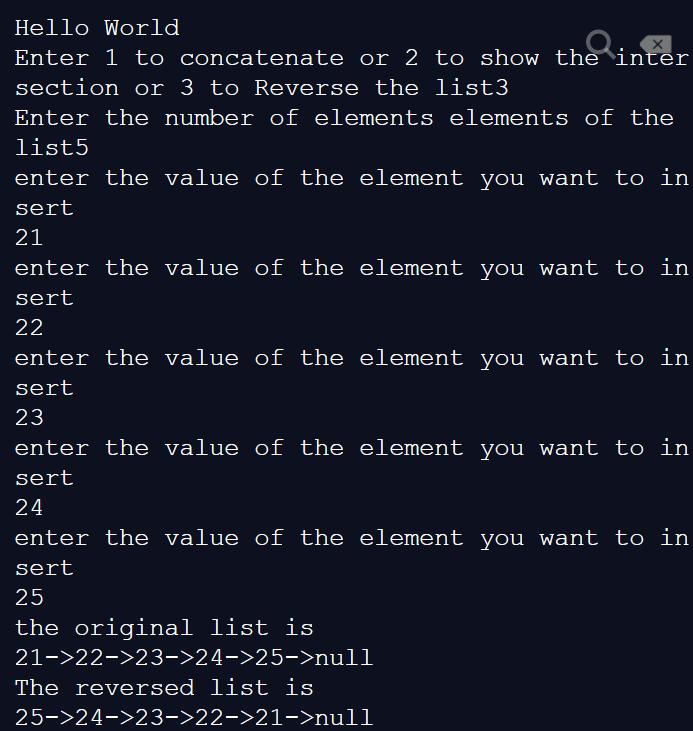


3: Reversing the linked list :

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



Case 2: reversing a normal linked list :



**Explain the Importance of the approach followed by you**

.

1. In this approach, we have made use of the basic operations on a linked list such as AddEnd() , Display() to implement the more advanced operations on a linked list/s.
2. We have taken into consideration most of the possible edge cases that might arise due to which the program might not work , such as when a linked list is empty ,etc , and printed appropriate messages for them.
3. The dynamic memory allocation of each node of a linked list using a struct, allows us to perform operations accordingly on the n number of nodes according to value entered by user.

**Conclusion:-** In this experiment , we have learnt how to accept two linked lists and perform advanced operations such as concatenation , intersection and reversal of linked list. The resultant list is displayed with the help of a pointer to the start of the list

**PostLab Questions:**

1. **Illustrate 2 Applications of Linked Lists.**
2. **Web Browsers:** A good example of linked lists is in web browsers, where it creates a linked list of web pages visited , so that when we want to fetch history , or when we press the back button, the previous node’s data is fetched.
3. **Implementation of graphs**: Adjacency list representation of graphs is most popular which uses linked list to store the adjacent vertices
4. **Image browser**: Previous and next images are linked , hence they can be accessed by the previous and next button
5. **Compare and Contrast between Arrays and Linked Lists.**

|  |  |
| --- | --- |
| **ARRAY** | **LINKED LIST** |
| 1. Array is a collection of elements of similar data type. | Linked List is an ordered collection of elements of same type, which are connected to each other using pointers. |
| 1. In an array, elements are stored in contiguous memory location or consecutive manner in the memory. | In a linked list, new elements can be stored anywhere in the memory.  Address of the memory location allocated to the new element is stored in the previous node of linked list, hence forming a link between the two nodes/elements. |
| 1. Memory is allocated as soon as the array is declared, at compile time. It's also known as Static Memory Allocation | Memory is allocated at runtime, as and when a new node is added. It's also known as Dynamic Memory Allocation. |
| 1. Array can single dimensional, two dimensional or multidimensional | Linked list can be Linear(Singly), Doubly or Circular linked list. |
| 1. Array gets memory allocated in the Stack section. | Whereas, linked list gets memory allocated in Heap section. |