**Batch:\_\_\_\_\_\_\_A3\_\_\_ Roll No.:\_\_\_\_\_\_\_\_\_1911034\_\_\_\_\_**

**Experiment No. 1**

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

|  |
| --- |
| **Title:**  Implementation of different operations on Linked List – creation, insertion, deletion, traversal, searching an element |

**Objective:** To understand the advantage of linked list over other structures like arrays in implementing the general linear list

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| CO 1 | To understand the advantage of linked list over other structures like arrays in implementing the general linear list |

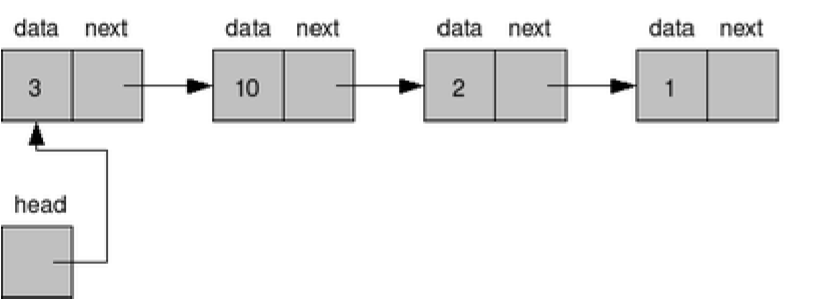
**Books/ Journals/ Websites referred:**

1. Data Structures A Pseudocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, second edition, CENGAGE learning.

2. Data Structures Using C & C++, Rajesh K. Shukla, Wiley- india.

**Abstract**:-

- A linear list is a list where each element has a unique successor. There are four common operations associated with linear list: insertion, deletion, retrieval, and traversal. Linear list can be divided into two categories: general list and restricted list. In general list the data can be inserted or deleted without any restriction whereas in restricted list there is restrictions for these operations. Linked list and arrays are commonly used to implement general linear list. A linked list is simply a chain of structures which contain a pointer to the next element. It is dynamic in nature. Items may be added to it or deleted from it at will.



A list item has a pointer to the next element, or to NULL if the current element is the tail (end of the list). This pointer points to a structure of the same type as itself. This Structure that contains elements and pointers to the next structure is called a Node.

**Related Theory: -**

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.

**Advantages of linked list over arrays:**

**The advantages are as follows ;**

1. Since the size of the array is always fixed. Therefore it has to be specified in advance. The array cannot grow dynamically and it leads also leads to wastage of space if all blocks of memory are not utilised. Whereas in Linked List , we can add elements dynamically if we wish to do so.
2. In order to insert an element in an array in an sorted order , some elements have to be shifted / moved , but in a linked list that is not required.
3. The same goes with deletion of an element in an array which is more difficult than a linked list.

**Drawbacks of linked list over arrays**:

The drawbacks are as follows ;

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do a binary search with linked lists.  
2) Extra memory space for a pointer is required with each element of the list.  
3) Arrays have better cache locality that can make a pretty big difference in performance.

**Linked List ADT:**

1. A linked list is a data structure consisting of nodes, where each node has a data field , and an address field , pointing to the next node.
2. The data field can be used to store the value of any data type.
3. The entire linked list can be accessed through the help of a pointer pointing to the first node.
4. The address field of the last node , points to a memory location ‘null’ in order to indicate that there are no more nodes after this node.
5. A Linked List can be implemented with the help of structs or arrays.
6. Many operations can be performed on a linked list , such as Creation of new nodes, adding an element to the beginning of the list , deleting and element from the list , searching for an element or counting the number of nodes , to name a few.
7. A linked list , is dynamic data structure which means that the nodes may be added or removed as and when required.

**Algorithm for creation, insertion, deletion, traversal and searching an element in linked list:**

1. **Algorithm to Create List**

STEP1: Let n = Number of Nodes in the linked list

STEP2: REPEAT STEPS 3 TO 10 FOR i=1 to N

STEP 3: ALLOCATE MEMORY FOR NEW\_NODE

STEP 4: IF i=1 , SET NEW\_NODE-->DATA = VAL

STEP 5: SET PTR = NEW\_NODE, SET P1 = PTR

[END OF IF]

STEP 6: ELSE REPEAT WHILE P1-->NEXT !=NULL

STEP7 : SET P1= P1-->NEXT

[END OF LOOP]

STEP 8: SET NEW\_NODE-->DATA = VAL

STEP 9: SET NEW\_NODE-->NEXT = NULL

STEP 10: SET P1-->NEXT = NEW\_NODE

[END OF LOOP]

1. Algorithm to insert a new node at the beginning of the linked list.

STEP1: Allocate memory for NEW\_NODE

STEP2: Set NEW\_NODE-->DATA =VAL;

STEP3: SET NEW\_NODE-->NEXT =START

STEP4: SET START = NEW\_NODE

1. Algorithm to insert at the end of a linked list

STEP1 : Allocate memory for NEW\_NODE

STEP2:SET NEW\_NODE-->DATA = VAL

STEP3: SET NEW\_NODE-->NEXT = NULL;

STEP 4: SET PTR= START

STEP 5: REPEAT STEP 6 WHILE PTR-->NEXT!=NULL

STEP 6:SET PTR= PTR--> NEXT

[END OF LOOP]

STEP 7: SET PTR-->NEXT = NEW\_NODE

1. ALGORITHM TO SEARCH FOR AN ELEMENT :

STEP1: SET COUNT=0

STEP2: SET PTR= START

STEP3: REPEAT STEPS 4 AND 5 WHILE PTR!=NULL

SET COUNT = COUNT+1

SET PTR=PTR-->NEXT

[END OF LOOP]

STEP 6: WRITE COUNT

STEP7 : EXIT

1. ALGORITHM TO TRAVERSE A LINKED LIST

STEP1: SET PTR=START

STEP2: REPEAT STEPS 3 AND 4 WHILE PTR!=NULL

STEP3: Apply PROCESS to PTR-->DATA

STEP4: SET PTR= PTR-->NEXT

[END OF LOOP]

STEP 5: EXIT

1. ALGORITHM TO DELETE AN ELEMENT FROM THE LINKED LIST.

STEP1: LET VAL = VALUE OF ELEMENT AT THE NODE TO BE DELETED.

STEP 2: SET PTR = START , SET PTR1= PTR

STEP 3: REPEAT STEP 4 WHILE PTR-->DATA != VAL

STEP 4: SET PTR = PTR-->NEXT

[END OF LOOP]

STEP 5: REPEAT STEP 6 WHILE PTR1--> NEXT !=PTR

STEP 6 : PTR1=PTR1-->NEXT

[END OF LOOP]

STEP 7: SET PTR1-->NEXT = PTR-->NEXT

STEP 8 : FREE PTR

1. ALGORITHM TO DELETE THE FIRST ELEMENT.

STEP 1: SET P = START , SET Q = P

STEP 2: SET P = P-->NEXT

STEP 3: FREE(Q)

**Program source code:**

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*ptr;

};

struct node\*AddBegin(struct node\*,int val);

void Display(struct node\*q);

void Search(struct node\*);

struct node\* Delete(struct node\*);

struct node\* AddEnd(struct node\*);

struct node\* InsertAfter(struct node\*);

void Count(struct node\*);

struct node\* CreateList(struct node\*,int n);

int main(void) {

printf("Hello World\n");

int ch,n,val,i;

struct node \*p=NULL;

do

{printf("Enter any of the following\n1 to Create List\n2 to display\n3 to search\n4 to delete any element\n5 to add an element at the end\n6 to add an element after a node\n7 to count the number of nodes\n8 to add an element at the beggining\n-1 to Exit\n");

scanf("%d",&ch);

switch (ch)

{

case 1:

{printf("Enter the number of elements you want to insert\n");

scanf("%d",&n);

p=CreateList(p,n);

}

break;

case 2:

{

Display(p);

}

break;

case 3:

{

Search(p);

}

break;

case 4:

{

p= Delete(p);

}

break;

case 5:

{

p = AddEnd(p);

}

break;

case 6:

{

p = InsertAfter(p);

}

break;

case 7:

{

Count(p);

}

break;

case 8:

{

printf("Enter the value of the element you'd like to insert\n");

scanf("%d",&val);

p = AddBegin(p,val);

}

}

}

while(ch!=-1);

return 0;

}

struct node\* AddBegin(struct node\*q,int val)

{

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

new->data=val;

if(q==NULL)

{

//to initialize if node is first node

new->ptr=NULL;

q=new;

}

else

{

new->ptr=q;

q=new;

}

return q;

}

void Display(struct node \*q)

{

while (q!=NULL)

{

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

q=q->ptr;

printf("->");

}

printf("null\n");

}

void Search(struct node \*q)

{

int i, num,count=0;

printf("Enter the element whose position you want to find\n");

scanf("%d",&num);

struct node \*temp =q;

while(temp!=NULL)

{

if(temp->data==num)

{

printf("the element is found at node %d\n",count+1);//gives count according to 1-based indexing

break;

}

count++;

temp= temp->ptr;

}

if(temp==NULL)

{

printf("Element does not exist in the linked list. Please enter a valid element\n");

}

}

struct node\* Delete (struct node\*q)

{

int num;

struct node \*temp=q;

int count=0,flag=0;

printf("enter the element you want to delete\n");

scanf("%d",&num);

if(temp==NULL)

{

printf("linked list is already empty.\n");

}

else{

while(temp!=NULL)

{

if(temp->data==num)

{

printf("the element is found at node %d\n",count+1);

flag=1;

break;

}

count++;

temp= temp->ptr;

}

if(flag==0)

{

printf("element not found in the linked list, please try again\n");

}

else{

struct node \*p=q;

if(count==0)

{//special case to delete first element.

q=q->ptr;

free(p);

}

else{

while(p->ptr!=temp)

{

if(p->ptr==temp)

{

break;

}

else p = p->ptr;

}

p->ptr=temp->ptr;

free(temp);//free will be used to free the memory allocated by the current node

}

}

}

printf("element deleted successfully\n");

return q;

}

struct node\* AddEnd(struct node\*q)

{

struct node \*p1=q;

int num;

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

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

scanf("%d",&num);

new->data = num;

if(q==NULL)

{

printf("Linked List contains no nodes. Please Add some nodes first\n");

}

else

{

while(p1->ptr!=NULL)

{

p1= p1->ptr;

}

p1->ptr = new;

new->ptr = NULL;

}

return q;

}

struct node\*InsertAfter(struct node\*q)

{

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

int num,loc,i=0;

struct node\*temp =q;

struct node\*temp1=q;

int count =0;

printf("enter the value of the element\n");

scanf("%d",&num);

new->data= num;

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

scanf("%d",&loc);

while(temp1!=NULL)

{

temp1= temp1->ptr;

count++;

}

if(loc==0)

{

printf("the element cannot be inserted at the first position\n");

}

else if(loc>count)

{

printf("the linked list has insufficient elements\n");

}

else

{

while(i<loc-1)

{

temp=temp->ptr;

i++;

}

new->ptr= temp->ptr;

temp->ptr= new;

}

return q;

}

void Count(struct node\*q)

{

struct node \*temp =q;

int count =0;

if(temp ==NULL)

{

printf("number of elements is the list are %d\n",count);

}

else while(temp!=NULL)

{

count++;

temp= temp->ptr;

}

printf("the number of elements in the list are %d\n",count);

}

struct node\* CreateList(struct node\*q,int n)

{

int i;

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

{

struct node \*p1=q;

int num;

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

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

scanf("%d",&num);

new->data = num;

if(q==NULL)

{

new->ptr=NULL;

q=new;

}

else

{

while(p1->ptr!=NULL)

{

p1= p1->ptr;

}

p1->ptr = new;

new->ptr = NULL;

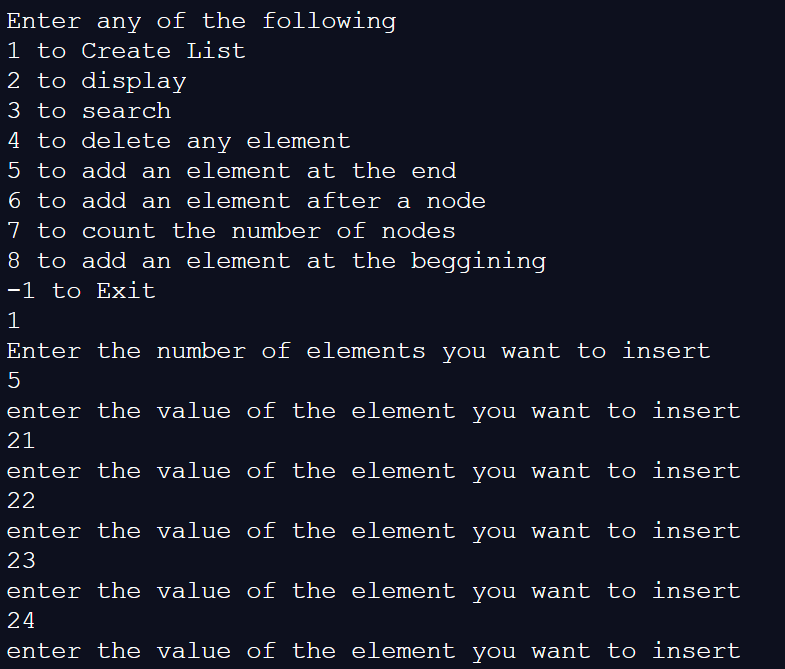
}

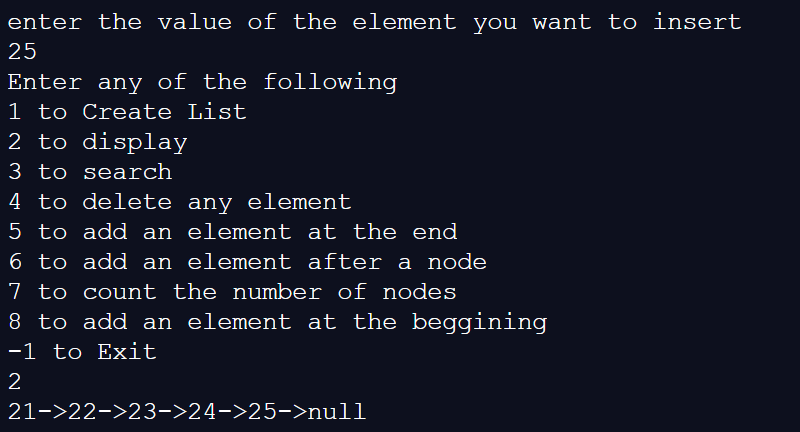
}

return q;

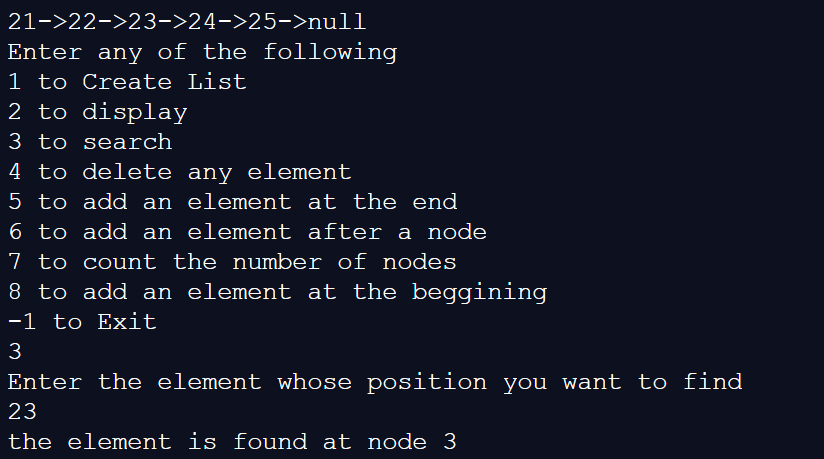
}

**Output Screenshots:**

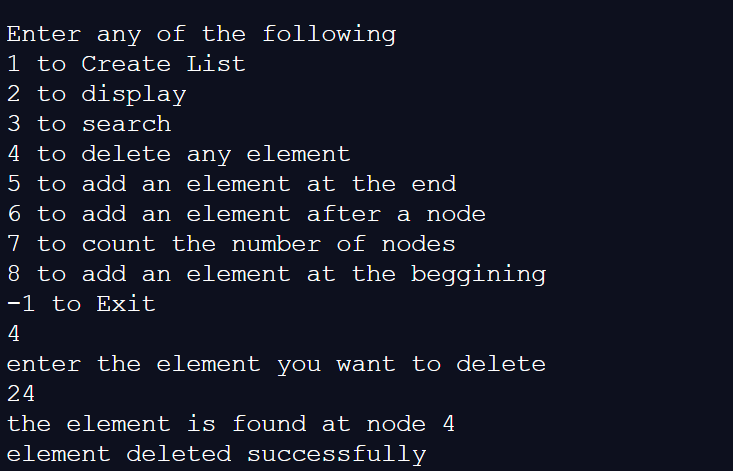




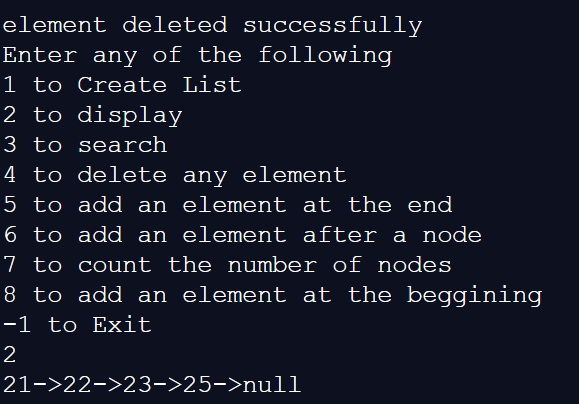
Searching where the value of a node is (by the data at that node)



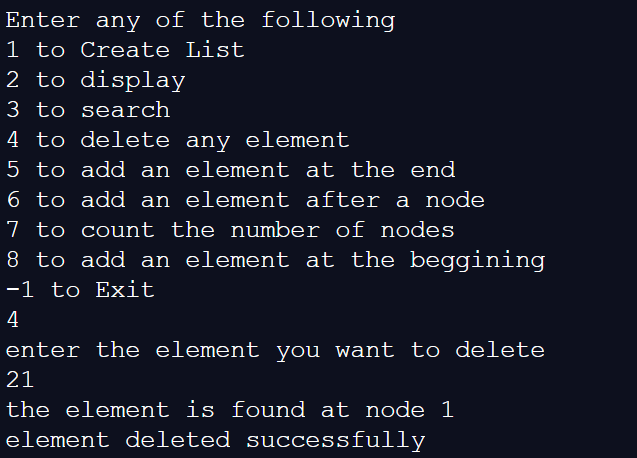
Deleting a node in the middle



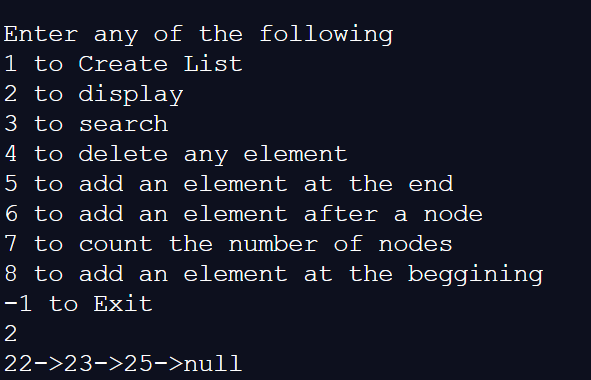
After deleting :



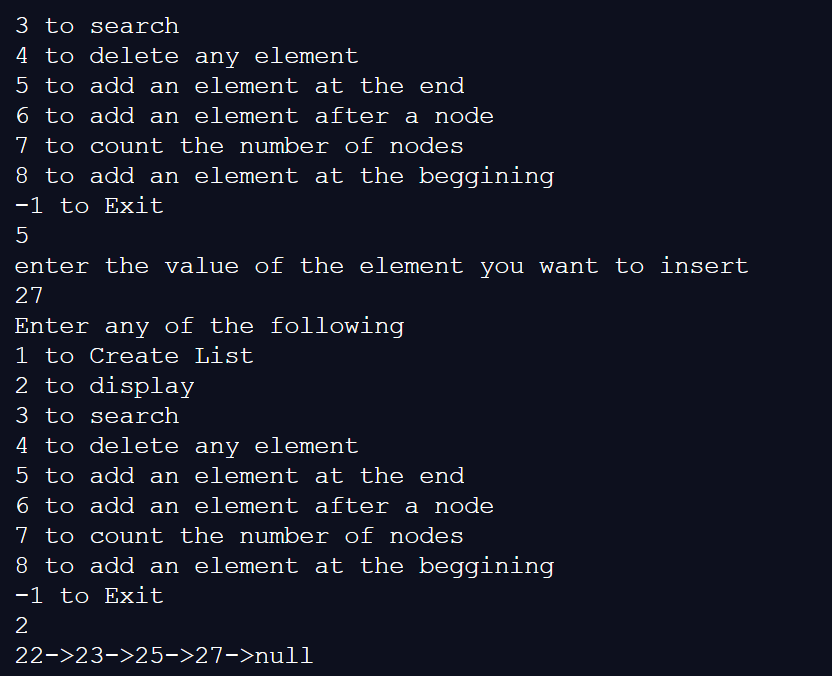
Special case of deleting first element :



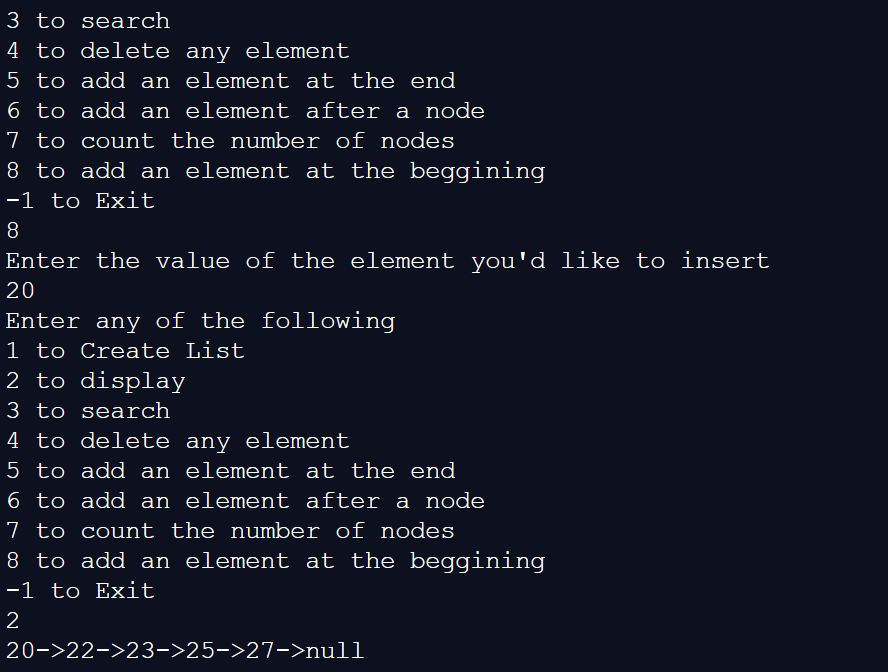
After deleting :



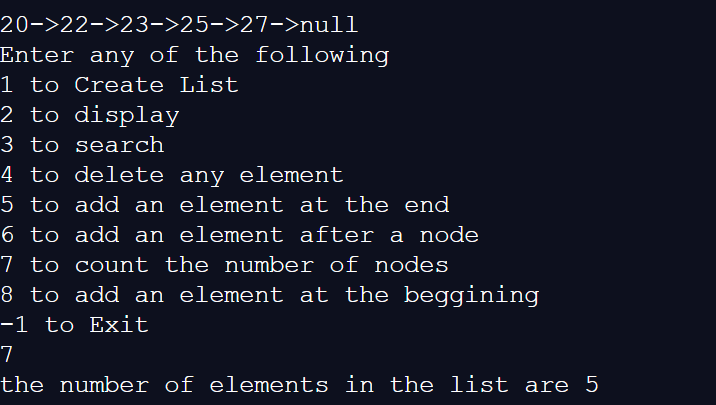
Adding an element at the end:



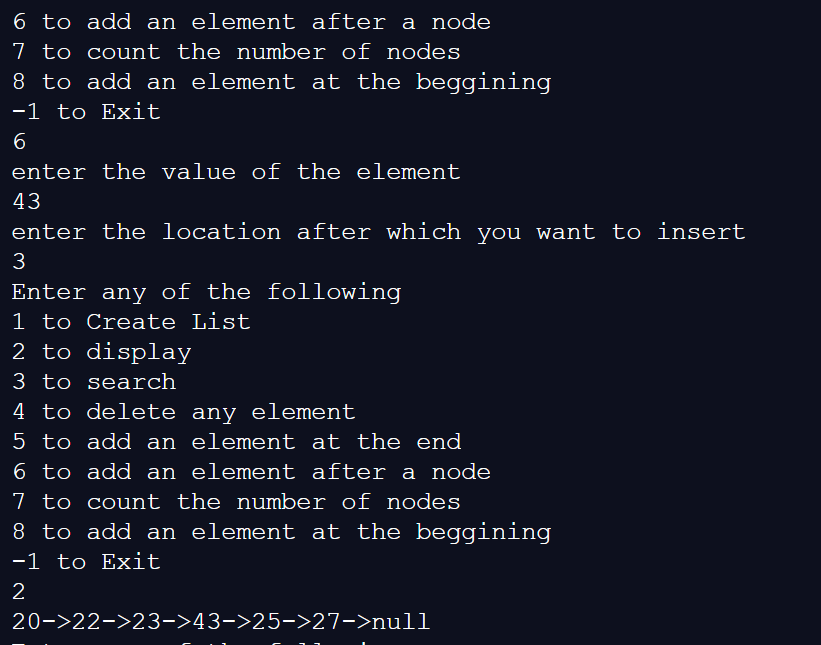
Adding an element at the beginning :



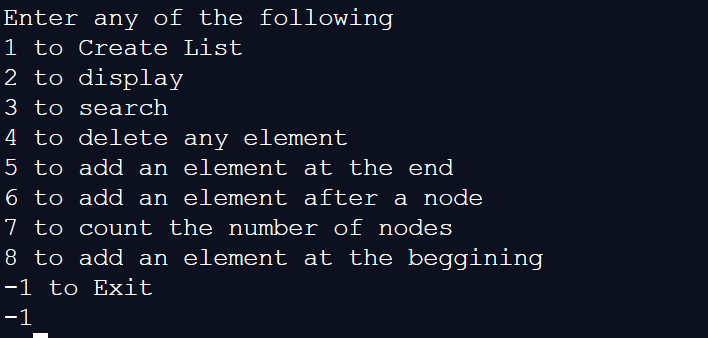
Counting the number of nodes:



Adding an element after a node whose number is specified by the user:



Exit



**Implementation Details:**

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

Steps followed:

1. Each node is defined with the help of a struct which contains an int value for the data and a struct node type pointer for pointing to the next field.
2. Initially the user is given a menu driven program with different options such as create list, insert , delete , etc.
3. After choosing option 1, user can enter the number of nodes to be entered and value of each node is asked after which linked list is created by adding nodes sequentially one after the other at the end.
4. The various options provided are as follows:
5. In searching for the element , in case the element is not found , the ‘not found element is displayed’
6. In deletion, there is a special case for deleting the first element apart from the usual case of deleting the other elements.
7. In the InsertAfter() function , special message for is displayed if there are no preceding nodes, then the message “Please enter some nodes first” is displayed.
8. In the InsertAfter() Method , in case the user enters a location that is greater than the number of nodes , error message is displayed.
9. In case the number of nodes in the list is zero , then the Count() method displays zero nodes instead of printing an error message.
10. **Explain your program logic, classes and methods used.**

**Program Logic :** The program works as a menu driven program , where the user is provided with an option CreateList , Insert at beginning , Insert at the end, Delete a node, Count the number of nodes or search for the value of a particular node in a inked list.

Each of the methods is implemented by the Algorithms as already described above.

Classes : No classes are used .

**Methods :**

**Built in methods**

free() : to free a block of memory after deletion

malloc(): to allocate memory dynamically.

**User defined methods used :**

struct node\*AddBegin(struct node\*,int val);// to insert a node at the beginning of the list

void Display(struct node\*q);// to display the entire list

void Search(struct node\*);// to search for the node from the value of the element provided at that node.

struct node\* Delete(struct node\*);// to delete any node from the linked list

struct node\* AddEnd(struct node\*);// to add a node towards the end of the linked list

struct node\* InsertAfter(struct node\*);// to insert after any node whose number will be provided by the user

void Count(struct node\*);// to count the total number of nodes.

struct node\* CreateList(struct node\*,int n);// to create he initial list containing n nodes whose value is specified by the user.

1. **Explain the Importance of the approach followed by you**
2. In this approach , we followed a menu driven based system so that any method could be accessed , and memory could be allocated or freed as and when the user requires.
3. Special cases such as deleting the first node , or inserting a new node when there are no nodes , or displaying an error message in case no nodes are left to delete is also shown.
4. The control is also transferred between different functions as and when required.
5. Since memory is allocated dynamically and also freed at runtime , it ensures that there is no wastage or overuse of memory space available.
6. The structs are used for creating nodes , where the data field contains the actual data and the next field points to the next node, which is one of the easier ways of forming a linked list.

**Conclusion:- Through this experiment , we have learnt the following**

1. We have learnt that Linked List is a data structure consisting of nodes ,where each node has two fields , a data field and a next field pointing to the new node.
2. We have learnt about Linked List as an ADT and the various operations that can be performed on it.
3. We have learnt how to implement these various operations such as create a list , searching , deleting with the help of various methods.
4. Finally we have learnt how to implement these methods by the use of a menu driven program , to operate on the linked list , by passing a pointer to the linked list that always points to the first node of the list.