**Experiment No. (2)**

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

**Batch: B3 Roll No.: 16010120135**

**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:**

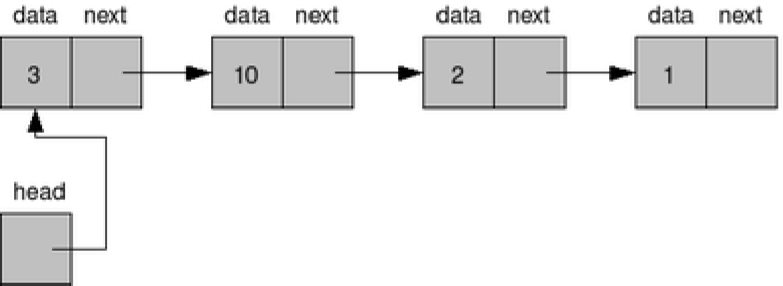
1. Data Structures A Pseudocode Approach with C, Richard F. Gilberg & Behrouz

A. Forouzan, second edition, CENGAGE learning.

1. 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 principal benefit of a linked list over a conventional [array](https://en.wikipedia.org/wiki/Array_data_structure) is that the list elements can be easily inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored [contiguously](https://en.wiktionary.org/wiki/contiguous) in memory or on disk, while restructuring an array at [run-time](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)) is a much more expensive operation. Linked lists allow insertion and removal of nodes at any point in the list, and allow doing so with a constant number of operations by keeping the link previous to the link being added or removed in memory during list traversal.

# Drawbacks of linked list over arrays:

* **Memory Usage**

More memory is required to store elements in linked list as compared to array. Because in linked list each node contains a pointer and it requires extra memory for itself.

# Traversal

Elements or nodes traversal is difficult in linked list. We cannot randomly access any element as we do in array by index. For example, if we want to access a node at position n then we have to traverse all the nodes before it. So, time required to access a node is large.

# Reverse Traversing

In linked list reverse traversing is really difficult. In case of [doubly linked list](https://www.thecrazyprogrammer.com/2015/09/doubly-linked-list-in-c-and-cpp.html) its easier but extra memory is required for back pointer hence wastage of memory.

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

**Creation**

1.first=new node;{create the 1st node of the list pointed by first}; 2.Read(Data(first))

1. NEXT(First)=NULL;
2. Far a First; [point Far to the First]
3. For I=1 to N-1 repeat steps 6 to 10 6.X=new node;

7.Read(Data(X)) 8.NEXT(X)=NULL;

1. NEXT(Far)=X; {connect the nodes}
2. Far=X;[shift the pointer to the last node of the list]

[end of For Loop]

1. END

**Traversing**

1.If First=NULL then {print “List empty” STOP}; 2.count=0;

3.ptr=First; {point ptr to the 1st node} 4.While ptr<> NULL repeat Steps 5 to 6 5.count=count+1;

6.ptr=NEXT(ptr) [shift ptr to the next node] 7.print (‘Number of nodes=’, count)

8.END

**Searching**

1.If first=NULL then{

Print “List empty”; STOP;} 2.ptr=First; [point ptr to the 1st node] 3.while (ptr<>NULL) repeat steps 4 to 5 4.If (DATA (ptr)= ‘X’)

Then {print “item found”; STOP

}

1. ptr=NEXT (ptr); [shift ptr to the next node] [end of while]
2. Print “item not found”; 7.END

**Insertion** 1.X=new node; 2.Read(DATA(X);

1. If (FIRST=NULL) then

{

1. END

}

Else

{

}

First=X; NEXT(X)=NULL;

NEXT(X)=First;

First=X;

**Deletion**

1. If (DATA(list)=’VAL’)then

{

Ptr=LIST; LIST=NEXT(list);

Delete ptr; Stop;

}

Back=list;

Ptr=list;

1. while(ptr<>NULL) repeat step 3 to 5 3.If(DATA(ptr)=’VAL’) then

{

4.back=ptr; 5.ptr=next(ptr)

NEXT(back)=NEXT(ptr); Delete ptr;

Exit;}

[end of while loop] 6.END

**Program source code:**

**DOUBLY LINKED LIST**

#include<stdio.h>

#include<stdlib.h>

*struct* Node  {

*int* data;

*struct* Node\* next;

*struct* Node\* prev;

};

*struct* Node\* head; *// global variable - pointer to head node.*

*//Creates a new Node and returns pointer to it.*

*struct* Node\* GetNewNode(*int* *x*) {

*struct* Node\* newNode

        = (*struct* Node\*)malloc(sizeof(*struct* Node));

    newNode->data = x;

    newNode->prev = NULL;

    newNode->next = NULL;

    return newNode;

}

*//Inserts a Node at head of doubly linked list*

*void* InsertAtHead(*int* *x*) {

*struct* Node\* newNode = GetNewNode(x);

    if(head == NULL) {

        head = newNode;

        return;

    }

    head->prev = newNode;

    newNode->next = head;

    head = newNode;

}

*//Inserts a Node at tail of Doubly linked list*

*void* InsertAtTail(*int* *x*) {

*struct* Node\* temp = head;

*struct* Node\* newNode = GetNewNode(x);

    if(head == NULL) {

        head = newNode;

        return;

    }

    while(temp->next != NULL) temp = temp->next; *// Go To last Node*

    temp->next = newNode;

    newNode->prev = temp;

}

*void* insertPos(*int* *x*, *int* *pos*){ *// 1   2   3   4   5*

*struct* Node\* temp=GetNewNode(x);                  *// 10  20  30  40  50*

*struct* Node\* curr=head;                   *// insert 5 at pos 4 ----> we have to run a loop till we reach pos 3*

    if(pos==1){                              *// start @ 1 ...we need 2 jumps/iterations =>for(i=1;i<=pos-2;i++)*

    temp->next=head;

    head=temp;

    return;

    }

    for (*int* i = 1; i <=pos-2 ; i++)

    {

        curr=curr->next;

        if (curr==NULL){

        printf("position does not exist returning initial LL : \n");

        return;

    }

    }

    if(curr->next==NULL){      *//HANDLES INSERT AT END*

        curr->next=temp;

        temp->prev=curr;

        return;

    }

    temp->next=curr->next; *//temp->next =(curr.next)*

    curr->next->prev=temp; *//(curr.next)->prev = temp*

    curr->next=temp;

    temp->prev=curr;

    return;

}

*void* deletePos(*int* *pos*){

*struct* Node\* curr=head;

if(pos==1){

    free (head);

    head=curr->next;

    return;

}

for (*int* i = 1; i <=pos-2 && curr!=NULL; i++)

{

    curr=curr->next;

}

*struct* Node\* temp=curr->next;

curr->next=temp->next;

temp->next->prev=curr;

free (temp);

}

*//Prints all the elements in linked list in forward traversal order*

*void* Print() {

*struct* Node\* temp = head;

    printf("Forward: ");

    while(temp != NULL) {

        printf("%d ",temp->data);

        temp = temp->next;

    }

    printf("\n");

}

*//Prints all elements in linked list in reverse traversal order.*

*void* ReversePrint() {

*struct* Node\* temp = head;

    if(temp == NULL) return; *// empty list, exit*

*// Going to last Node*

    while(temp->next != NULL) {

        temp = temp->next;

    }

*// Traversing backward using prev pointer*

    printf("Reverse: ");

    while(temp != NULL) {

        printf("%d ",temp->data);

        temp = temp->prev;

    }

    printf("\n");

}

*int* main() {

*//Driver code to test the implementation*

    head = NULL; *// empty list. set head as NULL.*

*// Calling an Insert and printing list both in forward as well as reverse direction.*

    InsertAtTail(2);

    InsertAtTail(4);

    InsertAtTail(6);

    InsertAtTail(8);

    insertPos(10,5);

Print();

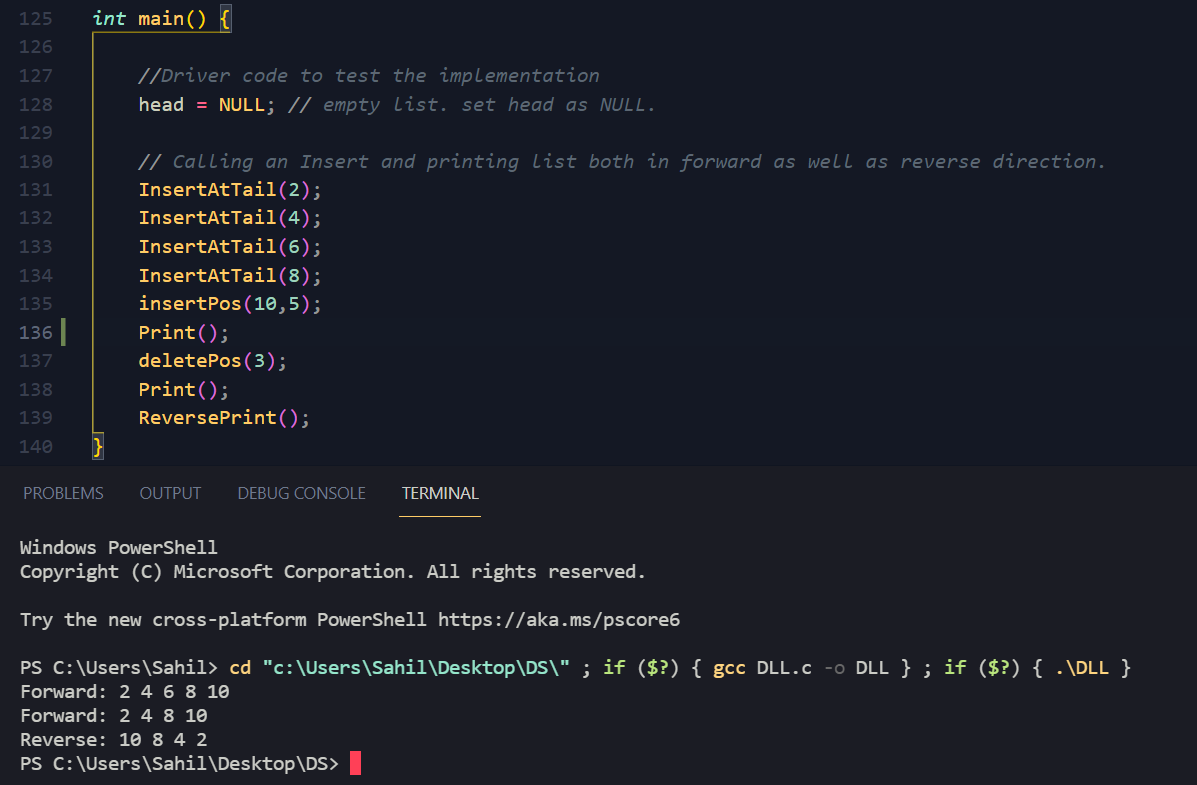
    deletePos(3);

    Print();

    ReversePrint();

}

**Output Screenshots:**



**Implementation Details:**

1. **Enlist all the Steps followed and various options explored**
   * Switch case was used to determine the function that would be performed.
   * The functions were created outside the main function
   * If else loop can also be used but it makes the program lengthy and it is less convenient.
   * GoTo can be used instead of break
   * The program could have also been performed statically i.e instead of asking the user for the number we could have written the numbers to be added or deleted in the code itself
   * User can be asked for the numbers in the create list function

# Explain your program logic, classes and methods used.

User is asked for the function to be performed. Create list is suggested as the changes due to other functions would be easily recognisable

The function is performed using a switch case and displayed using the display function

# Functions:

* InsertatHead-inserts node at head
* insertPos-inserts node at nth pos
* insertTail-inserts node at tail
* deletepos-deletes node from nth pos
* print() – prints DLL
* reverseprint()- reverse prints DLL

# Explain the Importance of the approach followed by you.

* + Using a switch case makes the program more convenient to understand and use.
  + It is a menu driven program so user has the control over which functions have to be used and in what order.
  + Even in most of the function’s user has the control to decide which number has to be added, searched or deleted.
  + After each function display() displays the whole list so user can see the changes after each function.

**Conclusion:- Various functionalities like add, delete, search, create of linked lists is understood and implemented.**