# EXPERIMENT - 13

# Aim: Write a menu driven program to show linked list implementation. It should consist of delete from head , delete from tail and delete from middle.

## Pseudo code

## Define a struct Node:

## Data

## Next

## Define a function insertAtHead that takes head and val as parameters:

## Create a new node n with data set to val

## Set n's Next pointer to head

## Set head to n

## Define a function insertAtTail that takes head and val as parameters:

## Create a new node n with data set to val

## If head is NULL:

## Set head to n

## Return

## Initialize a node pointer temp to head

## While temp's Next pointer is not NULL:

## Set temp to temp's Next pointer

## Set temp's Next pointer to n

## Define a function insertAtMiddle that takes head, val, and position as parameters:

## Create a new node n with data set to val

## If head is NULL or position is 0:

## Set n's Next pointer to head

## Set head to n

## Return

## Initialize a node pointer temp to head

## Initialize a counter count to 0

## While temp is not NULL and count is less than position - 1:

## Set temp to temp's Next pointer

## Increment count

## Set n's Next pointer to temp's Next pointer

## Set temp's Next pointer to n

## Define a function deleteAtHead that takes head as a parameter:

## Create a node pointer todelete and set it to head

## Update head to head's Next pointer

## Delete todelete

## Define a function deleteAtEnd that takes head as a parameter:

## If head is NULL or head's Next pointer is NULL:

## Call deleteAtHead with parameter head

## Return

## Initialize a node pointer temp to head

## While temp's Next pointer's Next pointer is not NULL:

## Set temp to temp's Next pointer

## Create a node pointer todelete and set it to temp's Next pointer

## Set temp's Next pointer to NULL

## Delete todelete

## Define a function deleteAtMiddle that takes head and val as parameters:

## If head is NULL or head's Next pointer is NULL:

## Call deleteAtHead with parameter head

## Return

## Initialize a node pointer temp to head

## While temp's Next pointer's data is not equal to val and temp's Next pointer is not NULL:

## Set temp to temp's Next pointer

## If temp's Next pointer is not NULL:

## Create a node pointer todelete and set it to temp's Next pointer

## Set temp's Next pointer to temp's Next pointer's Next pointer

## Delete todelete

## Define a function display that takes head as a parameter:

## Initialize a node pointer temp to head

## While temp is not NULL:

## Display temp's data

## Display "->"

## Set temp to temp's Next pointer

## Display "NULL"

## Display a new line

## Define the main function:

## Initialize head to NULL

## Call insertAtTail with parameters (head, 2)

## Call insertAtTail with parameters (head, 3)

## Call insertAtTail with parameters (head, 4)

## Call display with parameter (head)

## Call insertAtHead with parameters (head, 1)

## Call display with parameter (head)

## Call insertAtMiddle with parameters (head, 5, 4)

## Call display with parameter (head)

## Call deleteAtMiddle with parameters (head, 3)

## Call display with parameter (head)

## Call deleteAtHead with parameter (head)

## Call display with parameter (head)

## Call deleteAtEnd with parameter (head)

## Call display with parameter (head)

## Source code:

#include <iostream>

using namespace std;

struct node{

    int data ;

    node\* next;

    node (int val){

        data = val;

        next = NULL;

    }

};

void insertathead(node\* &head , int val){      // FUNCTION FOR

    node\* n = new node(val);                   //  INSERT AT

    n->next=head;                              //  BEGINNING

    head=n;

}

void insertattail(node\* &head,int val){

node\* n = new node(val);

if (head==NULL){                               // FUNCTION FOR

    head=n;                                    //  INSERT AT

    return;                                    //    END

}

node\* temp = head;

while ( temp->next!=NULL){

    temp = temp ->next;

}

temp->next=n;

}

void insertatmiddle(node\* &head ,int val, int position){

    int count = 0;

    node\* n = new node(val);

    if (head == NULL){                      // FUNCTION TO

        head=n;                             //  INSERT AT

        return;                             //   MIDDLE

    }

    if (position == 0){

        head=n;

        return;

    }

    node\* temp = head;

    while (temp!=NULL && count < position - 1){

              temp=temp->next;

               count++;

    }

    n->next=temp->next;

    temp->next=n;;

}

// DELETION OF FORMED LINKED LIST

void deleteathead(node\* &head){       // FUNCTION TO

  // DELETE FROM

    head=head->next;                  //    HEAD

}

void deletionatmiddle (node\* &head,int val){    // FUNCTION TO

    if (head ==NULL || head->next==NULL){       // DELETE FROM

        deleteathead(head);                     //   MIDDLE

        return;

    }

    node\* temp = head;

    while(temp->next->data!=val){

        temp=temp->next;

    }

    temp->next=temp->next->next;

}

void deleteatend(node\* &head){          // FUNCTION TO

    if (head==NULL){                    // DELETE FROM

        return;                         //    END

    }

    node\* temp =head;

    while(temp->next->next!=NULL){

        temp=temp->next;

    }

    temp->next=NULL;

}

void display(node\* head){

    node\* temp = head;                        // FUNCTION TO

    while (temp!=NULL){                       //   DISPLAY

        cout<< temp->data <<"->";             // LINKED LIST

        temp = temp->next;

    }

    cout<<"NULL"<< endl;

}

int main(){

    node\* head=NULL;

    insertattail(head,2);

    insertattail(head,3);

    insertattail(head,4);

  // display(head);

    insertathead(head,1);

   // display(head);

    insertatmiddle(head ,5,4);

   // display(head);

    //DELETION

    deletionatmiddle(head,3);

    display(head);

    deleteathead(head);

    display(head);

    deleteatend(head);

    display (head);

}

## Output:

## 1->2->4->5->NULL

## 2->4->5->NULL

## 2->4->NULL

## Learning from experiment

* Linked list operations.
* Insertion and deletion.