**NAME: ADWAIT S PURAO**

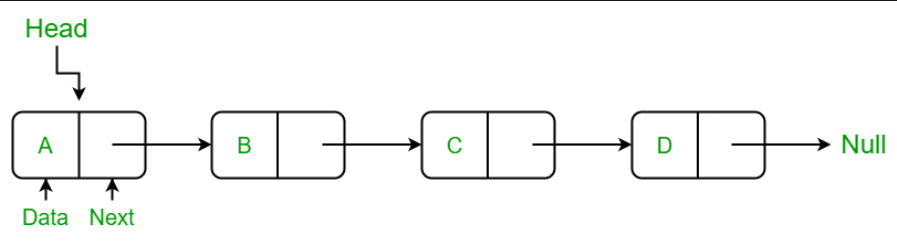
**UID: 2021300101**

**EXP NO. :3**

**AIM: To check whether a string is a Pallindrome or not using linked list**

**THEORY:**

**Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. They include a series of connected nodes. Here, each node stores the data and the address of the next node.**

****

**Why Linked List?**

**The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.**

**Insertion of a new element / Deletion of a existing element in an array of elements is expensive: The room has to be created for the new elements and to create room existing elements have to be shifted but in Linked list if we have the head node then we can traverse to any node through it and insert new node at the required position.**

**Advantages of Linked Lists over arrays:**

**Dynamic Array.**

**Ease of Insertion/Deletion.**

**Drawbacks of Linked Lists:**

**Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a binary search with linked lists efficiently with its default implementation.**

**Extra memory space for a pointer is required with each element of the list.**

**Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.**

**Types of Linked Lists:**

**Simple Linked List – In this type of linked list, one can move or traverse the linked list in only one direction**

**Doubly Linked List – In this type of linked list, one can move or traverse the linked list in both directions (Forward and Backward)**

**Circular Linked List – In this type of linked list, the last node of the linked list contains the link of the first/head node of the linked list in its next pointer and the first/head node contains the link of the last node of the linked list in its prev pointer**

**Representation of Linked Lists:**

**A linked list is represented by a pointer to the first node of the linked list. The first node is called the head of the linked list. If the linked list is empty, then the value of the head points to NULL.**

**Each node in a list consists of at least two parts:**

**A Data Item (we can store integer, strings, or any type of data).**

**Pointer (Or Reference) to the next node (connects one node to another) or An address of another node**

**In C, we can represent a node using structures. Below is an example of a linked list node with integer data.**

**struct Node {**

**int data;**

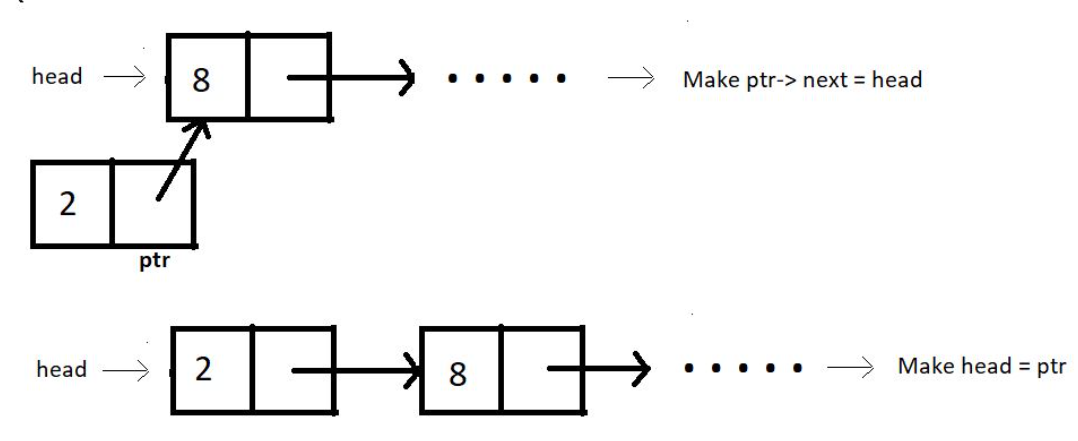
**struct Node\* next;**

**};**

**Linked List Insertion:**

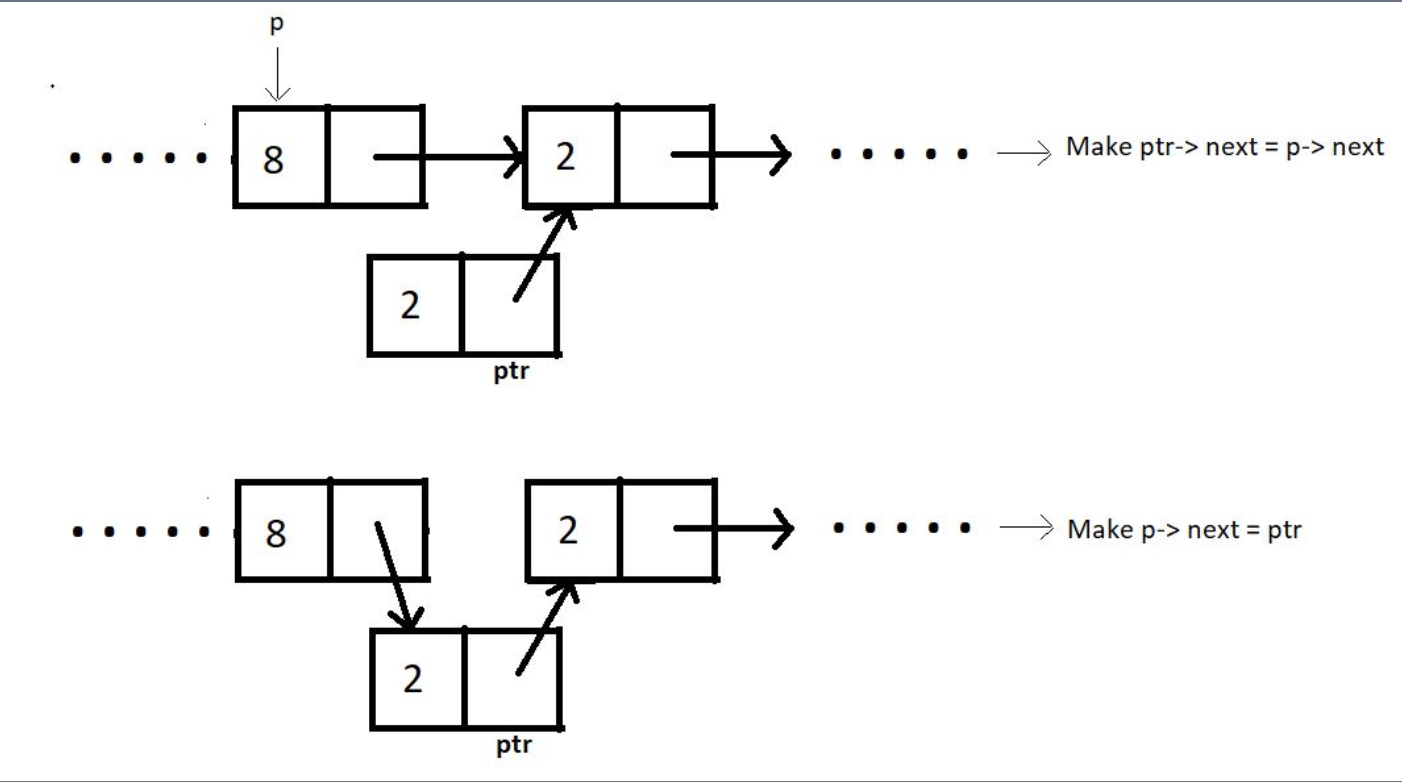
**1.Insertion at first**

**Allocate memory for ptr via malloc. Set data of ptr to data. Set next of ptr to head. Store the value of ptr in head. Return head**

****

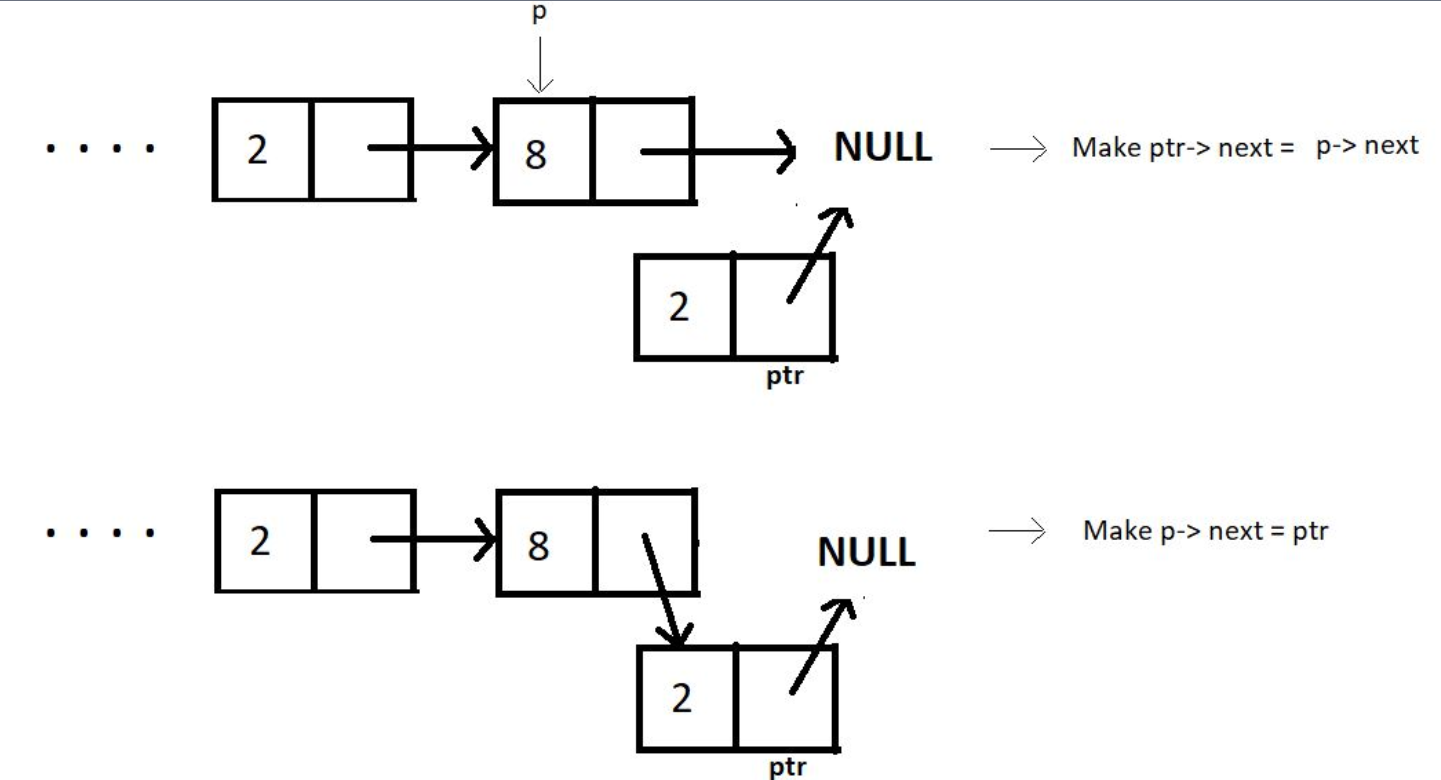
**2.Insertion in between**

**We would bring a temporary pointer p to the node after which we want to enter the node.We would allocate space for a new node ptr which we want to insert then make next of ptr equal to next of p after we would make next of p equal to ptr.**

****

**3.Insertion at end**

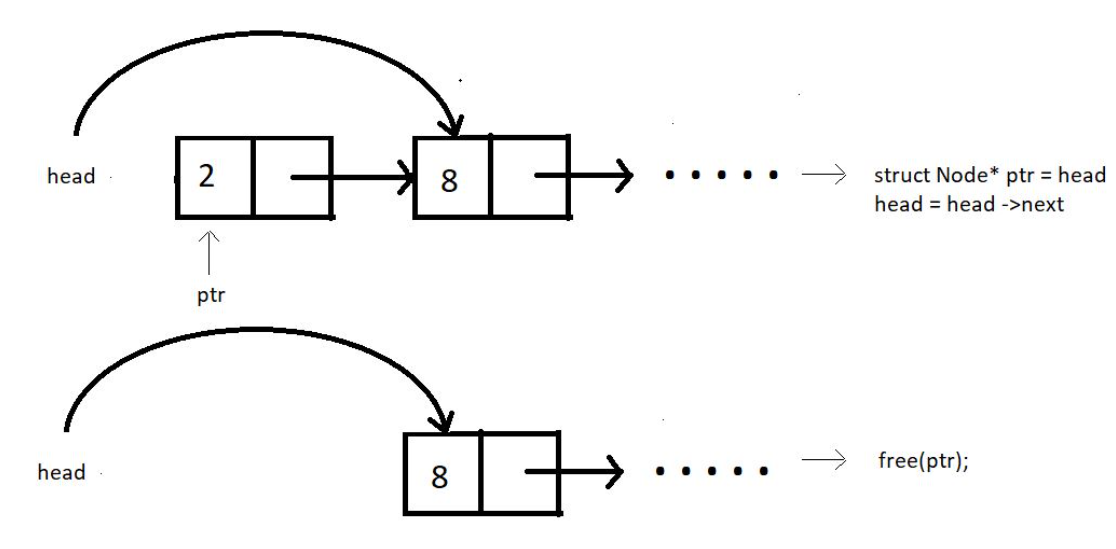
**Allocate memory for ptr via malloc .Set data of ptr to data Set next of ptr to NULL .If head is equal to NULL. Store the value of ptr in head .Return head .Else Set a temporary pointer p to head. Bring it to the last element with the help of a while loop .Set next of p to ptr. Return head**

****

**Deletion in Linked List**

**1.Deletion at beginning**

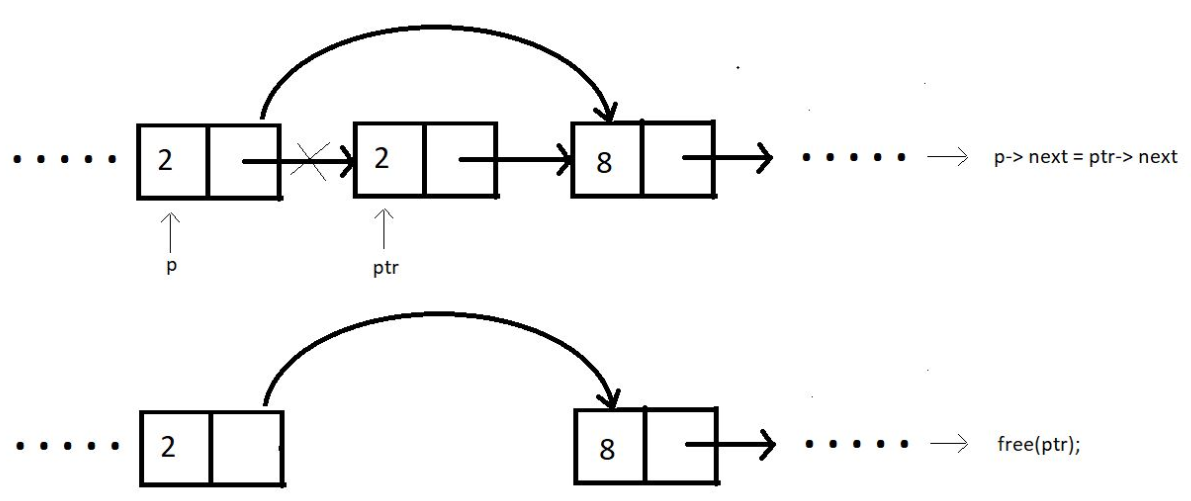
**Store head in ptr.Make head equal to next of head. Free ptr.**

****

**2.Delete at some index**

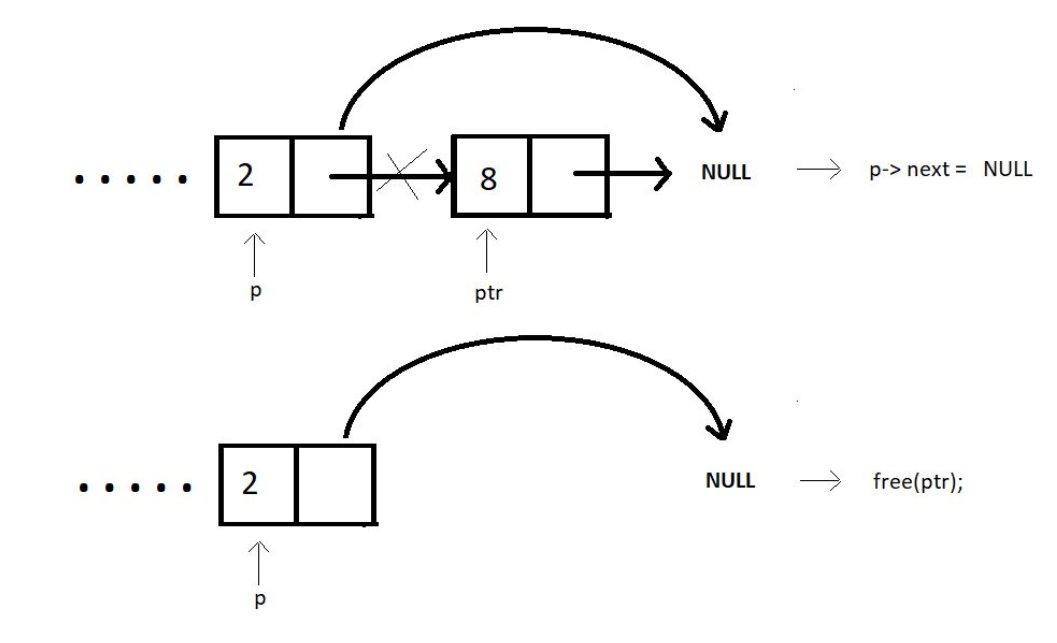
**Assuming index starts from 0, we can delete an element from index i>0 as follows:**

**Bring a temporary pointer p pointing to the node before the element you want to delete in the linked list. Since we want to delete between 2 and 8, we bring pointer p to 2.Assuming ptr points at the element we want to delete. We make pointer p point to the next node after pointer ptr skipping ptr. We can now free the pointer skipped.**

****

**3.Delete at end**

**In order to delete an element at the end of the linked list, we bring a temporary pointer ptr to the last element. And a pointer p to the second last. We make the second last element to point at NULL. And we free the pointer ptr.**

****

**ALGORITHM:**

**1.Struct Node**

**Data members**

**Char data**

**Struct node\*next**

**2)Function struct node\*InsertAtFirst(struct node\*head,int data)**

**1.Allocate memory for ptr via malloc**

**2.Set data of ptr to data**

**3.Set next of ptr to head**

**4.Store the value of ptr in head**

**5.Return head**

**3.Function struct node\*InsertAtEnd(struct node\*head,int data)**

**1.Allocate memory for ptr via malloc**

**2.Set data of ptr to data**

**3.Set next of ptr to NULL**

**4.If head==NULL**

**Store the value of ptr in head**

**Return head**

**5.Else**

**Set a temporary pointer p to head**

**Bring it to the last element with the help of a while loop**

**Set next of p to ptr**

**Return head**

**4.Function void DeleteAtEnd(struct node\*head)**

**1.If head==NULL**

**Print List is empty**

**2.Else**

**Set a temporary pointer p to head**

**Set a temporary pointer q to next of head**

**3.Bring q to the last element and p to the second last element with the help of a while loop**

**4.Set next of p to NULL**

**5.free q**

**5.Function void LinkedListTraversal(struct node\*head)**

**1.Set a temporary pointer ptr to head**

**2.Traverse and print the entire linked list with the help of a while loop and print the elements**

**6.Function void Pallindrome\_check(struct node\*q,struct node\*head,int mid)**

**1.Set flag to 0**

**2.Set temporary pointer q1 to head**

**3.Traverse the linked list from i=0 to i<mid**

**4.If data of q is not equal to data of q1**

**Print String is not a Pallindrome**

**Set flag =1 and return(come out of function)**

**5.Else**

**Set q to next of q**

**Set q1 to next of q1**

**6.If flag==0**

**Print String is a Pallindrome and break**

**7.Function struct node\*InsertReverted(struct node\*head, int mid, int n, struct node\*p2)**

**Struct node\*temp=p2;**

**1.For i=mid to i<n**

**Call Insert at first**

**Set temp to next of temp**

**2.Return head**

**8.Function struct node\*MidPointer(struct node\*head,int mid,int n)**

**1.Set temporary pointer q to head**

**2.For k=0 to k<mid**

**Set q to next of q**

**3.If n%2 is not equal to 0**

**Set q to next of q**

**4.Return q**

**9.Function void Delete\_mid\_last(struct node\*head,int mid)**

**1.For i=0 to i<mid**

**Call function DeleteAtEnd(head)**

**10.Main function**

**1.Initialize data members n and i**

**2.Take input of the length of the string**

**3.Take input of the string**

**4.For i=0 to i<n**

**Call function InsertAtEnd(head,arr[i])**

**5.Call Function LinkedListTraversal(head)**

**6.Initialize mid to n/2**

**7.Set ind to 0 and pointer p1 to head**

**8.Bring p1 to element just before the mid with the help of a while loop**

**Set p2 to next of p1**

**9.Call function InsertReverted(head,mid,n,p2)**

**10.Call function Delete\_mid\_last(head,mid)**

**11.Set the next of p1 to NULL to break the list into half**

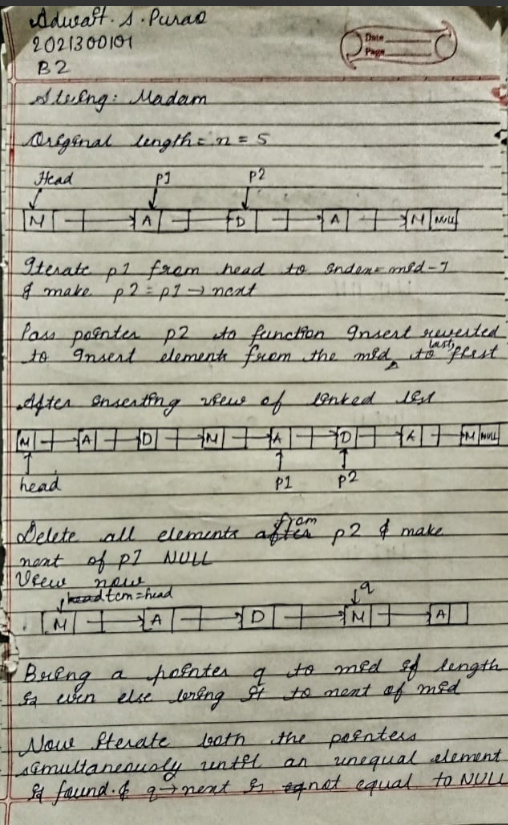
**12.Set a pointer q to mid with the help of MidPointer(head,mid,n)**

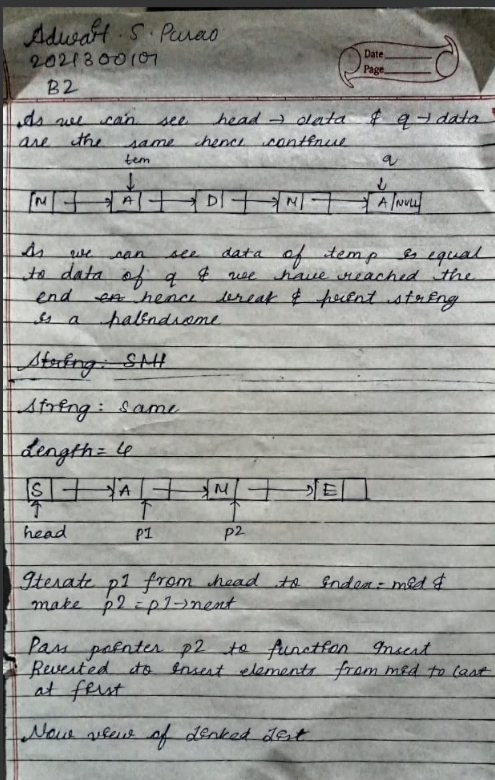
**13.Call the function LinkedListTraversal(head)**

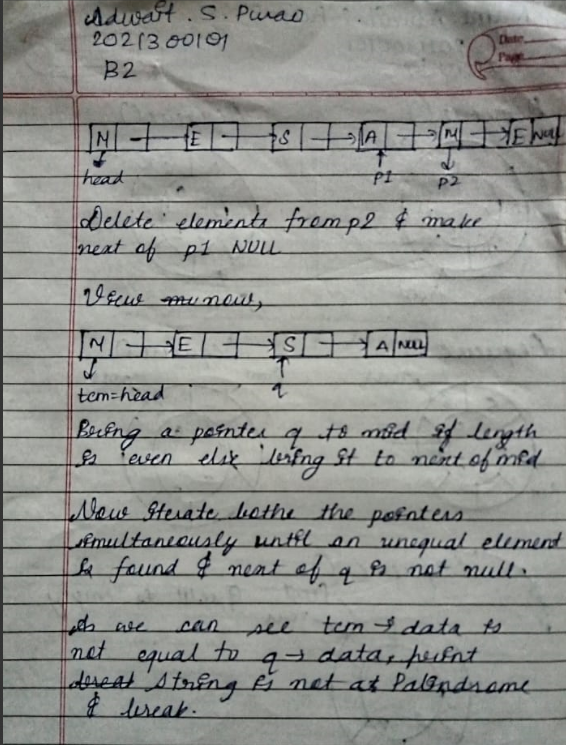
**14.Call function Pallindrome\_check(q,head,mid)**

**15.Program ends**

**PROBLEM SOLVING ON CONCEPT:**

****

****

****

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node{

char data;

struct node\*next;

};

struct node\*InsertAtFirst(struct node\*head,int data){

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

    ptr->data=data;

    ptr->next=head;

    head=ptr;

    return head;

}

struct node\*InsertAtEnd(struct node\*head,int data){

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

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL){

        head=ptr;

        return head;

    }

    else{

    struct node\*p=head;

    while(p->next!=NULL){

        p=p->next;

    }

    p->next=ptr;

    return head;

    }

}

void DeleteAtEnd(struct node\*head){

    if(head==NULL)

        printf("List is empty!\n");

    else{

        struct node\*p=head;

        struct node\*q=head->next;

        while(q->next!=NULL){

            q=q->next;

            p=p->next;

        }

        p->next=NULL;

        free(q);

    }

}

void LinkedListTraversal(struct node\*head){

    struct node\*ptr=head;

    printf("Traversal of entire linked list\n");

    while(ptr!=NULL){

        printf("%c ",ptr->data);

        ptr=ptr->next;

    }

}

void Pallindrome\_check(struct node\*q,struct node\*head,int mid){

    int flag=0;

    printf("Final Answer:\n");

    struct node\*q1=head;

    for(int i=0;i<mid;i++){

        if(q1->data!=q->data){

            printf("\nBreaking point\n");

            printf("First part:%c , Second part=%c\n",q1->data,q->data);

            printf("The string is not a Pallindrome\n");

            flag=1;

            return;

        }

        else{

            q1=q1->next;

            q=q->next;

        }

    }

    if(flag==0){

        printf("The string is a Pallindrome\n");

        return;

    }

}

struct node\*InsertReverted(struct node\*head,int mid,int n,struct node\*p2){

    struct node\*temp=p2;

        for(int i=mid;i<n;i++){

            head=InsertAtFirst(head,temp->data);

            temp=temp->next;

        }

    return head;

}

struct node\*MidPointer(struct node\*head,int mid,int n){

    struct node\*q=head;

    for(int k=0;k<mid;k++)

        q=q->next;      //q has reached the mid

    if(n%2!=0){

        q=q->next;

    }

    return q;

}

void Delete\_mid\_last(struct node\*head,int mid){

    for(int i=0;i<mid;i++){

        DeleteAtEnd(head);

    }

}

int main(){

    int n,i;

    printf("Enter the length of the string:\n");

    scanf("%d",&n);

    struct node\*head=NULL;

    char arr[n];

    printf("Enter a string\n");

    scanf("%s",arr);

    //Inserting elements into the linked list

    for(i=0;i<n;i++)

       head=InsertAtEnd(head,arr[i]);

    //view of linked list after inserting elements

    LinkedListTraversal(head);

    printf("\n");

    int mid=n/2;

    int ind=0;

    struct node\*p1=head;

        while(ind!=mid-1){

            p1=p1->next;

            ind++;

        }

    //Pointer to add the last half of the list to first

    struct node\*p2=p1->next;

    //Inserting elements from mid to last at the beginning in reverse order

    head=InsertReverted(head,mid,n,p2);

    //Deleting the elements from mid to last which are not required

    Delete\_mid\_last(head,mid);

    //For breaking the list into half

    p1->next=NULL;

    //Bringing a pointer to the middlemost element

    struct node\*q=MidPointer(head,mid,n);

    printf("\n");

    printf("After entering the last half first:\n");

    LinkedListTraversal(head);

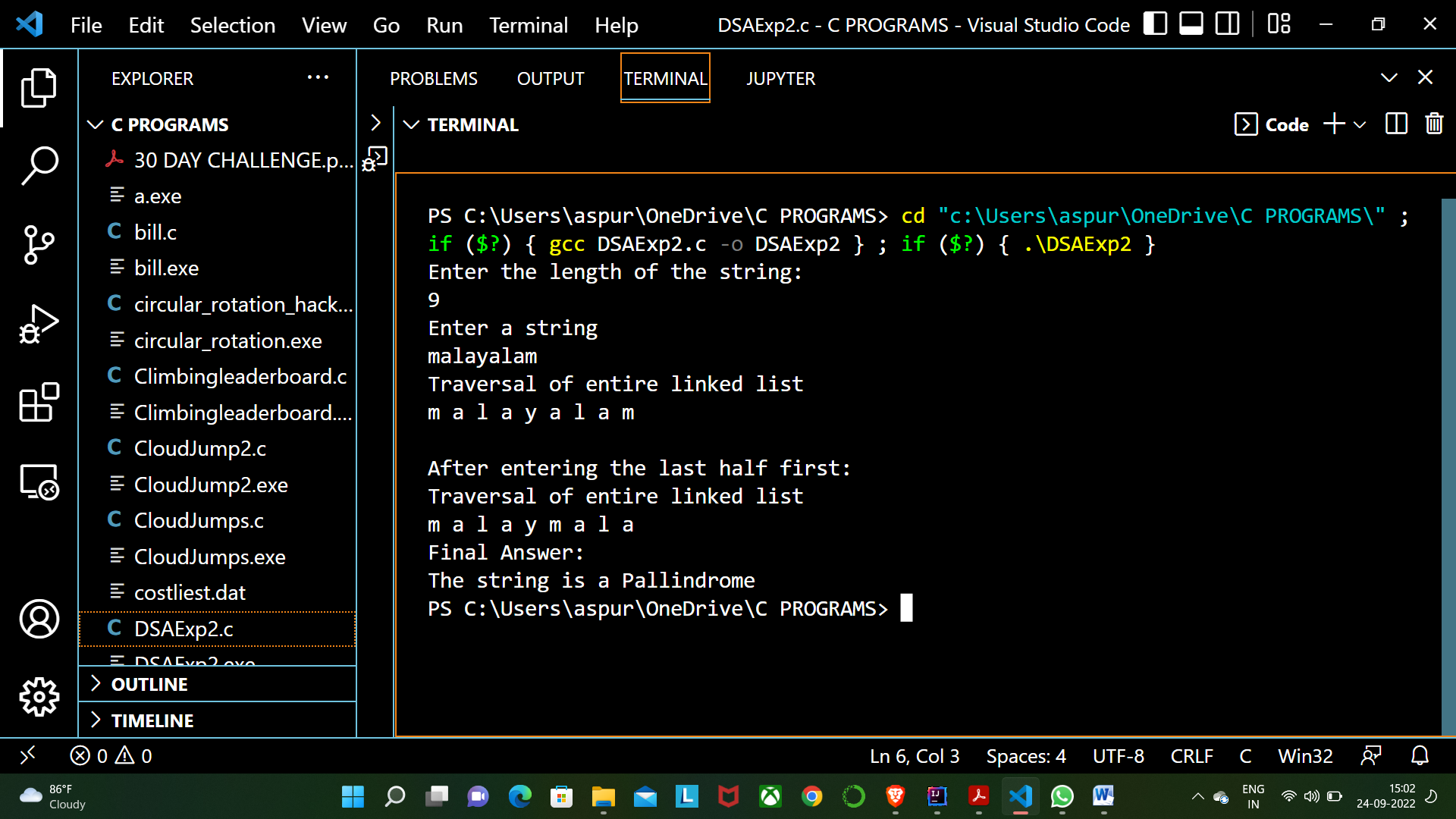
    printf("\n");

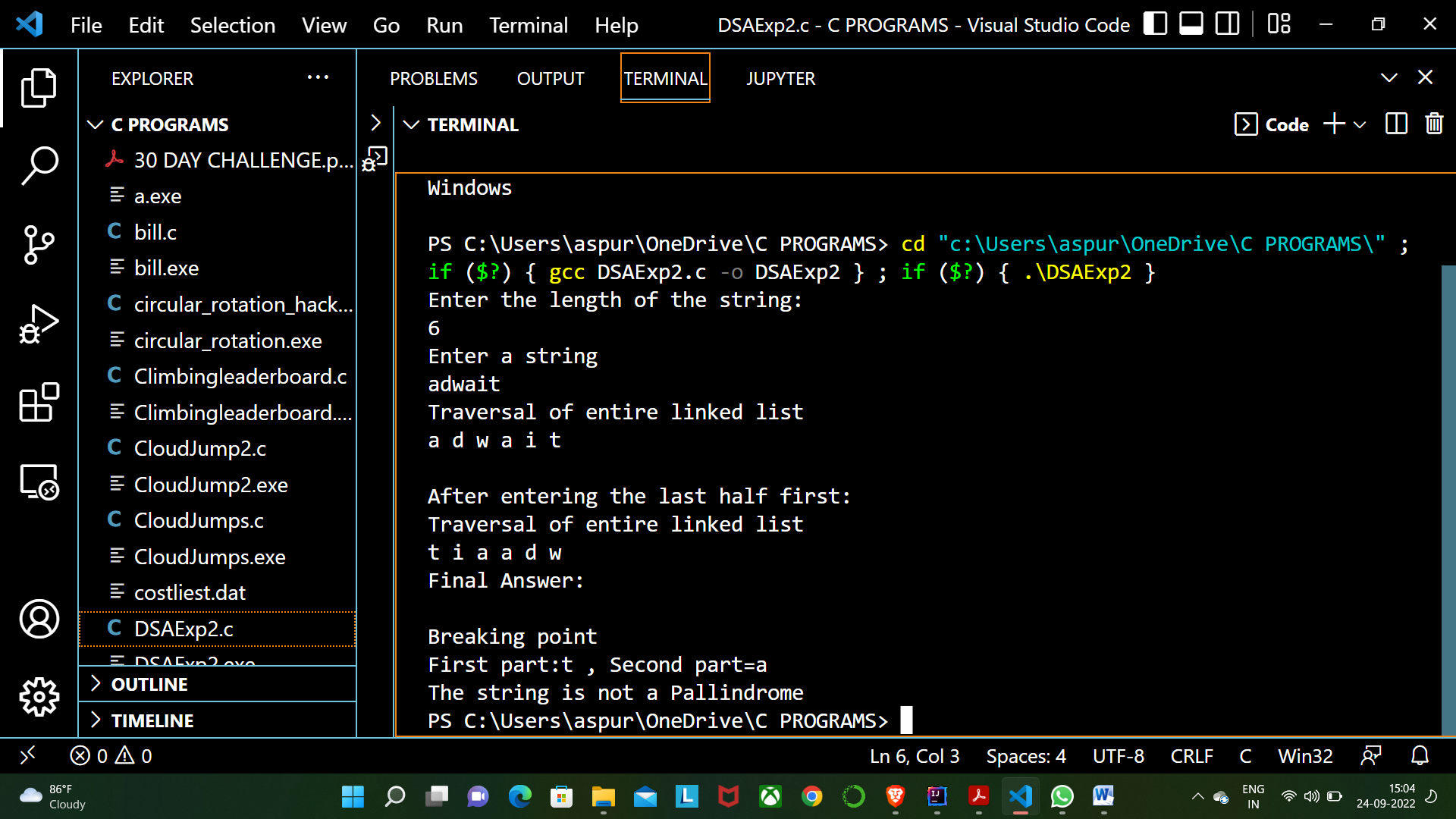
    Pallindrome\_check(q,head,mid);

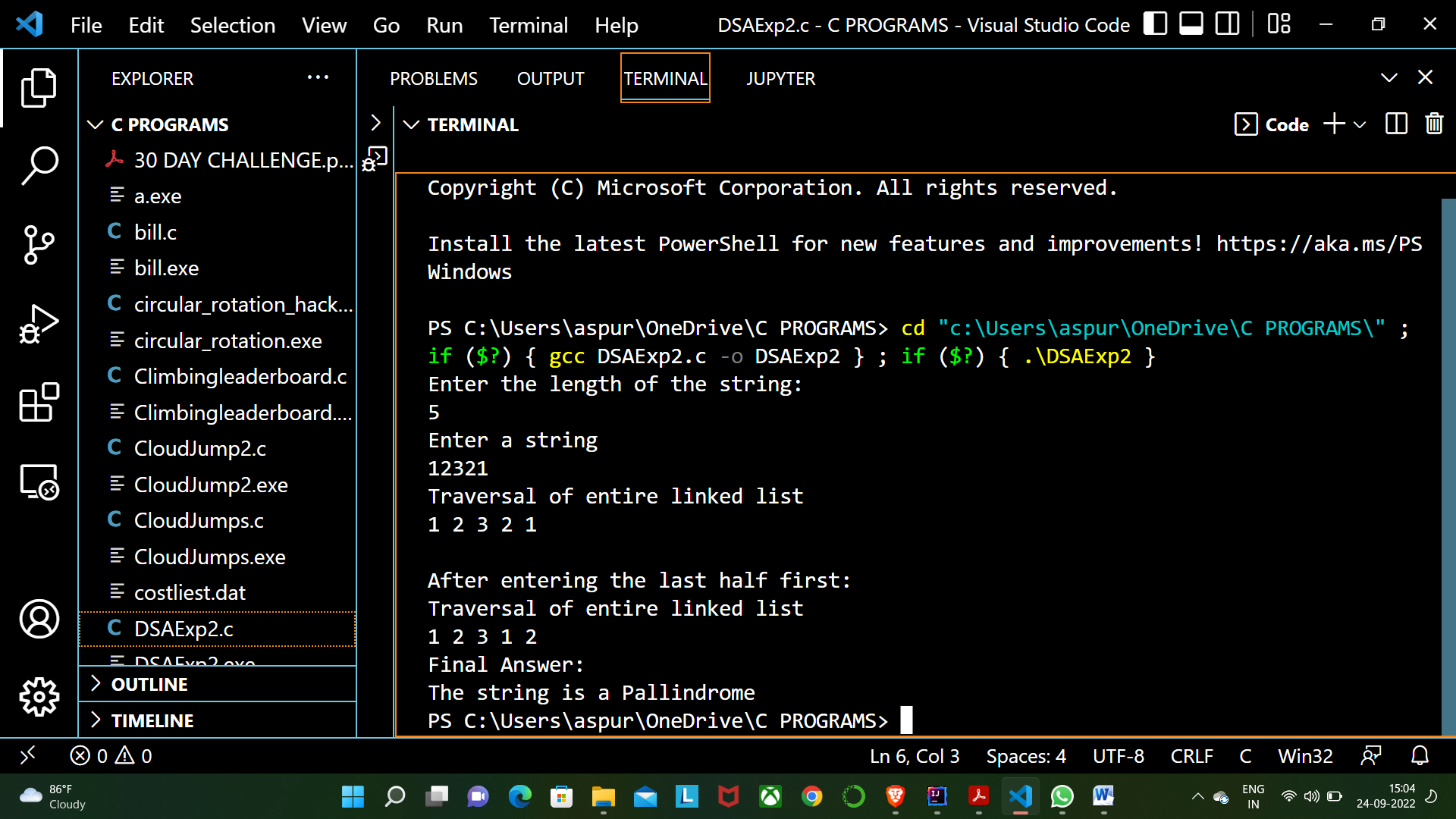
return 0;

}

**OUTPUT SCREENSHOT:**







**CONCLUSION:**

In the help of this experiment we learnt about the Linked Lists. We understood the internal structure of a linked list and the links in it. We learnt that a linked list ends with a NULL. We learnt about the malloc function used to allocate memory. We performed various functions on linked list like Insertion at first, Delete at end, Insert at end and Traversal of linked list.

We combined these all functions and checked whether a linked list is a Palindrome or not.