**NAME: ADWAIT S PURAO**

**UID: 2021300101**

**EXP NO. :4**

**AIM: To perform Insert and Delete operations on Doubly Linked List**

**THEORY:**

**Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.**

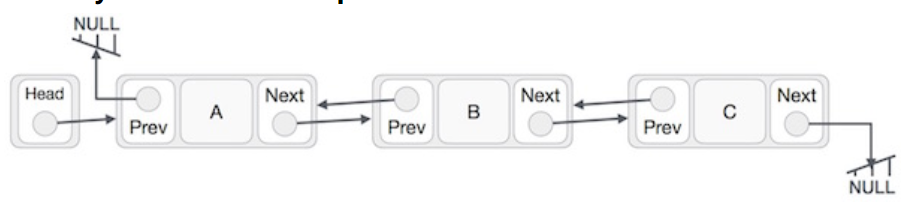
**Link − Each link of a linked list can store a data called an element.**

**Next − Each link of a linked list contains a link to the next link called Next.**

**Prev − Each link of a linked list contains a link to the previous link called Prev.**

**LinkedList − A Linked List contains the connection link to the first link called First and to the last link called Last.**

**Doubly Linked List Representation**

****

**As per the above illustration, following are the important points to be considered.**

**Doubly Linked List contains a link element called first and last.**

**Each link carries a data field(s) and two link fields called next and prev.**

**Each link is linked with its next link using its next link.**

**Each link is linked with its previous link using its previous link.**

**The last link carries a link as null to mark the end of the list.**

**Advantages of DLL over the singly linked list:**

**A DLL can be traversed in both forward and backward directions.**

**The delete operation in DLL is more efficient if a pointer to the node to be deleted is given.**

**We can quickly insert a new node before a given node.**

**In a singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.**

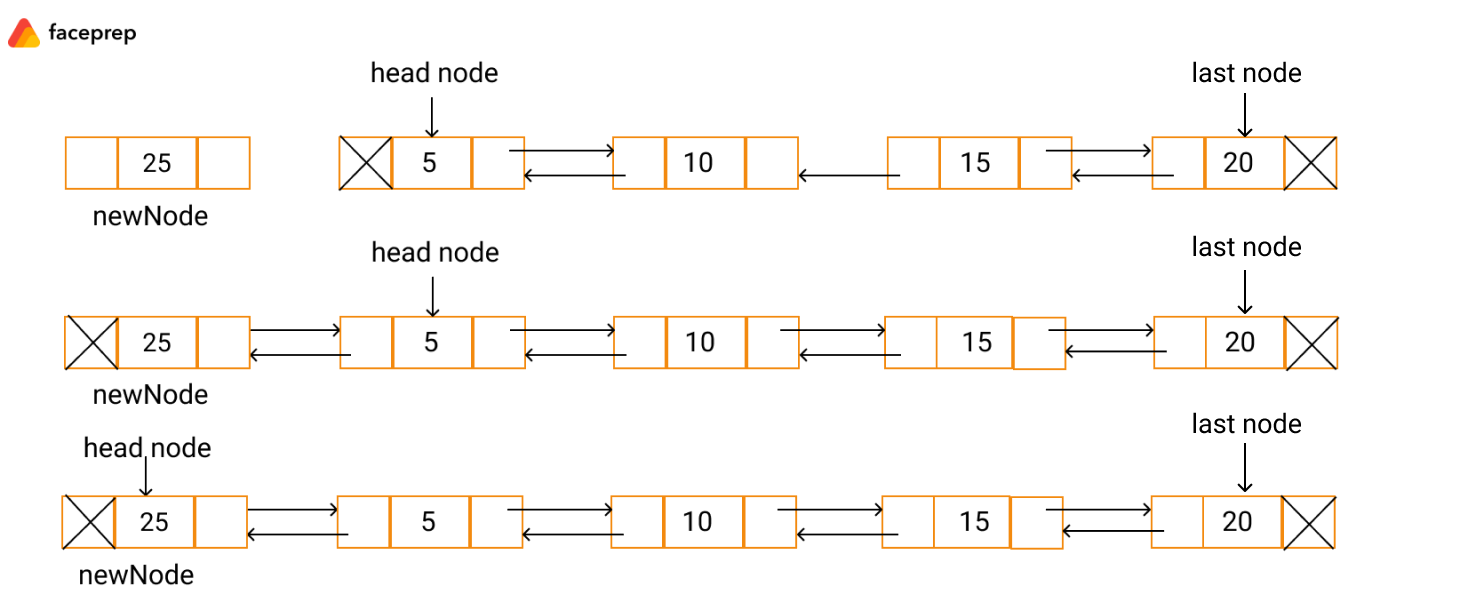
**Disadvantages of DLL over the singly linked list:**

**Every node of DLL Requires extra space for a previous pointer. It is possible to implement DLL with a single pointer though (See this and this).**

**All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with the next pointers. For example in the following functions for insertions at different positions, we need 1 or 2 extra steps to set the previous pointer.**

**Insertion in DLL:**

**1) Insert a node at the front:**



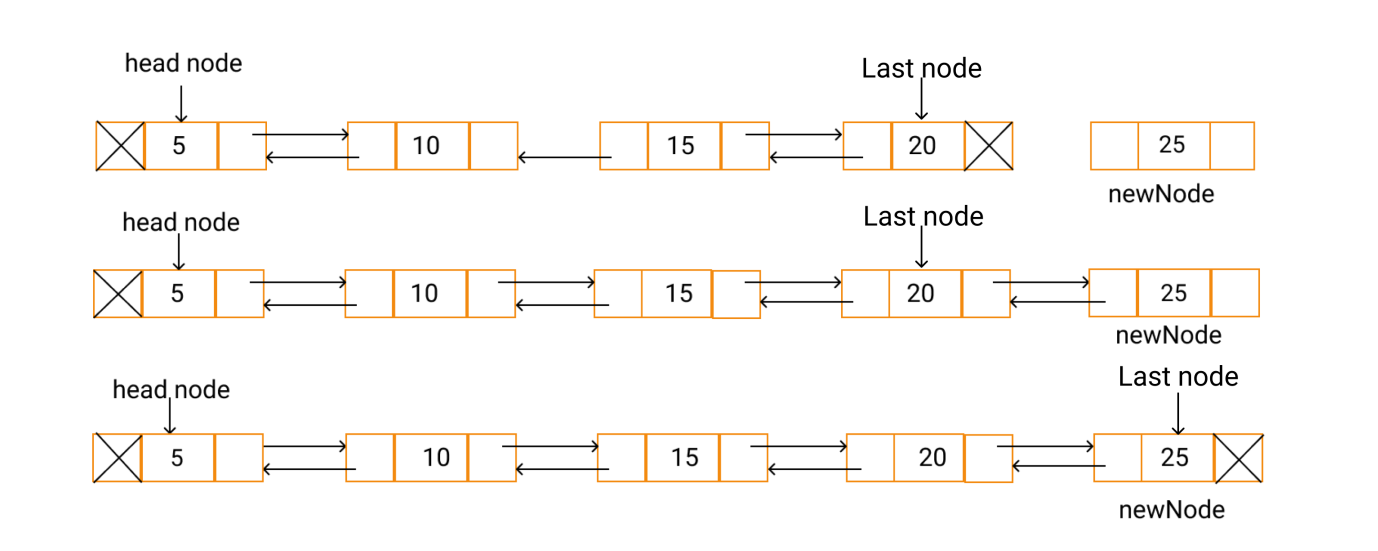
**Let us assume a newNode as shown above. The newNode with data = 25 has to be inserted at the beginning of the list.**

**The next pointer of the newNode is referenced to the head node and its previous pointer is referenced to NULL.**

**The previous pointer of the head node is referenced to the newNode.**

**The newNode is then made as the head node.**

**2) Insert a node at the End:**



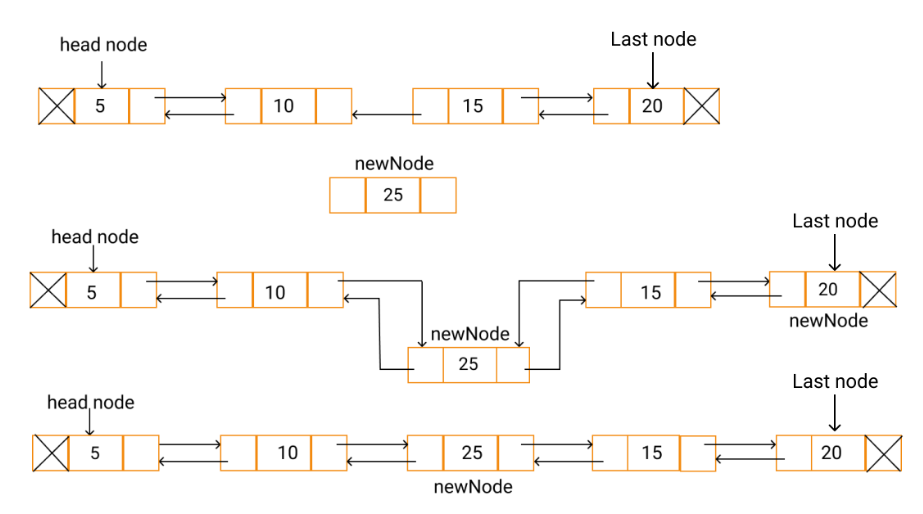
**Now, let us assume a newNode as shown above. The newNode with data = 25 has to be inserted at the end of the linked list.**

**Make the next pointer of the last node to point to the newNode .**

**The next pointer of the newNode is referenced to NULL and its prev pointer is made to point to the last node.**

**Then, the newNode is made as the last node.**

**3) Insert a node before a given node:**

****

**Bring a pointer to the node(element) before which you want to insert the node**

**Make the previous of newNode to the previous of p**

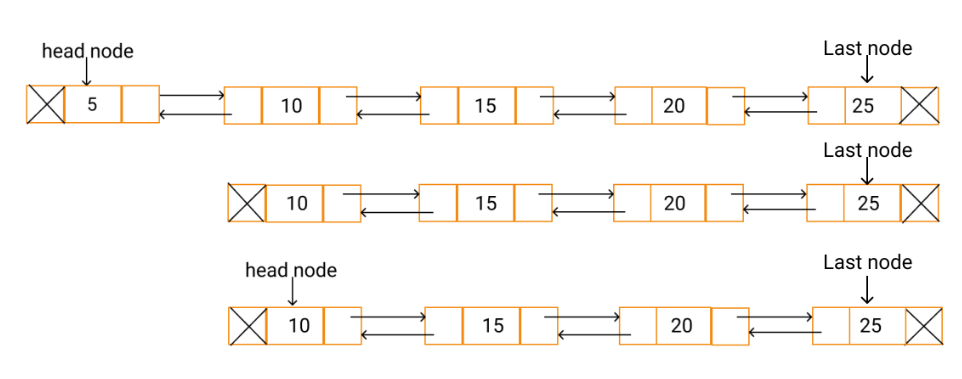
**Make the next of previous of p to newNode**

**Make the next of newNode to p**

**Set the previous of p to newNode**

**Deletion in a DLL:**

**4)Deletion at start**

****

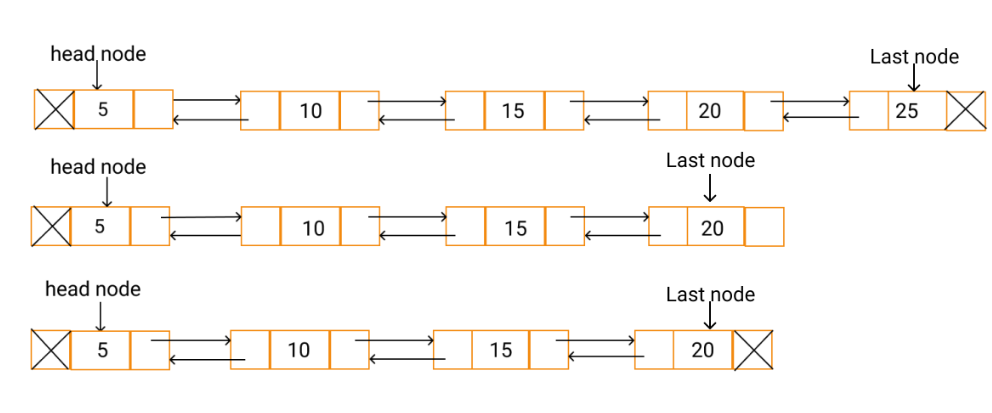
**Copy the head node in some temporary node.**

**Make the second node as the head node.**

**The prev pointer of the head node is referenced to NULL.**

**Delete the temporary node.**

**5) Deletion at end**

****

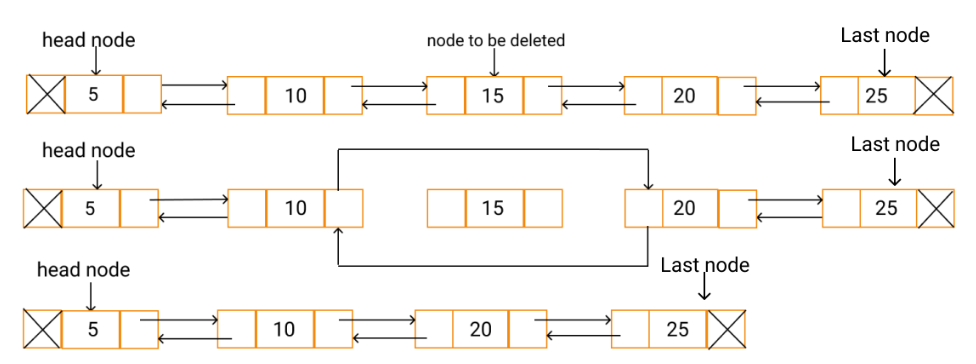
**Copy the last node to a temporary node.**

**Shift the last node to the second last position.**

**Make the last node's next pointer as NULL.**

**Delete the temporary node.**

**6) Delete a node with a given value**

****

**Bring a pointer p to the element which you want to delete**

**Set a temporary pointer r to previous of p**

**Make next of r to next of p**

**Make the previous of next of p to the previous of p**

**Free p**

**Return head**

**ALGORITHM:**

**1)Struct node**

**Data members**

**struct node\*prev**

**int data**

**struct node\*next**

**2)Function struct node\*DeleteAtEnd(struct node\*head)**

**If head==NULL**

**Print Can’t delete because list is empty**

**Return null**

**Else if head->next==null**

**Head=null**

**Free head**

**Print list is empty now**

**Return null**

**Else**

**Bring a pointer to the last node with the help of a while loop**

**Set struct node\*q to p**

**Make p=p->prev**

**Free q**

**Return head**

**3) Function struct node\*DeleteAtFront(struct node\*head)**

**If head==null**

**Print Can’t delete because the list is empty**

**Return null**

**Else if head->next==null and head->prev==null**

**Free head**

**Print list is empty now**

**Return null**

**Else**

**Set struct node\*p =head**

**Set head to next of head**

**Make head->prev==null**

**Free p**

**Return head**

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

**Allocate memory for a pointer ptr**

**Set ptr->data =data**

**If head == null**

**Make ptr->prev=null**

**Make ptr->next=null**

**Store ptr in head**

**Return head**

**Else**

**Set struct node\*p = head**

**Bring a pointer p to the last node**

**Make ptr->prev=p**

**Make p->next =ptr**

**Make ptr->next =null**

**Return head**

**5) Function struct node\*InsertAtFront(struct node\*head,int data)**

**Allocate memory for ptr**

**Set ptr->data=data**

**If head==null**

**Make ptr->prev=null**

**Make ptr->next=null**

**Store ptr in head**

**Return head**

**Else**

**Make ptr->next=head**

**Make head->prev = ptr**

**Store head in ptr**

**Return head**

**6) Function struct node\*InsertBeforePosition(struct node\*head,int checkData,int insData)**

**Allocate memory for ptr**

**Set struct node\*p = head**

**Store insData in ptr->data**

**If head==null**

**Print List is empty so can’t insert anything**

**Bring a pointer to the data before which you want to insert**

**Set struct node\*q=p->prev**

**If p->next ==null and p->data is not equal to checkData**

**Print No such element exists**

**Return head**

**Else if p->prev==null**

**Set ptr->prev=null**

**Set ptr->next=head**

**Set head->prev=ptr**

**Return head**

**Else**

**Set ptr->prev=p->prev**

**Set q->next=ptr**

**Set ptr->next=p**

**Set p->prev=ptr**

**Return head**

**7)Function void Display**

**Set struct node\*temp=head**

**If head==null**

**Print List is empty**

**Else**

**Print Traversal of entire linked list**

**Traverse the list while temp is not equal to null**

**Print temp->data**

**Set temp=temp->next**

**8)Function struct node\*DeleteAtPosition(struct node\*head,int checkData)**

**Set struct node\*p =head**

**If head==null**

**Print No such element exists**

**Return null**

**Traverse the list to bring pointer p to checkData**

**If p->next==null and p->data is not equal to checkData**

**Print no such element exists**

**Return head**

**Else If p->next==null and p->prev==null**

**Free p**

**Print List is empty now**

**Return null**

**Else if p->next==null**

**Call function DeleteAtEnd(head)**

**Else If p->prev==null**

**Call function DeleteAtFront(head)**

**Return head**

**Else If p is not equal to null**

**Set struct node\*r= p->prev**

**Make r->next=p->next**

**Make p->next->prev=p->prev**

**Free p**

**Return head**

**9) Main Function**

**Set struct node\*head=null**

**Initialize flag to 0**

**Do while (flag is not equal to 1)**

**Print the menu**

**Take user input ch**

**Switch (ch)**

**Case 1:**

**Take input ele**

**Call function InsertAtFront(head,ele)**

**Call function Display(head)**

**Case 2:**

**Take input ele**

**Call function InsertAtEnd(head,ele)**

**Call function Display(head)**

**Case 3:**

**Take input the element you want to insert(ele)**

**Take input the element before which you want to insert(check)**

**Call function InsertBeforePosition(head,check,ele)**

**Call function Display(head)**

**Case 4:**

**Call function DeleteAtFront(head)**

**Call function Display(head)**

**Case 5:**

**Call function DeleteAtEnd(head)**

**Call function Display**

**Case 6:**

**Take input of the element which you want to delete(check)**

**Call function DeleteAtPosition(head,check)**

**Call function Display**

**Case 7:**

**Set flag=1**

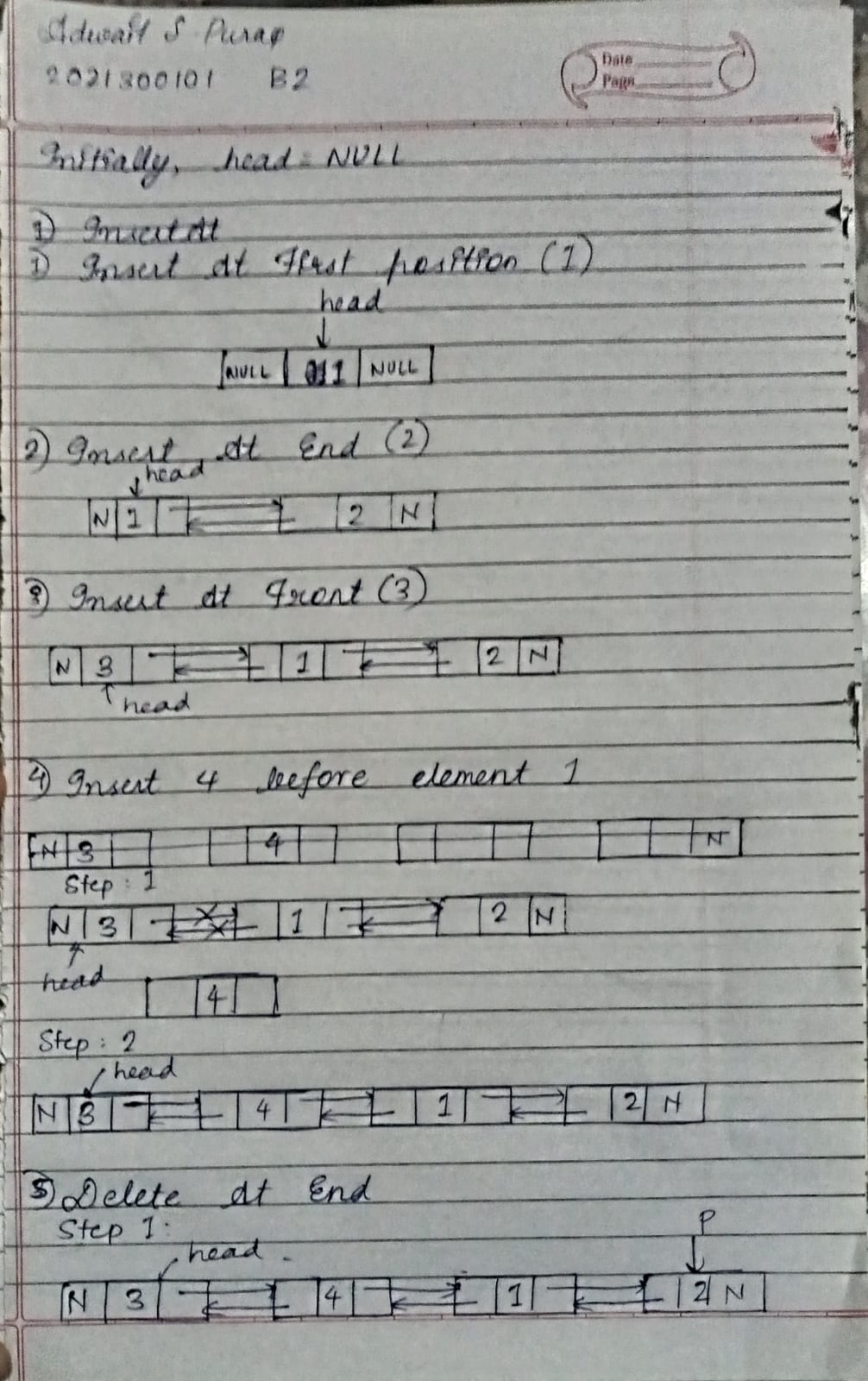
**Print Program finished**

**Break**

**Default:**

**Print Invalid choice**

**PROBLEM SOLVING ON CONCEPT:**

****

****

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node{

    struct node\*prev;

    int data;

    struct node\*next;

};

struct node\*DeleteAtEnd(struct node\*head){

    struct node\*p=head;

    if(head==NULL){

        printf("Can't delete because list is empty\n");

        return NULL;

    }

    else if(head->next==NULL){

        head=NULL;

        free(head);

        printf("List is empty now\n");

        return NULL;

    }

    else{

         while(p->next!=NULL){

            p=p->next;

        }

    struct node\*q=p;

    p=p->prev;

    if(p!=NULL)

        p->next=NULL;

    free(q);

    return head;

    }

}

struct node\*DeleteAtFront(struct node\*head){

    if(head==NULL){

        printf("Can't delete because the list is empty\n");

        return NULL;

    }

    else if(head->next==NULL && head->prev==NULL){

        free(head);

        printf("List is empty now\n");

        return NULL;

    }

    else{

    struct node\*p=head;

    head=head->next;

    head->prev=NULL;

    free(p);

    return head;

    }

}

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

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

    ptr->data=data;

    if(head==NULL){

        ptr->prev=NULL;

        ptr->next=NULL;

        head=ptr;

        return head;

    }

    else{

        struct node\*p=head;

        while(p->next!=NULL){

        p=p->next;

        }

        ptr->prev=p;

        p->next=ptr;

        ptr->next=NULL;

        return head;

    }

}

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

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

    //struct node\*p=head;

    ptr->data=data;

    if(head==NULL){

        ptr->prev=NULL;

        ptr->next=NULL;

        head=ptr;

        return head;

    }

    else{

    ptr->next=head;

    head->prev=ptr;

    head=ptr;

    return head;

    }

}

struct node\*InsertBeforePosition(struct node\*head,int checkData,int insData){

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

    struct node\*p=head;

    ptr->data=insData;

    while(p->data!=checkData && p->next!=NULL){

        p=p->next;

    }

    struct node\*q=p->prev;

    if(p->next==NULL && p->data!=checkData){

        printf("No such element exists!\n");

        return head;

    }

    else if(p->prev==NULL){

        ptr->prev=NULL;

        ptr->next=head;

        head->prev=ptr;

        head=ptr;

        return head;

    }

    else{

        ptr->prev=p->prev;

        q->next=ptr;

        ptr->next=p;

        p->prev=ptr;

        return head;

    }

}

void Display(struct node\*head){

    struct node\*temp=head;

    if(head==NULL)

        printf("List is empty!\n");

    else{

    printf("Traversal of entire Linked List\n");

    while(temp!=NULL){

        printf("%d ",temp->data);

        temp=temp->next;

        }

    }

}

struct node\*DeleteAtPosition(struct node\*head,int checkData){

    struct node\*p=head;

    while(p->data!=checkData){

        p=p->next;

    }

    if(p->next==NULL && p->data!=checkData){

        printf("No such element exists!\n");

        return head;

    }

    else if(p->next==NULL && p->prev==NULL){

        free(p);

        printf("List is empty now\n");

        return NULL;

    }

    else if(p->next==NULL){

        head=DeleteAtEnd(head);

        return head;

    }

    else if(p->prev==NULL){

        head=DeleteAtFront(head);

        return head;

    }

    else if(p!=NULL){

        struct node\*r=p->prev;

        r->next=p->next;

        p->next->prev=p->prev;

        free(p);

        return head;

    }

}

int main(){

    struct node\*head=NULL;

    int flag=0;

    do {

    int ch;

    printf("\n\nEnter your choice:\n");

    printf("1)Insert At Front\n2)Insert At End\n3)Insert with a given value\n");

    printf("4)Delete At Front\n5)Delete At End\n6)Delete with a given element\n7)Exit\n");

    scanf("%d",&ch);

    switch(ch){

        case 1:

        {

            int ele;

            printf("Enter the element you want to insert:\n");

            scanf("%d",&ele);

            head=InsertAtFront(head,ele);

            printf("Current Status:\n");

            Display(head);

            break;

        }

        case 2:

        {

            int ele;

            printf("Enter the element you want to insert:\n");

            scanf("%d",&ele);

            head=InsertAtEnd(head,ele);

            printf("Current Status:\n");

            Display(head);

            break;

        }

        case 3:

        {

            int ele,check;

            printf("Enter the element you want to insert:\n");

            scanf("%d",&ele);

            printf("Enter the element before which you want to insert:\n");

            scanf("%d",&check);

            head=InsertBeforePosition(head,check,ele);

            printf("Current Status:\n");

            Display(head);

            break;

        }

        case 4:

        {

            head=DeleteAtFront(head);

            printf("Current Status:\n");

            Display(head);

            break;

        }

        case 5:

        {

            head=DeleteAtEnd(head);

            printf("Current Status:\n");

            Display(head);

            break;

        }

        case 6:

        {

            int check;

            printf("Enter the element which you want to delete:\n");

            scanf("%d",&check);

            head=DeleteAtPosition(head,check);

            printf("Current Status:\n");

            Display(head);

            break;

        }

        case 7:

        {

            flag=1;

            printf("Program finished\n");

            break;

        }

        default:

        {

            printf("Invalid choice!\n");

            break;

        }

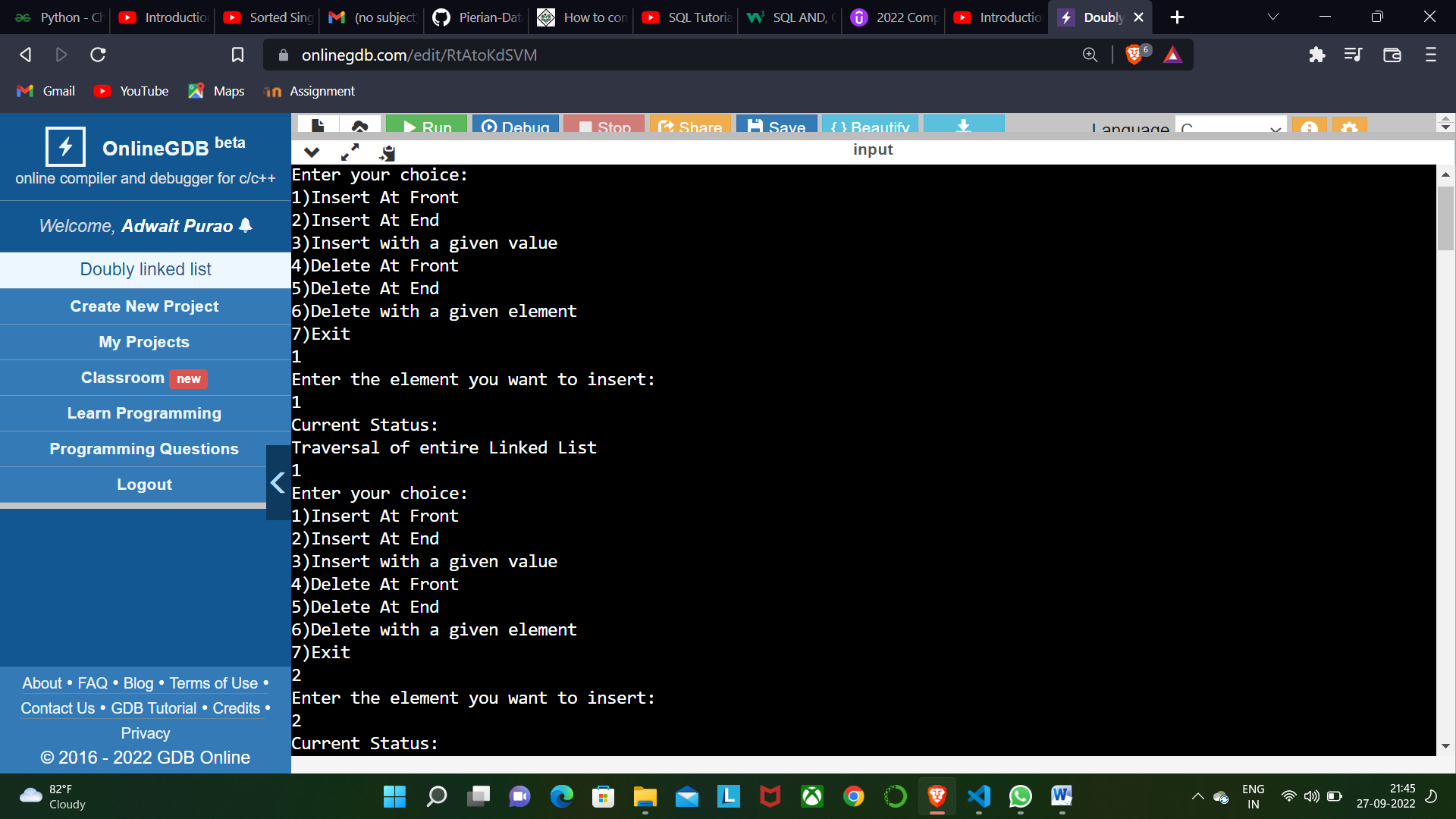
    }

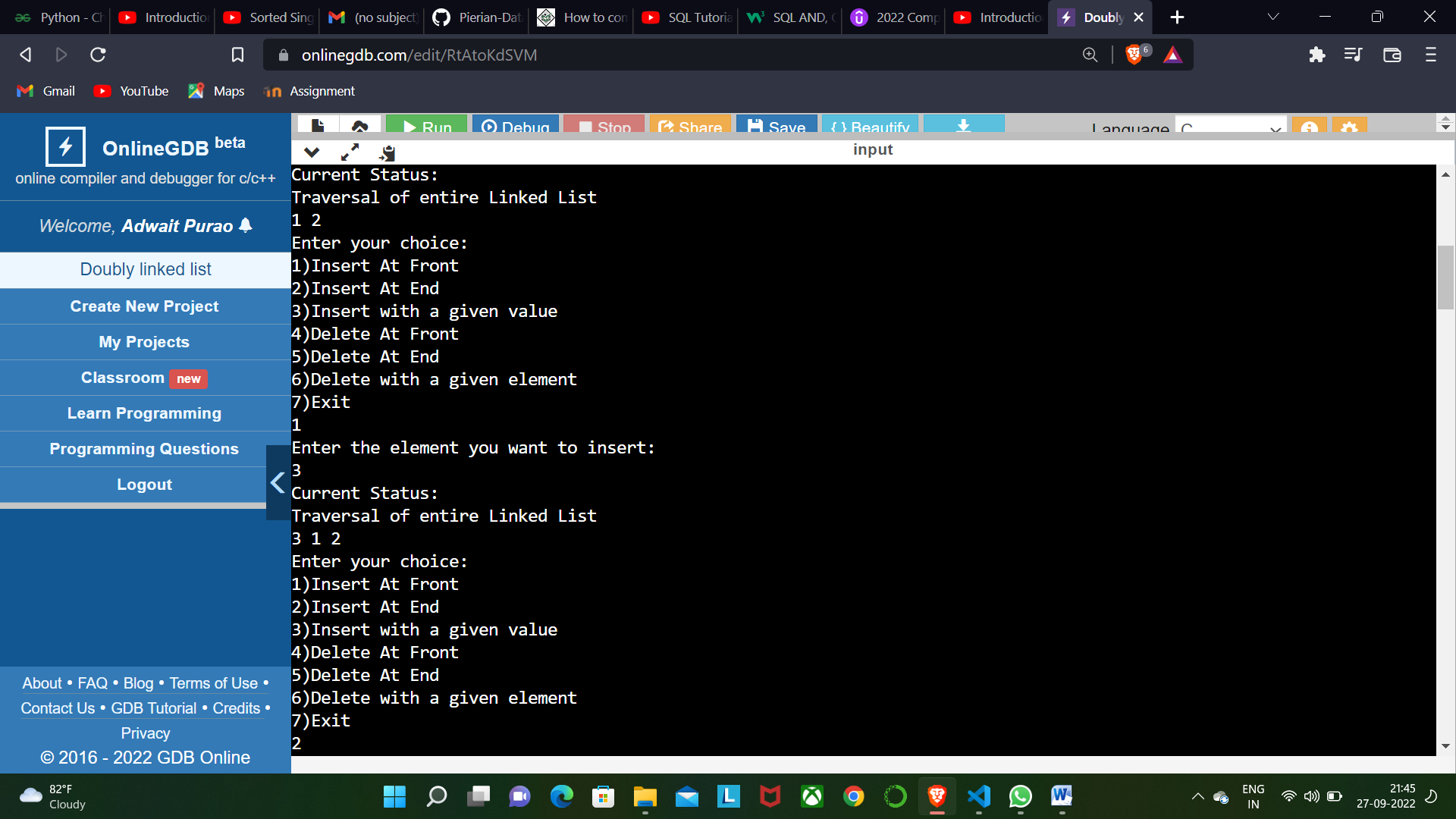
    }while(flag!=1);

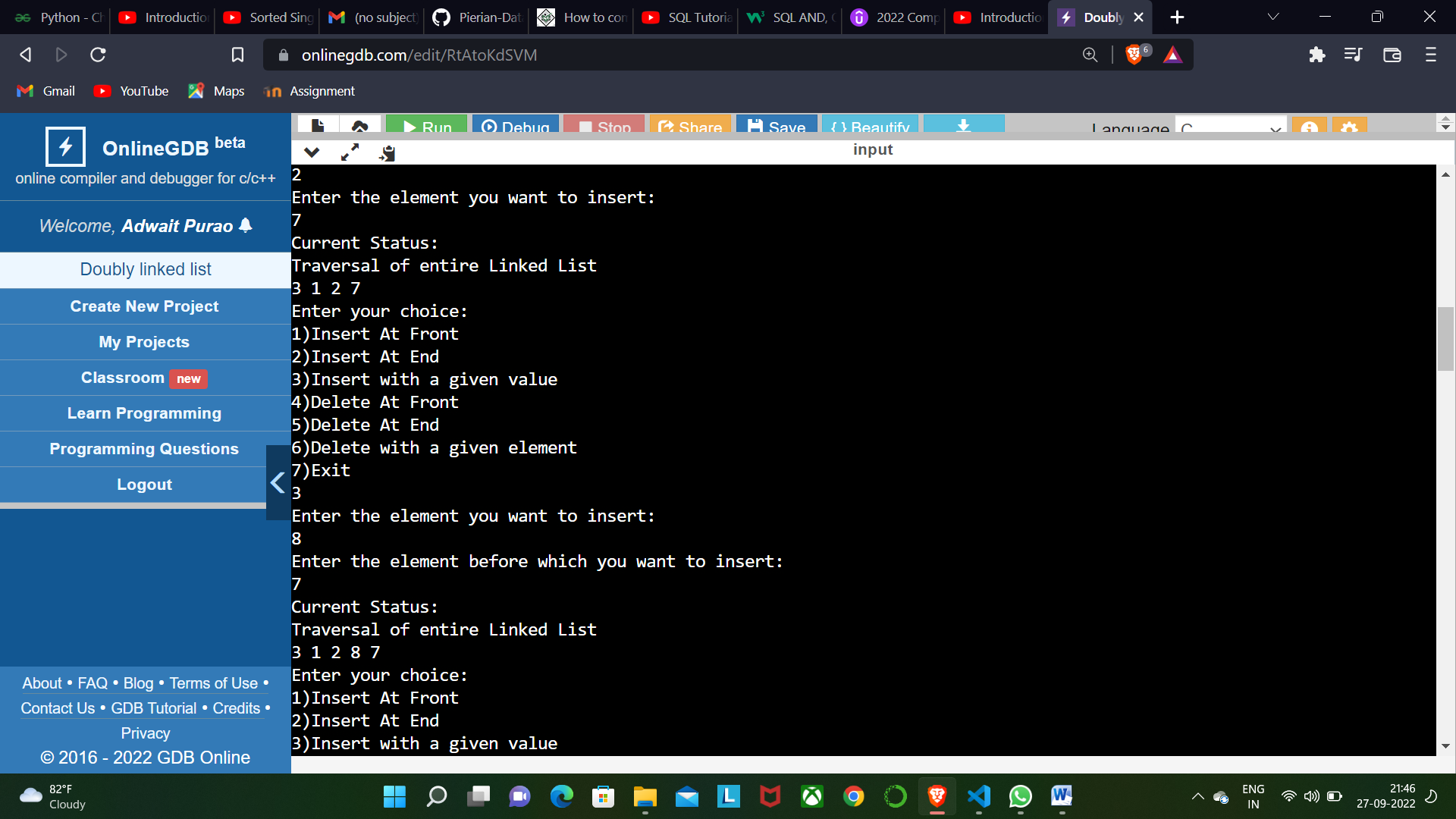
    return 0;

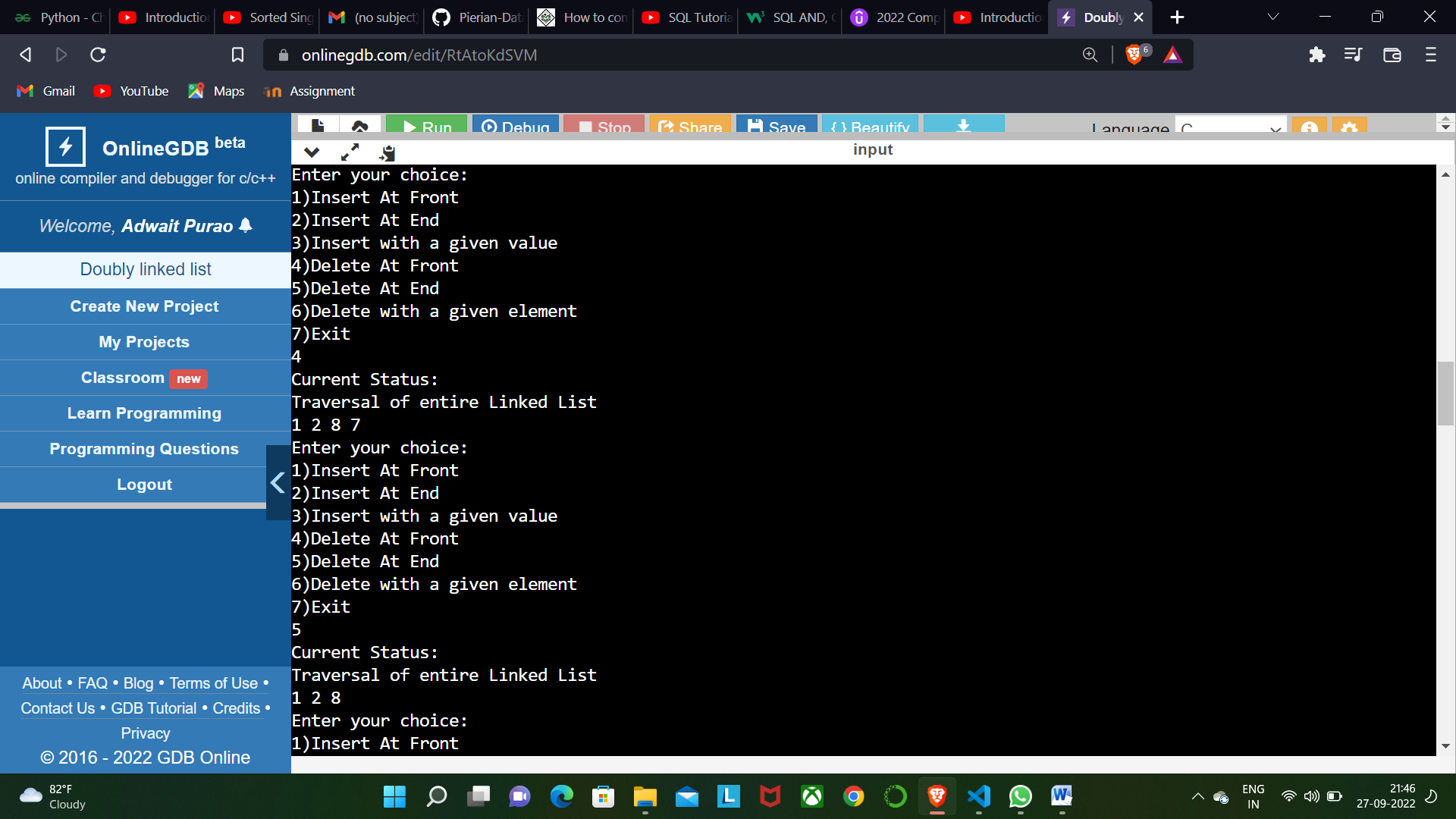
}

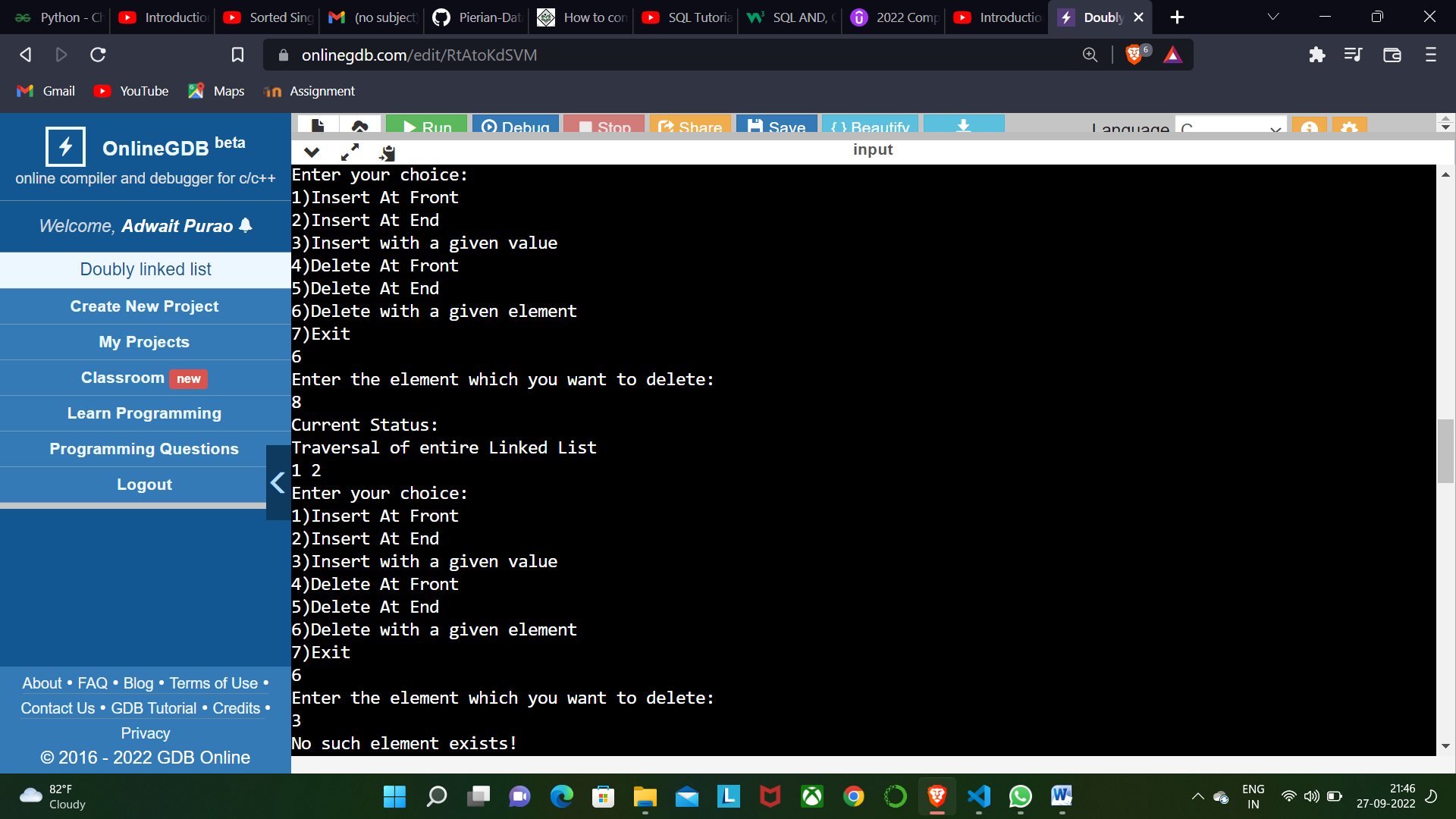
**OUTPUT SCREENSHOT:**

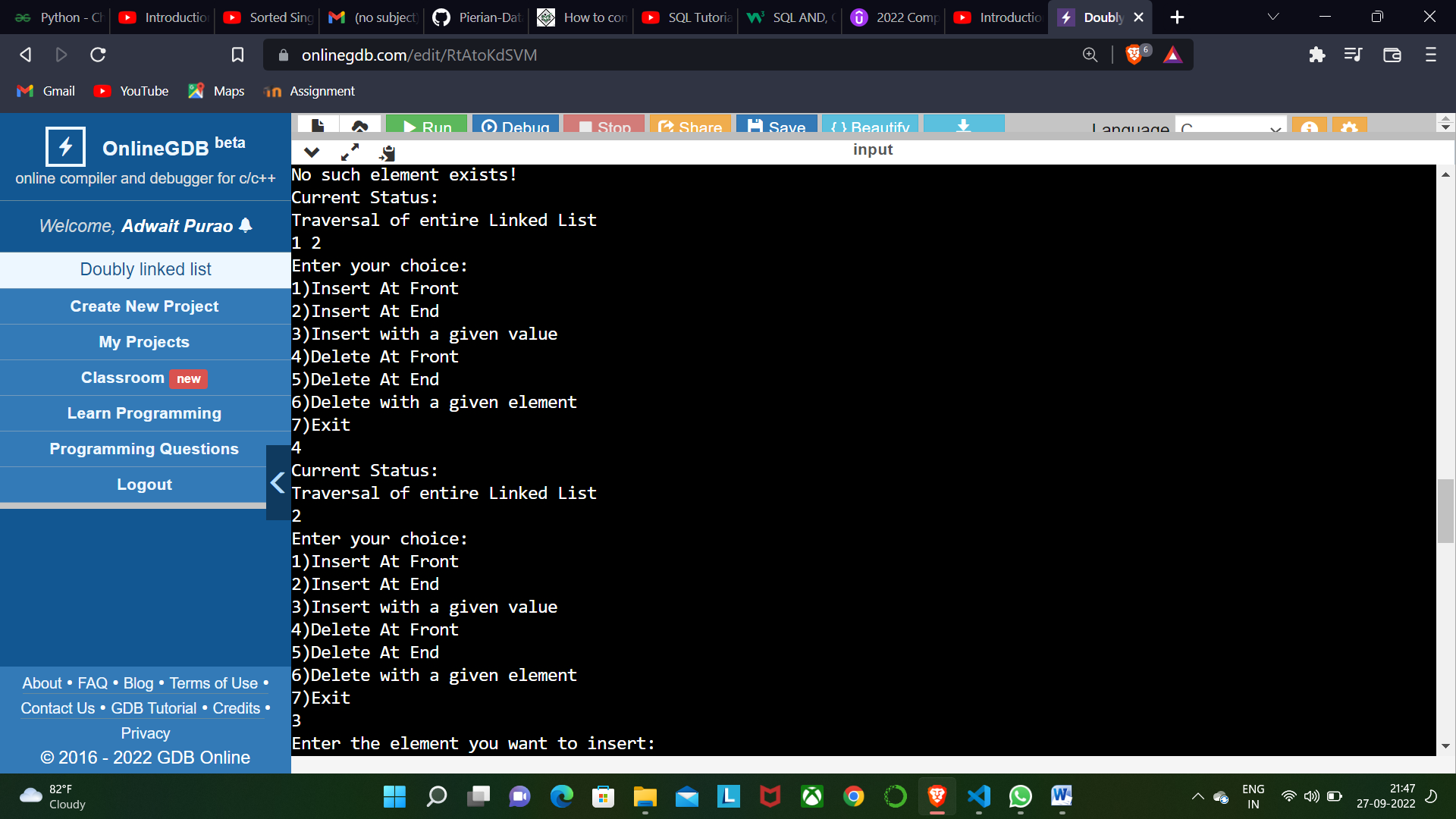


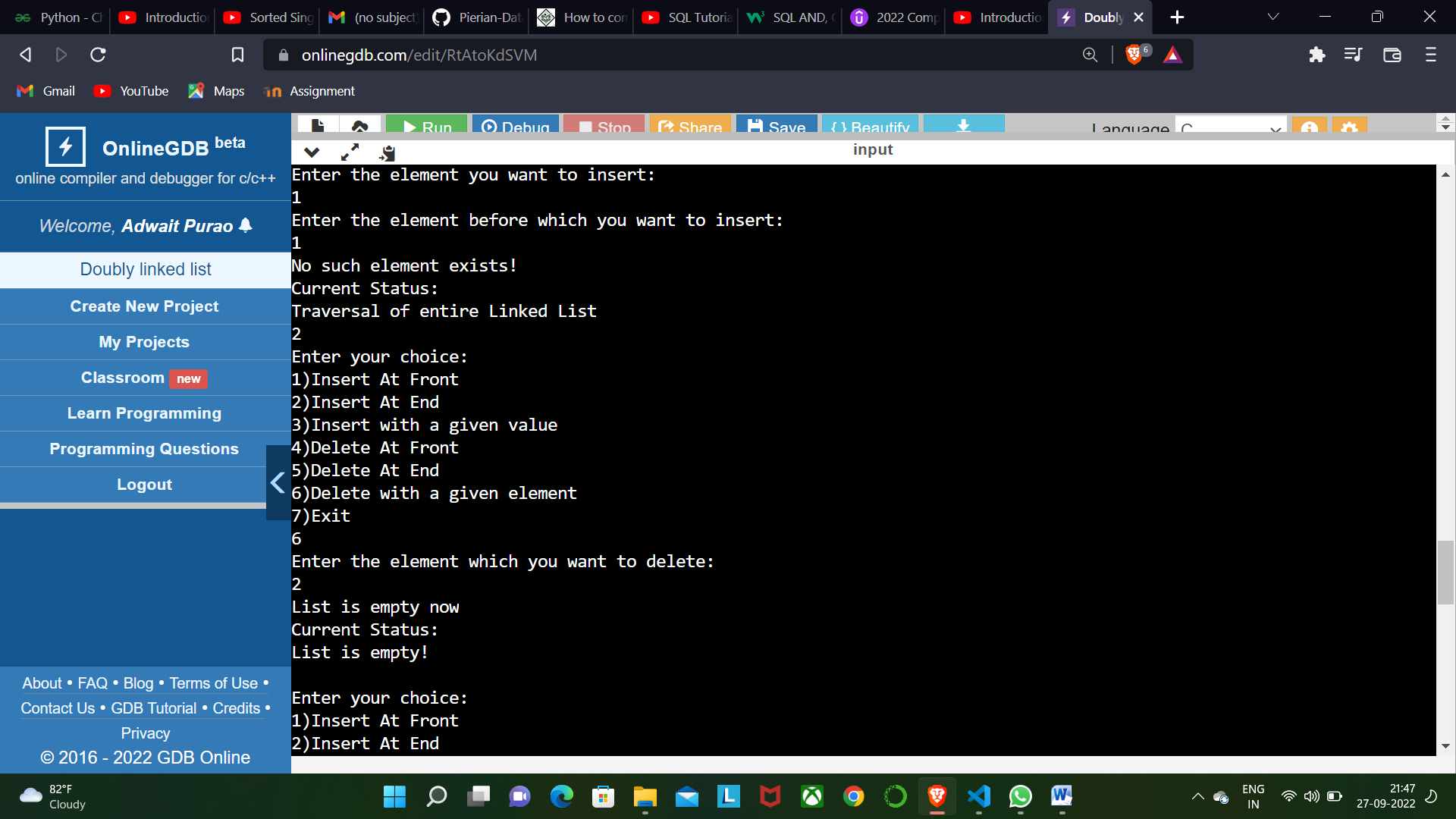


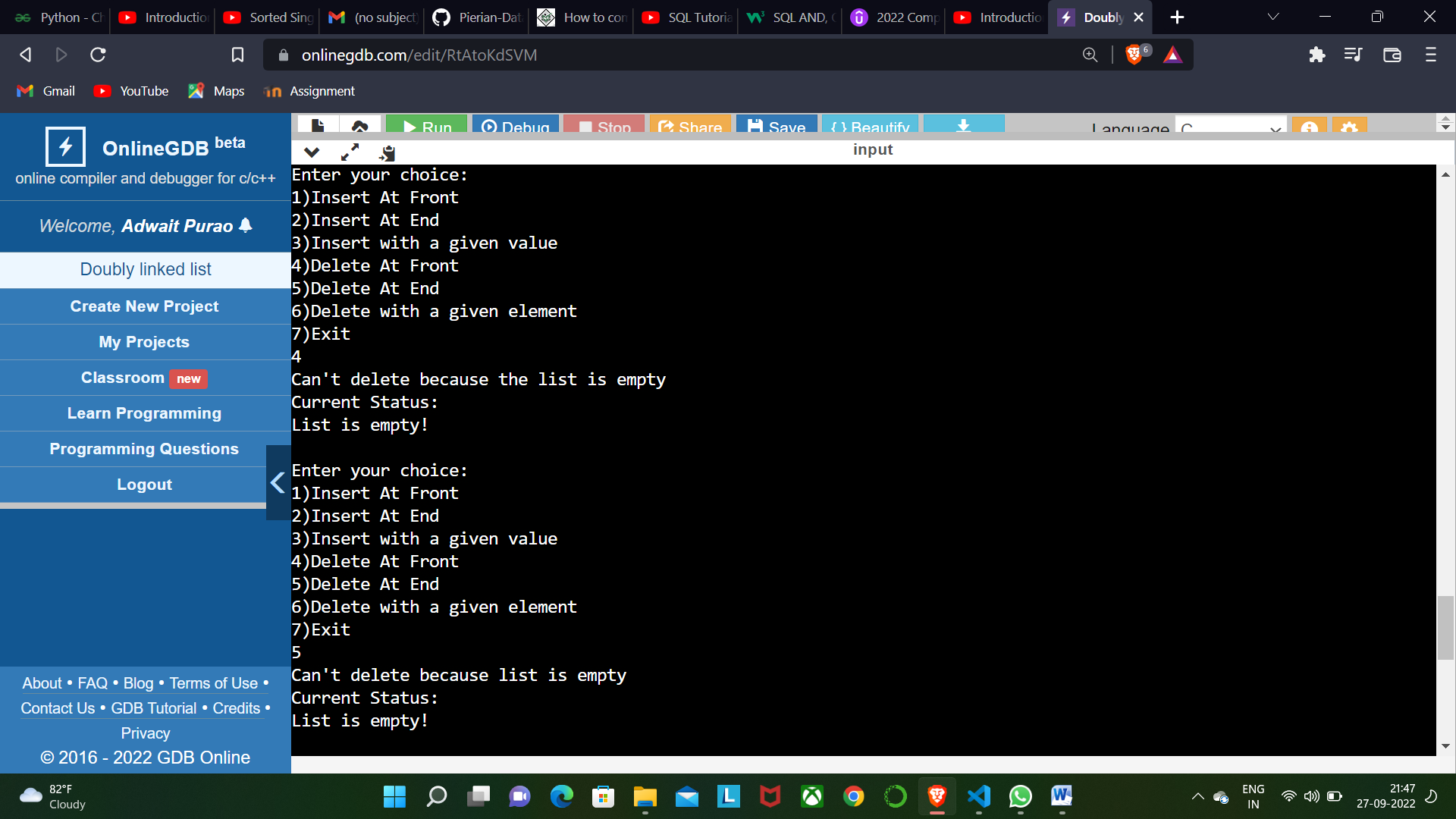


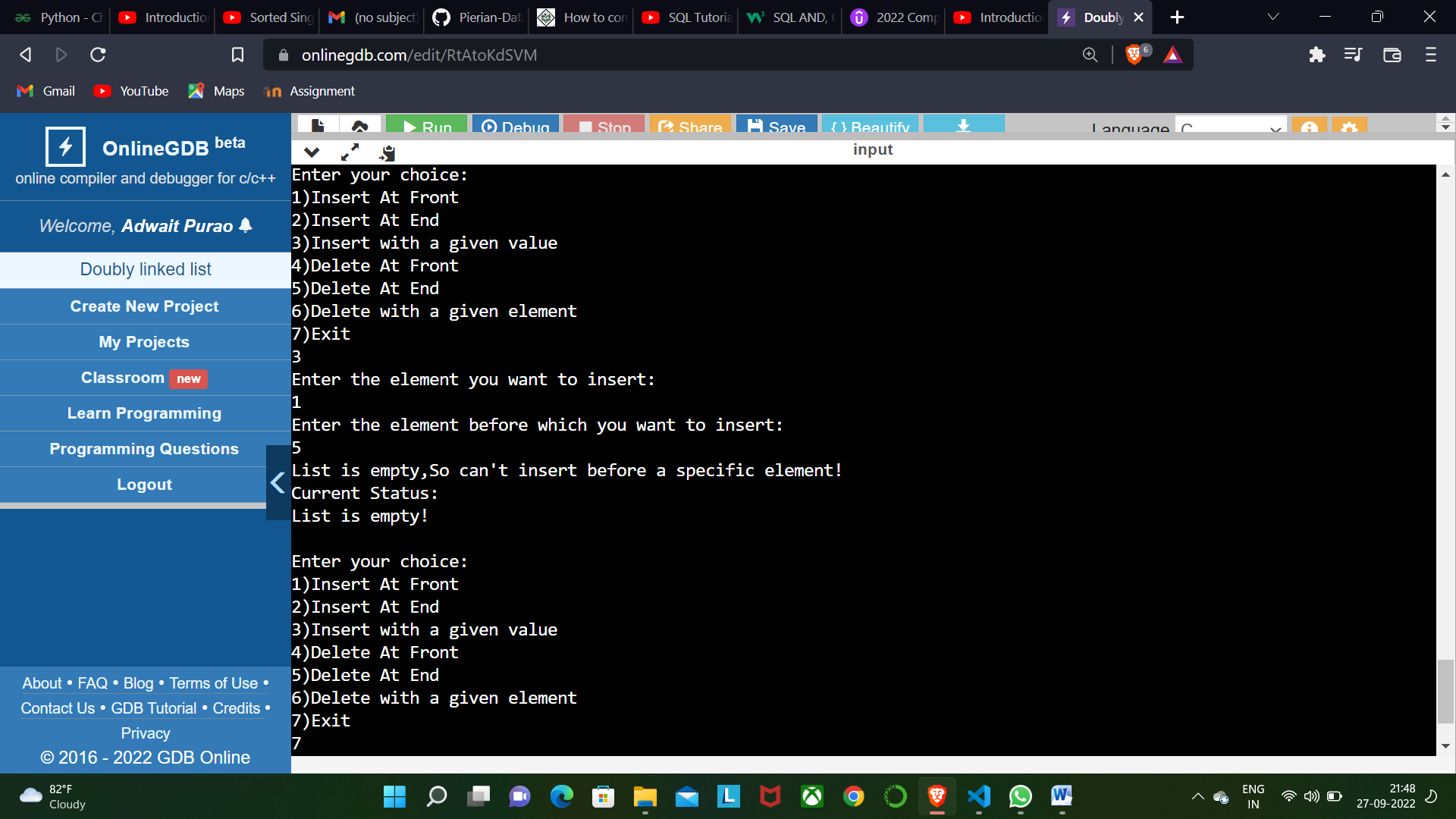


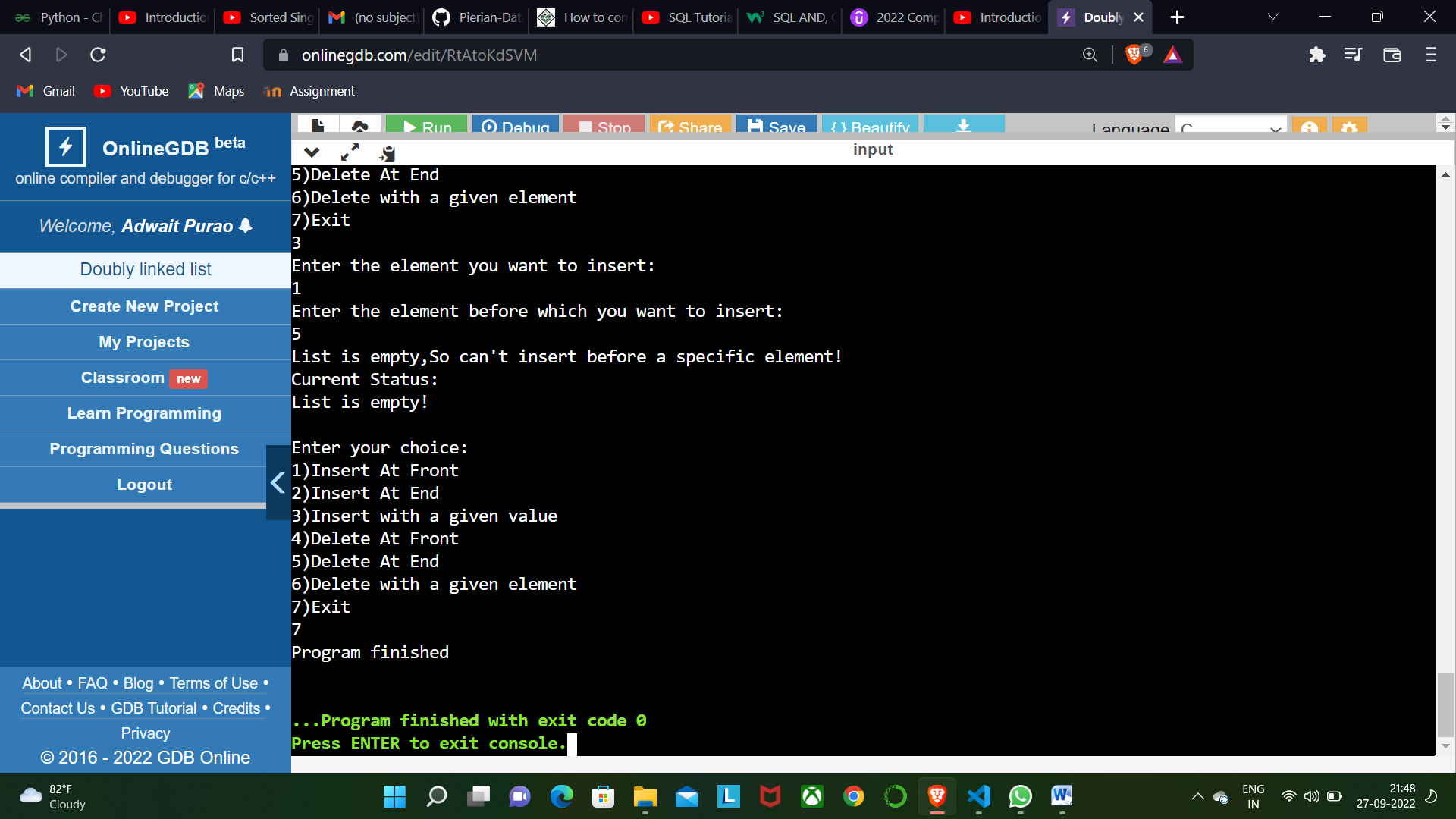












**CONCLUSION:**

**In the help of this experiment we learnt about the Doubly Linked Lists. We understood the internal structure of a Doubly linked list which contains data,pointer to the previous node and pointer to the next node. We learnt that a Doubly linked list has previous of head equal to NULL and next of last node equal to NULL. We learnt about the malloc function used to allocate memory. We performed various functions on Doubly linked list like Insertion at first, Insertion at end, Delete at end, Delete at end , Insert before a particular index, delete before a particular index and Traversal of Doubly linked list.**