Source Code For all Linked list Operation

```
#include<iostream>

using namespace std;

//defining the structure of a node

struct node{
        int data;
        struct node *link;
};

//defining class for performing all operation on singly linked list

class list{
        private:
                node * header;
        public:
                list(){
                        header=NULL;
                }

//insertion at beginning

                insertion()
                {
                        int value;
                        cout<<"Enter the roll no of student"<<endl;
                        cin>>value;
                        node *new_node=new node;
                        if(new_node==NULL)
                        cout<<"overflow,insertion is not possible"<<endl;
```

```
        else
        {
                if(header==NULL)
                {
                        header=new_node;
                        new_node->link=NULL;
                        new_node->data=value;
                }
                else
                {
                        node *ptr;
                        ptr=header;
                        header=new_node;
                        new_node->link=ptr;
                        new_node->data=value;
                }
        }
}
```

//traversing in singly linked list

```
traversing()
{
        node *ptr;
        ptr=header;
        cout<<"node in list are:-"<<endl;
        while(ptr!=NULL)
        {
                cout<<ptr->data<<endl;
                ptr=ptr->link;
        }

}
```

//Insertion at the end of a singly linked list

```
insertion_at_end()
{
        int value;
        cout<<"enter the roll"<<endl;
```

```
                cin>>value;

                node *new_node=new node;

                if(new_node==NULL)
                 cout<<"overflow,insertion not possible"<<endl;
                else
                {
                        node *ptr;
                        ptr=header;
                        while(ptr->link!=NULL)
                        {
                                ptr=ptr->link;
                        }
                        ptr->link=new_node;
                        new_node->link=NULL;
                        new_node->data=value;
                }
        }
```

//insertion after a given node

```
        insertion_after_node()
        {
                int key_value,value;
                cout<<"Enter a value to store in the new node"<<endl;
                cin>>value;
                cout<<"Enter a value after which you have to enter the new
node"<<endl;
                cin>>key_value;

                node * new_node=new node;

                if(new_node==NULL)
                cout<<"overflow"<<endl;
                else
                {
                        node *ptr;
                        ptr=header;
                        while(ptr->link!=NULL && ptr->data!=key_value)
```

```
                ptr=ptr->link;
            if(ptr->data!=key_value)
            cout<<"that insertion is not possible, key value not found"<<endl;
            else if(ptr->data==key_value && ptr->link==NULL)
            {
                ptr->link=new_node;
                new_node->link=NULL;
                new_node->data=value;
            }
            else
            {
                new_node->link=ptr->link;
                ptr->link=new_node;
                new_node->data=value;
            }
        }
    }
```

//deletion operation at the beginning;

```
    deletion_from_beginning()
    {
        if(header==NULL)
        cout<<"list is empty, No deletion"<<endl;
        else{
            node *ptr;
            ptr=header;
            header=ptr->link;
            cout<<"deletion has been performed"<<endl;
            cout<<ptr->data;
        }
    }
```

//deletion from the end;

```
    deletion_from_end(){
        if(header==NULL)
        cout<<"list is empty"<<endl;
        else
        {
```

```
                node *ptr=header;
                node *ptr1;
                while(ptr->link!=NULL)
                {
                        ptr1=ptr;
                        ptr=ptr->link;
                }
                ptr1->link=NULL;
                cout<<"deleted element is"<<endl;
                cout<<ptr->data;
        }
}
```

//deleting a given node

```
deletion(){
    int key;
    cout<<"enter the number to delete"<<endl;
    cin>>key;
    if(header==NULL)
    cout<<"deletion cant be perform,list is empty"<<endl;
    else
    {
        node *ptr,*ptr1;
        ptr=header;
        while(ptr->data!=key && ptr->link!=NULL)
        {
            ptr1=ptr;
            ptr=ptr->link;
            }
            if(ptr->data==key)
            {
                ptr1->link=ptr->link;
                cout<<"the data is deleted"<<endl;
                cout<<ptr->data;
            }
            else
            {
                cout<<"key is not found"<<endl;
            }
```

```
        }
}
```

//Linear Search in Singly Linked list

```
linear-search()
{
        int item;
        cout<<"Enter a value to search in the list"<<endl;
        cin>>item;
        if(header==NULL)
                cout<<"list is empty ,no searching"<<endl;
        else{
                node *ptr=header;
                while(ptr->data!=item && ptr->link!=NULL)
                {
                        ptr=ptr->link;
                }
            }
        if(ptr-data!=item)
                cout<<"Element is not found"<<endl;
        else
        {
                cout<<"Element is found"<<endl;
                cout<<Address of found element is <<ptr;
        }
}
```

//bubble sorting in singly linked list

```
bubbleSort()
{
        node *ptr,*ptr1;
        ptr1=NULL;
        int swap;
        if(header==NULL)
        cout<<"list is empty"<<endl;
        else
        {
```

```
                do {
                        ptr=header;
                        swap=0;
                        while(ptr->link!=ptr1)
                        {
                                if(ptr->data>ptr->link->data)
                                {
                                        int temp;
                                        temp=ptr->data;
                                        ptr->data=ptr->link->data;
                                        ptr->link->data=temp;

                                        swap=1;
                                }
                                ptr=ptr->link;
                        }
                        ptr1=ptr;
                }while(swap);
        }
}
```