Task # 1:
Implement a singly linked list with the following functions:
1. Insert a node at the head
2. Insert a node at the tail/end/back
3. Insert a node at any position
4. Delete a node by value
5. Delete head
6. Delete tail
7. Delete a node at any position.

Single Class Implementation of Linked List

```
#include<iostream>
using namespace std;

class node{
      public:
            node *head,*tail,*next;
            int data;
      node()
      {
            head=NULL;
            tail=NULL;
            }

      //-------------------> insert at end in sll

      void insertAtEnd(int d)
      {
            // creat a temprory node and assign memory.
            node* temp= new node;

            // assign data on it.
            temp->data=d;
            temp->next=NULL;

            if(head==NULL)
            {
                  head=temp;
                  tail=temp;
            }

            else
            {
                tail->next=temp;
                  tail=temp;
            }
```

```
}
//-----------------> ADD At Front.


void insetAtFrot(int d)
{
      node* temp=new node;
      temp->data=d;
      temp->next=head;
      head=temp;
}

//---------------->add at any poistion.

void insertAtAnyPosition(int pos,int val)
{
      node* temp=new node;
      node* current,* previous;
      current=head;
      temp->data=val;
      temp->next=NULL;
      for(int i=0;i<pos;i++)
      {
            previous=current;
            current=current->next;
      }
      previous->next=temp;
      temp->next=current;
}

//----------------------> Delete At front

void deleteAtFront()
{
      node*temp;
      temp=head;
      head=temp->next;
      delete temp;


}

//------------------------> Delete At Back

void deleteAtBack()
{
```

```
   node *current,*prev;
   current=head;
   while(current->next!=NULL)
   {
       prev=current;
       current=current->next;

   }

   tail=prev;
   tail->next=NULL;
   delete current;
}
//-----------------> Any position

void delet(int pos)
{
      node* current;
      node* pre;
      current=head;  // intilize current pointer with head.

      for(int i=0; i<pos;i++)
      {
            pre=current;
            current=current->next;
      }
            pre->next=current->next;
            current->next=NULL;
            delete current;
}


//---------------->delete by value

void deleteByValue(int d)
{

      node *current,*prev;
    current=head;
      while(current->next!=NULL)
      {
             prev=current;
           current=current->next;

           if(current->data==d)
           {
```

```
                        break;

                }

            }

              prev->next=current->next;
                  current->next=NULL;
                  delete current;


      }

      //-----------------> display.

      void disp()
      {
            node* temp;
            temp=head;

            while(temp!=NULL)
            {
                  cout<<temp->data<<" ";
                  temp=temp->next;
            }

            }

};

int main()
{
      node obj;
      cout<<"ADD AT Tail:\n";
      obj.insertAtEnd(4);
      obj.insertAtEnd(7);
      obj.insertAtEnd(0);
      obj.insertAtEnd(2);
      obj.disp();
      cout<<"\n\n ADD AT Front:\n";
      obj.insetAtFrot(2);
      obj.disp();
      cout<<"\n\nADD at your selective position:\n";
      obj.insertAtAnyPosition(2,5);
      obj.disp();
      cout<<"\n\nSearch By Value:\n";
      int val;
```

```
      cin>>val;
      obj.searchByValue(val);
      cout<<"\n\nDelete at Front:\n";
      obj.deleteAtFront();
      obj.disp();

      cout<<"\n\nDelete at back:\n";
      obj.deleteAtBack();
      obj.disp();

      cout<<"\n\nDelete at any position:\n";
      obj.delet(1);
      obj.disp();

    cout<<"\n\nDelete by value:\n";
      obj.deleteByValue(7);
      obj.disp();
}
```

Task 02:Alter a node’s data in a Singly Linked List
Input: S->A->T->A
Output: D->A->T->A

```
#include<iostream>
using namespace std;

class node{
      public:
            node* head,*tail,*next;
            char data;
            public:
                  node()
                  {
                        head=NULL;
                        tail=NULL;
                  }
            //--------------> node insertion..
            void addbacK(char d)
            {
                  node *temp=new node;
                  temp->data=d;
                  temp->next=NULL;

                  if(head==NULL)
                  {
                        head=temp;
                        tail=temp;
```

```
                }
                else
                {
                        tail->next=temp;
                        tail=temp;
                }
        }
        //---------------------> inter change;

        void interchange(char d,int pos)
        {
                node *temp;
                temp=head;
                for(int i=0;i<pos;i++)
                {
                        temp=temp->next;
                }
                temp->data=d;
        }


        //---------------------->display()
        void display()
        {
                node* temp;
                temp=head;
                while(temp!=NULL)
                {
                        cout<<temp->data<<"->";
                        temp=temp->next;
                }
        }
};
int main()
{
      node obj;
      obj.addbacK('S');
      obj.addbacK('A');
      obj.addbacK('T');
      obj.addbacK('A');
      obj.display();
      obj.interchange('D',0);
      cout<<"\n\nAfter Interchange:\n";
      obj.display();
}
```

Task 03: Solve the following problem using a Singly Linked List.
Given a singly linked list of characters, write a function to make a word out of given letters in the list
Example: **Input:** C->S->A->R->B->B->E->L->NULL
Output: S->C->R->A->B->B->L->E->NULL

```
#include<iostream>
using namespace std;

class node{
	public:
	node *head,*next,*tail;
	char data;
	node()
	{
		head=NULL;
		tail=NULL;
	}
	//-------------------->add front
	void Add(char d)
	{
		node* temp=new node;
		temp->next=NULL;
		temp->data=d;
		if(head==NULL)
		{
			head=temp;
			tail=temp;
		}
		else
		{
			tail->next=temp;
			tail=temp;
		}

	}

	//------------------->edit

	void edit()
	{
		node *current;
		current=head;
		node *prev;
		int i=2;
		while(current->next!=NULL)
		{
			prev=current;
			current=current->next;

			if(i%2==0)
			{
				char temp;
				temp=prev->data;
```

```
                        prev->data=current->data;
                        current->data=temp;
                    i++;
                   }
                   else
                   {
                        i++;
                   }

             }

       }
       //------------------>voide display

       void display()
       {
             node *temp;
             temp=head;
             while(temp!=NULL)
             {
                   cout<<temp->data<<"->";
                   temp=temp->next;
             }
       }
};
int main()
{
       node obj;
       cout<<"\nBefore Edit:\n";
       obj.Add('a');
       obj.Add('s');
       obj.Add('c');
       obj.Add('R');
       obj.Add('B');
       obj.Add('B');
       obj.Add('E');
       obj.Add('L');
       obj.display();
       obj.edit();
       cout<<"\n\n After Edit:\n";
       obj.display();
}
```

Task # 4:

Use the class of SLL created by you during lab task 1. Do the following:

1. Reverse the linked list
2. Sort the contents of the linked list
3. Find the duplicates in the linked list

```
#include<iostream>
using namespace std;
static int size=0;
class node{
```

```
    public:
            node* head,*tail,*next;
            int data;
            public:
                    node()
                    {
                            head=NULL;
                            tail=NULL;
                    }
            //--------------> node insertion..
            void addbacK(int d)
            {
                    size++;
                    node *temp=new node;
                    temp->data=d;
                    temp->next=NULL;

                    if(head==NULL)
                    {
                            head=temp;
                            tail=temp;
                    }
                    else
                    {
                            tail->next=temp;
                            tail=temp;
                    }
            }
// ------------------> void reverse()
void reverse()
{

    node *current,*temp,*prev;
    current=head;
    prev=NULL;
    while(current!=NULL)
    {
            temp=current->next;
            current->next=prev;
            prev=current;
            current=temp;
                     }
                     head=prev;


             }
             //------------------->void duplicate

             void duplicate()
             {
                    int counter=0;
```

```
        node *temp,*current;
        temp=head;
        while(temp->next!=NULL)
        {
                current=temp;
                temp=temp->next;
                if(temp->data==current->data)
                {
                        counter++;
                 }
         }
         cout<<"\n\nNumber of duplicate are : "<<counter;
 }

 void sort()
 {
        node *current=NULL,*prev=NULL;
        int temp=0;
        for(int i=0;i<size-1;i++)
        {
                current=head;

                for(int j=0;j<size-i-1;j++)
                {
                                prev=current;
                      current=current->next;

                        if(prev->data > current->data)
                        {
                                temp=prev->data;
                                prev->data=current->data;
                                current->data=temp;
                         }

                 }
         }
 }

//---------------------->display()
void display()
{
        node* temp;
        temp=head;
        while(temp!=NULL)
        {
                cout<<temp->data<<" ";
                temp=temp->next;
        }
}
};
```

```
int main()
{
	node obj;
	obj.addbacK(9);
	obj.addbacK(5);
	obj.addbacK(5);
	obj.addbacK(3);
	obj.addbacK(4);
	obj.display();
	obj.reverse();
	cout<<"\nLinkedList After Reverse!!!\n\n";
	obj.display();
	obj.duplicate();
	cout<<"\nLinkedList After Reverse!!!\n\n";
	obj.sort();
	obj.display();
}
```