TASK-06

Create a menu driven program for Doubly Linked List.
Include all operations in the menu, which are as follows:
1. Insert at Head
2. Insert at Tail
3. Insert After
4. Insert Before
5. Delete from Head
6. Delete from Tail
7. Delete Node
8. Traverse List

9. Reverse Traverse List

Code:

- **Dnode.h File**

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

class Dnode
{
  public:
        double data;
        Dnode* next;

        Dnode* prev;

  Dnode(double i=0, Dnode* n=0, Dnode* p=0)

        {

              data = i;
              next = n;
              prev = p;
        }
};
```

- **Dlinkedlist.h file**

```
#include "Dnode.h"
#include <iostream>
using namespace std;

class Dlinkedlist
{
	private:
		Dnode* head;
		Dnode* tail;
	public:
		Dlinkedlist()
		{
			head = 0;
			tail = 0;
		}
		void insertathead(double value);
		void insertattail(double value);
		void insertafter(double existing, double value);
		void insertbefore(double existing, double value);
		void deletefromhead();
		void deletefromtail();
		void deleteSnode(double value);
		void traverselist();
		void reversetraverselist();
		bool isempty();
};
bool Dlinkedlist::isempty()
	{
		if(head == 0 && tail == 0)
		{
			return true;
		}
```

```
        else
        {
            return false;
        }
    }
void Dlinkedlist::insertathead(double value)
    {
        Dnode* newnode = new Dnode(value);
        if(isempty())
        {
            head = tail = newnode;
        }
        else
        {
            newnode->next = head;
            head->prev = newnode;
            head = newnode;
        }
    }
void Dlinkedlist::insertattail(double value)
    {
        Dnode* newnode = new Dnode(value);
        if(isempty())
        {
            head = tail = newnode;
        }
        else
        {
            newnode->prev = tail;
            tail->next = newnode;
            tail = newnode;
        }
    }
void Dlinkedlist::insertafter(double existing, double value)
```

```
	{
		if(isempty())
		{
			cout<<"\nList is empty.";
		}
		else if(existing == tail->data)
		{
			insertattail(value);
		}
		else
		{
			Dnode* currnode = head;
			while(currnode != 0 && currnode->data != 
existing)
			{
				currnode = currnode->next;
			}
			if(currnode==0)
			{
				cout<<"\nInsertion is not possible in the list 
because existing element in not present in the list.";
			}
			else
			{
				Dnode* newnode = new Dnode(value);
				newnode->next = currnode->next;
				newnode->prev = currnode;
				currnode->next->prev = newnode;
				currnode->next = newnode;
			}
		}
	}
void Dlinkedlist::insertbefore(double existing, double value)
	{
		if(isempty())
		{
			cout<<"\nList is empty.";
		}
		else if(existing == head->data)
		{
			insertathead(value);
		}
		else
		{
			Dnode* currnode = head;
			while(currnode != 0 && currnode->data != 
existing)
			{
				currnode = currnode->next;
			}
			if(currnode==0)
			{
				cout<<"\nInsertion is not possible in the list 
because existing element in not present in the list.";
			}
			else
			{
				Dnode* newnode = new Dnode(value);
				newnode->next = currnode;
				newnode->prev = currnode->prev;
				currnode->prev->next = newnode;
				currnode->prev = newnode;
			}
		}
	}
void Dlinkedlist::traverselist()
	{
		if(isempty())
```

```
				{
					cout<<"\nList is empty.";
				}
				else
				{
					cout<<"\nValues in list are: "<<endl;
					Dnode* currnode = head;
					while(currnode != 0)
					{
						cout<<currnode->data<<endl;
						currnode = currnode->next;
					}
				}
			}
		void Dlinkedlist::reversetraverselist()
			{
				if(isempty())
				{
					cout<<"\nList is empty.";
				}
				else
				{
					cout<<"\nValues in list are: "<<endl;
					Dnode* currnode = tail;
					while(currnode != 0)
					{
						cout<<currnode->data<<endl;
						currnode = currnode->prev;
					}
				}
			}
		void Dlinkedlist::deletefromhead()
			{
				if(isempty())
```

```
                {
                        cout<<"\nList is empty.";
                }
                else if(head==tail)
                {
                        Dnode* dellnode = head;
                        head = tail = 0;
                        delete dellnode;
                }
                else
                {
                        Dnode* dellnode = head;
                        head->next->prev = 0;
                        head = head->next;
                        dellnode->next = 0;
                        delete dellnode;
                }
        }
void Dlinkedlist::deletefromtail()
        {
                if(isempty())
                {
                        cout<<"\nList is empty.";
                }
                else if(tail==head)
                {
                        Dnode* dellnode = tail;
                        tail = head = 0;
                        delete dellnode;
                }
                else
                {
                        Dnode* dellnode = tail;
                        tail->prev->next = 0;
```

```
                    tail = tail->prev;
                    dellnode->prev = 0;
                    delete dellnode;
                }
            }
        void Dlinkedlist::deleteSnode(double value)
            {
                if(isempty())
                {
                    cout<<"\nList is empty.";
                }
                else if(head->data == value)
                {
                    deletefromhead();
                }
                else if(tail->data == value)
                {
                    deletefromtail();
                }
                else
                {
                    Dnode* dellnode = head;
                    while(dellnode!=0 && dellnode->data!=value)
                    {
                        dellnode = dellnode->next;
                    }
                    if(dellnode==0)
                    {
                        cout<<"\nValue doesn't exist in the list.";
                    }
                    else
                    {
                        dellnode->prev->next = dellnode->next;
                        dellnode->next->prev = dellnode->prev;
                        dellnode->next = 0;
                        dellnode->prev = 0;
                        delete dellnode;
                    }
                }

            }
```

- **.cpp file**

```
#include <iostream>
#include "Dlinkedlist.h"
using namespace std;
int main()
{
    double val;
    double existing;
    char con;
    int choice;
    Dlinkedlist list;
    do
    {

        cout<<"\tPress 1 for insert at head"<<endl;
        cout<<"\tPress 2 for insert at tail"<<endl;
        cout<<"\tPress 3 for insert after"<<endl;
        cout<<"\tPress 4 for insert before"<<endl;
        cout<<"\tPress 5 for delete from head"<<endl;
        cout<<"\tPress 6 for delete from tail"<<endl;
        cout<<"\tPress 7 for delete from specific node"<<endl;
        cout<<"\tPress 8 for traverse node"<<endl;
        cout<<"\tPress 9 for reverse traverse node"<<endl;
        cout<<"Enter choice: ";
        cin>>choice;
```

```
                    switch (choice)
                    {
                            case 1:
                            cout<<"Enter value to insert at head: ";
                            cin>>val;
                            list.insertathead(val);
                            break;
                case 2:
                            cout<<"Enter value to insert at tail: ";
                            cin>>val;
                            list.insertattail(val);
                        break;
                    case 3:
                            cout<<"Enter value to insert after(1st enter existing
value then enter new value): ";
                            cin>>existing;
                            cin>>val;
                            list.insertafter(existing,val);
                        break;
                case 4:
                            cout<<"Enter value to insert before(1st enter
existing value then enter new value): ";
                            cin>>existing;
                            cin>>val;
                            list.insertbefore(existing,val);
                        break;
                    case 5:
                            list.deletefromhead();
                        break;
                    case 6:
                            list.deletefromtail();
                            break;
                    case 7:
                            cout<<"Enter value for specific node deletion: ";
                cin>>val;
                list.deleteSnode(val);
                break;
            case 8:
                list.traverselist();
              break;
            case 9:
                list.reversetraverselist();
              break;
            default:
                cout<<"Sorry! Wrong choise"<<endl;
                break;
            }
            cout<<"\nPress (y) for again continue the program and
press any key except (y) for exit: ";
            cin>>con;

        }
        while(con == 'y');
}
```