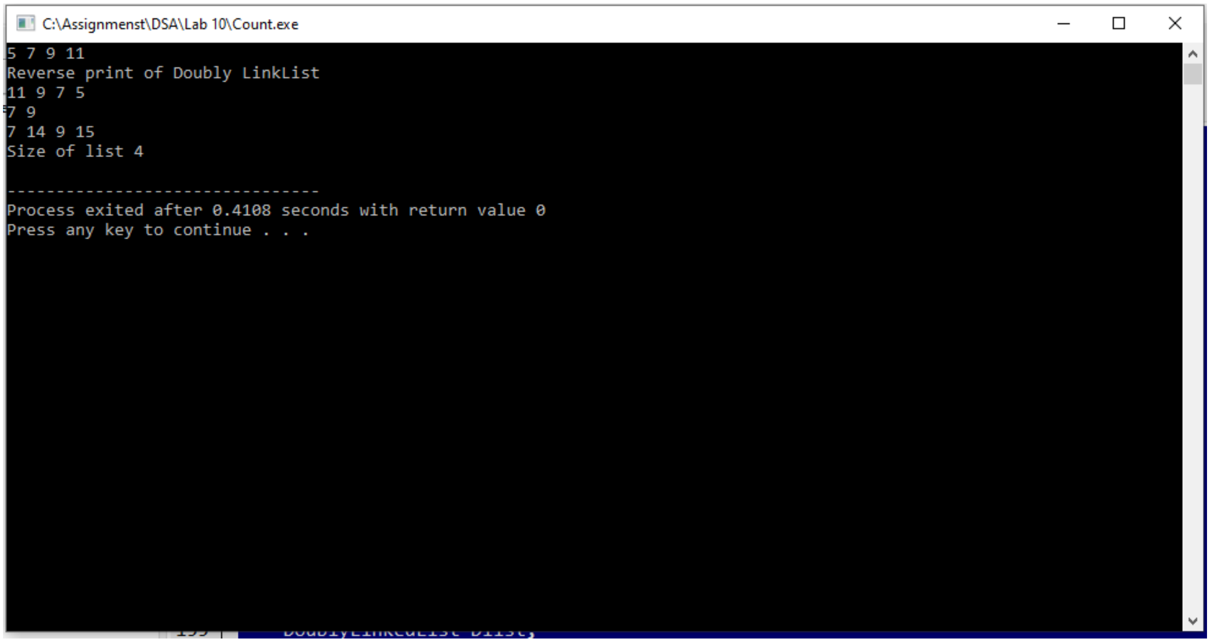
Q. Length of List

```
#include <iostream>
using namespace std;
class Node {
public:
  int data;
  Node* next;
  Node* prev;
 Node(int data)
        {
        this->data = data;
        this->next = NULL;
        this->prev = NULL;
        }

};

class DoublyLinkedList {
private:
  Node* head;
  Node* tail;

public:
  DoublyLinkedList()
        { head = NULL;
          tail = NULL;
        }

  void insertAtStart(int val) {
    Node* newNode = new Node(val);
    if (!head) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

void insertAtEnd(int val) {
    Node* newNode = new Node(val);
    if (!tail) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void insertAtPosition(int val, int position) {
    if (position < 1) {
        cout << "Position Invalid." <<endl;
        return;
    }
    Node* newNode = new Node(val);
    if (position == 1) {
        insertAtStart(val);
    } else {
        Node* current = head;
        int currentPosition = 1;
        while (current && currentPosition < position - 1) {
            current = current->next;
            currentPosition++;
        }
        if (!current) {
            cout << "Invalid Position." <<endl;
            delete newNode;
            return;
        }
        newNode->next = current->next;
        newNode->prev = current;
        if (current->next) {
            current->next->prev = newNode;
        }
        current->next = newNode;
    }
}

void deleteFromStart() {
    if (!head) {
        cout << "List is empty." <<endl;
        return;
    }

    Node* temp = head;
    head = head->next;
    if (head) {
        head->prev = NULL;
    } else {
        tail = NULL;
    }
    delete temp;
}

void deleteFromEnd() {
    if (!tail) {
        cout << "List is empty." <<endl;
        return;
    }

    Node* temp = tail;
    tail = tail->prev;
    if (tail) {
        tail->next = NULL;
    } else {
        head = NULL;
    }
    delete temp;
}

void printList() {
    Node* current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout <<endl;
}
int count(){
          int cc=0;
          Node* front=head;
                        if (!head) {
```

```
            cout << "List is empty." <<endl;
            exit;
        }else{

              while(front){
                            front =front->next;
                            cc++;
                                          }
                            }
              return cc;
              }
    void insertm(){
              int half =count()/2;
              if (!head) {
            cout << "List is empty." <<endl;
            return;
        }
        else{
                            }
              }
    void printreverseList() {
        Node* reverse = tail;
        while (reverse) {
            cout << reverse->data << " ";
            reverse = reverse->prev;
        }
        cout <<endl;
    }
};

int main() {
    DoublyLinkedList Dlist;


    Dlist.insertAtStart(9);

    Dlist.insertAtEnd(11);

    Dlist.insertAtStart(5);

    Dlist.insertAtPosition(7, 2);

    Dlist.printList();

    cout<<"Reverse print of Doubly LinkList"<<endl;

    Dlist.printreverseList();

    Dlist.deleteFromStart();

    Dlist.deleteFromEnd();

    Dlist.printList();

    Dlist.insertAtEnd(15);

    Dlist.insertAtPosition(14, 2);

    Dlist.printList();

    int count=Dlist.count();

    cout<<"Size of list "<<count<<endl;

    return 0;

}
```



```
5 7 9 11
Reverse print of Doubly LinkList
11 9 7 5
7 9
7 14 9 15
Size of list 4

--------------------------------
Process exited after 0.4108 seconds with return value 0
Press any key to continue . . .
```

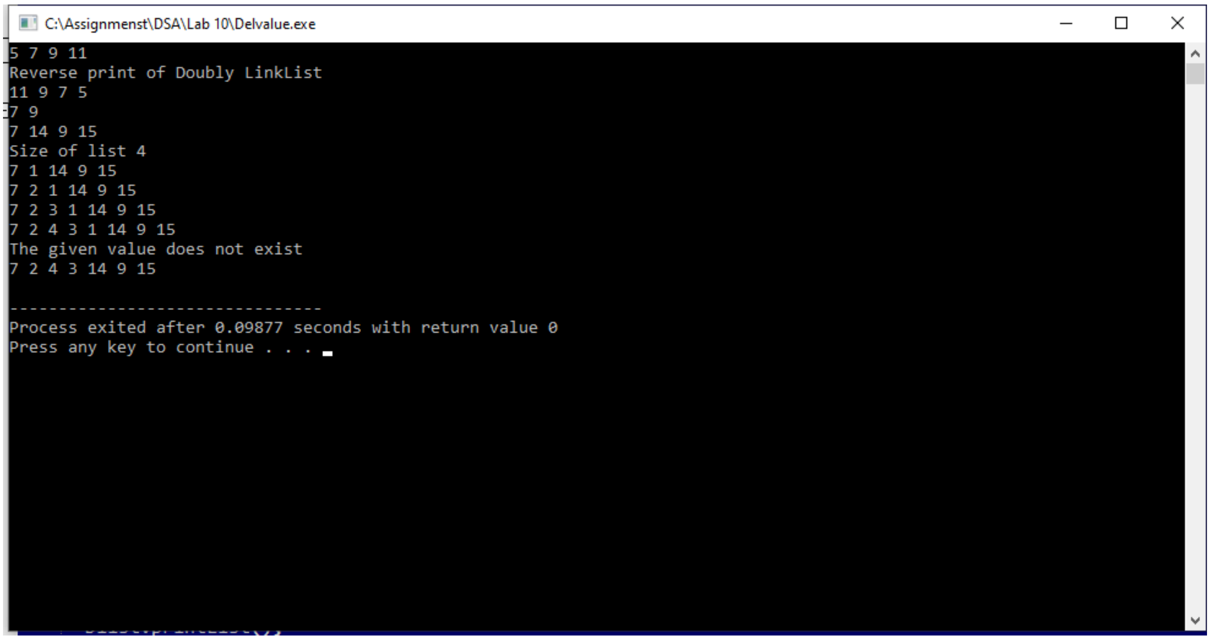
Q. Delete by Value

```
#include <iostream>

using namespace std;

class Node {

public:

   int data;

   Node* next;

   Node* prev;

  Node(int data)

              {

              this->data = data;

              this->next = NULL;

              this->prev = NULL;

              }


};


class DoublyLinkedList {

private:

   Node* head;

   Node* tail;


public:

   DoublyLinkedList()

              { head = NULL;

                tail = NULL;

              }


   void insertAtStart(int val) {

       Node* newNode = new Node(val);

       if (!head) {
           head = tail = newNode;
       } else {
           newNode->next = head;
           head->prev = newNode;
           head = newNode;
       }
   }

   void insertAtEnd(int val) {
       Node* newNode = new Node(val);
       if (!tail) {
           head = tail = newNode;
       } else {
           tail->next = newNode;
           newNode->prev = tail;
           tail = newNode;
       }
   }

   void insertAtPosition(int val, int position) {
       if (position < 1) {
           cout << "Position Invalid." <<endl;
           return;
       }
       Node* newNode = new Node(val);
       if (position == 1) {
           insertAtStart(val);
       } else {
           Node* current = head;
           int currentPosition = 1;
           while (current && currentPosition < position - 1) {
               current = current->next;
               currentPosition++;
           }
           if (!current) {
               cout << "Invalid Position." <<endl;
               delete newNode;
               return;
           }
           newNode->next = current->next;
           newNode->prev = current;
           if (current->next) {
               current->next->prev = newNode;
           }
           current->next = newNode;
       }
   }

   void deleteFromStart() {
       if (!head) {
           cout << "List is empty." <<endl;
           return;
       }

       Node* temp = head;
       head = head->next;
       if (head) {
           head->prev = NULL;
       } else {
           tail = NULL;
       }
       delete temp;
   }

   void deleteFromEnd() {
       if (!tail) {
           cout << "List is empty." <<endl;
           return;
       }

       Node* temp = tail;
       tail = tail->prev;
       if (tail) {
           tail->next = NULL;
       } else {
           head = NULL;
       }
       delete temp;
   }

   void printList() {
       Node* current = head;
       while (current) {
           cout << current->data << " ";
           current = current->next;
       }
       cout <<endl;
   }
   int count(){
           int cc=0;
           Node* front=head;
                       if (!head) {
```

```
    cout << "List is empty." <<endl;
    exit;
  }else{

     while(front){
            front =front->next;
            cc++;
                   }
            }
     return cc;
     }
void insertm(int data){
     int half =count()/2;
     Node* position=head;
     if (!head) {
    cout << "List is empty." <<endl;
    return;
  }
  else{

     for(int i=1;i<half-1;i++){
            position=position->next;
                   }
                   Node* newnode=new Node(data);
             newnode->next = position->next;
    newnode->prev = position;
    if (position->next) {
      position->next->prev = newnode;
    }
    position->next = newnode;
            }
```

```
          }
void printreverseList() {
    Node* reverse = tail;
    while (reverse) {
        cout << reverse->data << " ";
        reverse = reverse->prev;
    }
    cout <<endl;
}
 void deletevalue(int data) {
    if (!head) {
        cout << "List is empty." <<endl;
        return;
    }

    Node* temp = head;
  int cc=count();

   if(!head){
           cout << "List is empty." <<endl;
        return;

             }

             else{
             while(temp){

    if (temp->data==data) {
          temp->prev->next=temp->next;
       temp->next=NULL;
            temp->prev=NULL;
```

```
         return;
     }
     else if(temp==tail){
         cout<<"The given value does not exist"<<endl;
                  }
         temp=temp->next;




                   }
  delete temp;
   }
         return;
         }
};

int main() {
   DoublyLinkedList Dlist;

   Dlist.insertAtStart(9);
   Dlist.insertAtEnd(11);
   Dlist.insertAtStart(5);
   Dlist.insertAtPosition(7, 2);
   Dlist.printList();
   cout<<"Reverse print of Doubly LinkList"<<endl;
   Dlist.printreverseList();
   Dlist.deleteFromStart();
   Dlist.deleteFromEnd();
   Dlist.printList();
   Dlist.insertAtEnd(15);
   Dlist.insertAtPosition(14, 2);

   Dlist.printList();

   int count=Dlist.count();

   cout<<"Size of list "<<count<<endl;

   Dlist.insertm(1);

     Dlist.printList();

   Dlist.insertm(2);

     Dlist.printList();

   Dlist.insertm(3);

     Dlist.printList();

   Dlist.insertm(4);

     Dlist.printList();

     Dlist.deletevalue(18);

       Dlist.deletevalue(1);

   Dlist.printList();

   return 0;

}
```



```
5 7 9 11
Reverse print of Doubly LinkList
11 9 7 5
7 9
7 14 9 15
Size of list 4
7 1 14 9 15
7 2 1 14 9 15
7 2 3 1 14 9 15
7 2 4 3 1 14 9 15
The given value does not exist
7 2 4 3 14 9 15

--------------------------------
Process exited after 0.09877 seconds with return value 0
Press any key to continue . . .
```

Q. Insert Middle

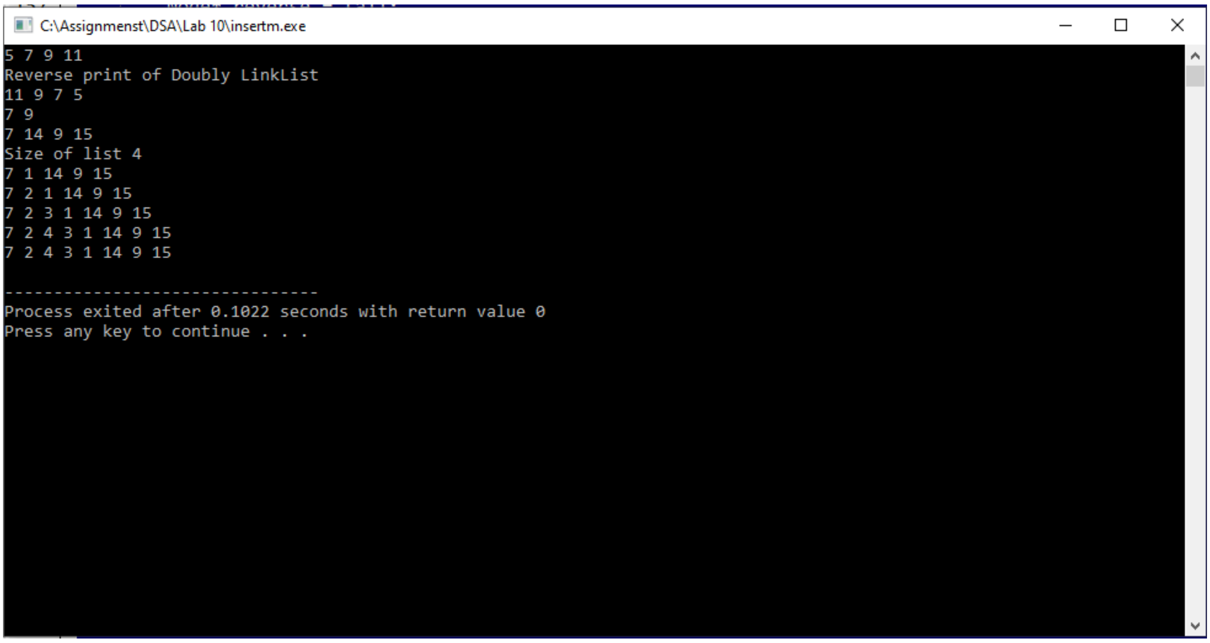
```
#include <iostream>

using namespace std;
```

```
class Node {
public:
    int data;
    Node* next;
    Node* prev;
  Node(int data)
              {
              this->data = data;
              this->next = NULL;
              this->prev = NULL;
              }

};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList()
              { head = NULL;
                tail = NULL;
              }

    void insertAtStart(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void insertAtEnd(int val) {
        Node* newNode = new Node(val);
        if (!tail) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void insertAtPosition(int val, int position) {
        if (position < 1) {
            cout << "Position Invalid." <<endl;
            return;
        }
        Node* newNode = new Node(val);
        if (position == 1) {
            insertAtStart(val);
        } else {
            Node* current = head;
            int currentPosition = 1;

            while (current && currentPosition < position - 1) {
                current = current->next;
                currentPosition++;
            }
            if (!current) {
                cout << "Invalid Position." <<endl;
                delete newNode;
                return;
            }
            newNode->next = current->next;
            newNode->prev = current;
            if (current->next) {
                current->next->prev = newNode;
            }
            current->next = newNode;
        }
    }

    void deleteFromStart() {
        if (!head) {
            cout << "List is empty." <<endl;
            return;
        }

        Node* temp = head;
        head = head->next;
        if (head) {
            head->prev = NULL;
        } else {
            tail = NULL;
        }
        delete temp;
    }
```

```
void deleteFromEnd() {
    if (!tail) {
        cout << "List is empty." <<endl;
        return;
    }

    Node* temp = tail;
    tail = tail->prev;
    if (tail) {
        tail->next = NULL;
    } else {
        head = NULL;
    }
    delete temp;
}

void printList() {
    Node* current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout <<endl;
}
int count(){
          int cc=0;
          Node* front=head;
                        if (!head) {
        cout << "List is empty." <<endl;
        exit;
    }else{
       while(front){
                 front =front->next;
                 cc++;
                           }
                 }
       return cc;
       }
void insertm(int data){
       int half =count()/2;
       Node* position=head;
       if (!head) {
     cout << "List is empty." <<endl;
     return;
   }
   else{

       for(int i=1;i<half-1;i++){
                 position=position->next;
                           }
                           Node* newnode=new Node(data);
                  newnode->next = position->next;
     newnode->prev = position;
     if (position->next) {
        position->next->prev = newnode;
     }
     position->next = newnode;
                 }
       }
void printreverseList() {
   Node* reverse = tail;
```

```
        while (reverse) {
            cout << reverse->data << " ";
            reverse = reverse->prev;
        }
        cout <<endl;
    }
};

int main() {
    DoublyLinkedList Dlist;

    Dlist.insertAtStart(9);
    Dlist.insertAtEnd(11);
    Dlist.insertAtStart(5);
    Dlist.insertAtPosition(7, 2);
    Dlist.printList();
    cout<<"Reverse print of Doubly LinkList"<<endl;
    Dlist.printreverseList();
    Dlist.deleteFromStart();
    Dlist.deleteFromEnd();
    Dlist.printList();
    Dlist.insertAtEnd(15);
    Dlist.insertAtPosition(14, 2);
    Dlist.printList();
    int count=Dlist.count();
    cout<<"Size of list "<<count<<endl;
    Dlist.insertm(1);
      Dlist.printList();
    Dlist.insertm(2);
      Dlist.printList();
    Dlist.insertm(3);
```

```
      Dlist.printList();
    Dlist.insertm(4);
      Dlist.printList();
    Dlist.printList();
    return 0;
}
```



C:\Assignmenst\DSA\Lab 10\insertm.exe

```
5 7 9 11
Reverse print of Doubly LinkList
11 9 7 5
7 9
7 14 9 15
Size of list 4
7 1 14 9 15
7 2 1 14 9 15
7 2 3 1 14 9 15
7 2 4 3 1 14 9 15
7 2 4 3 1 14 9 15

--------------------------------
Process exited after 0.1022 seconds with return value 0
Press any key to continue . . .
```

Q. Merge Lists

```
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node* prev;
  Node(int data)
            {
            this->data = data;
            this->next = NULL;
            this->prev = NULL;
            }
```

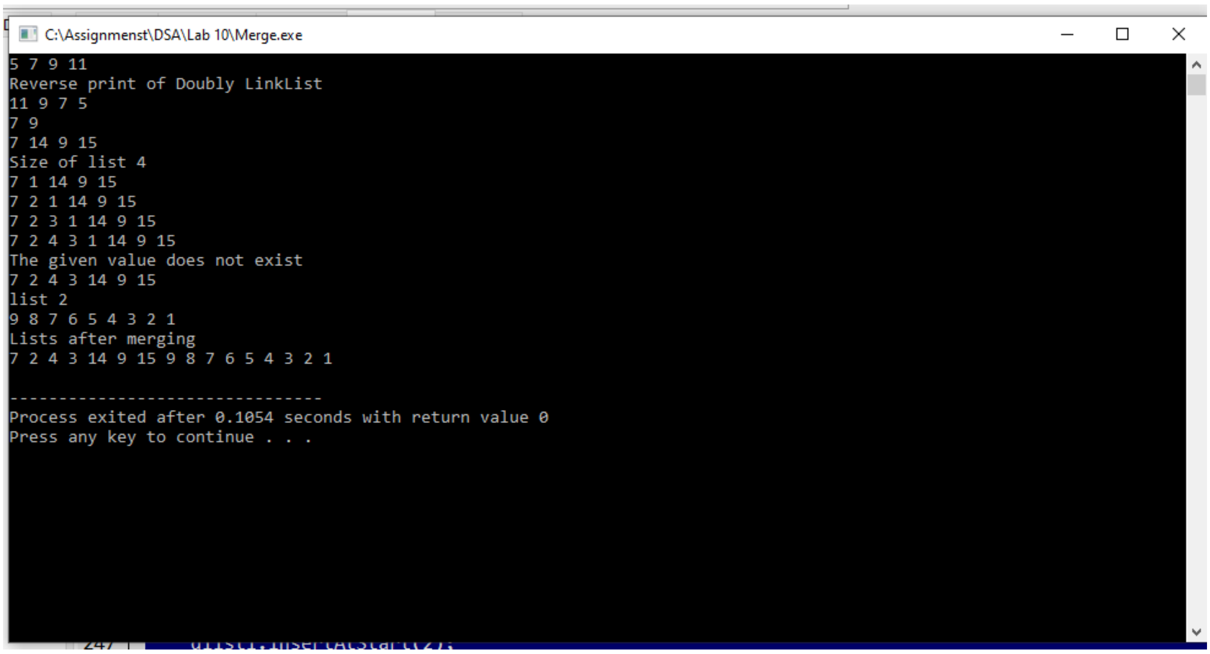
```
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList()
              { head = NULL;
                tail = NULL;
              }

    void insertAtStart(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void insertAtEnd(int val) {
        Node* newNode = new Node(val);
        if (!tail) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void insertAtPosition(int val, int position) {
        if (position < 1) {
            cout << "Position Invalid." <<endl;
            return;
        }
        Node* newNode = new Node(val);
        if (position == 1) {
            insertAtStart(val);
        } else {
            Node* current = head;
            int currentPosition = 1;

            while (current && currentPosition < position - 1) {
                current = current->next;
                currentPosition++;
            }
            if (!current) {
                cout << "Invalid Position." <<endl;
                delete newNode;
                return;
            }
            newNode->next = current->next;
            newNode->prev = current;
            if (current->next) {
                current->next->prev = newNode;
            }
            current->next = newNode;
        }
    }

    void deleteFromStart() {
        if (!head) {
            cout << "List is empty." <<endl;
            return;
        }

        Node* temp = head;
        head = head->next;
        if (head) {
            head->prev = NULL;
        } else {
            tail = NULL;
        }
        delete temp;
    }

    void deleteFromEnd() {
        if (!tail) {
            cout << "List is empty." <<endl;
            return;
        }

        Node* temp = tail;
        tail = tail->prev;
        if (tail) {
            tail->next = NULL;
        } else {
          head = NULL;
       }
       delete temp;
    }

    void printList() {
       Node* current = head;
       while (current) {
          cout << current->data << " ";
          current = current->next;
       }
       cout <<endl;
    }
    int count(){
            int cc=0;
            Node* front=head;
                      if (!head) {
          cout << "List is empty." <<endl;
          exit;
       }else{

            while(front){
                      front =front->next;
                      cc++;
                                }
                      }
            return cc;
            }
    void insertm(int data){
            int half =count()/2;
            Node* position=head;
```

```
          if (!head) {
        cout << "List is empty." <<endl;
        return;
    }
    else{


          for(int i=1;i<half-1;i++){
                         position=position->next;
                                        }
                                        Node* newnode=new Node(data);
                           newnode->next = position->next;
        newnode->prev = position;
        if (position->next) {
            position->next->prev = newnode;
        }
        position->next = newnode;
                         }
           }
void printreverseList() {
    Node* reverse = tail;
    while (reverse) {
        cout << reverse->data << " ";
        reverse = reverse->prev;
    }
    cout <<endl;
}
 void deletevalue(int data) {
    if (!head) {
        cout << "List is empty." <<endl;
        return;
    }
```

```
  Node* temp = head;
 int cc=count();

  if(!head){
      cout << "List is empty." <<endl;
    return;

       }

       else{
       while(temp){

  if (temp->data==data) {
      temp->prev->next=temp->next;
    temp->next=NULL;
      temp->prev=NULL;

      return;
  }
  else if(temp==tail){
      cout<<"The given value does not exist"<<endl;
            }
      temp=temp->next;



             }
delete temp;
}
      return;
```

```
        }
        Node* acctail(){
                return tail;
        }
        Node* acchead(){
                return head;
        }
};

void merge(DoublyLinkedList L1,DoublyLinkedList L2){
Node* temp1;
Node* temp2;
temp1=L1.acctail();
temp2=L2.acchead();
        temp1->next=temp2;
        temp2->prev=temp1;
}
int main() {
  DoublyLinkedList Dlist;
DoublyLinkedList dlist1;
  Dlist.insertAtStart(9);
  Dlist.insertAtEnd(11);
  Dlist.insertAtStart(5);
  Dlist.insertAtPosition(7, 2);
  Dlist.printList();
  cout<<"Reverse print of Doubly LinkList"<<endl;
  Dlist.printreverseList();
  Dlist.deleteFromStart();
  Dlist.deleteFromEnd();
  Dlist.printList();
  Dlist.insertAtEnd(15);
```

```
    Dlist.insertAtPosition(14, 2);
    Dlist.printList();
    int count=Dlist.count();
    cout<<"Size of list "<<count<<endl;
    Dlist.insertm(1);
      Dlist.printList();
    Dlist.insertm(2);
      Dlist.printList();
    Dlist.insertm(3);
      Dlist.printList();
    Dlist.insertm(4);
      Dlist.printList();
      Dlist.deletevalue(18);
         Dlist.deletevalue(1);
    Dlist.printList();
    dlist1.insertAtStart(1);
    dlist1.insertAtStart(2);
    dlist1.insertAtStart(3);
    dlist1.insertAtStart(4);
    dlist1.insertAtStart(5);
    dlist1.insertAtStart(6);
    dlist1.insertAtStart(7);
    dlist1.insertAtStart(8);
    dlist1.insertAtStart(9);
    cout<<"list 2"<<endl;
    dlist1.printList();
    merge(Dlist,dlist1);
    cout<<"Lists after merging"<<endl;
    Dlist.printList();
    return 0;
}
```



```
5 7 9 11
Reverse print of Doubly LinkList
11 9 7 5
7 9
7 14 9 15
Size of list 4
7 1 14 9 15
7 2 1 14 9 15
7 2 3 1 14 9 15
7 2 4 3 1 14 9 15
The given value does not exist
7 2 4 3 14 9 15
list 2
9 8 7 6 5 4 3 2 1
Lists after merging
7 2 4 3 14 9 15 9 8 7 6 5 4 3 2 1

--------------------------------
Process exited after 0.1054 seconds with return value 0
Press any key to continue . . .
```

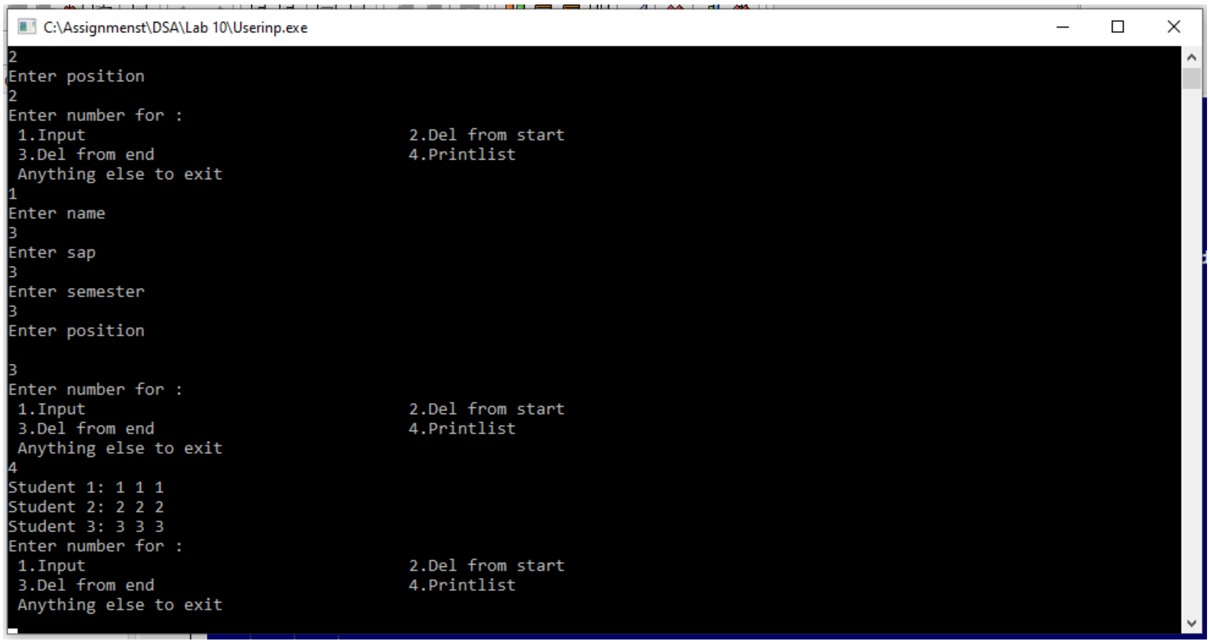
Q.User inp

```
#include <iostream>

using namespace std;

class Node {

public:

   int sem,sap;

   string name;

   Node* next;

   Node* prev;

  Node(int sem,int sap,string name)

        {

        this->name=name;

        this->sap=sap;

        this->sem = sem;

        this->next = NULL;

        this->prev = NULL;

        }


};
```

```
class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList()
              { head = NULL;
                tail = NULL;
              }

    void insertAtStart(int sem,int sap,string name) {
        Node* newNode = new Node(sem,sap,name);
        if (!head) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
        return;
    }

    void insertAtEnd(int sem,int sap,string name) {
        Node* newNode = new Node(sem,sap,name);
        if (!tail) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;

        }
        return;
    }

    void insertAtPosition(int sem,int sap,string name, int position) {
        if (position < 1) {
            cout << "Position Invalid." <<endl;
            return;
        }
        Node* newNode = new Node(sem,sap,name);
        if (position == 1) {
            insertAtStart(sem,sap,name);
        } else {
            Node* current = head;
            int currentPosition = 1;

            while (current && currentPosition < position - 1) {
                current = current->next;
                currentPosition++;
            }
            if (!current) {
                cout << "Invalid Position." <<endl;
                delete newNode;
                return;
            }
            newNode->next = current->next;
            newNode->prev = current;
            if (current->next) {
                current->next->prev = newNode;
            }
            current->next = newNode;
        }
        return;
    }

    void deleteFromStart() {
        if (!head) {
            cout << "List is empty." <<endl;
            return;
        }

        Node* temp = head;
        head = head->next;
        if (head) {
            head->prev = NULL;
        } else {
            tail = NULL;
        }
        delete temp;
        return;
    }

    void deleteFromEnd() {
        if (!tail) {
            cout << "List is empty." <<endl;
            return;
        }


                        Node* temp = tail;
```

```
   tail = tail->prev;
   if (tail) {
      tail->next = NULL;
   } else {
      head = NULL;
   }
                  delete temp;



return;
        }


void printList() {
          if (!head) {
      cout << "List is empty." <<endl;
      return;
   }
   else{
                  Node* current = head;
  int cc=1;
   while (current) {
         cout << "Student "<<cc<<": " ;
      cout <<current->sem << " ";
       cout<<current->sap << " ";
        cout<<current->name << " ";
      cout <<endl;
      cc++;
      current = current->next;
   }}
   }
};

int main() {
    DoublyLinkedList lis;
    int choice=1;
    string name;
    int sap,sem,pos;
        lis.insertAtPosition(1,51,"ab",1);
            lis.insertAtPosition(2,52,"ac",2);

lis.deleteFromEnd(); // Check state after deletion

    while(choice>0&&choice<5){
        cout<<"Enter number for :"<<"\n 1.Input \t\t\t\t 2.Del from start \n 3.Del from end \t\t\t 4.Printlist \n Anything else to exit"<<endl;
        cin>>choice;
        switch(choice){
            case 1:
                cout<<"Enter name"<<endl;
                cin.ignore();
                getline(cin,name);
                cout<<"Enter sap"<<endl;
                cin>>sap;
                cout<<"Enter semester"<<endl;
                cin>>sem;
                cout<<"Enter position"<<endl;
                cin>>pos;
                lis.insertAtPosition(sem,sap,name,pos);
            break;
            case 2:
```

```
                                        lis.deleteFromStart();
                                break;
                                case 3:
                                        lis.deleteFromEnd();
                                break;
                                case 4:
                                        lis.printList();
                                break;
                                default:
                                        cout<<"\nExiting"<<endl;
                                break;
                        }
                }

  return 0;
}
```



```
2
Enter position
2
Enter number for :
 1.Input                                 2.Del from start
 3.Del from end                          4.Printlist
 Anything else to exit
1
Enter name
3
Enter sap
3
Enter semester
3
Enter position

3
Enter number for :
 1.Input                                 2.Del from start
 3.Del from end                          4.Printlist
 Anything else to exit
4
Student 1: 1 1 1
Student 2: 2 2 2
Student 3: 3 3 3
Enter number for :
 1.Input                                 2.Del from start
 3.Del from end                          4.Printlist
 Anything else to exit
```