```
#include <iostream>
#include <cstdlib>
#include<string.h>
using namespace std;



class node{

      public:
         int data;
         node *next, *prev;

         node(){
             next=NULL;
             prev=NULL;
             data=0;
         }

         node(int i, node *in, node *ip) {
             data = i;
               next = in;
               prev=ip;
         }

             node(int i){
              data=i;
              next=NULL;
              prev=NULL;
         }
};


class list
{
    private:
         node *head, *tail;

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

      list(const list &rhs){   //copy constructor
             this->head=NULL;
           this->tail=NULL;
           node *q=rhs.head;
           while(q!=NULL){
               add_node_tail(q->data);
               q=q->next;
           }
      }

      list& operator=(const list& rhs){    //assignment operator
           if(this!=&rhs){
                     node *ptr;
                     /*instead of writing this line and while loop
                     we can simply use destructor as this->~list();*/
                while(this->head!=NULL){
```

```
            ptr=this->head->next;
            delete this->head;
            this->head=ptr;
        }

        this->head=NULL;
      this->tail=NULL;
      node *q=rhs.head;
      while(q!=NULL){
          add_node_tail(q->data);
          q=q->next;
      }
    }
              return *this;


}

void add_node_head(int n)
{
    if(head == NULL)  {
       head=tail=new node(n,NULL,NULL);
    }
    else{
       head=new node(n,head,NULL);
       head->next->prev=head;
    }
}



    void add_node_tail(int n){
//add to tail
       if(tail!=NULL)
    {
        tail= new node(n,NULL,tail);
        tail->prev->next=tail;
       }
    else{
        head=tail=new node(n);
       }
}


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

void delete_head() {           //delete first node
    if (head!=NULL){
        int delnode = head->data;
        node *tmp = head;
        if (head == tail){
            head = tail = NULL;
        }
        else{
            head = head->next;
            delete tmp;
            head->prev=NULL;
        }

    }
    else{
        cout<<"list is empty";
    }
}


void delete_tail() {    //delete last node
    int delnode = tail->data;
    if (head == tail) {
        delete head;
        head=tail=NULL;
    }
    else{
        tail=tail->prev;
        delete tail->next;
        tail->next=NULL;

    }
}

~list(){               //destructor
    node *ptr;
    while(head!=NULL){
        ptr=head->next;
        delete head;
        head=ptr;
    }
}

 bool searching(int a){
    node *tmp;
    for(tmp=head;tmp!=0 && tmp->data!=a;tmp=tmp->next);
    return tmp!=0;
}

void add_somewhere(int n, int a){
    if(head==tail){
        head=tail=new node(n,NULL,NULL);

    }
    else{
        node *tmp, *place;
        for(tmp=head;tmp!=0;tmp=tmp->next){
            if(tmp->data==a){
                place=new node(n,tmp->next,tmp);
                if(tmp->next!=NULL){
                    tmp->next->prev=place;
                }
                else{
                    tail=place;
                }
                tmp->next=place;
```

```
                    }


                }

            }

        }

        void reversing(){
            node *temp = NULL;
            node *current = head;

            while (current !=  NULL)
            {
                temp = current->prev;
                current->prev = current->next;
                current->next = temp;
                current = current->prev;
            }


            if(temp != NULL ) {
                head = temp->prev;
            }
        }
};

int main()
{
    list a;
    a.add_node_head(1);
    //adding nodes to a
      a.add_node_head(2);
    a.add_node_tail(4);
    a.add_node_tail(5);
    a.display();
    a.add_somewhere(8,5);
    a.display();
    a.reversing();



    list b(a);  //case of copy constructor
      b.display();
      list c;
      c.add_node_head(6);
      c.add_node_head(7);
      c=a;        //case of operator assignment
      c.display();


      return 0;

}
```