DATA STRUCTURES AND ALGORITHMS LAB
BCSE202P

DIGITAL ASSIGNMENT – 3

ABHINAV LV – 21BCE0192

Node Structure and Linked List Class:

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

struct node{
    int val;
    struct node *next;
};

class LinkedList{
    node *head = NULL;
public:
    void append(int e){
        if(!head){
            head = new node;
            head->val = e;
            head->next = NULL;
            return;
        }
        node *newNode = new node;
        newNode->val = e;
        newNode->next = NULL;

        node *temp;
        for(temp=head;temp->next;temp=temp->next);

        temp->next = newNode;
    }

    void dis(){
        cout<<"\nLinked List: ";
        node *temp;
        for(temp=head;temp->next;temp=temp->next){
            cout<<temp->val<<" --> ";
        }
        cout<<temp->val<<endl;
    }
};
```

Question 1:

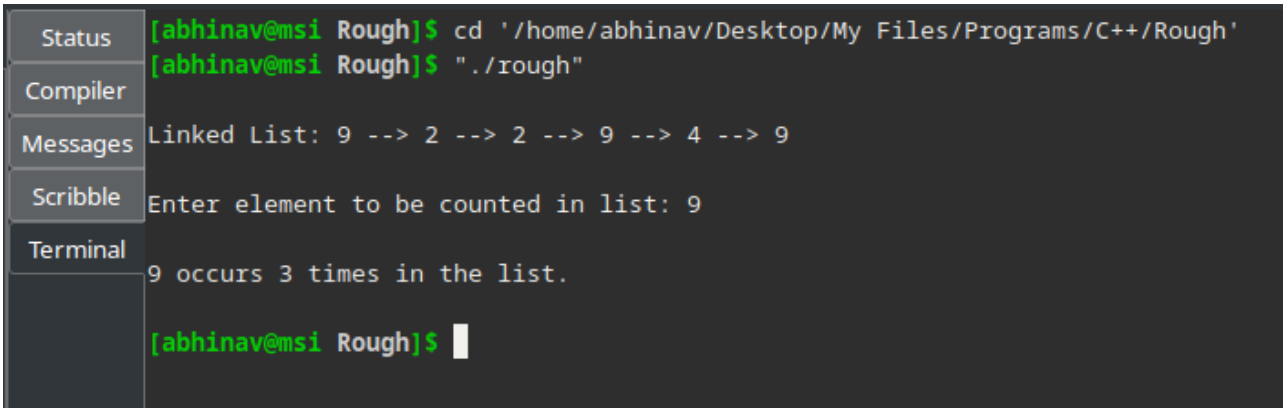
1. Write a Count() function that counts the number of times a given int occurs in a linked list. The code for this has the classic list traversal structure as demonstrated in Length().

Code:

```
    //Part of LinkedList class
    void count(){
        node *temp;
        int c=0,e;

        cout<<"\nEnter element to be counted in list: ";
        cin>>e;

        for(temp=head;temp;temp=temp->next)
            if(temp->val == e) c++;

        if(!c) cout<<"\n"<<e<<" does not occur in the list."<<endl;
        else if(c==1) cout<<"\n"<<e<<" occurs "<<c<<" time in the list."<<endl;
        else cout<<"\n"<<e<<" occurs "<<c<<" times in the list."<<endl;
    }
```

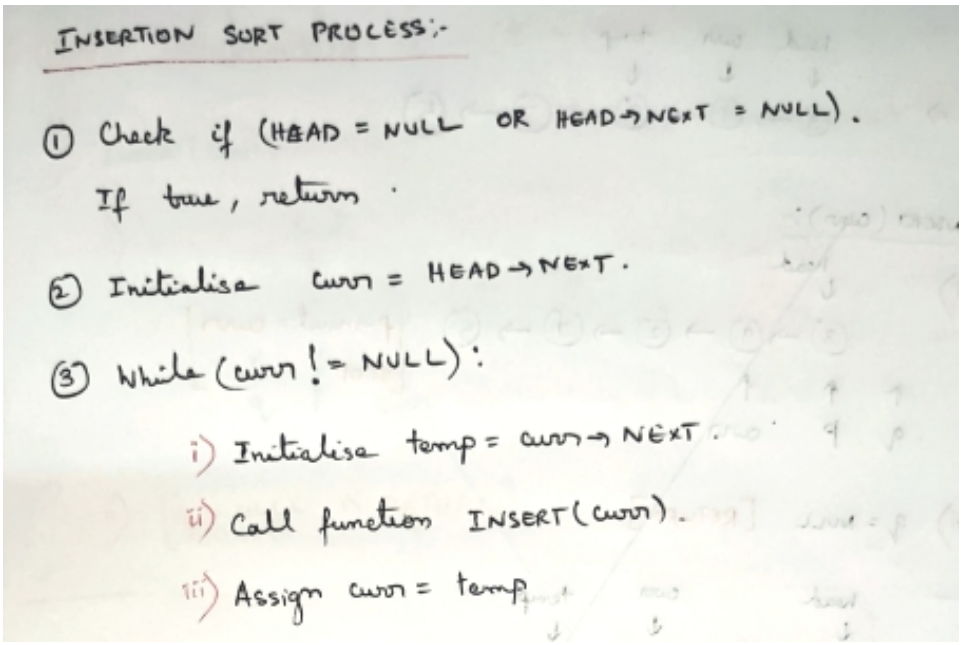
```
int main(){

    LinkedList lst;
    lst.append(3);
    lst.append(2);
    lst.append(2);
    lst.append(3);
    lst.append(3);
    lst.append(4);
    lst.dis();

    lst.count();

    cout<<endl;
    return 0;
}
```

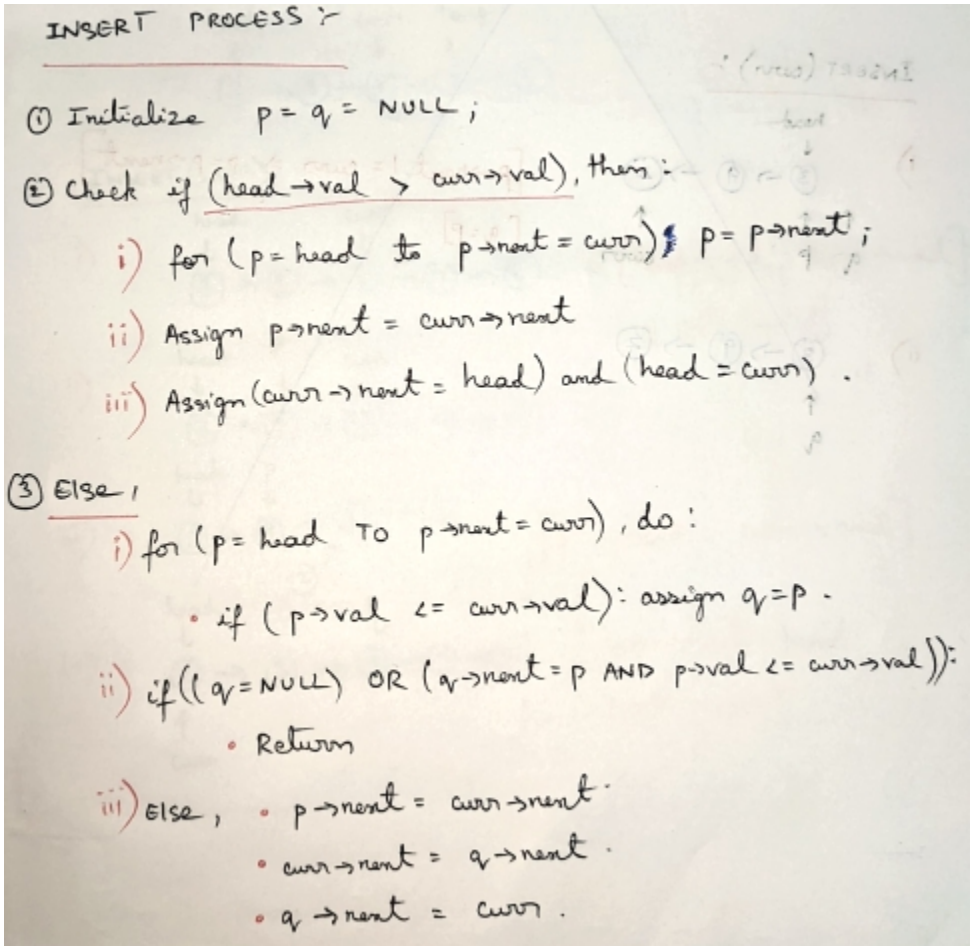
Output:

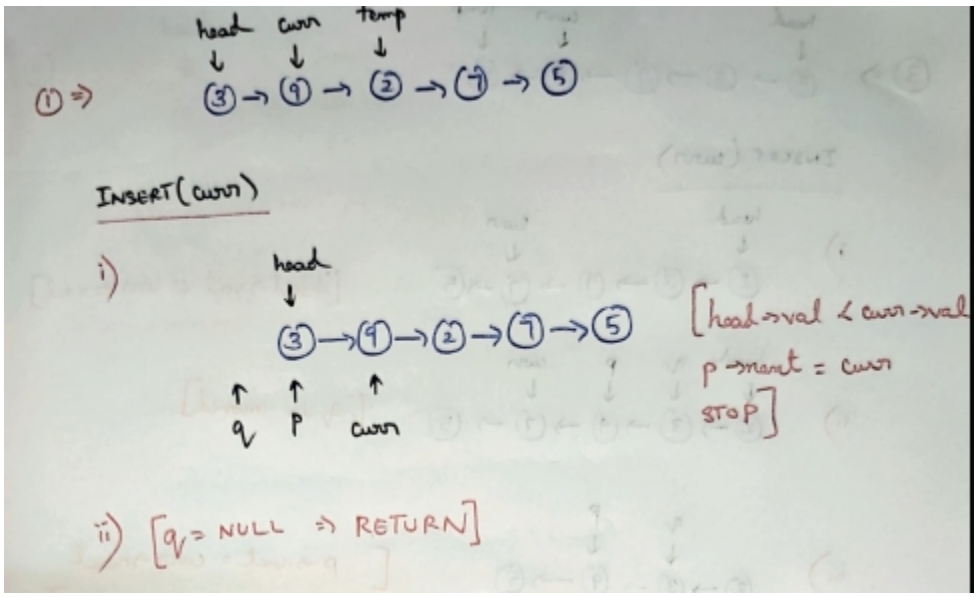

Status
Compiler
Messages
Scribble
Terminal
[abhinav@msi Rough]$ cd '/home/abhinav/Desktop/My Files/Programs/C++/Rough'
[abhinav@msi Rough]$ "./rough"
Linked List: 9 --> 2 --> 2 --> 9 --> 4 --> 9
Enter element to be counted in list: 9
9 occurs 3 times in the list.
[abhinav@msi Rough]$

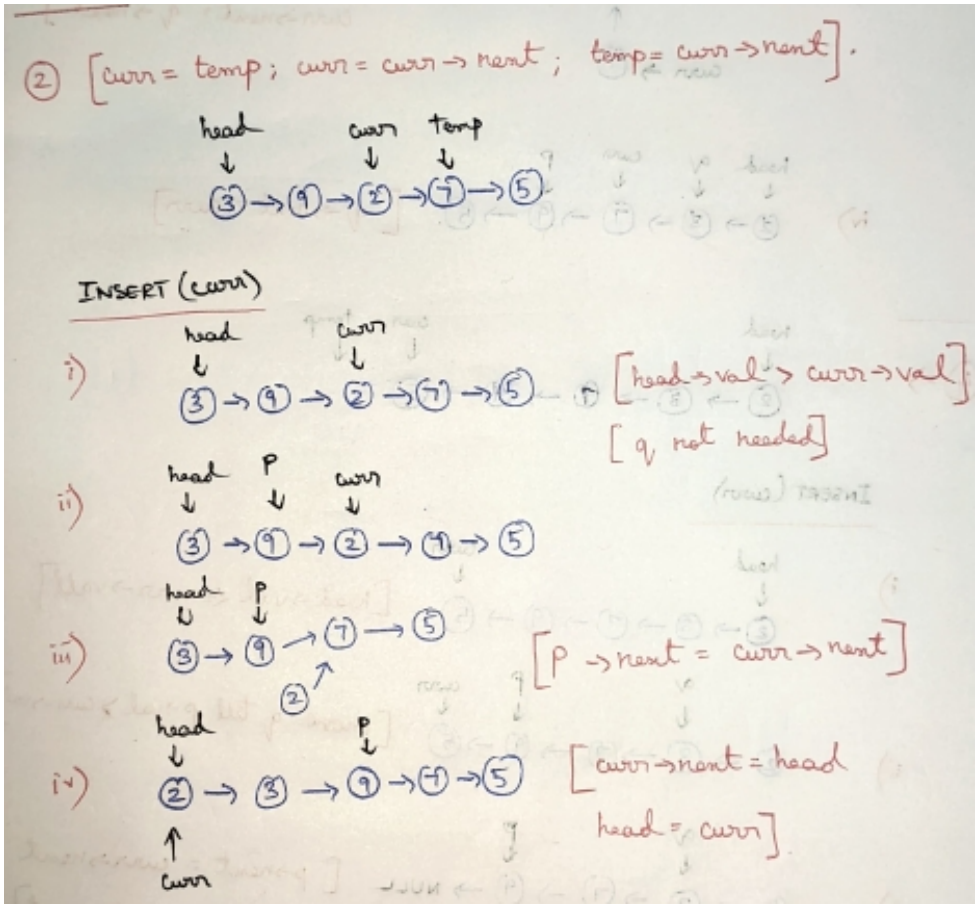
Question 2:

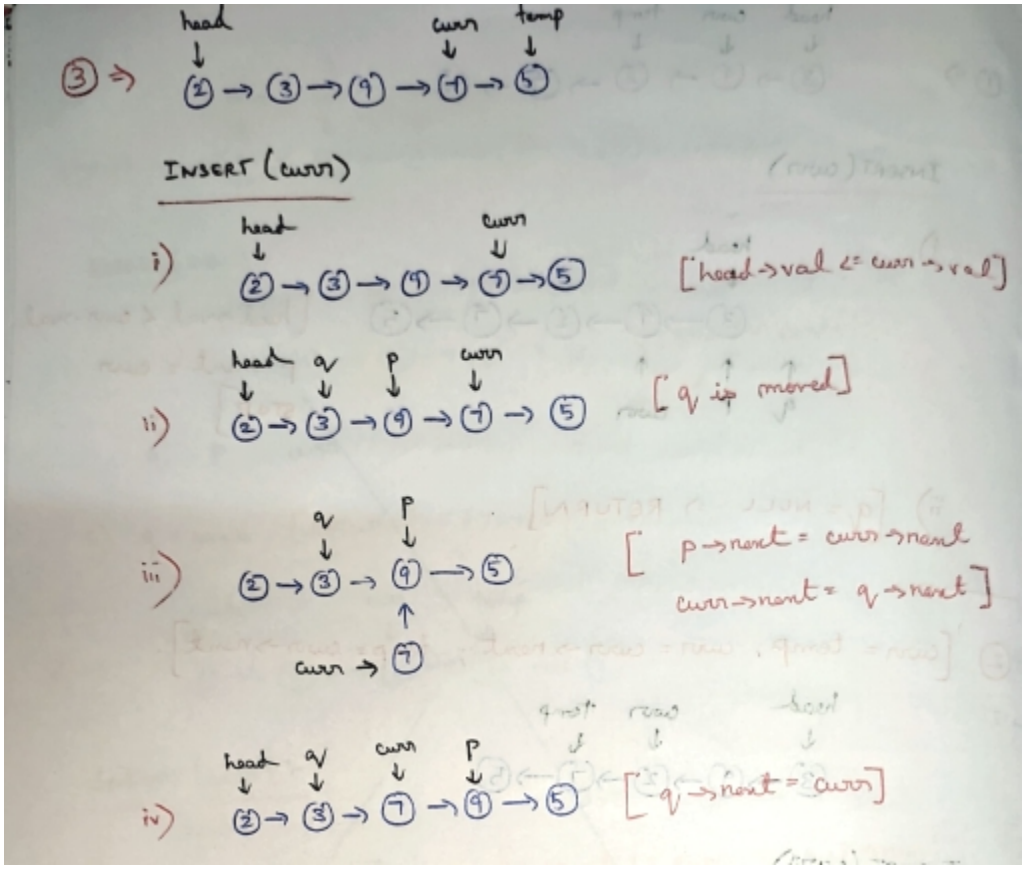
2. Implement InsertionSort() algorithm on a linked list and explain the steps before writing the code.

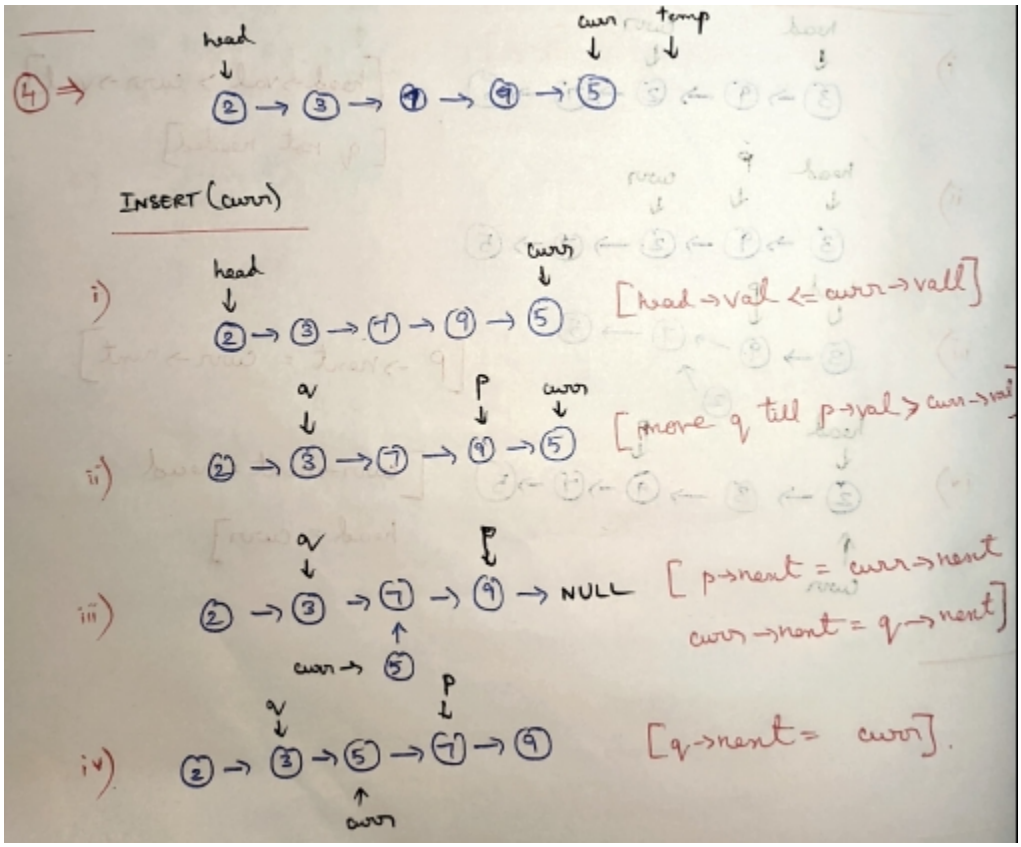


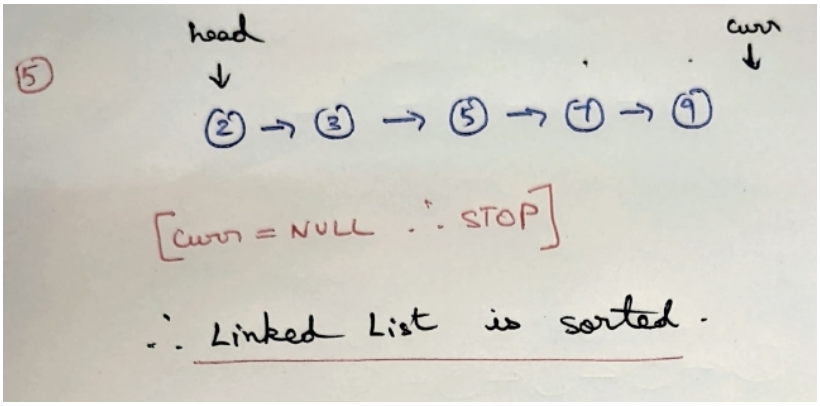



head
curr
temp
(1) =>
3 → 9 → 2 → 7 → 5
INSERT(curr)
i)
head
3 → 9 → 2 → 7 → 5
q
p
curr
[head->val < curr->val
p->next = curr
STOP]
ii) [q = NULL => RETURN]


(2) [curr = temp ; curr = curr -> next ; temp = curr -> next].
head
curr
temp
3 → 9 → 2 → 7 → 5
INSERT (curr)
i)
head
curr
3 → 9 → 2 → 7 → 5
[head->val > curr->val]
[q not headed]
ii)
head
p
curr
3 → 9 → 2 → 7 → 5
iii)
head
p
3 → 9 → 7 → 5
2
[p->next = curr->next]
iv)
head
p
2 → 3 → 9 → 7 → 5
curr
[curr->next = head
head = curr]


head
curr
temp
INSERT (curr)
i)
[head->val <= curr->val]
ii)
q
p
[q is moved]
iii)
[p->next = curr->next
curr->next = q->next]
iv)
[q->next = curr]


head
curr
temp
INSERT (curr)
i)
[head->val <= curr->val]
ii)
[move q till p->val > curr->val]
iii)
NULL
[p->next = curr->next
curr->next = q->next]
iv)
[q->next = curr].

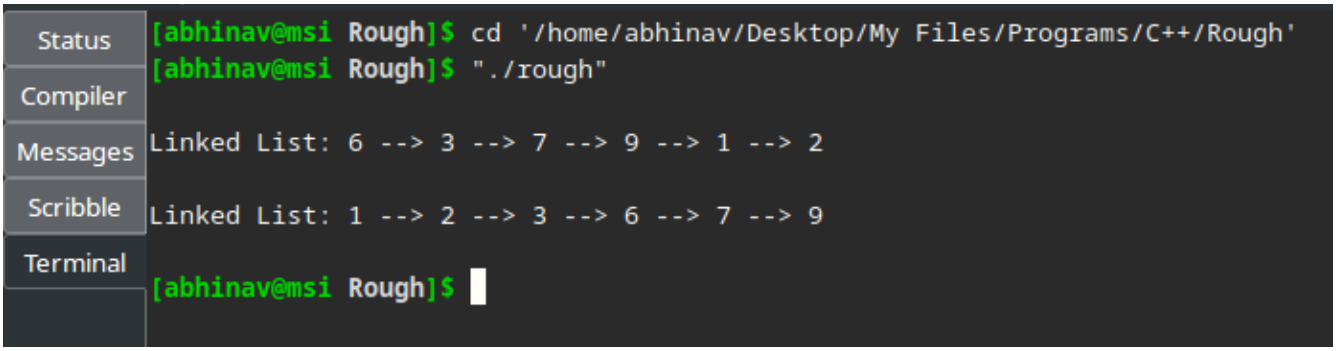


Code:

```
void insert(node *stop){
    node *p, *q = NULL;
    if(head->val > stop->val){
        for(p=head;p->next!=stop;p=p->next);
        p->next = stop->next;
        stop->next = head;
        head = stop;
        return;
    }

    //find element before stop
    for(p=head;p->next!=stop;p=p->next) {
        if(p->val <= stop->val) q=p;
    }
    if(!q || (q->next == p && p->val <= stop->val)) return;

    p->next = stop->next;
    stop->next = q->next;
    q->next = stop;
}

void insertionSort(){
    if(!head || !head->next) return;

    node *stop = head->next;
    while(stop!=NULL){
        node *temp = stop->next;
        insert(stop);
        stop = temp;
    }
}
```

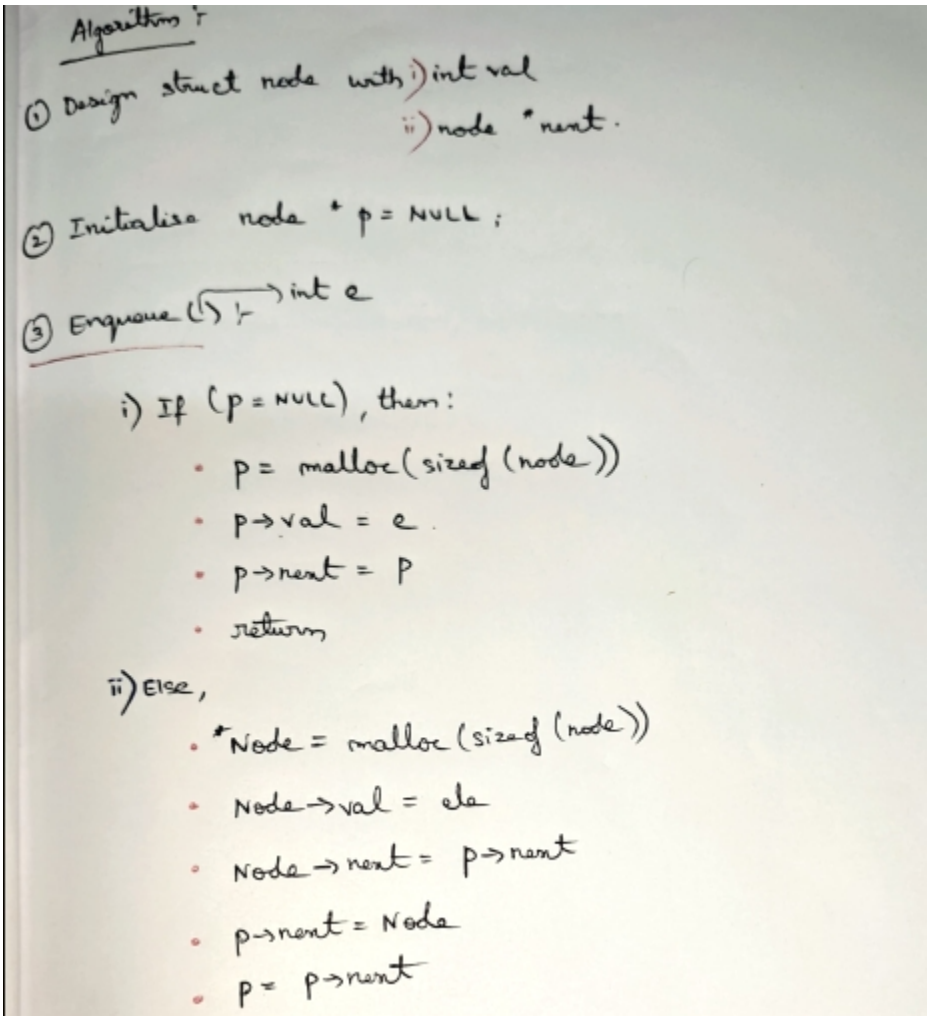
```
int main(){

    LinkedList lst;
    lst.append(6);
    lst.append(3);
    lst.append(7);
    lst.append(9);
    lst.append(1);
    lst.append(2);
    lst.dis();

    lst.insertionSort();
    lst.dis();

    cout<<endl;
    return 0;
}
```

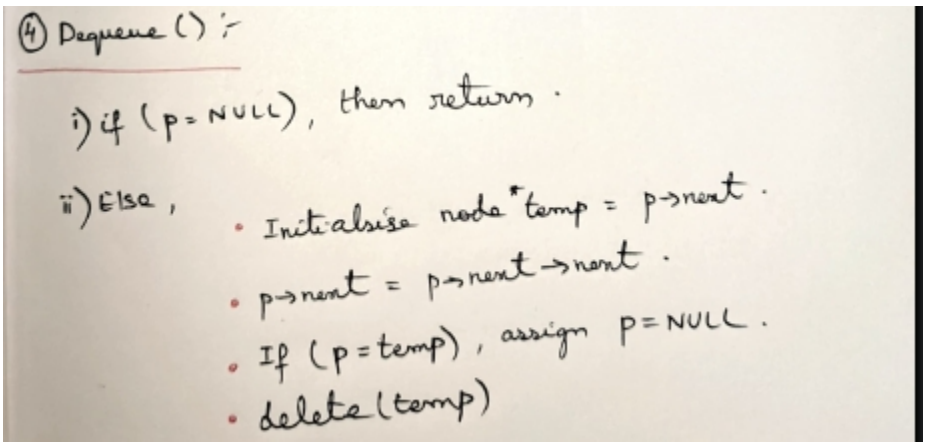
Output:



Question 3:

3. Circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time? Explain the same with the relevant code and description.





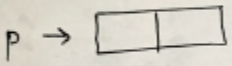
For performing enQueue() and deQueue() in constant time, p should point to the last element of the circular linked list.

∴ 'p' points to last, 'p→next' points to first node.

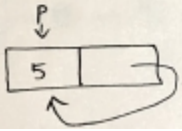
① Empty list: p → NULL

② enQueue(5): [First element]

i) make new node and make p point to it.

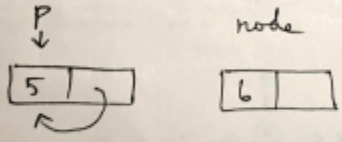


ii) Assigne node→val = 5 and since its circular linked list, node→next = node.

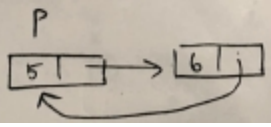


③ enQueue(6):

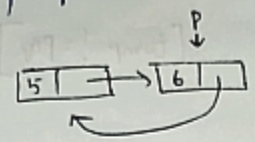
i) make new node and assign node→val = 6.



ii) node→next = p→next, AND p→next = node

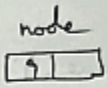


iii) p = p → next

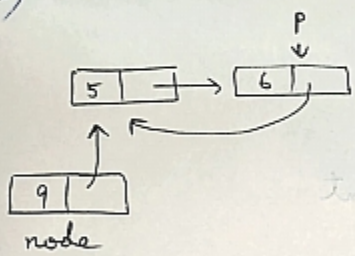


④ enQueue (9) :

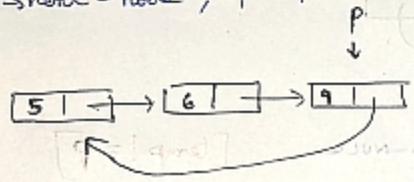
i) make new node with value 9



ii) node → next = p → next

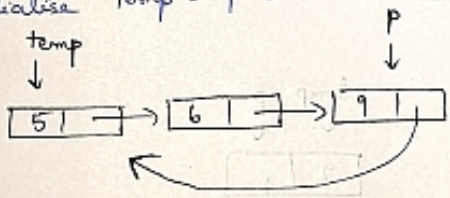


iii) p → next = node , p = p → next

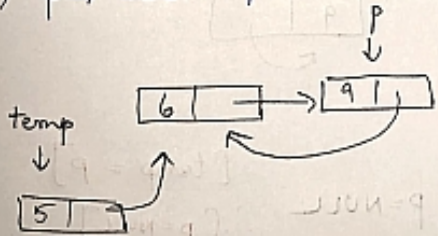


⑤ deQueue ( ) :

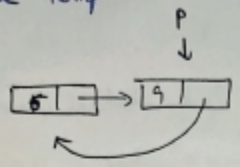
i) Initialise temp = p → next



ii) p → next = p → next → next

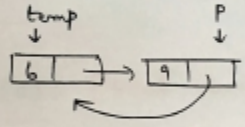


iii) If (temp = p), then assign p = NULL.
~~Else~~ Delete temp. [temp != p]

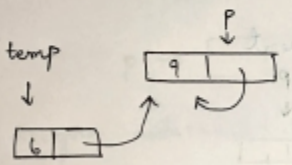


(6) deQueue():

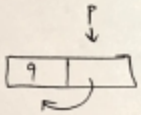
i) temp = p->next



ii) p->next = p->next->next



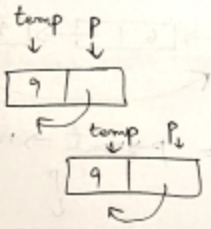
iii) If (temp = p), p = NULL [temp != p]
Delete temp.



(7) deQueue() :-

i) temp = p->next

ii) p->next = p->next->next



iii) If (temp = p) p = NULL;
Delete temp.

[temp = p]
p = NULL
∴ [p = NULL]

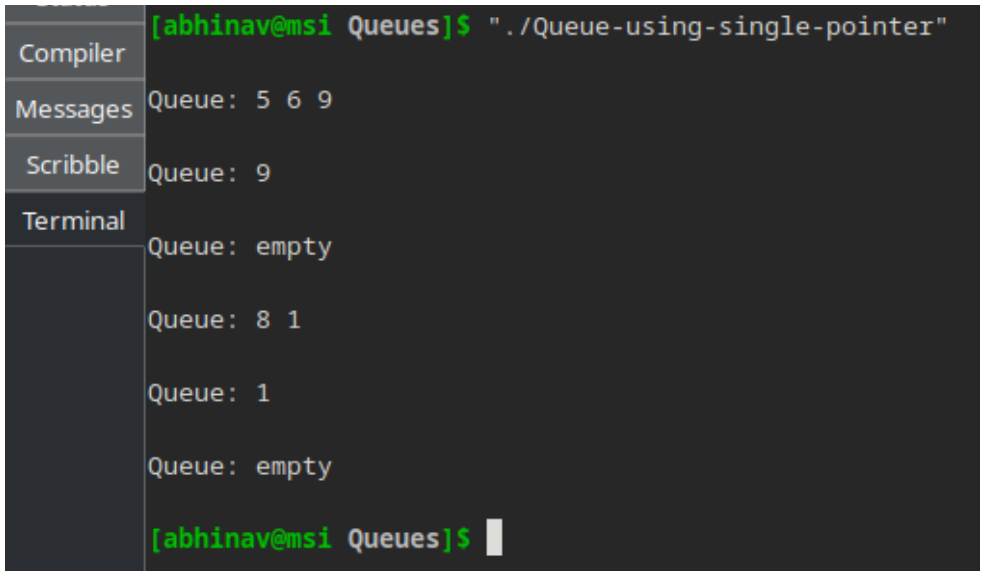
[p = NULL ⇒ Queue is empty].

Code:

```
4   struct node{
5       int val;
6       node *next;
7   };
8
9   class Queue{
10      node *p = NULL;
11
12  public:
13      void enqueue(int ele){
14          if(!p){
15              p = new node;
16              p->val = ele;
17              p->next = p;
18              return;
19          }
20
21          node *newnode = new node;
22          newnode->val = ele;
23          newnode->next = p->next;
24          p->next = newnode;
25          p = p->next;
26      }
27
28      void dequeue(){
29          if(!p) return;
30
31          node *temp = p->next;
32          p->next = p->next->next;
33
34          if(temp==p) p = NULL;
35          delete temp;
36      }
37
38      void dis(){
39          node *temp;
40          cout<<"\nQueue: ";
41          if(!p) {cout<<"empty"<<endl; return;}
42
43          for(temp=p->next;temp!=p;temp=temp->next) cout<<temp->val<<" ";
44          cout<<temp->val<<endl;
45      }
46  };
```

```
47
48  int main(){
49
50      Queue q;
51      q.enqueue(5);
52      q.enqueue(6);
53      q.enqueue(9);
54      q.dis();
55
56      q.dequeue();
57      q.dequeue();
58      q.dis();
59
60      q.dequeue();
61      q.dis();
62
63      q.enqueue(8);
64      q.enqueue(1);
65      q.dis();
66
67      q.dequeue();
68      q.dis();
69
70      q.dequeue();
71      q.dis();
72
73      cout<<endl;
74      return 0;
75  }
76
```

Output:



Question 4:

4. Write a program to remove duplicates from a sorted linked list.

Code:

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

struct node{
    int val;
    node *prev, *next;
};

class DoublyLinkedList{
    node *head = NULL;

public:
    void append(int e){
        if(!head){
            head = new node;
            head->val = e;
            head->next = head->prev = NULL;
            return;
        }

        node *newnode = new node, *temp;
        newnode->val = e;
        newnode->next = NULL;

        for(temp=head;temp->next;temp=temp->next);
        temp->next = newnode;
        newnode->prev = temp;
    }

    void removeDuplicates(){
        if(!head) return;
        node *temp;

        for(temp=head; temp->next; temp=temp->next){
            if(temp->val == temp->next->val){

                temp->next->prev = temp->prev;
                if(temp == head) head = head->next;
                else temp->prev->next = temp->next;

                delete temp;
            }
        }
    }
```

```
46      void dis(){
47          node *temp;
48          cout<<"\nLinked List: ";
49          for(temp=head;temp->next;temp=temp->next) cout<<temp->val<<" <--> ";
50          cout<<temp->val<<endl;
51      }
52  };
53
54  int main(){
55
56      DoublyLinkedList l;
57      l.append(4);
58      l.append(4);
59      l.append(6);
60      l.append(7);
61      l.append(7);
62      l.append(8);
63      l.append(8);
64      l.append(8);
65      l.append(9);
66      l.append(9);
67      l.dis();
68
69      l.removeDuplicates();
70      l.dis();
71
72      cout<<endl;
73      return 0;
74  }
75
```

Output: