Data Structure & Algorithms

Sunbeam Infotech

Linked List → queue has a linked list in which add/del done from two different ends.

```
class node {
    ---
    ---
    ---
};
```

```
class singly_list {
    private:
        node * head;
    public:
        singly_list();
        void add_first(int);
        void add_last(int);
        void add_atpos(int, int);
        void del_first();
        void del_last();
        bool is_empty();
        int get_first();
        void del_all();
        ~singly_list();
};
```

```
class my_queue {
    private:
        singly_list l;
    public:
        void push(int ele) {
            l.add_last(ele);
        }
        void pop() {
            l.del_first();
        }
        bool is_empty() {
            return l.is_empty();
        }
        int peek() {
            return l.get_first();
        }
};
```

Linked List → queue is a special (restricted) linked list in which add/del done from two different ends.

```
class node {
    ——
    ——
    ——
};
```

```
class singly_list {
private:
    node * head;
public:
    singly_list();
    void add_first(int);
    void add_last(int);
    void add_atpos(int, int);
    void del_first();
    void del_last();
    bool is_empty();
    int get_first();
    void del_all();
    ~singly_list();
};
```

```
class my_queue : private singly_list {
public:
    void push(int ele) {
        add_last(ele);
    }
    void pop() {
        del_first();
    }
    bool is_empty() {
        return is_empty();
    }
    int peek() {
        return get_first();
    }
};
```

```
main() {
    my_queue q;
    q - push / pop / peek / Isempty()
}
```

Linked List → queue is a special (restricted) linked list in which add/del done from two different ends.

```
class node {
    ---
    ---
    ---
};
```

```
class singly_list {
private:
    node * head;
public:
    singly_list();
    void add_first(int);
    void add_last(int);
    void add_atpos(int, int);
    void del_first();
    void del_last();
    bool is_empty();
    int get_first();
    void del_all();
    ~singly_list();
};
```

```
class my_queue : private singly_list {
public:
    singly_list::add_last;
    singly_list::del_first;
    singly_list::is_empty;
    singly_list::get_first;
};
```

redeclaring members in derived class.

```
main() {
    my_queue q;
    q.add_last(-);
    q.del_first(-);
}
```

Linked List

```
class node {
    private:
        int data;
        node * next;
    public:
        node(){ }
        node(int) { }

    friend class singly_list;
};
```

```
class singly_list {
    private:
        node * head;
    public:
        singly_list();
        void add_first(int v) {
            node *nn = new node(v);
            nn->next = head;
            head = nn;
        }
        void display() {
            node * trav = head;
            while(trav != NULL) {
                cout << trav->data;
                trav = trav->next;
            }
        }
};
```

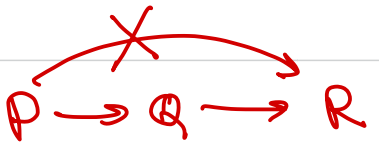
C++ friend keyword

friend fns

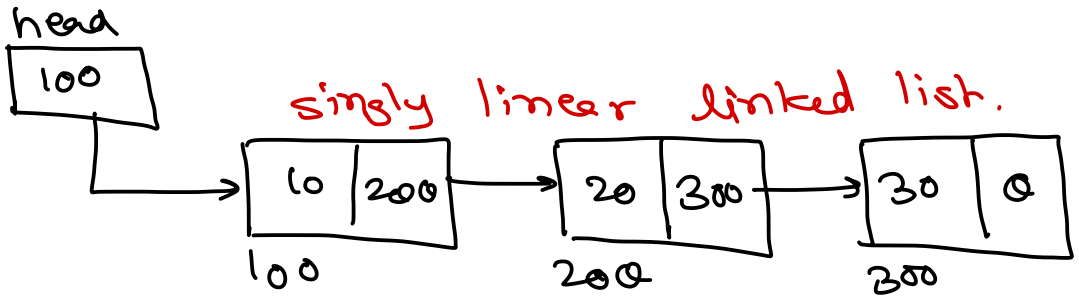
```
class X {
  private:
     int y;
  public:
     =
     =
   friend void fun();
};

void fun() {
    X obj;
    cout << obj.y;
}
```

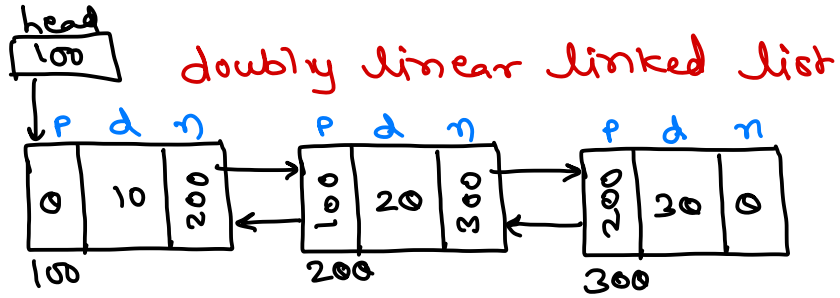
friend class

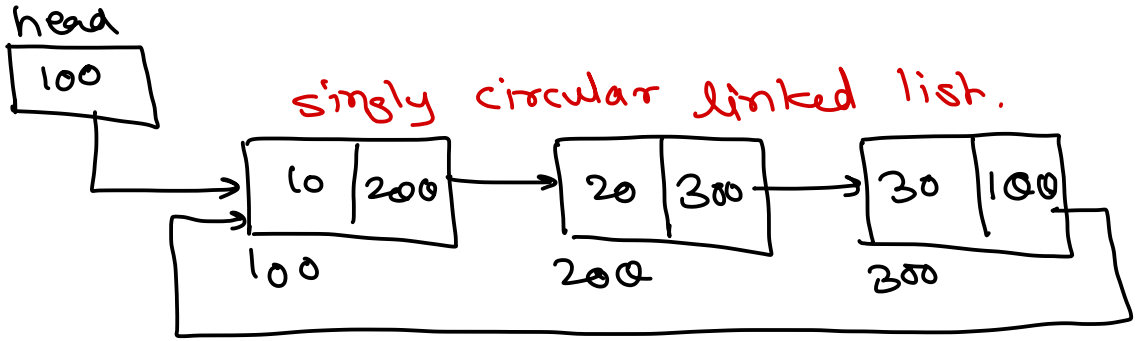
```
class A {
  private:
     int a;
  public:
     =
     =
     =
   friend class B;
};

class C: public A {
    =
};

class B {
  public:
    void f1(){
       =
    }
    void f2(){
       =
    }
    =
    =
};
```

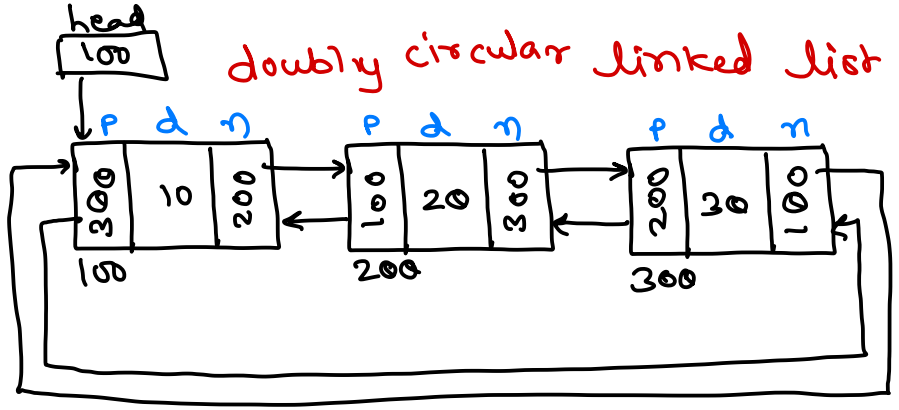


Linked List


head
100
singly linear linked list.
10
200
20
300
30
0
100
200
300

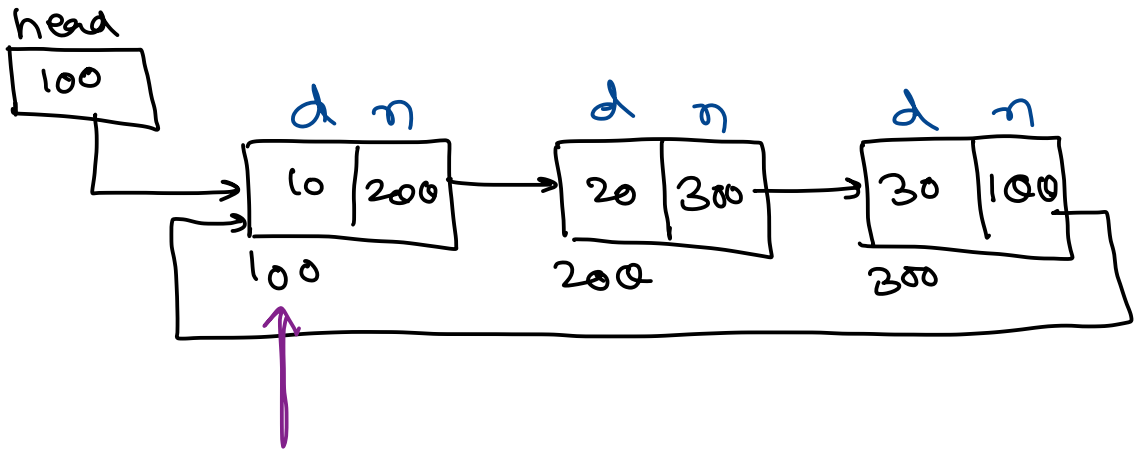

head
100
doubly linear linked list
p
d
n
0
10
200
100
20
300
200
30
0
100
200
300


head
100
singly circular linked list.
10
200
20
300
30
100
100
200
300


head
100
doubly circular linked list
p
d
n
300
10
200
100
20
300
200
30
100
100
200
300

Linked List

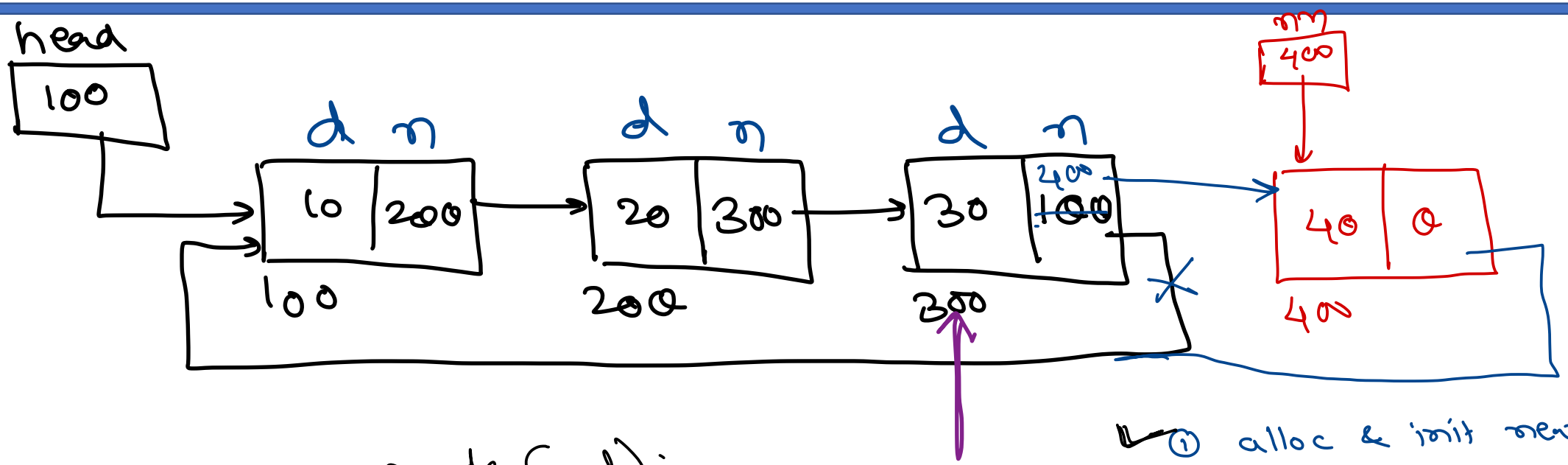
singly circular linked list.




```
display()
if (head != NULL) {
    trav = head;
    do {
        cout << trav -> data;
        trav = trav -> next;
    } while (trav != head);
}
```

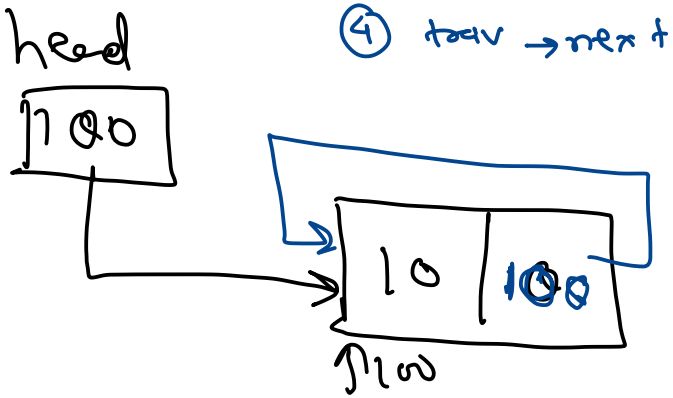
```
trav = head;
while (trav -> next != head) {
    cout << trav -> data;
    trav = trav -> next;
}
```

circular list → add last




```
nn = new node(val);
trav = head;
while (trav->next != head)
    trav = trav->next;
nn->next = head;
trav->next = nn;
```

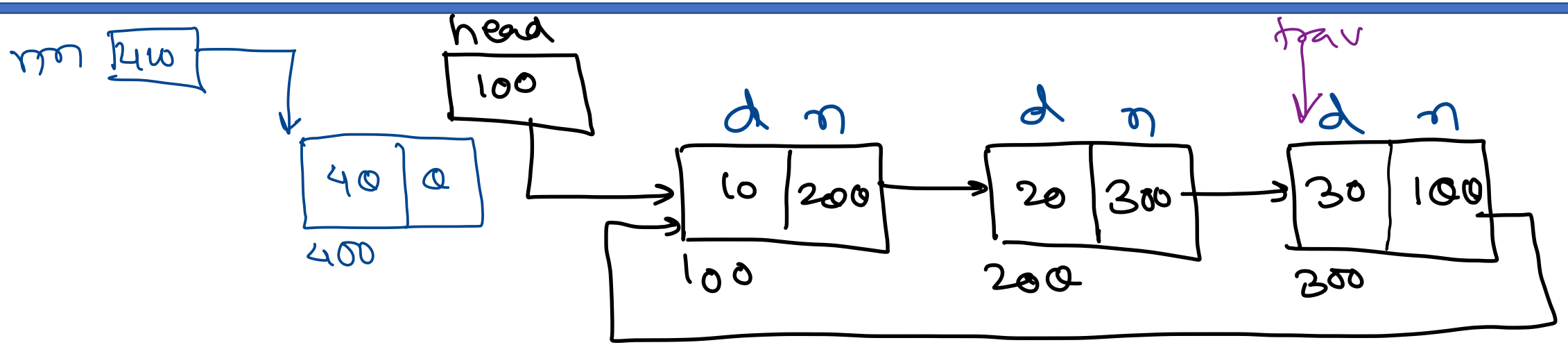
① alloc & init new node
② traverse till last node of list.
③ nn->next = head;
④ trav->next = nn;



Special: empty list

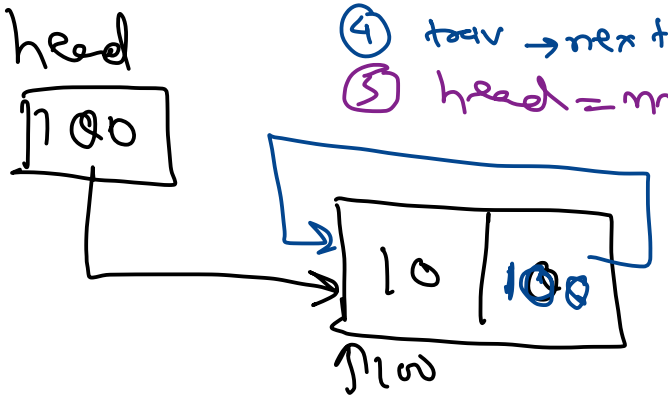
```
head = nn;
nn->next = head;
```

circular LL → add first




```
nn = new node(val);
trav = head;
while (trav->next != head)
    trav = trav->next;
nn->next = head;
trav->next = nn;
head = nn;
```

① alloc & init new node
② traverse till last node of list.
③ nn->next = head;
④ trav->next = nn;
⑤ head = nn;

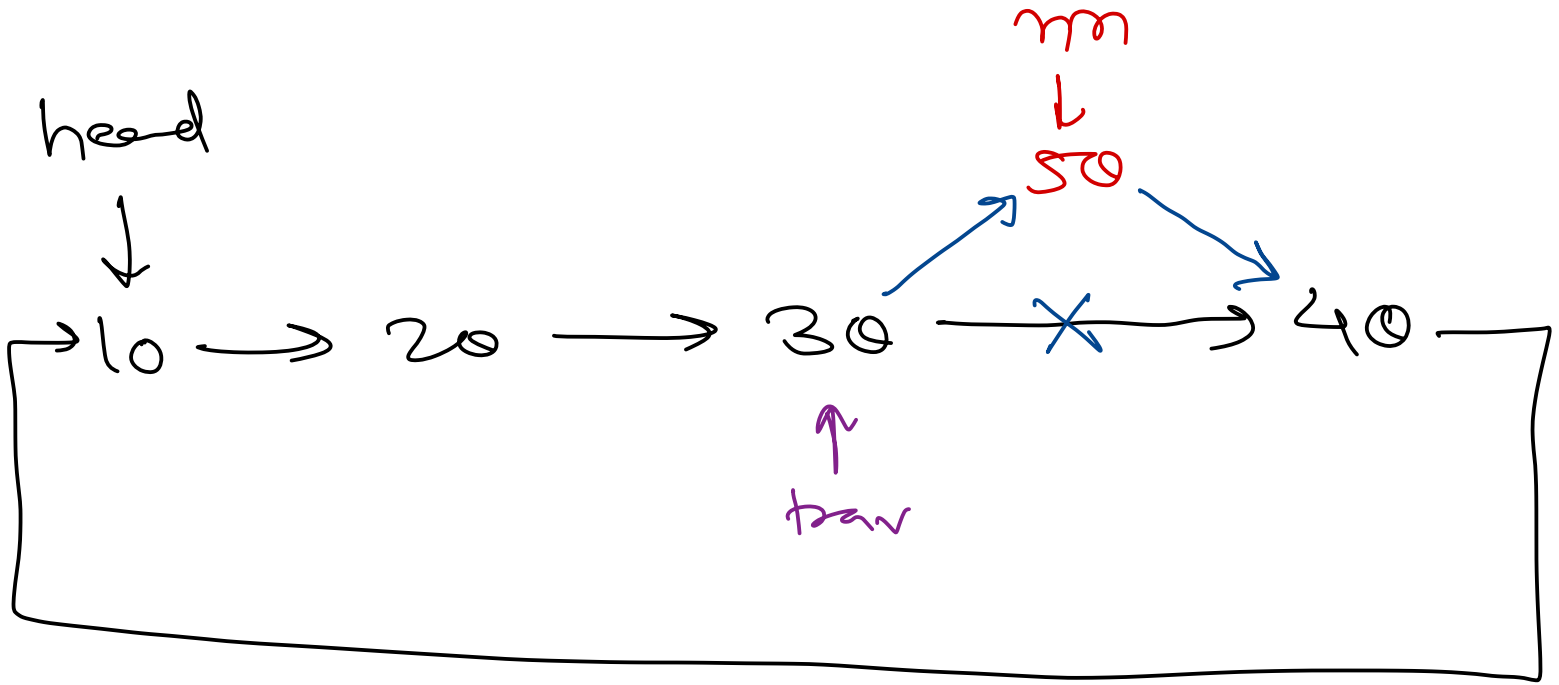


Special: empty list

```
head = nn;
nn->next = head;
```

Linked List

Singly circular linked list → add_at_pos.



Special 1 : list is empty

↓

add_first()

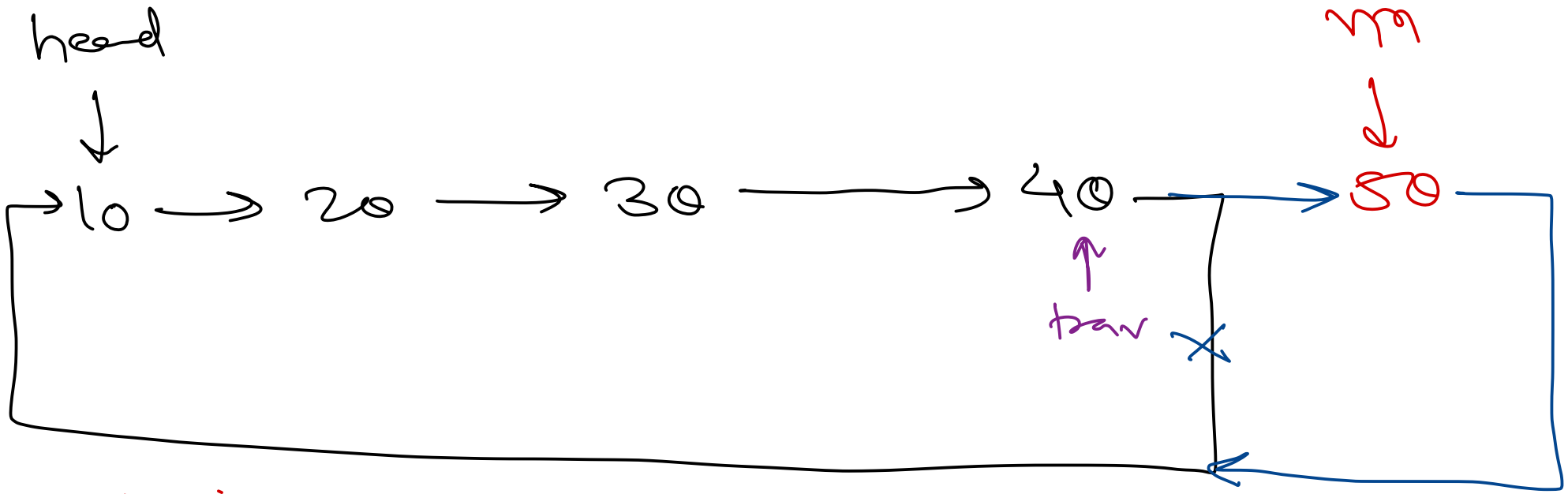
Special 2 : pos <= 1

↓

add_first()

Special : pos > size (beyond last)

Singly circular linked list → add_at_pos.



Special 1 : list is empty

↓

add_first()

Special 2 : pos <= 1

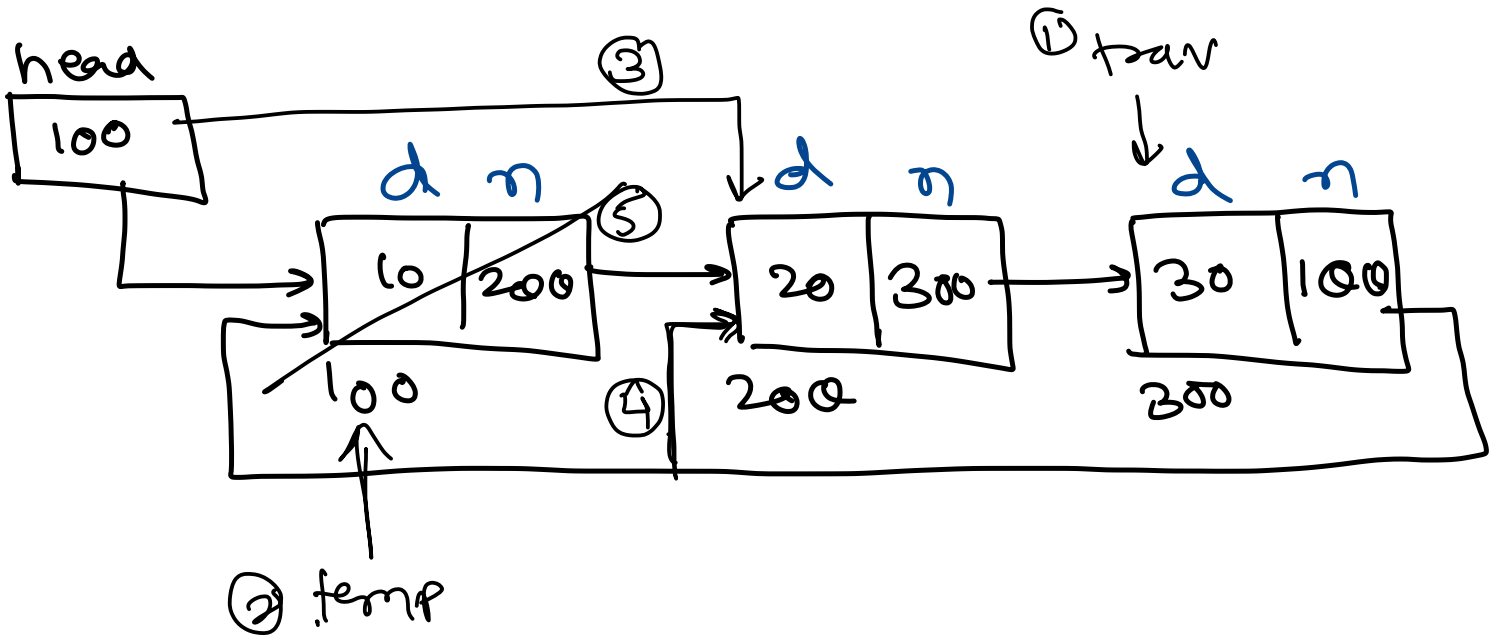
↓

add_first()

Special : pos > size (beyond last)

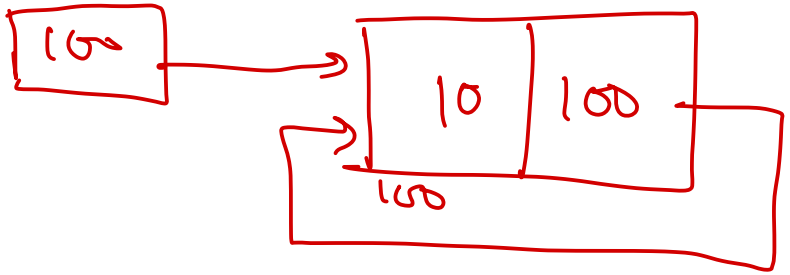
```
{ if (trav -> next == head)
      break;
}
nn -> next = head;
trav -> next = nn;
```

singly circular list – del_first()



① traverse till last node
② take first node addr in temp
③ take head to next node
④ last node point to new head.
⑤ delete temp node.

Special 2: single node in list.

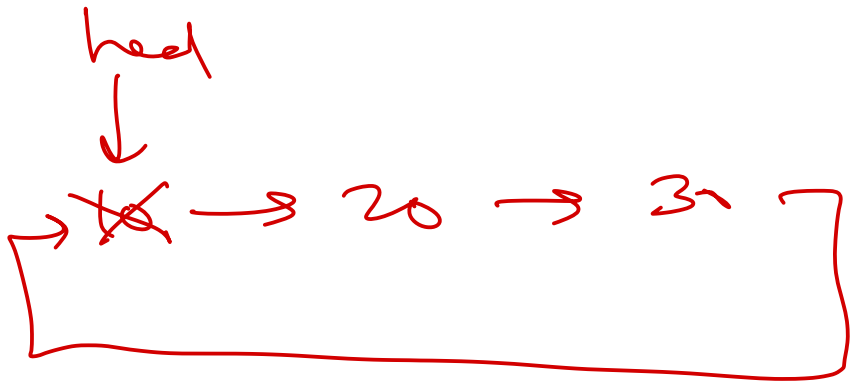


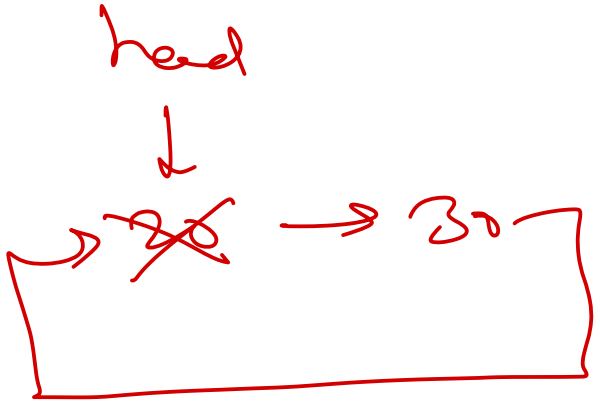

```
if (head -> next==head)
{
    delete head;
    head = NULL;
}
```

Special 1: if list is empty, return.

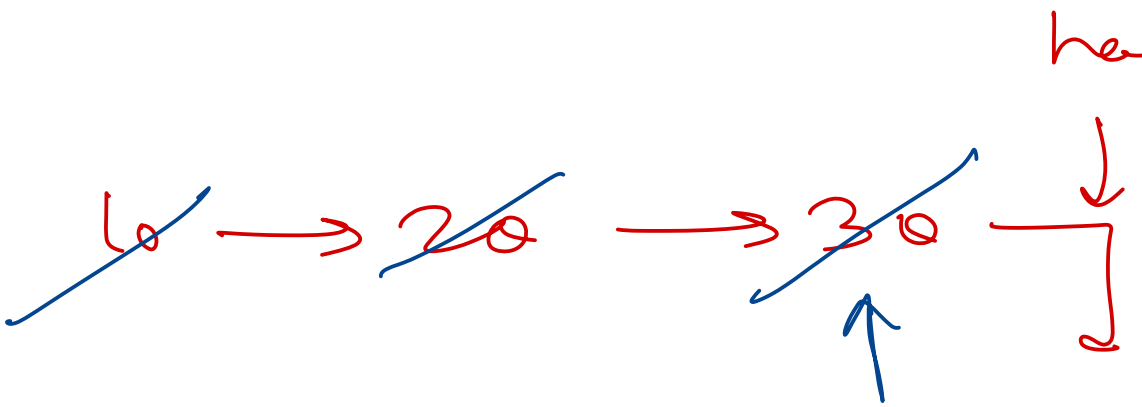
```
if (head==NULL)
    return;
```

Singly circular list - del all



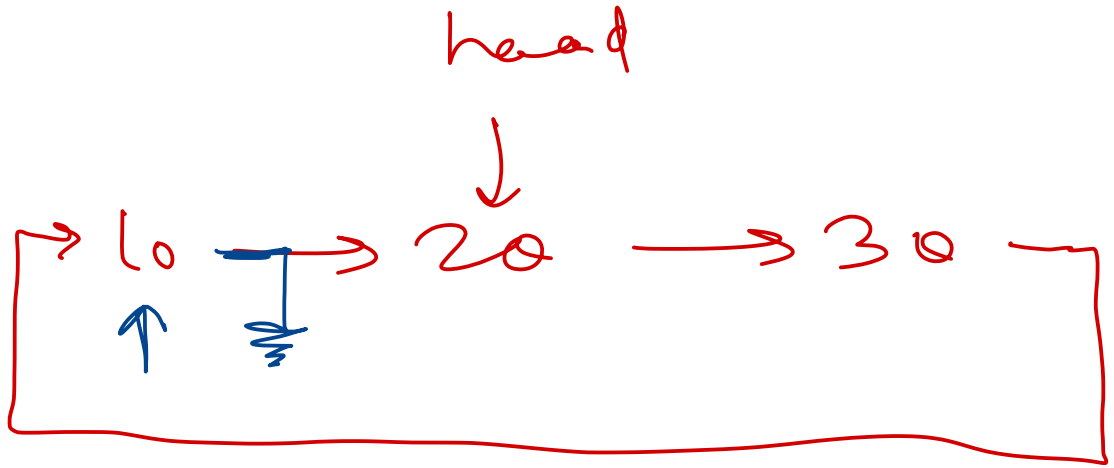


singly linear LL - del all




```
while (head != NULL)
{
    temp = head;
    head = head->next;
    delete temp;
}
```

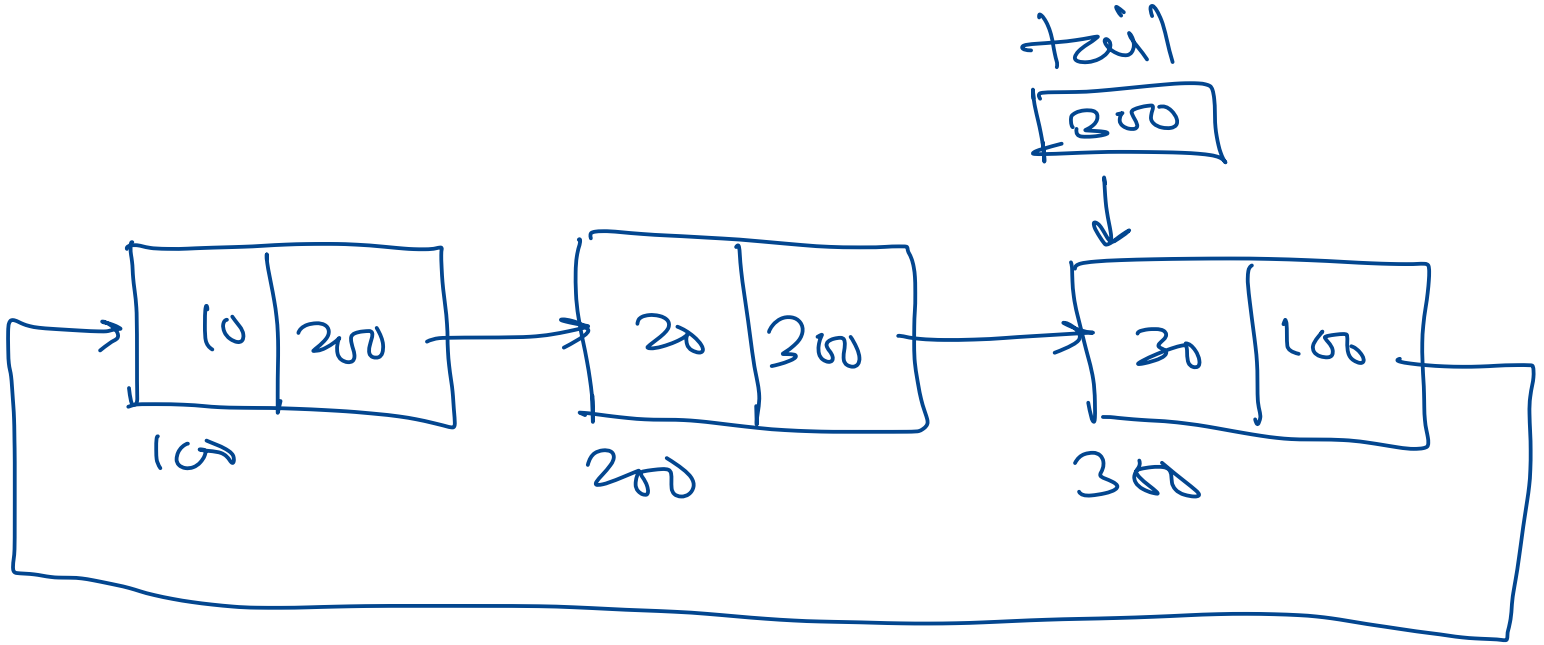
singly circular LL - del all




```
if( head != NULL ) {
    temp = head;
    head = head->next;
    temp->next = NULL;
}
while (head != NULL)
{
    temp = head;
    head = head->next;
    delete temp;
}
```

Linked List

singly circular LL - assignment.



1. addFirst()
2. addLast()
3. delFirst()
4. display()

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>