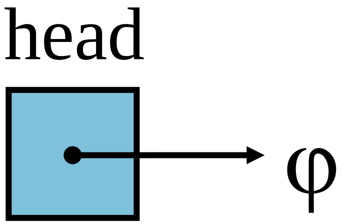
List Class

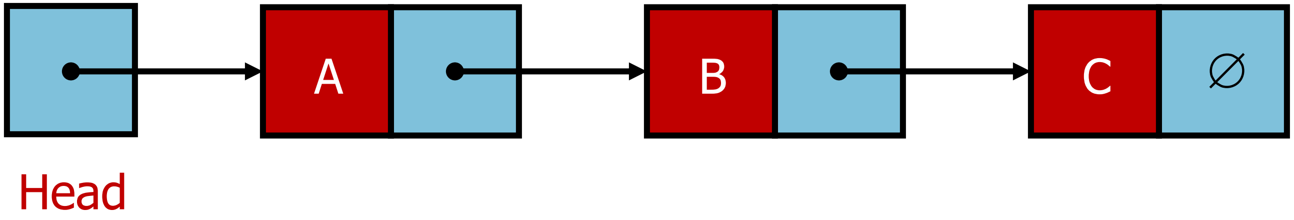
- Declare `List`, which contains
  - `head`: a pointer to the first node in the list.
    Since the list is empty initially, `head` is set to `NULL`
  - Operations on `List`

```
template<class type>
class List {
public:
    List() { head = 0; };
    ~List() ;

private:
        Node<type> * head;

};
```

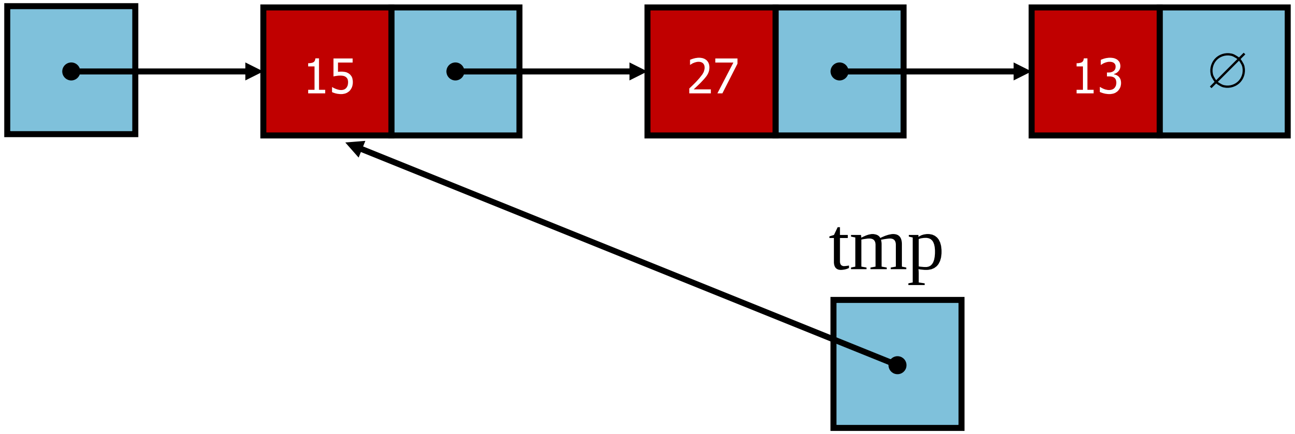


Is Empty

```
bool IsEmpty() {
    return head == 0;
}
```

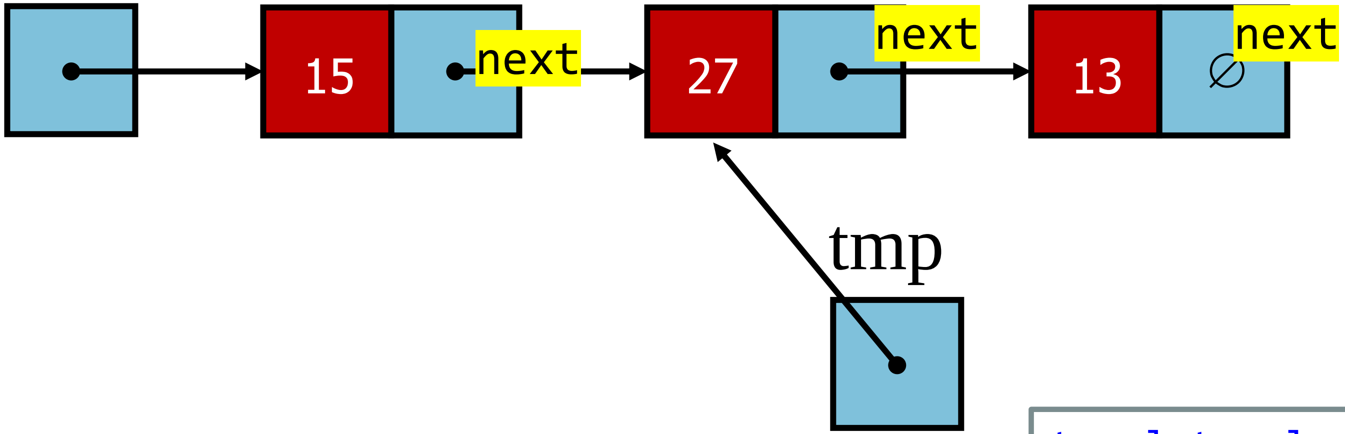


Find a node (data value) in List



```
template<class type>
bool List<type>::Find(type val) {
    Node<type> * tmp = head;
    while (tmp != NULL && tmp->data != val)
        tmp = tmp->next;

    return tmp != NULL;
}
```

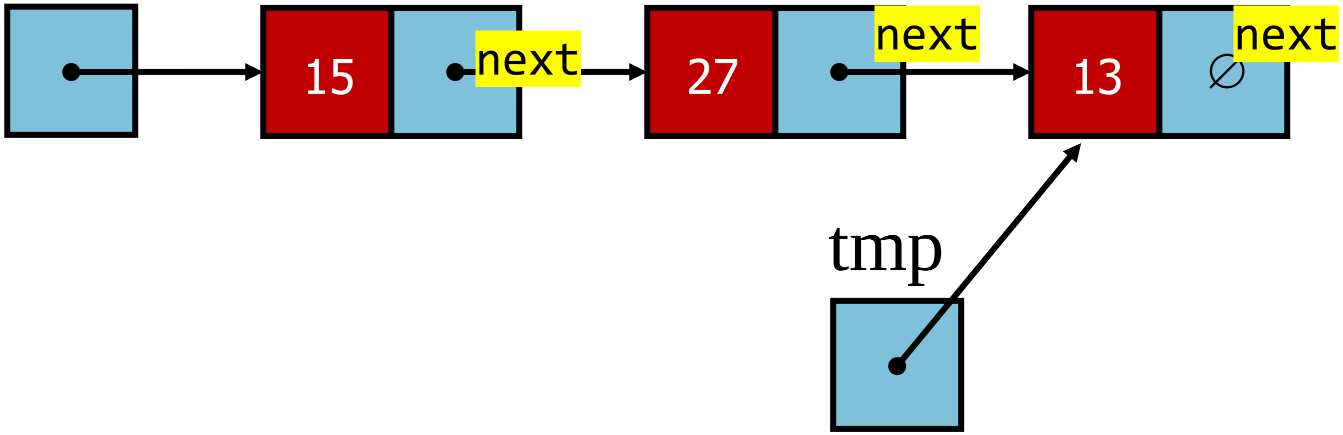
Find a node (data value) in List



```
template<class type>
bool List<type>::Find(type val) {
    Node<type> * tmp = head;
    while (tmp != NULL && tmp->data != val)
        tmp = tmp->next;

    return tmp != NULL;
}
```

```
template<class type>
class Node {
public:
    Node()
    Friend class List;
private:
    type data;
    Node<type> * next;

};
```

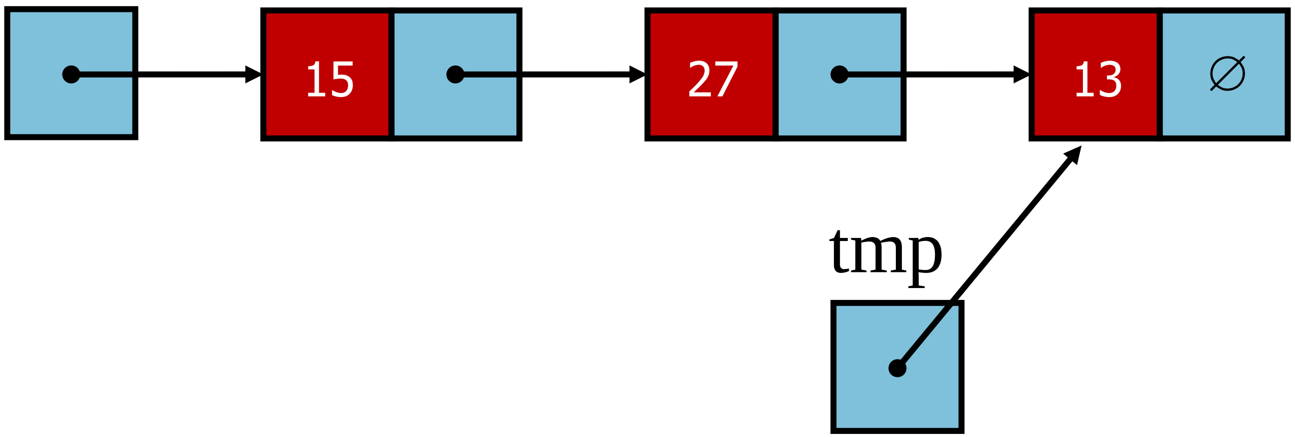
Find a node (data value) in List



```
template<class type>
bool List<type>::Find(type val) {
    Node<type> * tmp = head;
    while (tmp != NULL && tmp->data != val)
        tmp = tmp->next;

    return tmp != NULL;
}
```

```
template<class type>
class Node {
public:
    Node()
    Friend class List;
private:
    type data;
    Node<type> * next;

};
```

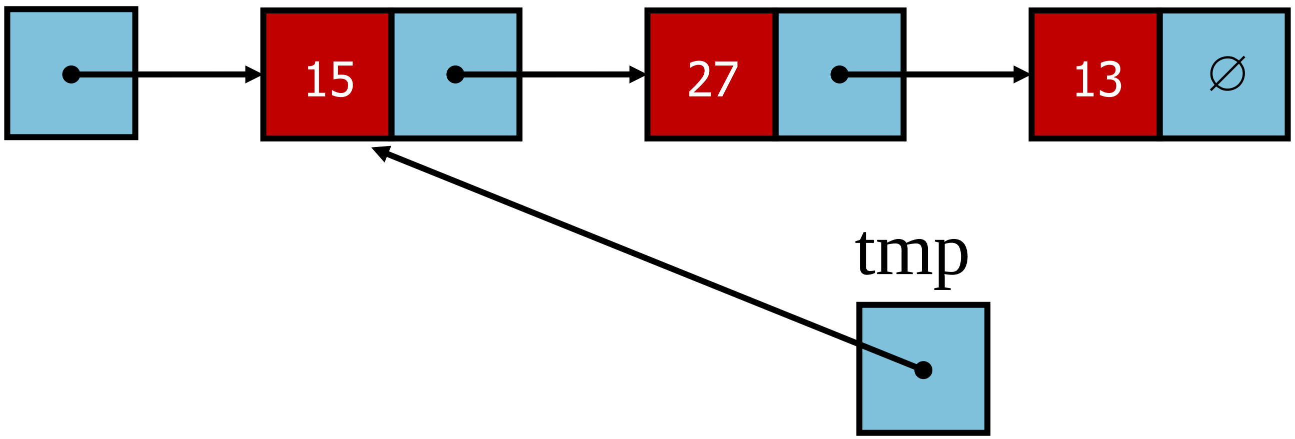
Find a node (data value) in List



```
template<class type>
bool List<type>::Find(type val) {
    Node<type> * tmp = head;
    while (tmp != NULL && tmp->data != val)
        tmp = tmp->next;

    return tmp != NULL;
}
```

Takes *O*(1) time in the best case and *O*(*n*) in the worst and average cases

Print SL List



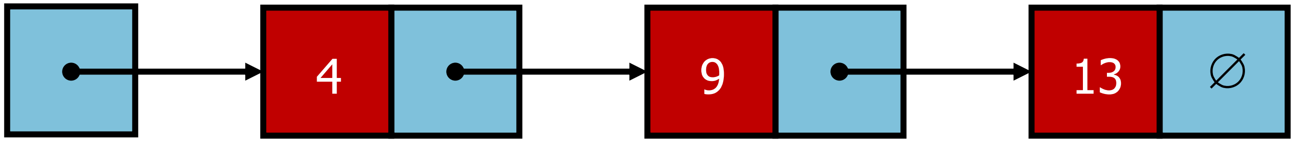
```
template<class type>
void List<type>::print() {
    Node<type> * tmp;
    for (tmp = head; tmp != 0; tmp = tmp->next)
        cout << tmp->data << "  ";
    cout << endl;
}
```

Takes *O*(*n*) time

SL List AddNode

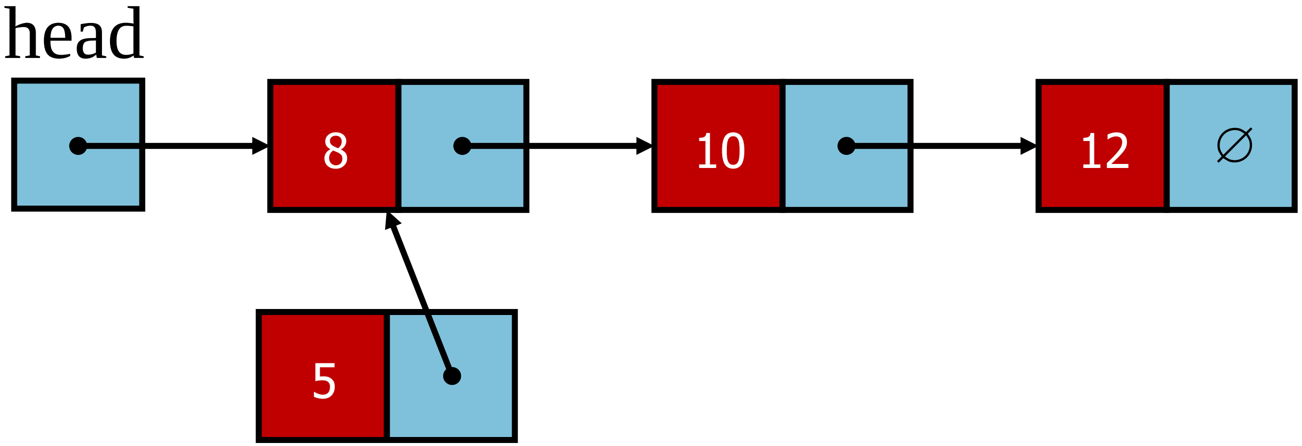
Let's implement some basic operations in class **List**

- **Add Node**
- **Where to** add Node
  - Start of the list
  - End of the list
  - Some where in the middle …after some particular data value (sorted list)
- **Which is most efficient ?**
- We provide all the options let user decide which to use



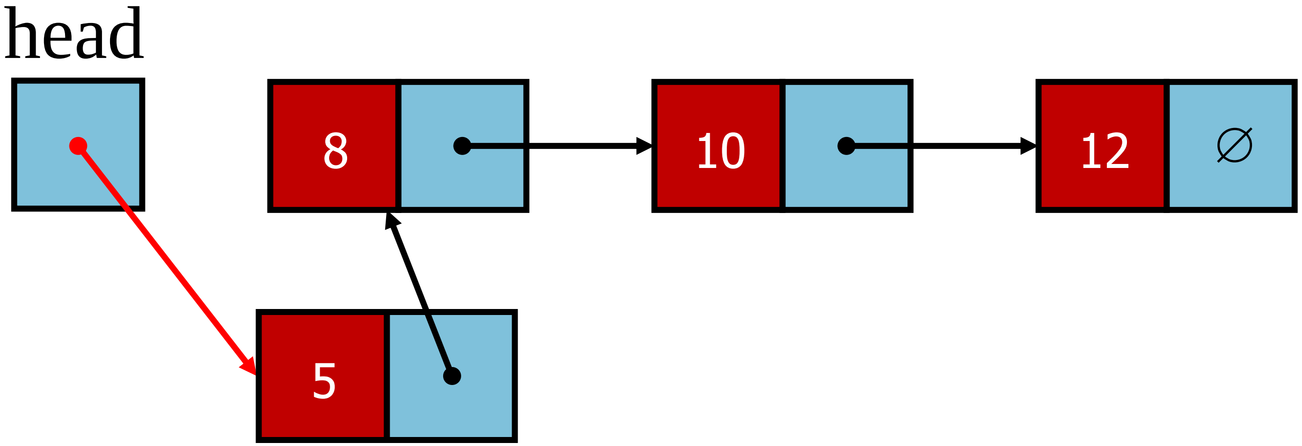
SL List AddNode

AddNode at start



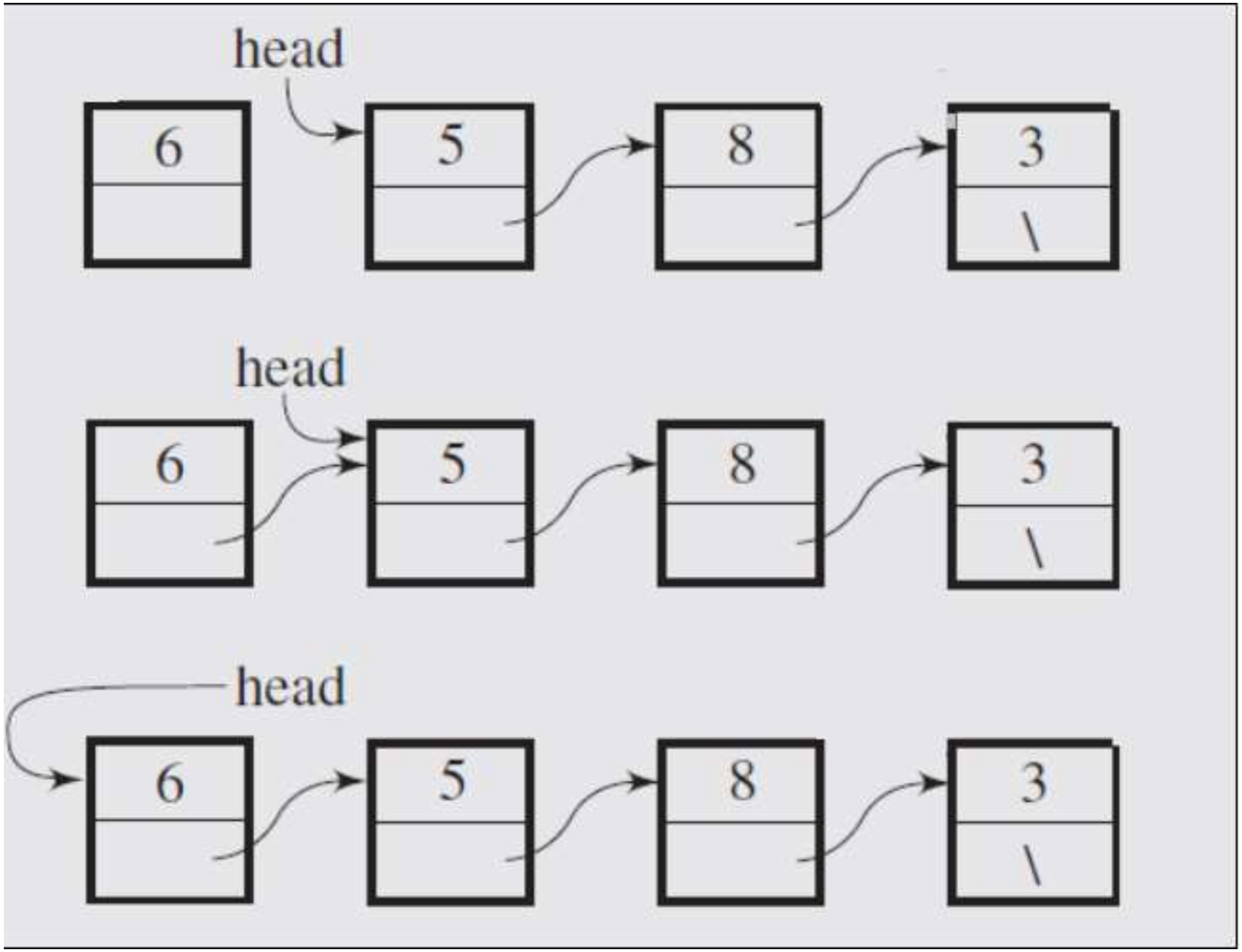
SL List AddNode

AddNode at start



SL List AddNode

AddNode at start

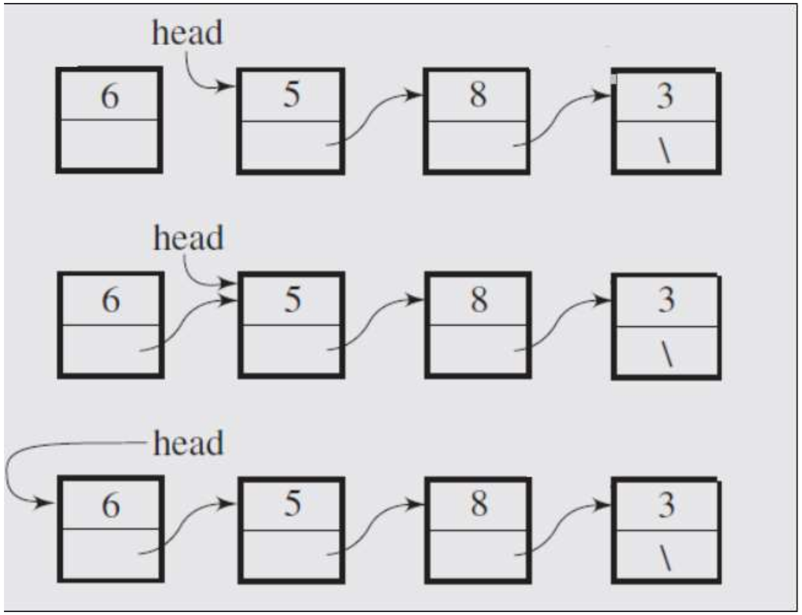


AddNode at start

```
template<class type>
void List<type>::addtoHead(type val)
{
    head = new Node<type>(val, head);
}
```

TIME COMPLEXITY ?

```
template<class type>
class Node {
public:
    Node(){ next = NULL; }
    Node(type val,Node<type>*nptr=0){
        data = val;
        next = nptr;
    }
    Friend class List;
private:
    type data;
    Node<type> * next;
};
```



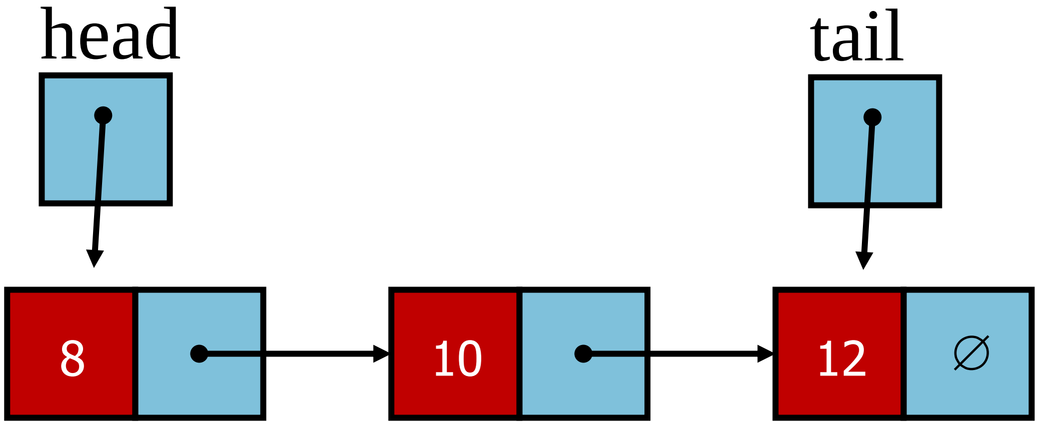
AddNode at start

```
template<class type>
class List {
public:
    List() { head = 0;
              tail= 0; };
    ~List() ;

    void addToStart(type val);

private:
       Node<type> * head;
       Node<type> * tail;
};
```

AddNode at End

Not a good idea to traverse the entire list and insert
Keep a pointer to the tail of the list.

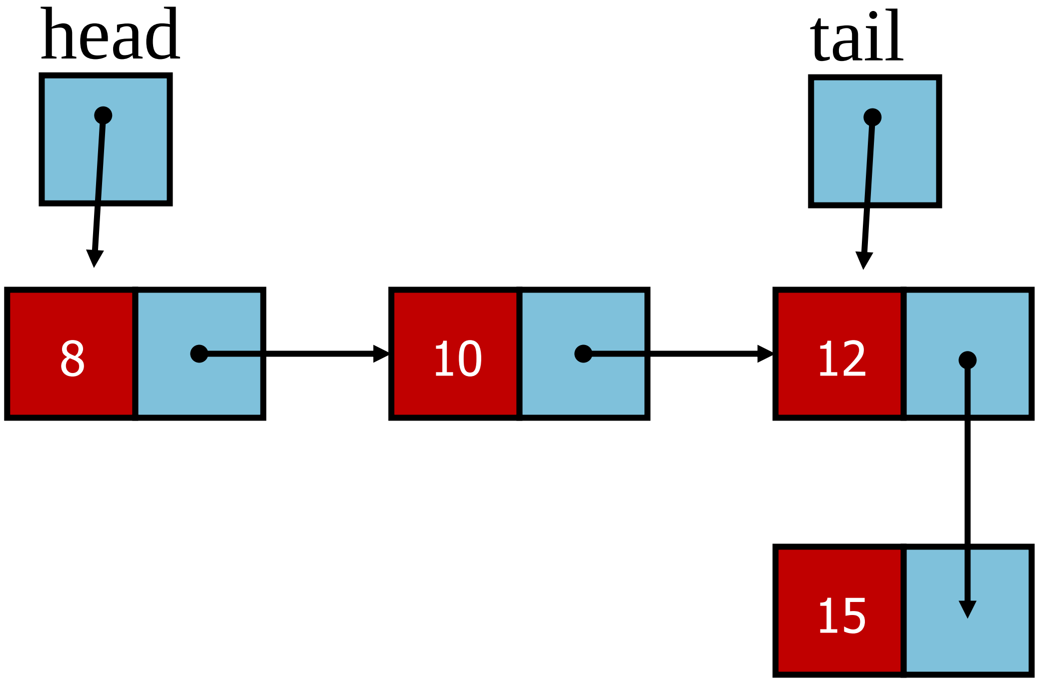
```
template<class type>
class List {
public:
    List() { head = 0;
                tail= 0; };
    ~List() ;

    void addToStart(type val);
    void addToTail(type val);

private:
        Node<type> * head;
        Node<type> * tail;
};
```



AddNode at End

```
Not a good idea to traverse the entire list and insert
Keep a pointer to the tail of the list.
```

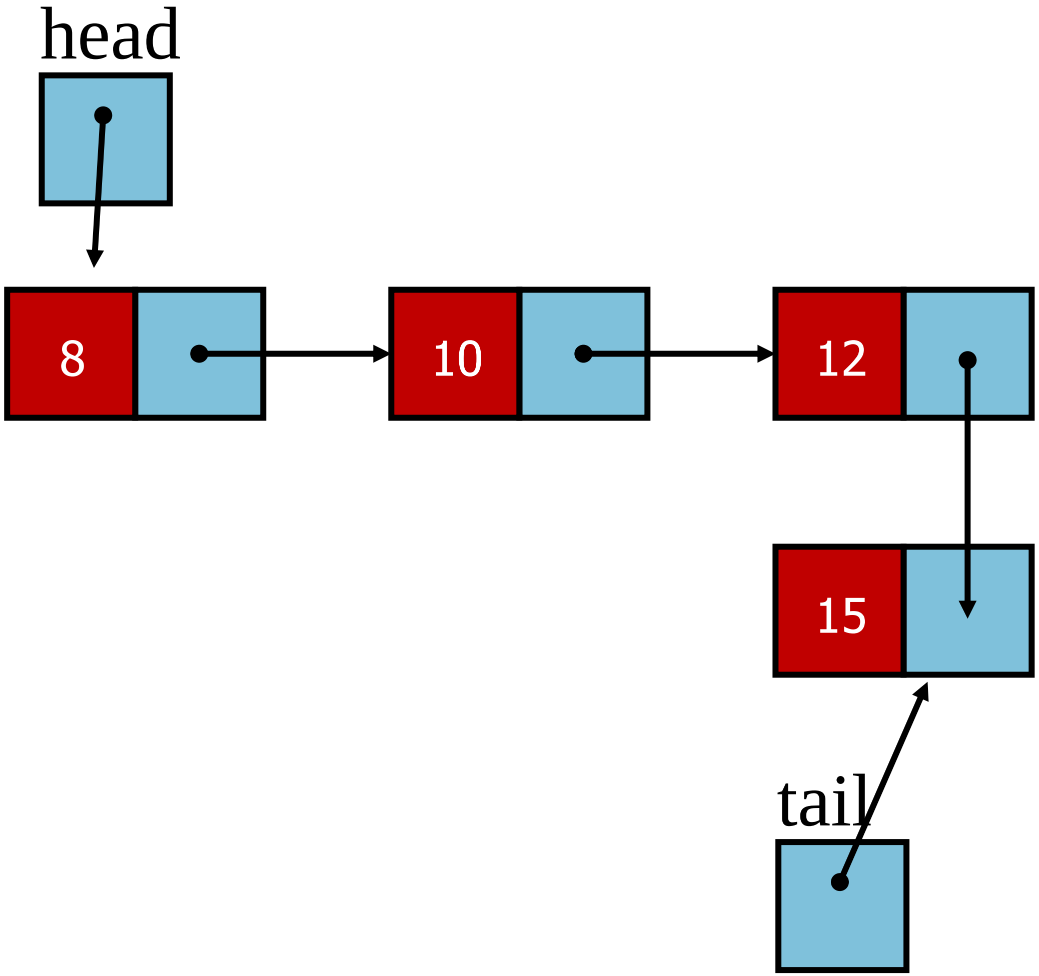
```
template<class type>
class List {
public:
    List() { head = 0;
                tail= 0; };
    ~List() ;

    void addToStart(type val);
    void addToTail(type val);

private:
        Node<type> * head;
        Node<type> * tail;
};
```



AddNode at End

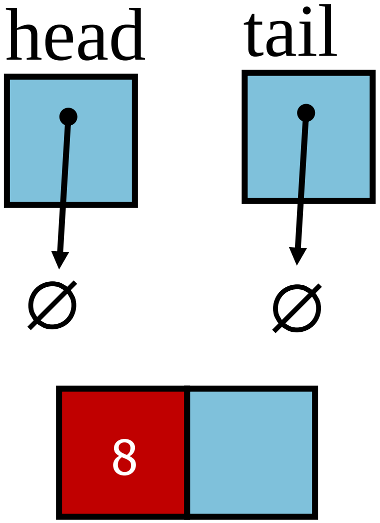
Not a good idea to traverse the entire list and insert
Keep a pointer to the tail of the list.

```
template<class type>
class List {
public:
    List() { head = 0;
             tail= 0; };
    ~List() ;

    void addToStart(type val);
    void addToTail(type val);

private:
        Node<type> * head;
        Node<type> * tail;
};
```



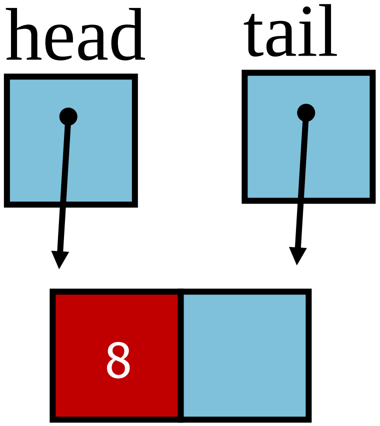
AddNode at End

```
template<class type>
void List<type>::addToTail(type val) {
    if (tail != NULL) {

    }
    else
        head = tail= new Node<type>( val);
}
```



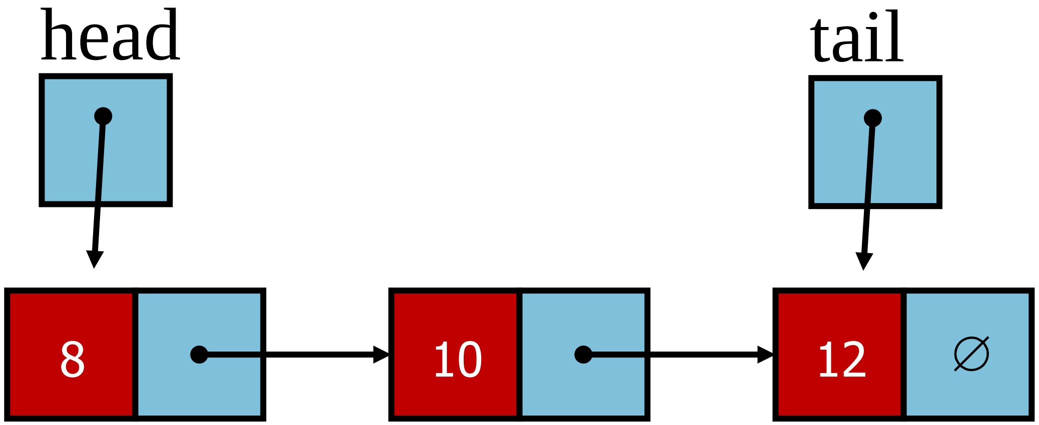
AddNode at End

```
template<class type>
void List<type>::addToTail(type val) {
    if (tail != NULL) {
    }
    else
        head = tail= new Node<type>( val);
}
```



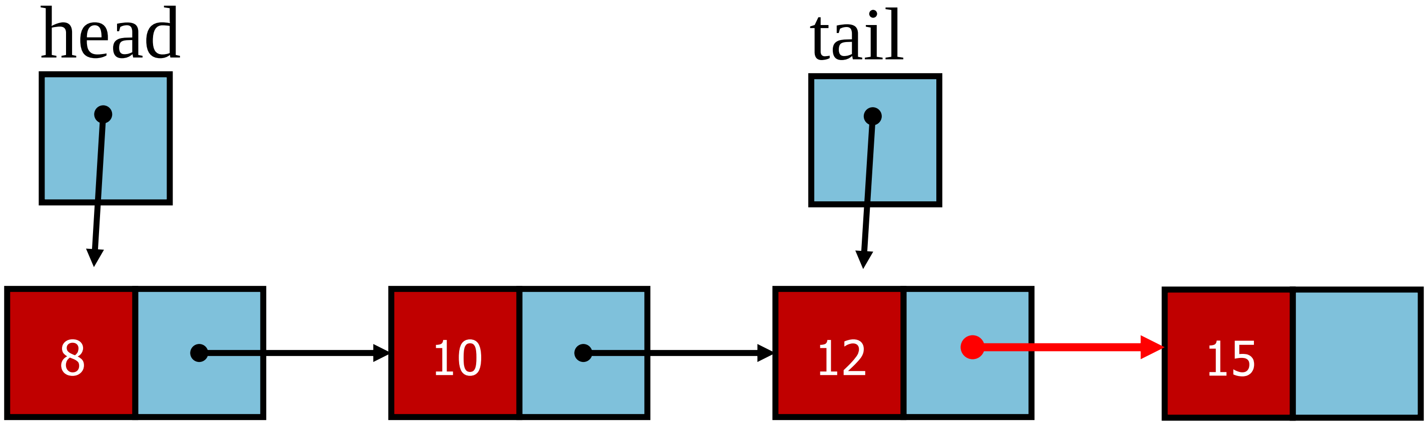
AddNode at End

```
template<class type>
void List<type>::addToTail(type val) {
    if (tail != NULL) {
    }
    else
        head = tail= new Node<type>( val);
}
```



AddNode at End

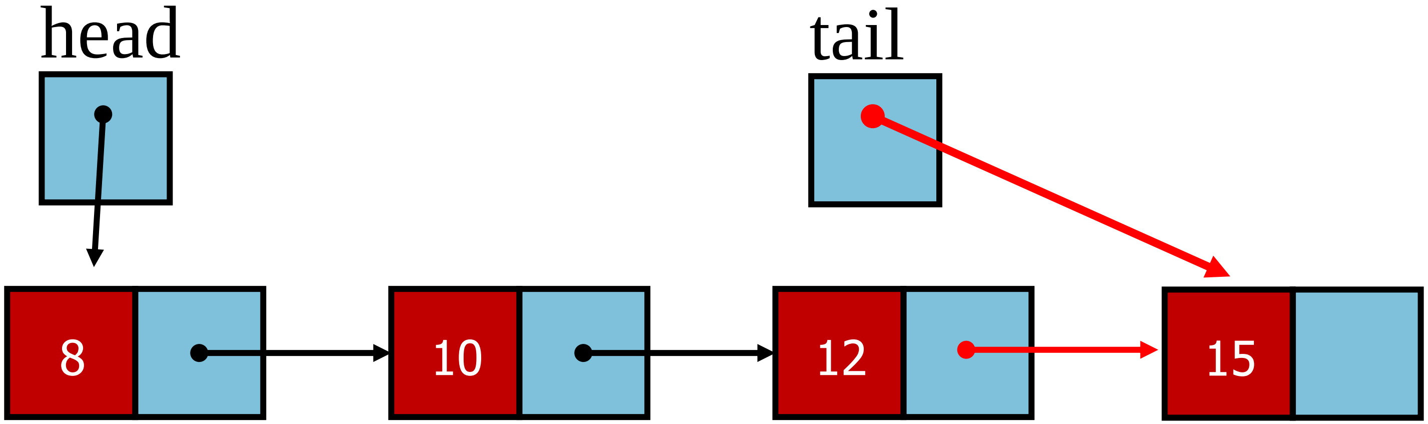
```
template<class type>
void List<type>::addToTail(type val) {
    if (tail != NULL) {
        tail->next = new Node<type>(val);
    }
    else
        head = tail= new Node<type>( val);
}
```



AddNode at End

```
template<class type>
void List<type>::addToTail(type val) {
    if (tail != NULL) {
        tail->next = new Node<type>(val);
        tail = tail->next;
    }
    else
        head = tail= new Node<type>( val);
}
```

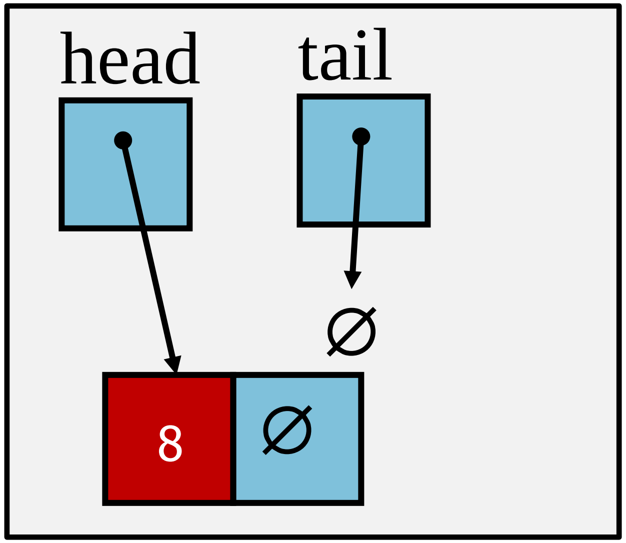
TIME COMPLEXITY ?

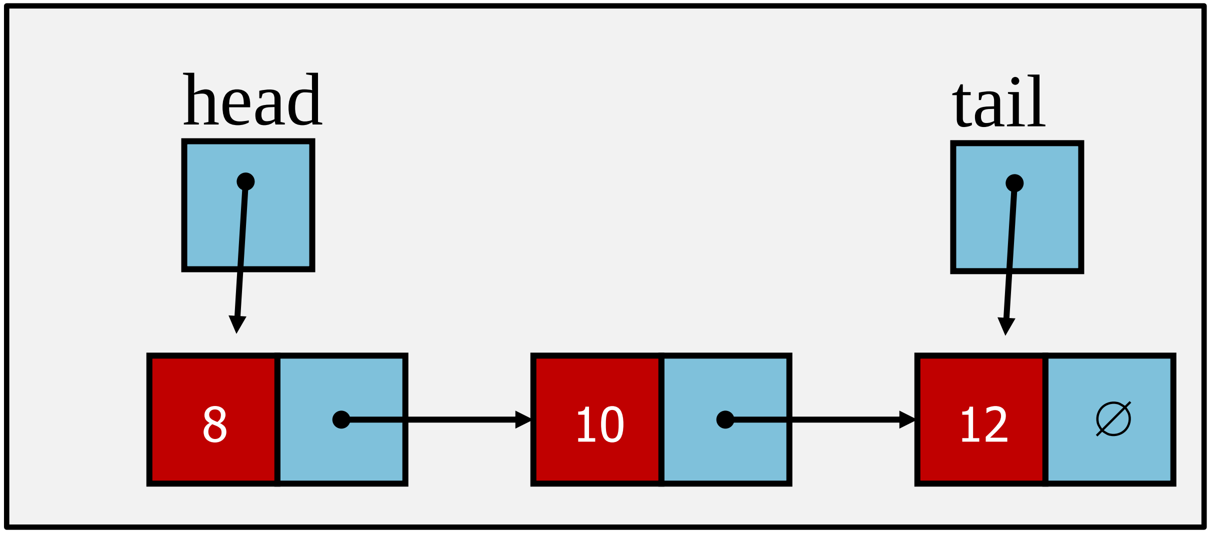


Update Method AddToHead

```
template<class type>
void List<type>::addtoHead(type val)
{
     head = new Node<type>(val, head);
}
```

ISSUES ??



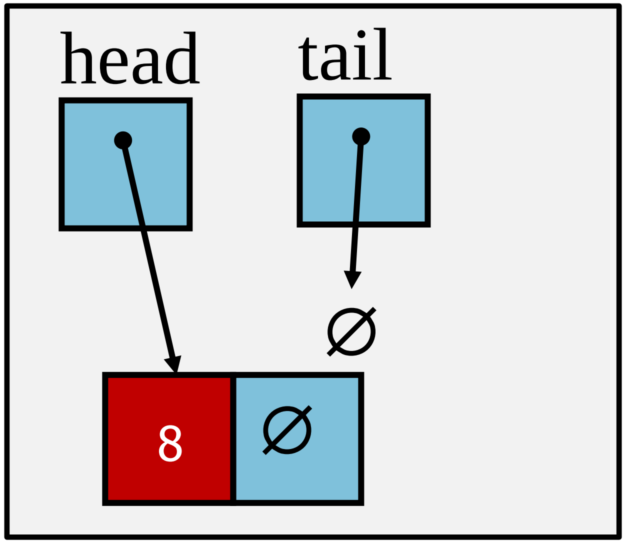


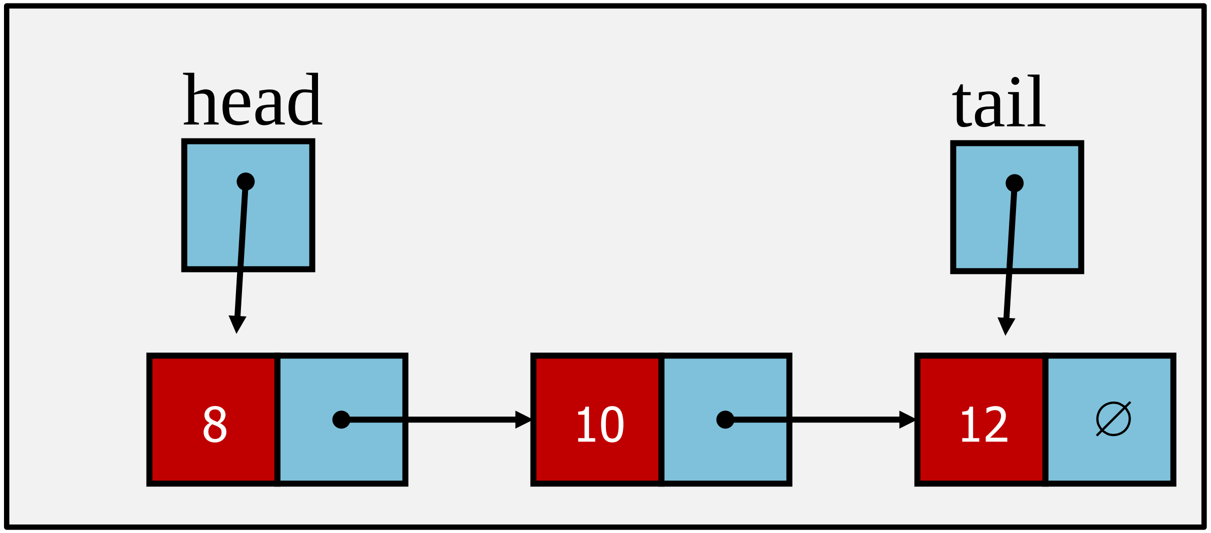
```
template<class type>
void List<type>:: addtoHead(type val){
   head = new Node<type>(val, head);
    if (tail == 0)
        tail = head;
}
```

Update Method AddToHead

```
template<class type>
void List<type>::addtoHead(type val)
{
    head = new Node<type>(val, head);
}
```

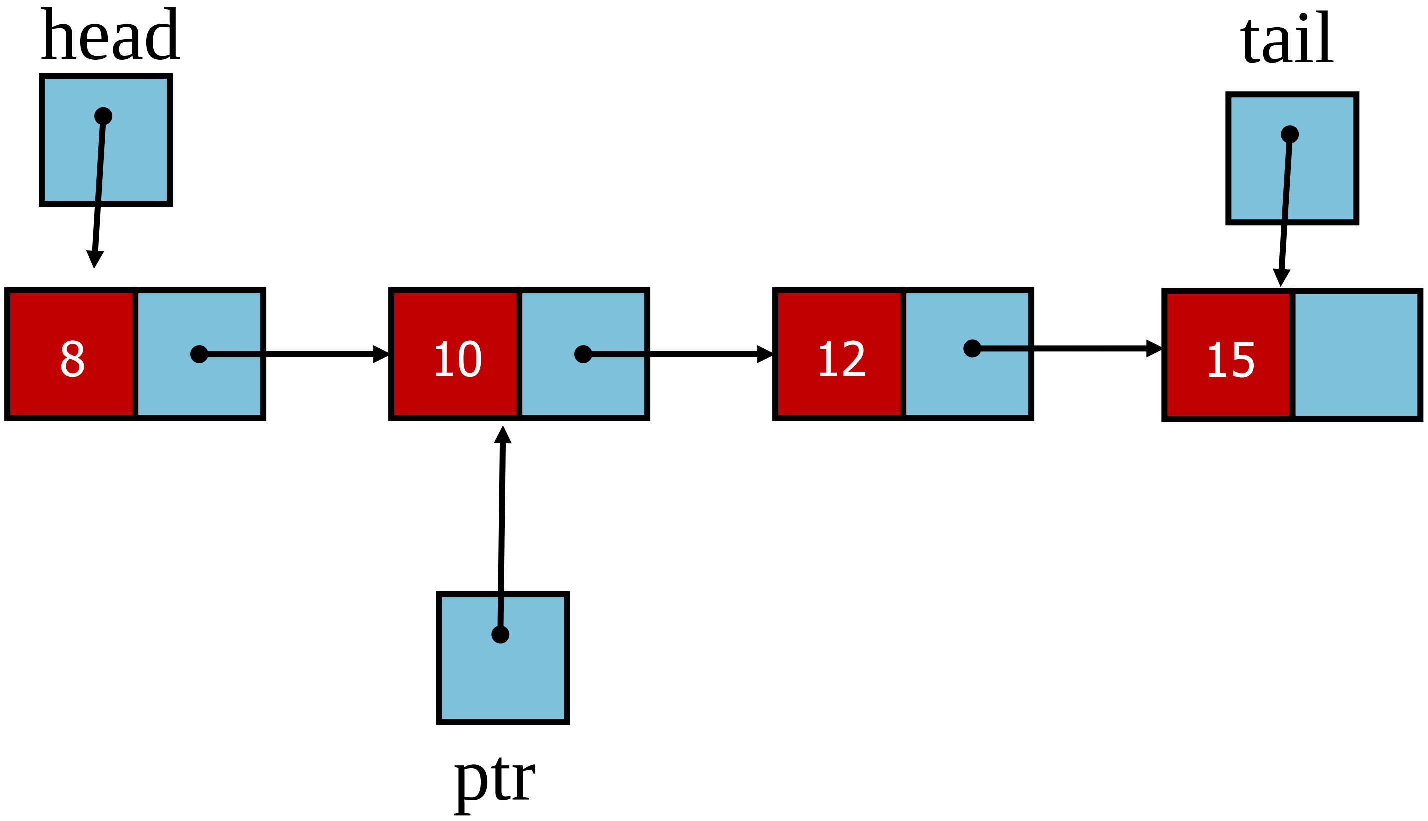
ISSUES ??





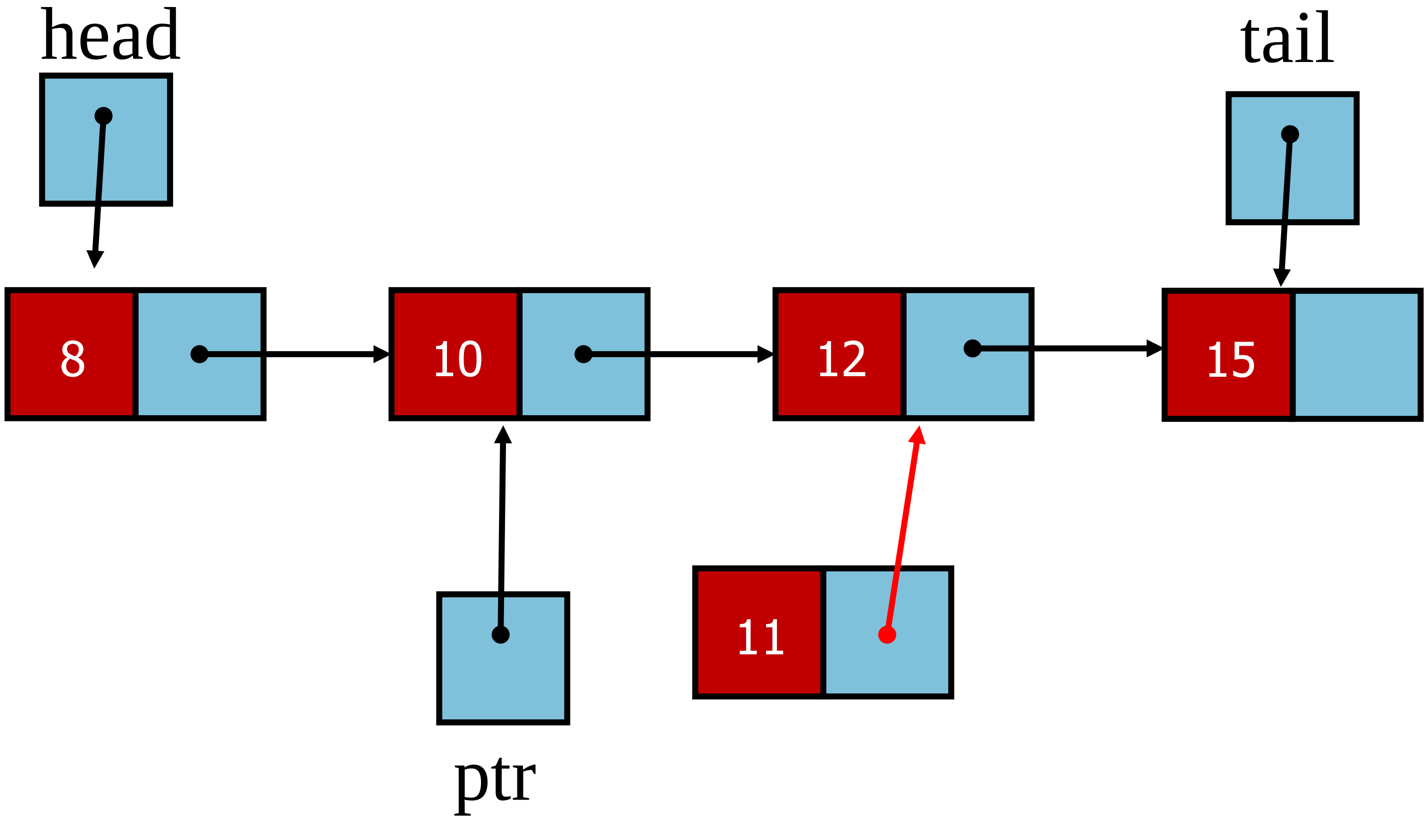
```
template<class type>
void List<type>:: addtoHead(type val){
   head = new Node<type>(val, head);
    if (tail == 0)
        tail = head;
}
```

AddNode after a particular Data item



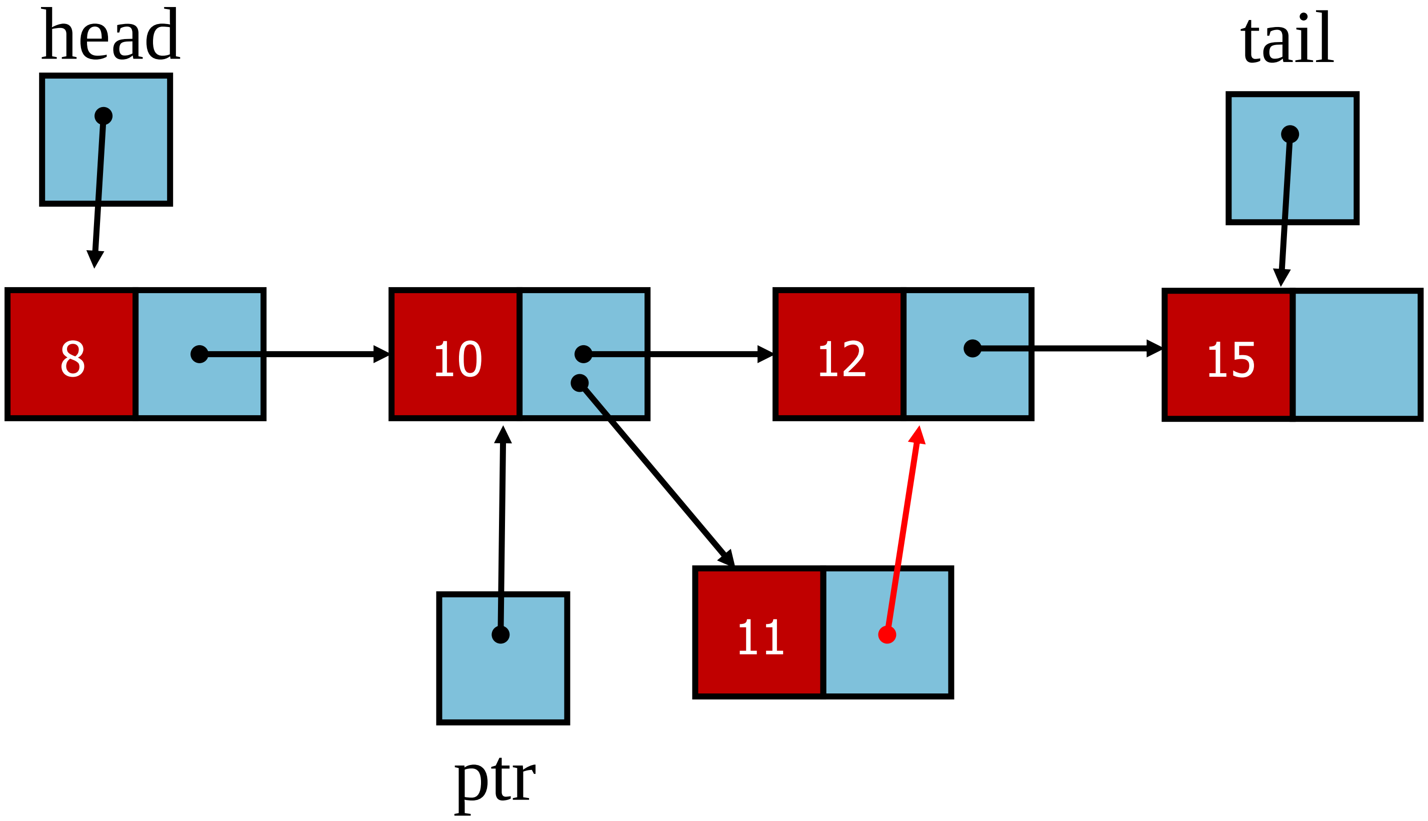
Add a new node after the node with value 10

AddNode after a particular Data item



Add a new node with data=11 after the node (with data= 10)

AddNode after a particular Data item



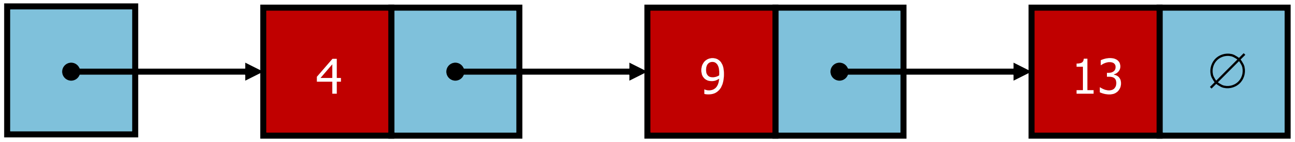
WRITE CODE

TIME COMPLEXITY ?

SL List Delete Node

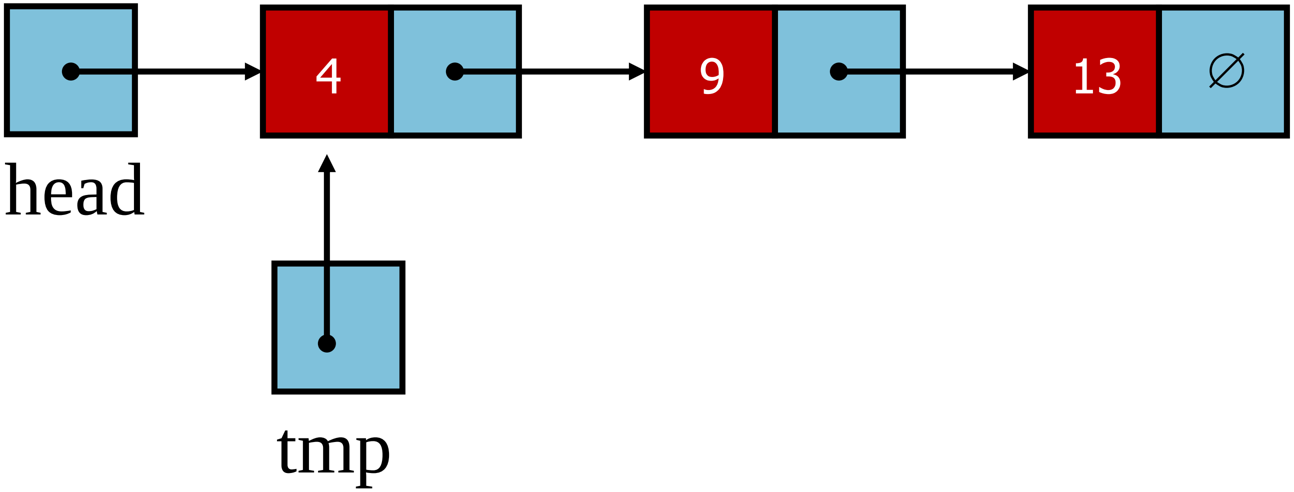
Delete Node

- **Which node to delete ?**
  - Start ?
  - End ?
  - Or the one with some particular data value ?
- **Which is most efficient ?**
- We provide all the options let user decide which to use



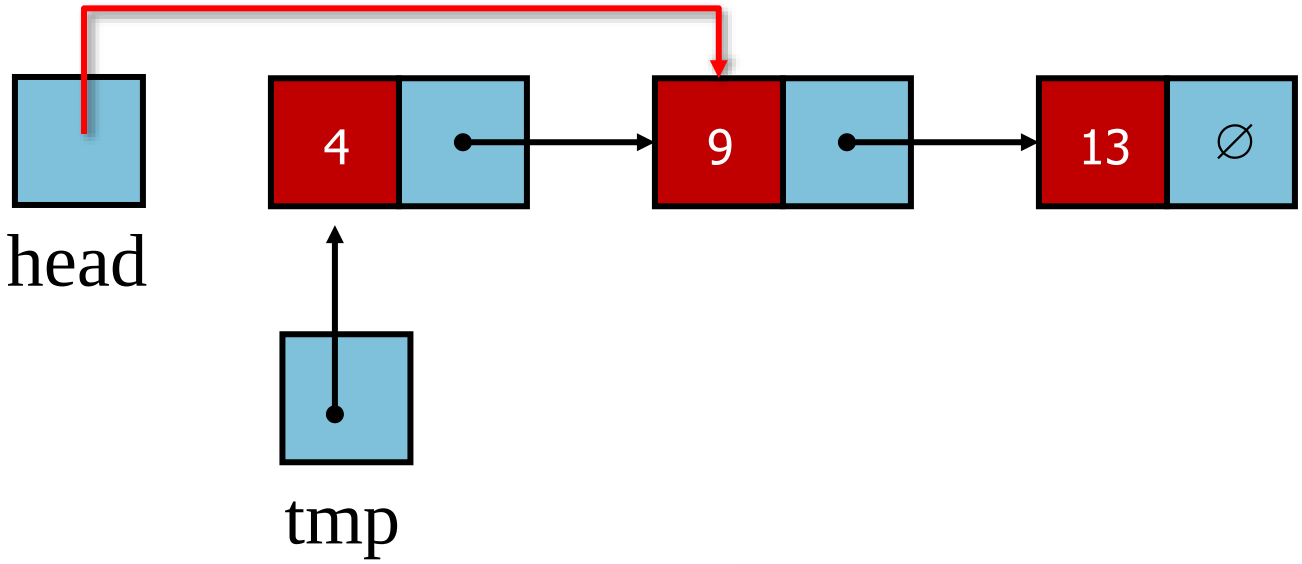
SL List Delete Node

DeleteNode From start



SL List Delete Node

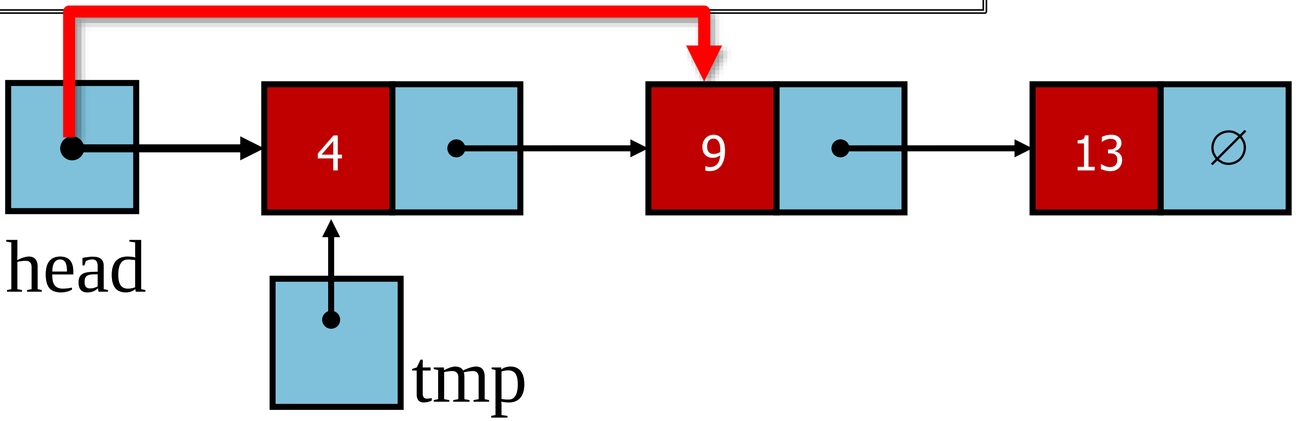
DeleteNode From start



TIME COMPLEXITY ?

DeleteNode From Start

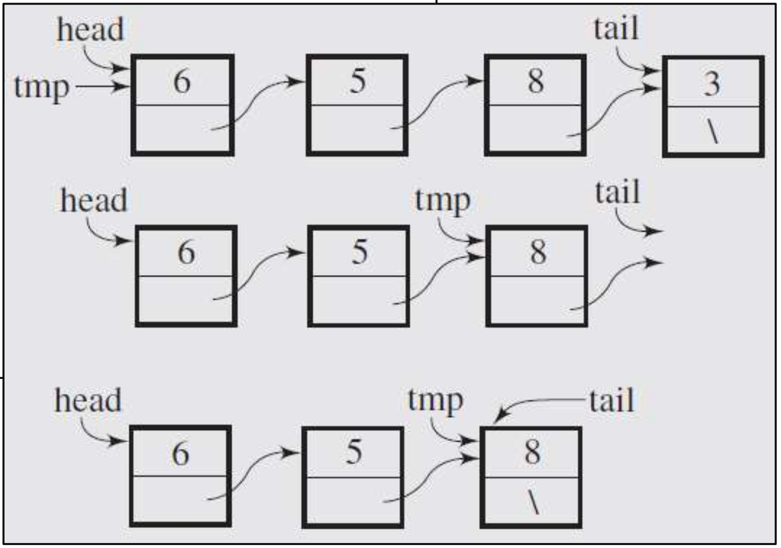
```
template<class type>
bool List<type>::deleteFromHead() {
    bool deleted = false;
    if (head != NULL) {//non empty list
        Node<type> * tmp = head;
        if (head == tail) {// only one node in the list
                head = tail = NULL;
        }
        else  //more than one node
                head = head->next;
        delete tmp;
        deleted = true;
    }
    return deleted;
}
```



Delete Node From End

- Can tail be helpful ?

```
template<class type>
bool List<type>::deleteFromTail() {
    bool deleted = false;
    if (head != NULL) {//non empty list
        if (head == tail) {// only one node in the list
            delete head;
            head = tail = NULL;
        }
        else { //more than one node transverse to the end
            Node<type> * tmp = head;
            for (; tmp->next != tail; tmp = tmp->next);
            delete tail;
            tail = tmp;
            tail->next = NULL;
        }
    }
    return deleted;
}
```



Delete Node with given input data

TRY IT YOURSELF