Double Linked List

CS212:Data Structure

ADT List: Specification

Elements: The elements are of generic type <Type> (The elements are placed in nodes for linked list implementation).

Structure: the elements are linearly arranged. The first element is called head, there is a element called current.

Domain: the number of elements in the list is bounded therefore the domain is finite. Type name of elements in the domain: List

ADT List: Specification

Operations: We assume all operations operate on a list L.

1. Method FindFirst ( )

   requires: list L is not empty. input: none

   results: first element set as the current element. output: none.

2. Method FindNext ( )

   requires: list L is not empty. Cur is not last. input: none

   results: element following the current element is made the current element.

   output: none.

2. Method FindPrevious ( )

   requires: list L is not empty. Cur is not Head. input: none

   results: element Previous to the current element is made the current element.

   output: none.

3. Method Retrieve (Type e)

   requires: list L is not empty. input: none

   results: current element is copied into e. output: element e.

ADT List: Specification

Operations:

4. Method Update (Type e).

   requires: list L is not empty. input: e.

   results: the element e is copied into the current node.

   output: none.

5. Method Insert (Type e).

   requires: list L is not full. input: e.

   results: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. output: none.

ADT List: Specification

Operations:

6. Method Remove ( )

   requires: list L is not empty. input: none

   results: the current element is removed from the list. If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element otherwise first element is made the new current element. output: none.

7. Method Full (boolean flag)

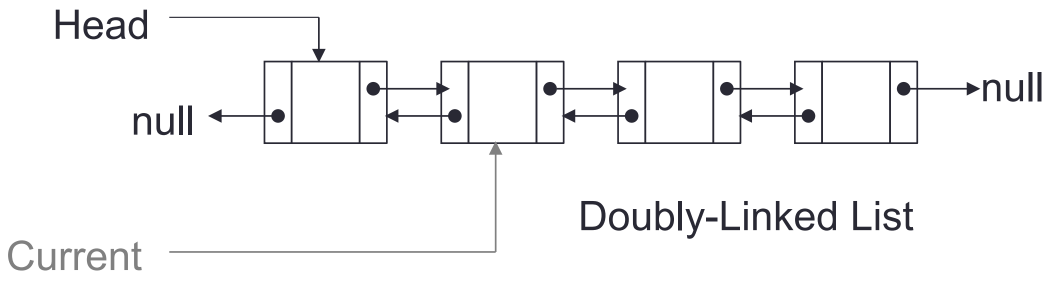
   input: none. returns: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. output: flag.

ADT List: Specification

Operations:

8. Method Empty (boolean flag).
   input: none. results: if the number of elements in L is zero, then flag is set to true otherwise false. Output: flag.
9. Method First (boolean flag).
   input: none. requires: L is not empty. Results: if the first element is the current element then flag is set to true otherwise false. Output: flag
10. Method Last (boolean flag).
   input: none. requires: L is not empty. Results: if the last element is the current element then flag is set to true otherwise false. Output: flag

List: Double-Linked List



ADT List (Double-Linked List): Element

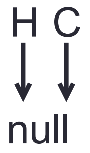
```
public class Node<T> {
        public T data;
        public Node<T> next;
        public Node<T> previous;

        public Node () {
                data = null;
                next = null;
                previous = null;
        }

        public Node (T val) {
                data = val;
                next = null;
                previous= null;
        }

        // Setters/Getters...
}
```

ADT List (Double-Linked List): Representation

```
public class DoubleLinkedList<T> {
        private Node<T> head;
        private Node<T> current;

        public DoubleLinkedList() {
                head = current = null;
        }

        public boolean empty() {
                return head == null;
        }

        public boolean last() {
                return current.next == null;
        }
  public boolean first() {
                return current.previous == null;
        }
```

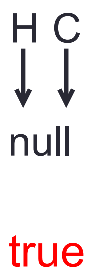
ADT List (Double-Linked List): Representation

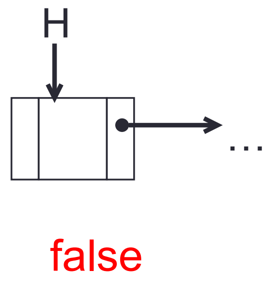
```
public class DoubleLinkedList<T> {
        private Node<T> head;
        private Node<T> current;

        public DoubleLinkedList() {
                head = current = null;
        }

        public boolean empty() {
                return head == null;
        }

        public boolean last() { isLast?
                return current.next == null;
        }
  public boolean first() { isFirst?
                return current.previous == null;
        }
```



ADT List (Double-Linked List): Representation

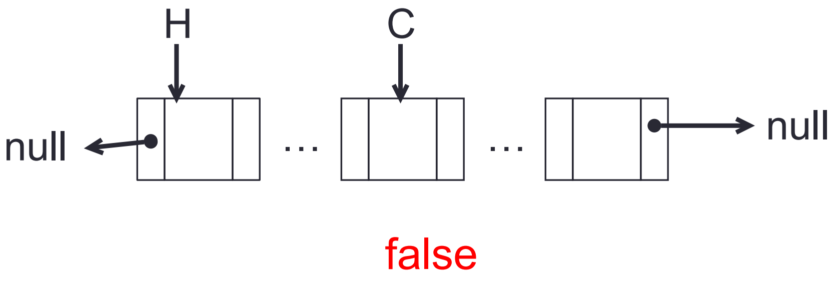
```
public class DoubleLinkedList<T> {
        private Node<T> head;
        private Node<T> current;

        public DoubleLinkedList() {
                head = current = null;
        }

        public boolean empty() {
                return head == null;
        }

        public boolean last() {
                return current.next == null;
        }
  public boolean first() {
                return current.previous == null;
        }
```

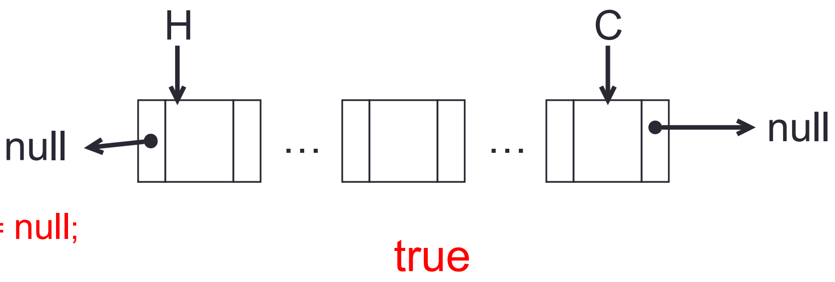




ADT List (Double-Linked List): Representation

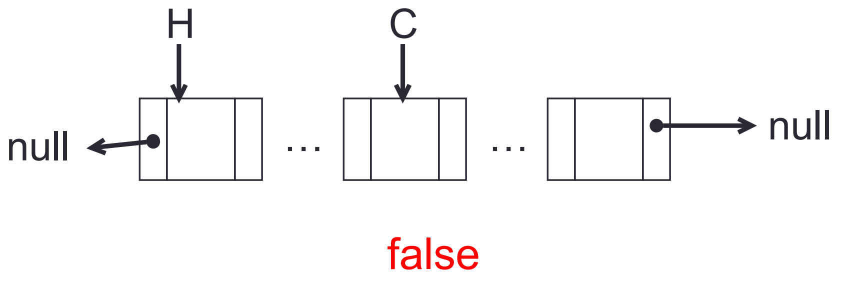
```
public class DoubleLinkedList<T> {
        private Node<T> head;
        private Node<T> current;

        public DoubleLinkedList() {
                head = current = null;
        }

        public boolean empty() {
                return head == null;
        }

        public boolean last() {
                return current.next == null;
        }
  public boolean first() {
                return current.previous == null;
        }
```

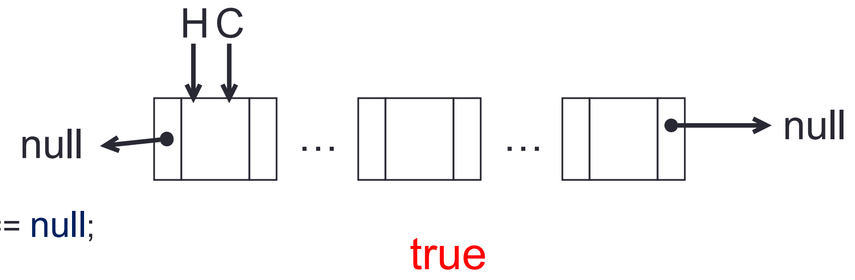




ADT List (Double-Linked List): Representation

```
public class DoubleLinkedList<T> {
        private Node<T> head;
        private Node<T> current;

        public DoubleLinkedList() {
                head = current = null;
        }

        public boolean empty() {
                return head == null;
        }

        public boolean last() {
                return current.next == null;
        }
  public boolean first() {
                return current.previous == null;
        }
```



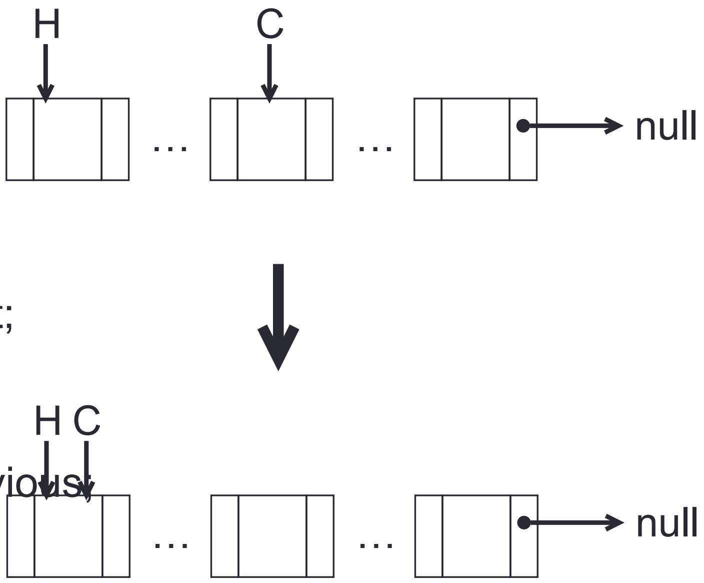


ADT List (Double-Linked List): Implementation

```
public boolean full() {
        return false;
}
public void findFirst() {
        current = head;
}
public void findNext() {
        current = current.next;
}
public void findPrevious() {
        current = current.previous;
}
public T retrieve() {
        return current.data;
}
public void update(T val) {
        current.data = val;
}
```

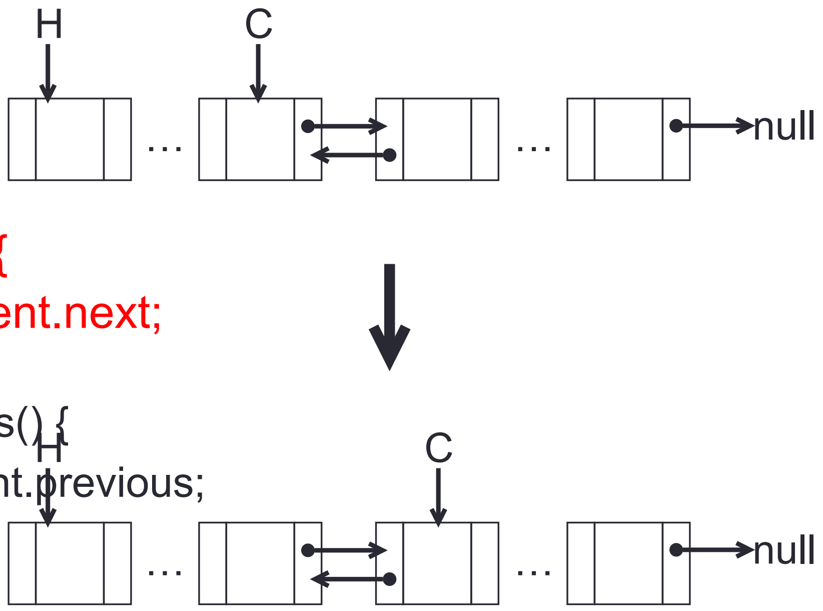
ADT List (Double-Linked List): Implementation

```
public boolean full() {
        return false;
}
public void findFirst() {
        current = head;
}
public void findNext() {
        current = current.next;
}
public void findPrevious() {
        current = current.previous;
}
public T retrieve() {
        return current.data;
}
public void update(T val) {
        current.data = val;
}
```



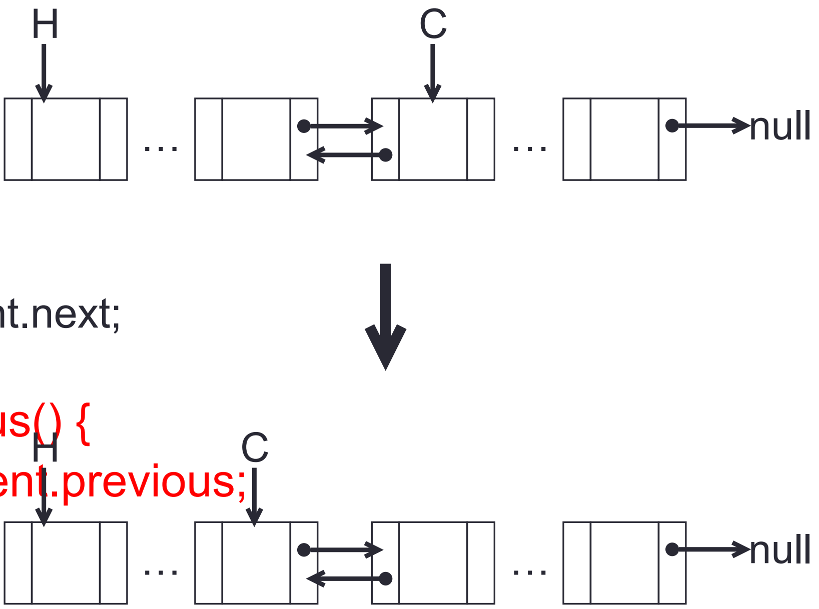
ADT List (Double-Linked List): Implementation

```
public boolean full() {
        return false;
}
public void findFirst() {
        current = head;
}
public void findNext() {
        current = current.next;
}
public void findPrevious() {
        current = current.previous;
}
public T retrieve() {
        return current.data;
}
public void update(T val) {
        current.data = val;
}
```



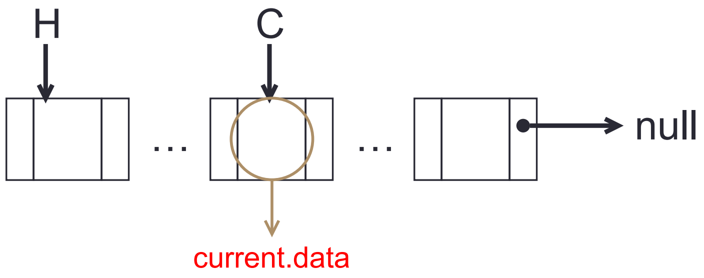
ADT List (Double-Linked List): Implementation

```
public boolean full() {
        return false;
}
public void findFirst() {
        current = head;
}
public void findNext() {
        current = current.next;
}
public void findPrevious() {
        current = current.previous;
}
public T retrieve() {
        return current.data;
}
public void update(T val) {
        current.data = val;
}
```



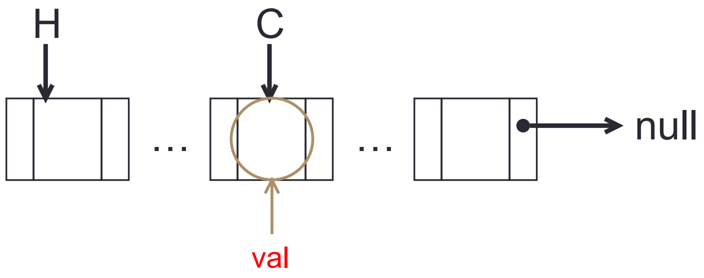
ADT List (Double-Linked List): Implementation

```
public boolean full() {
        return false;
}
public void findFirst() {
        current = head;
}
public void findNext() {
        current = current.next;
}
public void findPrevious() {
        current = current.previous;
}
public T retrieve() {
        return current.data;
}
public void update(T val) {
        current.data = val;
}
```



ADT List (Double-Linked List): Implementation

```
public boolean full() {
        return false;
}
public void findFirst() {
        current = head;
}
public void findNext() {
        current = current.next;
}
public void findPrevious() {
        current = current.previous;
}
public T retrieve() {
        return current.data;
}
public void update(T val) {
        current.data = val;
}
```



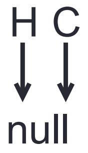
ADT List (Double-Linked List): Implementation

```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

ADT List (Double-Linked List): Implementation

Example #1

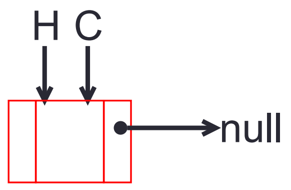
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #1

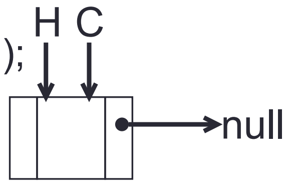
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #2

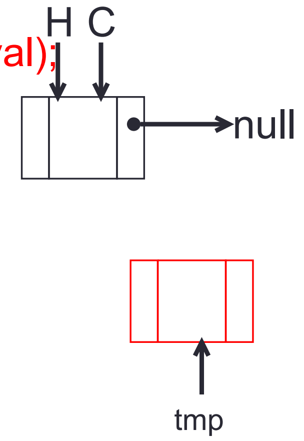
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #2

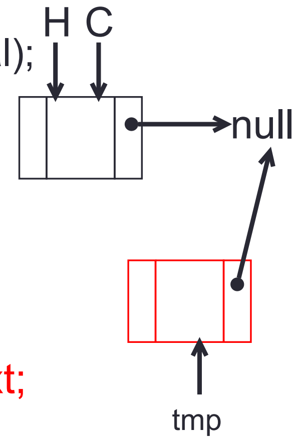
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #2

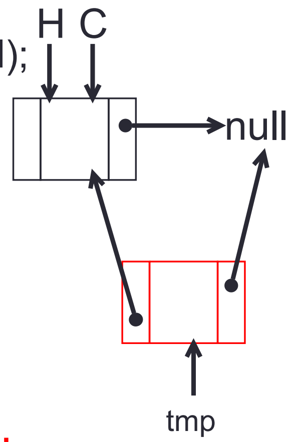
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #2

```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

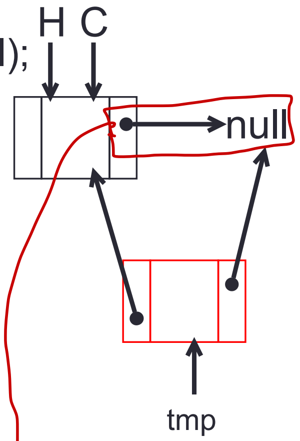


ADT List (Double-Linked List): Implementation

Example #2

```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

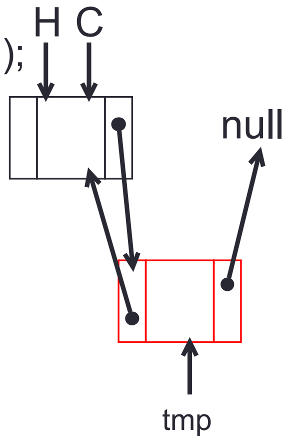
to avoid null pointer exception



ADT List (Double-Linked List): Implementation

Example #2

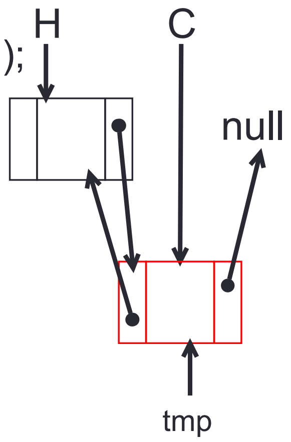
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

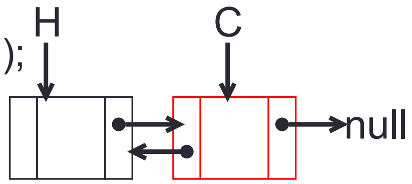
Example #2

```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

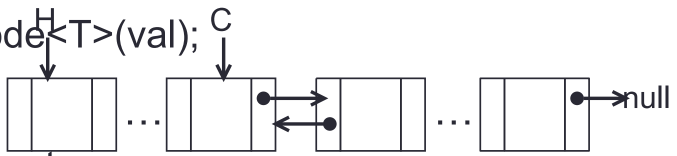
Example #2

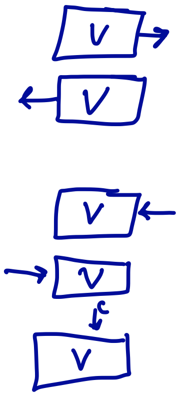


```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

ADT List (Double-Linked List): Implementation

Example #3





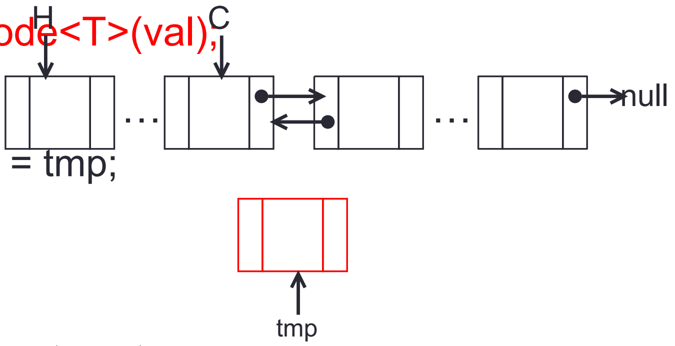
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

كيف أخلي النود اللي بعدي تأشر علي وهي مب موجودة ؟

ADT List (Double-Linked List): Implementation

Example #3

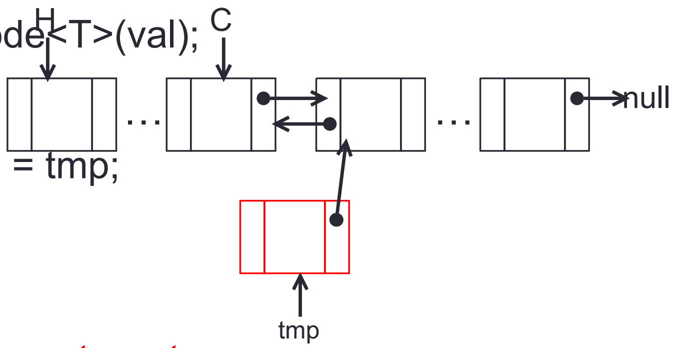
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

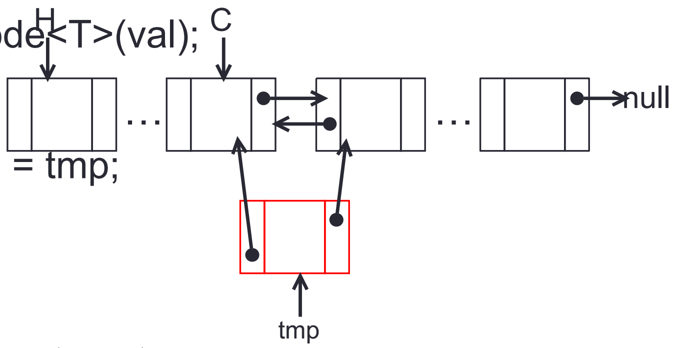
Example #3

```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

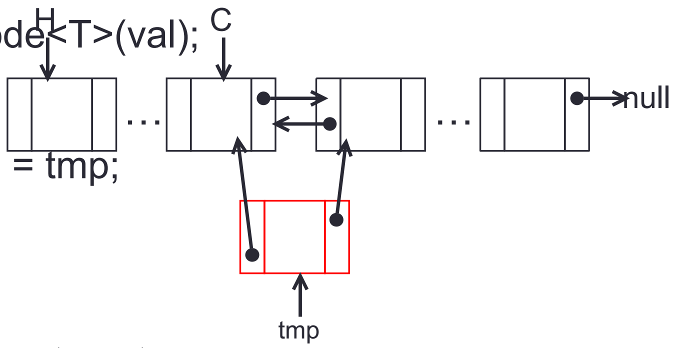
Example #3



```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

ADT List (Double-Linked List): Implementation

Example #3

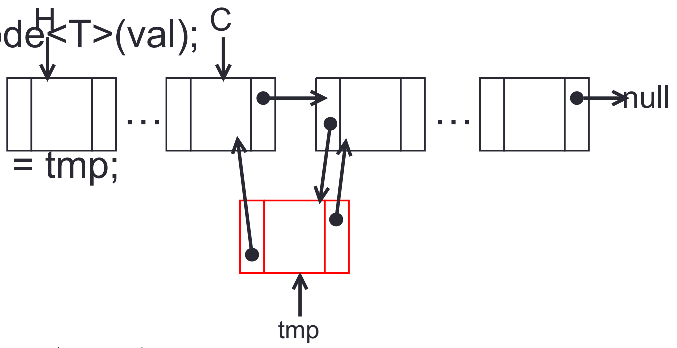


```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

ADT List (Double-Linked List): Implementation

Example #3

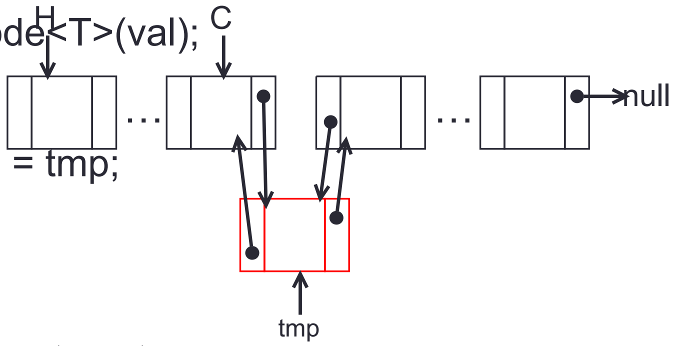
```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

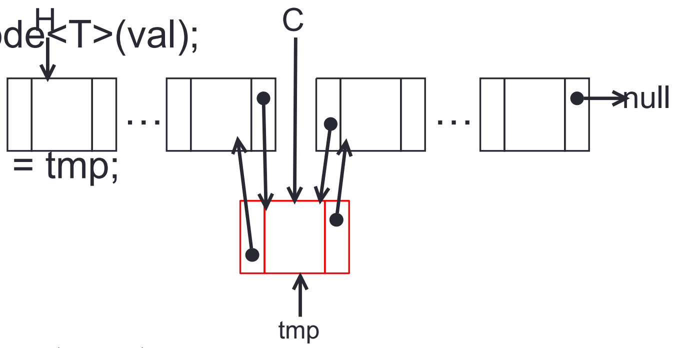
Example #3

```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```



ADT List (Double-Linked List): Implementation

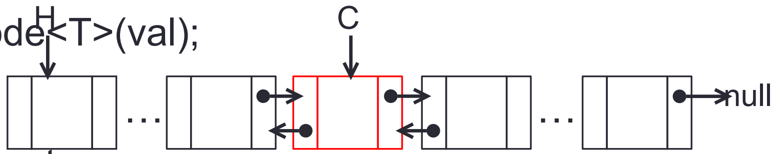
Example #3



```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

ADT List (Double-Linked List): Implementation

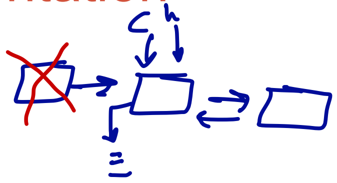
Example #3

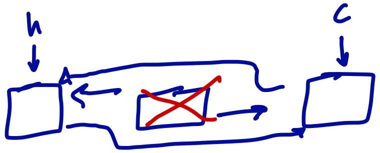


```
public void insert(T val) {
        Node<T> tmp = new Node<T>(val);
        if(empty()) {
                current = head = tmp;
        }
        else {
                tmp.next = current.next;
                tmp.previous = current;
                if(current.next != null)
                        current.next.previous = tmp;
                current.next = tmp;
                current = tmp;
        }
}
```

ADT List (Double-Linked List): Implementation

```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                   head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                   current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
}
}
```

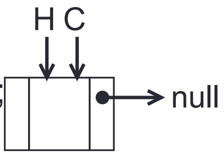




ADT List (Double-Linked List): Implementation

Example #1

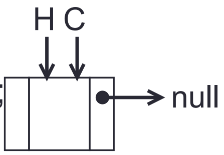
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

Example #1

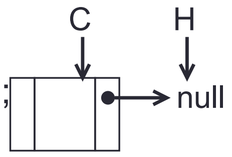
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
            head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
            current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

Example #1

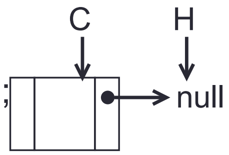
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                   head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                   current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

Example #1

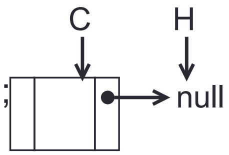
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
}
}
```



ADT List (Double-Linked List): Implementation

Example #1

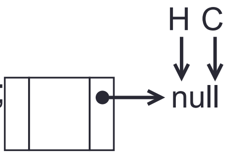
```
        public void remove() {
            if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
            }
            else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
            }

            if(current.next == null)
                current = head;
            else
                current = current.next;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #1

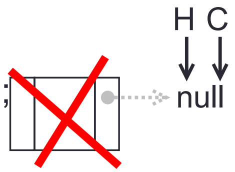
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

Example #1

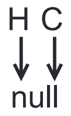
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
            head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
            current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```

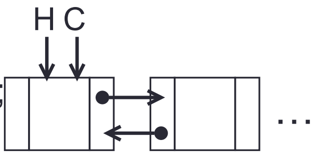
Example #1



ADT List (Double-Linked List): Implementation

Example #2

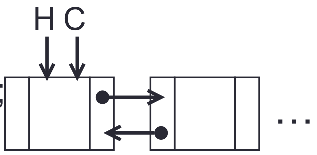
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
  }
}
```



ADT List (Double-Linked List): Implementation

Example #2

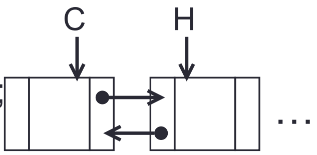
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                   head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                   current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
}
}
```



ADT List (Double-Linked List): Implementation

Example #2

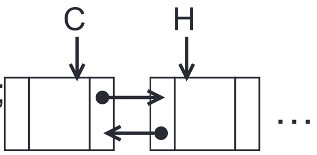
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

Example #2

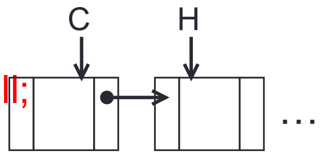
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```

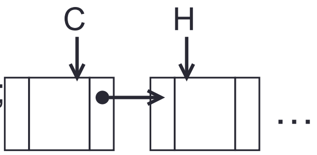
Example #2



ADT List (Double-Linked List): Implementation

Example #2

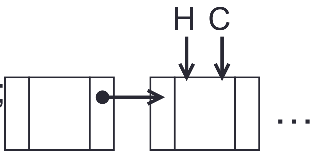
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
  }
}
```



ADT List (Double-Linked List): Implementation

Example #2

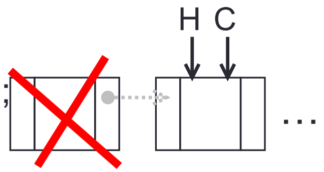
```
        public void remove() {
                if(current == head) {
                        head = head.next;
                        if(head != null)
                          head.previous = null;
                }
                else {
                        current.previous.next = current.next;
                        if(current.next != null)
                          current.next.previous = current.previous;
                }

                if(current.next == null)
                        current = head;
                else
                        current = current.next;
        }
}
```



ADT List (Double-Linked List): Implementation

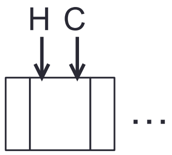
Example #2

```
        public void remove() {
            if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
            }
            else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
            }

            if(current.next == null)
                current = head;
            else
                current = current.next;
        }
}
```



ADT List (Double-Linked List): Implementation

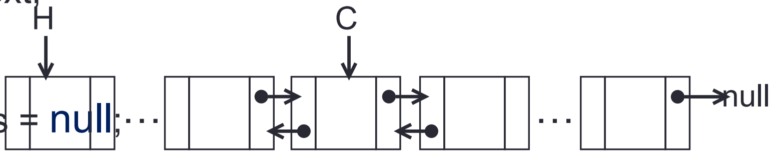
Example #2



```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

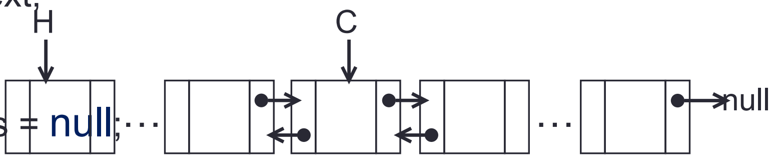
Example #3



```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;

    }
}
```

ADT List (Double-Linked List): Implementation

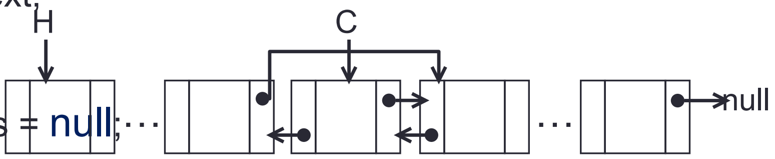
Example #3



```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

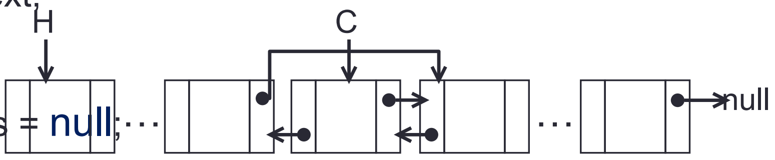
Example #3



```
        public void remove() {
            if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
            }
            else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
            }

            if(current.next == null)
                current = head;
            else
                current = current.next;
        }
}
```

ADT List (Double-Linked List): Implementation

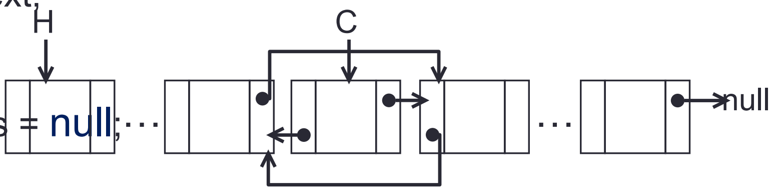
Example #3



```
          public void remove() {
                    if(current == head) {
                              head = head.next;
                              if(head != null)
                                head.previous = null;
                    }
                    else {
                              current.previous.next = current.next;
                              if(current.next != null)
                                current.next.previous = current.previous;
                    }

                    if(current.next == null)
                              current = head;
                    else
                              current = current.next;
          }
}
```

ADT List (Double-Linked List): Implementation

Example #3

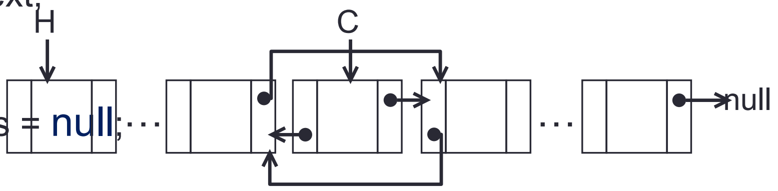


```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                   head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                   current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
        }
}
```

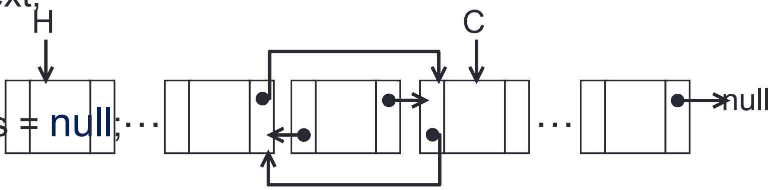
Example #3



ADT List (Double-Linked List): Implementation

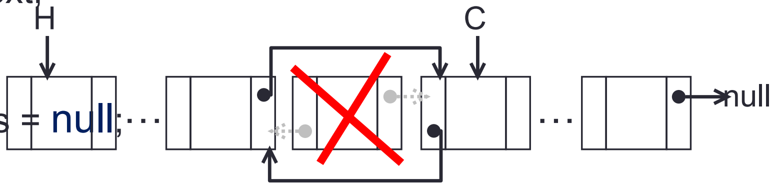
Example #3

```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
    }
}
```



ADT List (Double-Linked List): Implementation

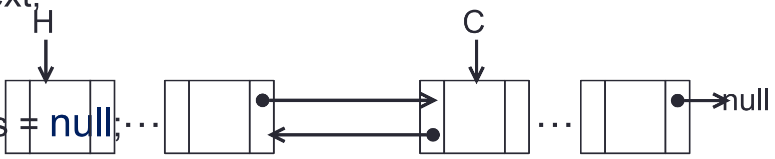
Example #3



```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

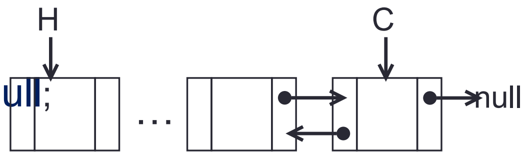
Example #3



```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

Example #4

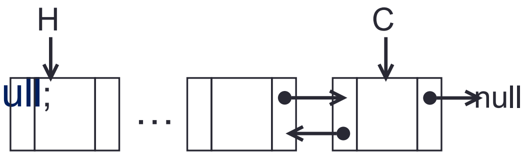


```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

Example #4

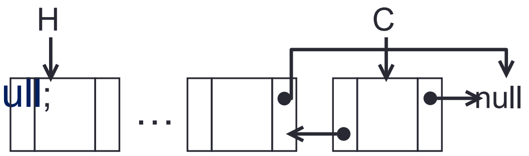
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
   }
}
```



ADT List (Double-Linked List): Implementation

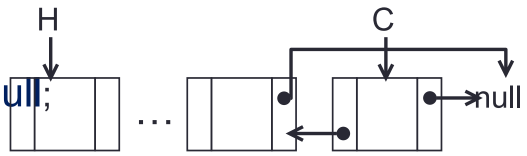
Example #4

```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
  }
}
```



ADT List (Double-Linked List): Implementation

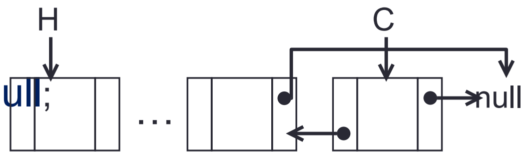
Example #4



```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

ADT List (Double-Linked List): Implementation

Example #4

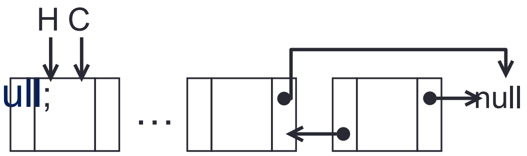


```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
  }
}
```

ADT List (Double-Linked List): Implementation

Example #4

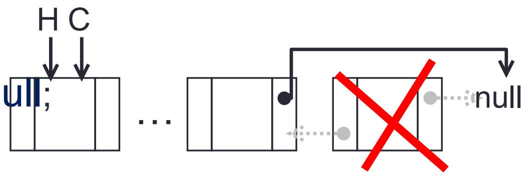
```
        public void remove() {
            if(current == head) {
                    head = head.next;
                    if(head != null)
                      head.previous = null;
            }
            else {
                    current.previous.next = current.next;
                    if(current.next != null)
                      current.next.previous = current.previous;
            }

            if(current.next == null)
                    current = head;
            else
                    current = current.next;
        }
}
```



ADT List (Double-Linked List): Implementation

Example #4

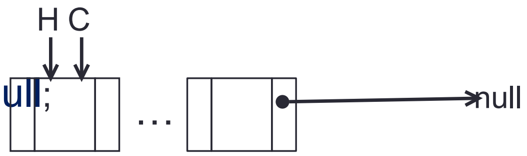
```
public void remove() {
    if(current == head) {
        head = head.next;
        if(head != null)
          head.previous = null;
    }
    else {
        current.previous.next = current.next;
        if(current.next != null)
          current.next.previous = current.previous;
    }

    if(current.next == null)
        current = head;
    else
        current = current.next;
  }
}
```



ADT List (Double-Linked List): Implementation

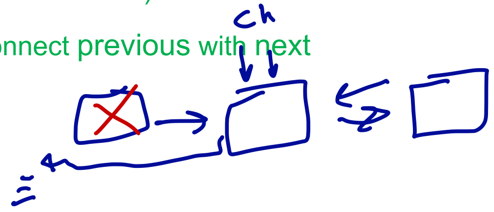
Example #4

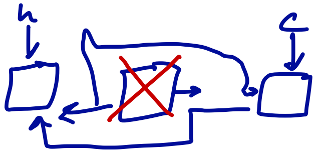
```
    public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                  head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
                  current.next.previous = current.previous;
        }

        if(current.next == null)
                current = head;
        else
                current = current.next;
    }
}
```

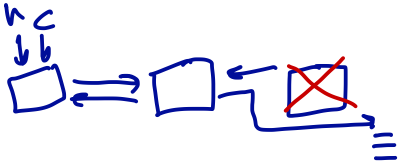


ADT List (Double-Linked List): Remove #2

```
// Another simpler implementation for remove (optional)
public void remove() {
        // if current is first only move right (no node before it)
        // otherwise (there is a node before it) connect previous with next
        if(current == head)
                head = head.next;
        else
                current.previous.next = current.next

        // if current is not last (there is a node after it), then connect next with previous
        if(current.next != null)
current.next.previous = current.previous;

        // move current either to first (when it is last)
        // otherwise, move it next
        if(current.next == null)
                current = head;
        else
                current = current.next;
}
```







Complexity so far?

Operation	Array List	Linked List	Double-Linked List
Empty			
Last			
Full			
FindFirst			
FindNext			
FindPrevious			
Retrieve			
Update			
Insert			
Remove			

Complexity so far?

Operation	Array List	Linked List	Double-Linked List
Empty	O(1)	O(1)	?
Last	O(1)	O(1)	?
Full	O(1)	O(1)	?
FindFirst	O(1)	O(1)	?
FindNext	O(1)	O(1)	?
FindPrevious	-	-	O(1) ?
Retrieve	O(1)	O(1)	?
Update	O(1)	O(1)	?
Insert	O(n)	O(1)	O(1) ?
Remove	O(n)	O(n)	O(1) ?

ToDo

- For Array List and Linked List:
  - Implement member method FindPrevious.
  - Find the complexity for both implementations.
- For Double-Linked List:
  - Find the complexity for all of the methods.
  - Implement the member method FindLast:

Method FindLast ( )

requires: list L is not empty. input: none

results: last element is set as the current element. output: none.

ADT List (Array List): FindPrevious

```
public void findPrevious() {
        current--;
}
```

ADT List (Linked List): FindPrevious

```
public void findPrevious() {
        Node<T> tmp = head;
        while(tmp.next != current)
                tmp = tmp.next;
        current = tmp;
}
```

ADT List (Double-Linked List): FindLast

```
public void findLast() {
        while(current.next != null)
                current = current.next;
}
```

O(n)

Complexity so far?

Operation	Array List	Linked List	Double-Linked List
Empty	O(1)	O(1)	O(1)
Last	O(1)	O(1)	O(1)
Full	O(1)	O(1)	O(1)
FindFirst	O(1)	O(1)	O(1)
FindNext	O(1)	O(1)	O(1)
FindPrevious	O(1)	O(n)	O(1)
Retrieve	O(1)	O(1)	O(1)
Update	O(1)	O(1)	O(1)
Insert	O(n)	O(1)	O(1)
Remove	O(n)	O(n)	O(1)