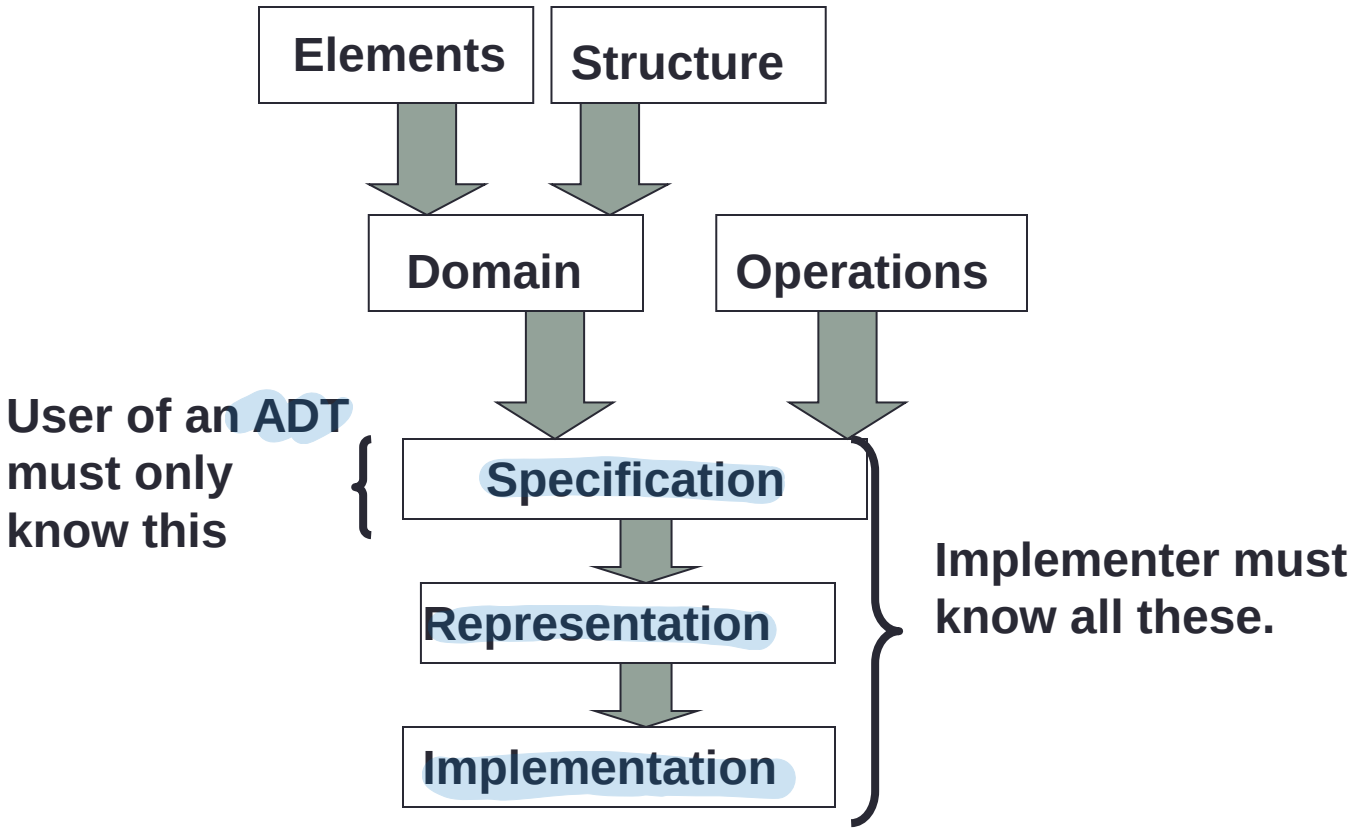
DATA STRUCTURES

ADT List

ADT List


Elements
Structure
Domain
Operations
Specification
Representation
Implementation
User of an ADT must only know this
Implementer must know all these.

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. Current is not last. **input**: none
   results: element following 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** last (boolean flag).

   input: none. **requires**: L is not empty.

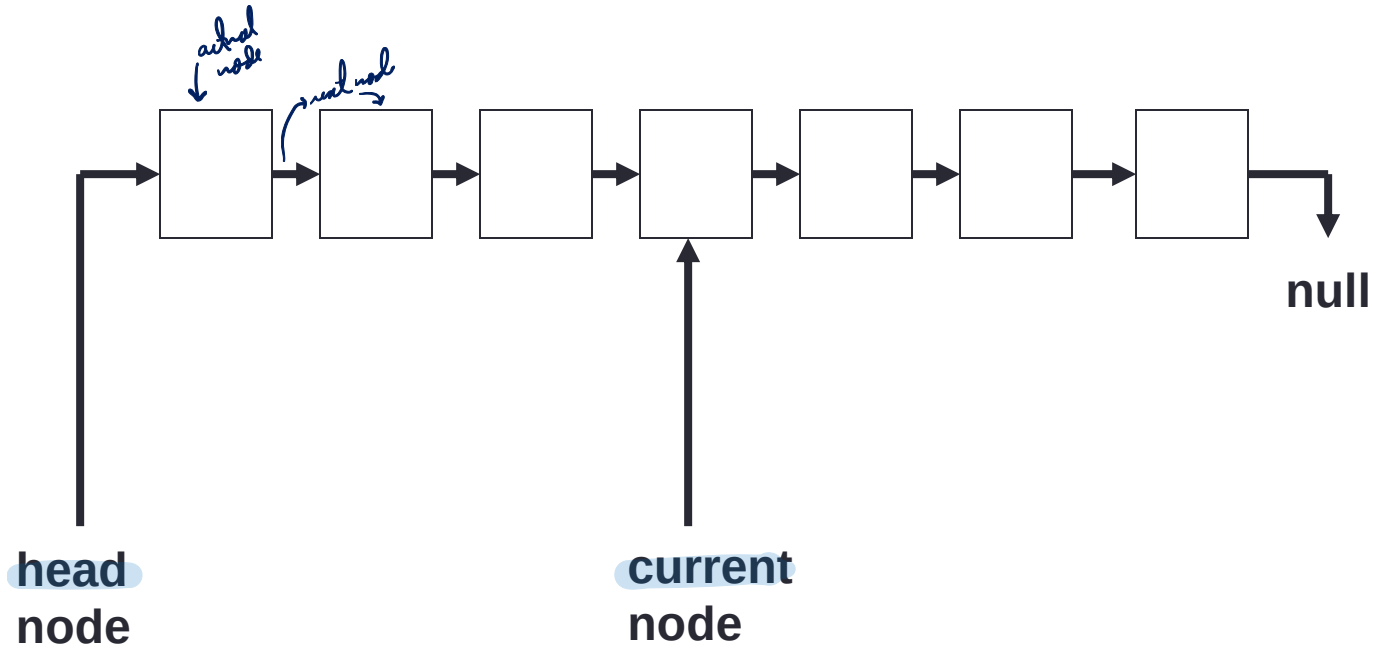
   Results: if the last element is the current element then flag is set to true otherwise false.

   Output: flag

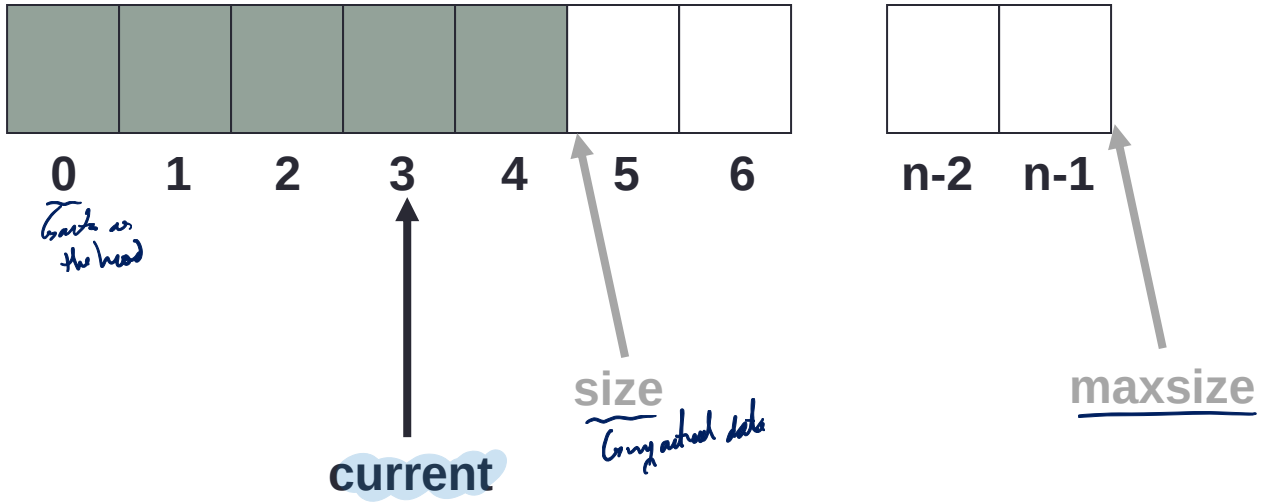
Representation/Implementation

- Programmers have to deal with the questions:
  - How to represent lists? Storage structure affects the efficiency of the operations. → linearly?
  - How to implement the operations? Algorithm chosen must be efficient.
- Lists can represented as
  1. Linked List
  2. Array based List

List (Linked List)



List (Array Based)



List Interface

```
public interface List<T>{
    public void findFirst( );
    public void findNext( );
    public T retrieve( );
    public void update(T e);
    public void insert(T e);
    public void remove( );
    public boolean full( );
    public boolean empty( );
    public boolean last( );
}
```

an interface is helpful to apply ADT

ADT List (Linked List): Element

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

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

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

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

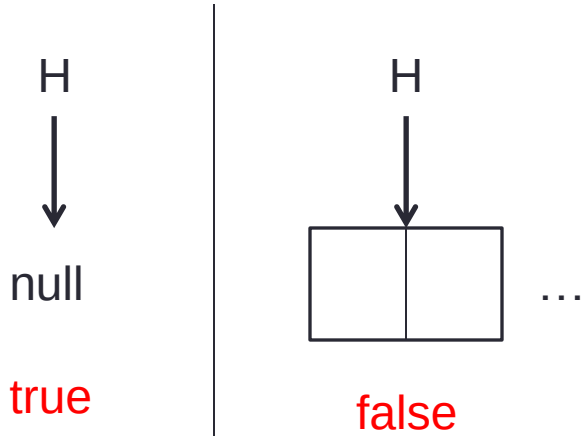
ADT List (Linked List): Representation

```
public class LinkedList<T> implements List<T>{
  private Node<T> head;
  private Node<T> current;

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

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

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

ADT List (Linked List): Representation

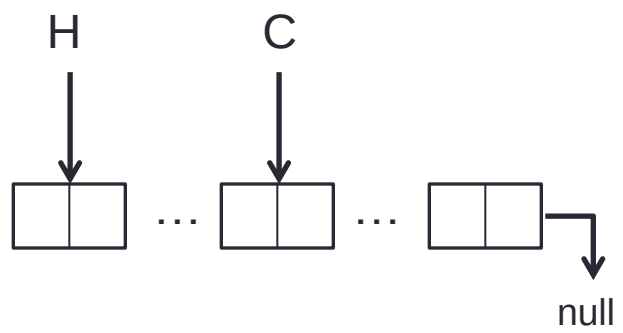
```
public class LinkedList<T> implements List<T>{
  private Node<T> head;
  private Node<T> current;

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

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

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

if current is the last elem return true
if current is NOT the last elem return false

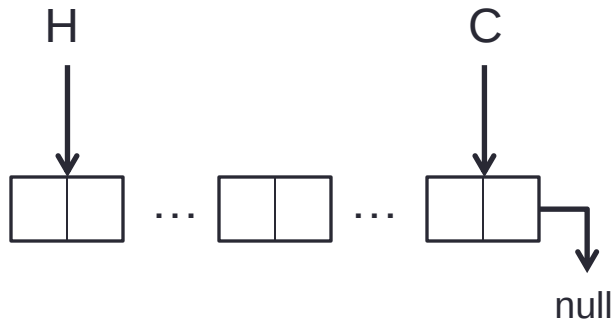


ADT List (Linked List): Representation

```
public class LinkedList<T> implements List<T>{
  private Node<T> head;
  private Node<T> current;

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

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

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



false



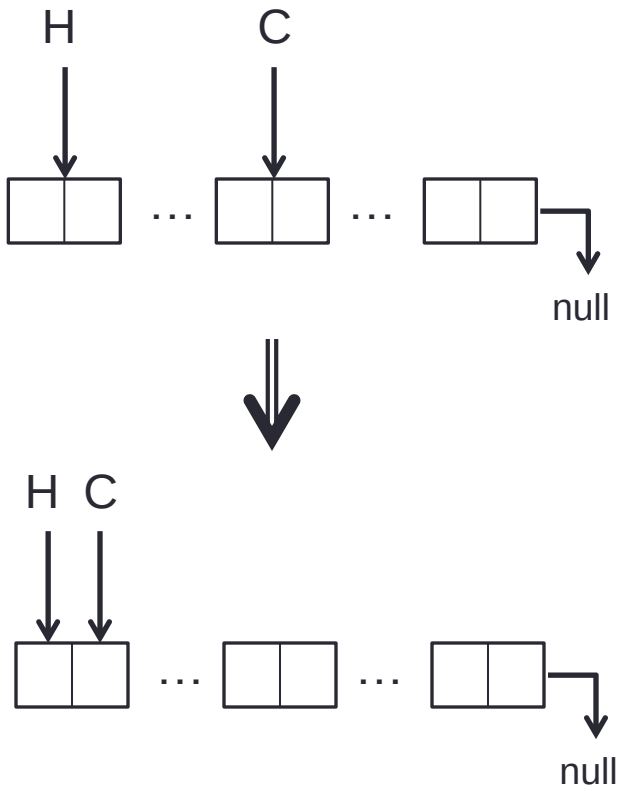
true

ADT List (Linked List): Implementation

```
public boolean full () {
      return false;
}
public void findfirst () {
      current = head;
}
public void findnext () {
      current = current.next;
}
public T retrieve () {
      return current.data;
}
public void update (T val) {
      current.data = val;
}
```

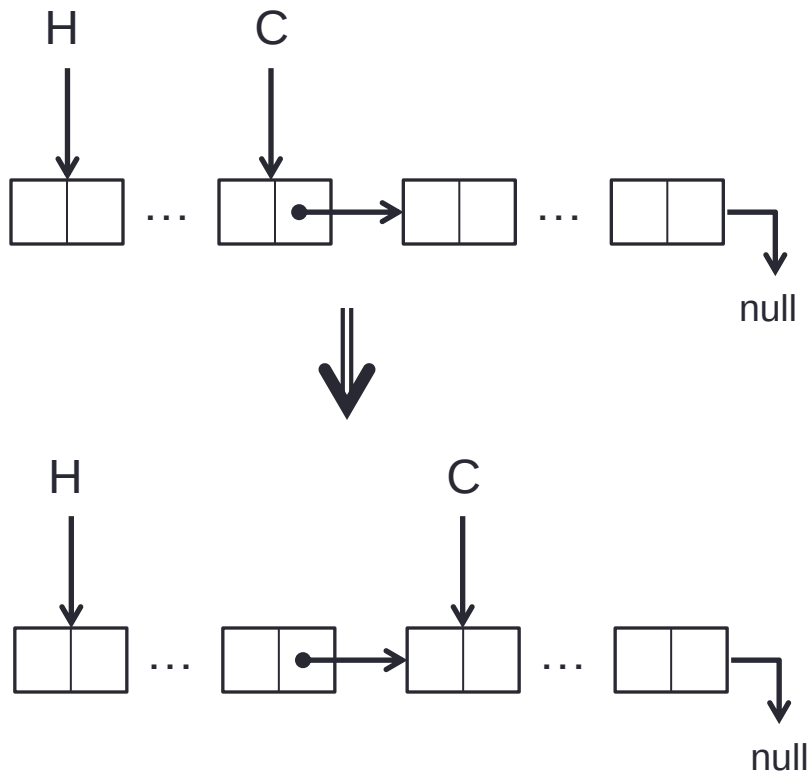
ADT List (Linked List): Implementation

```
public boolean full () {
      return false;
}
public void findfirst () {
      current = head;
}
public void findnext () {
      current = current.next;
}
public T retrieve () {
      return current.data;
}
public void update (T val) {
      current.data = val;
}
```



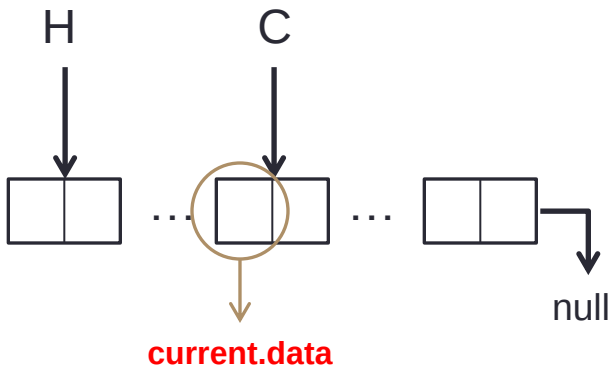
ADT List (Linked List): Implementation

```
public boolean full () {
      return false;
}
public void findfirst () {
      current = head;
}
public void findnext () {
      current = current.next;
}
public T retrieve () {
      return current.data;
}
public void update (T val) {
      current.data = val;
}
```



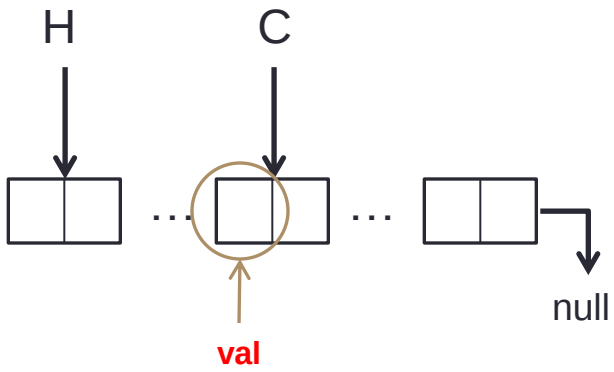
ADT List (Linked List): Implementation

```
public boolean full () {
      return false;
}
public void findfirst () {
      current = head;
}
public void findnext () {
      current = current.next;
}
public T retrieve () {
      return current.data;
}
public void update (T val) {
      current.data = val;
}
```



ADT List (Linked List): Implementation

```
public boolean full () {
      return false;
}
public void findfirst () {
      current = head;
}
public void findnext () {
      current = current.next;
}
public T retrieve () {
      return current.data;
}
public void update (T val) {
      current.data = val;
}
```



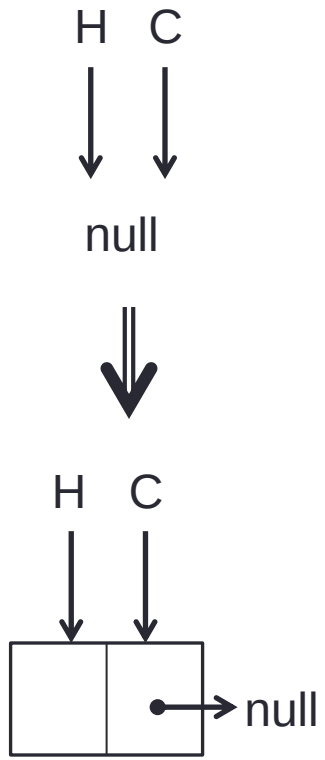
ADT List (Linked List): Implementation

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

ADT List (Linked List): Implementation

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

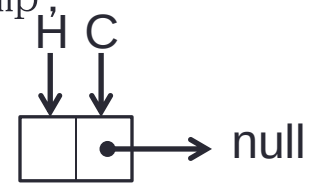
Example #1



ADT List (Linked List): Implementation

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

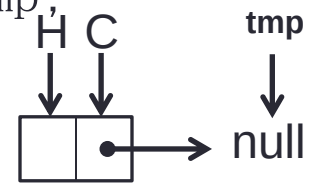
Example #2



ADT List (Linked List): Implementation

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

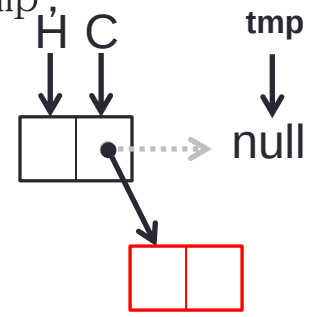
Example #2



ADT List (Linked List): Implementation

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

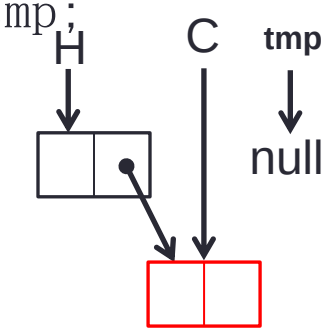
Example #2



ADT List (Linked List): Implementation

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

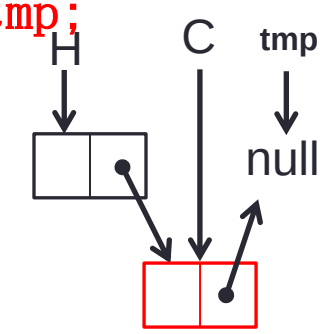
Example #2



ADT List (Linked List): Implementation

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

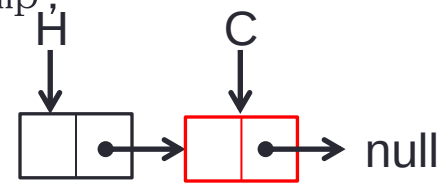
Example #2



ADT List (Linked List): Implementation

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

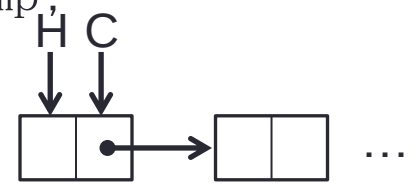
Example #2



ADT List (Linked List): Implementation

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

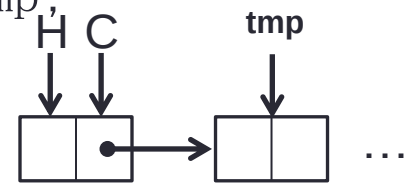
Example #3



ADT List (Linked List): Implementation

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

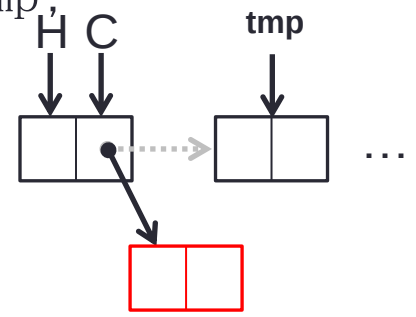
Example #3



ADT List (Linked List): Implementation

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

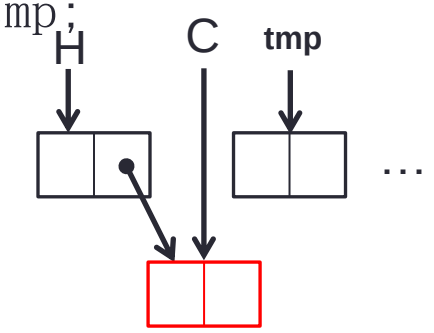
Example #3



ADT List (Linked List): Implementation

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

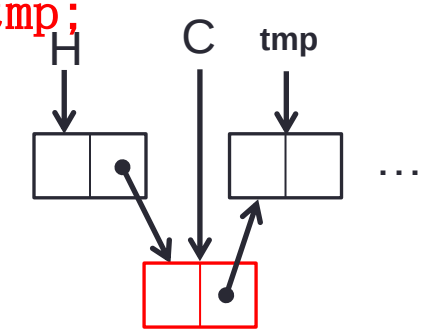
Example #3



ADT List (Linked List): Implementation

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

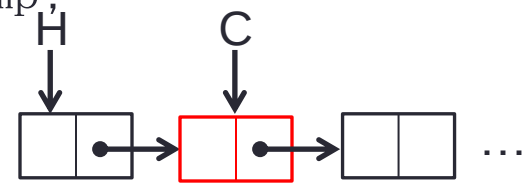
Example #3



ADT List (Linked List): Implementation

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

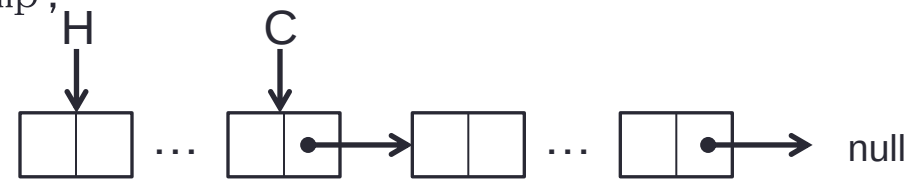
Example #3



ADT List (Linked List): Implementation

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

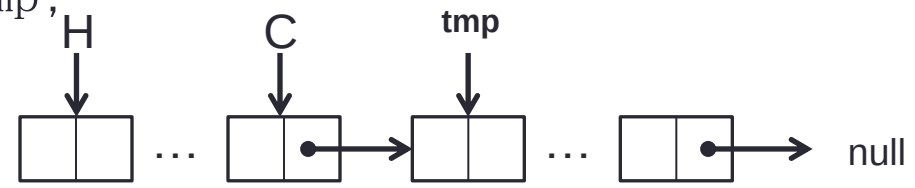
Example #4



ADT List (Linked List): Implementation

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

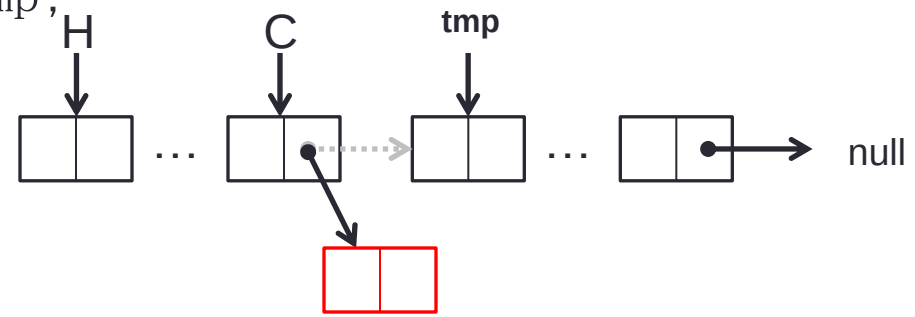
Example #4



ADT List (Linked List): Implementation

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

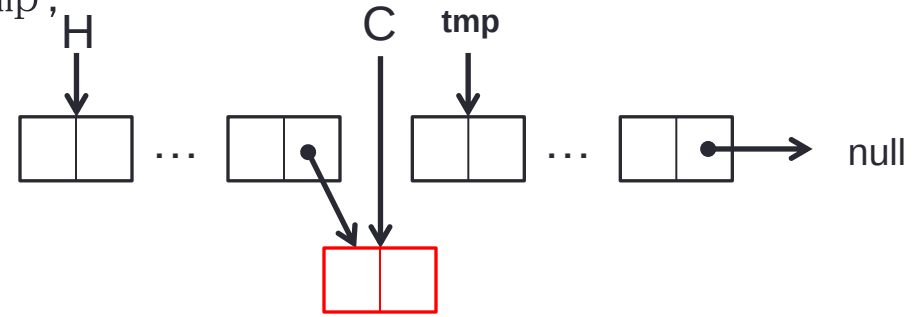
Example #4



ADT List (Linked List): Implementation

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

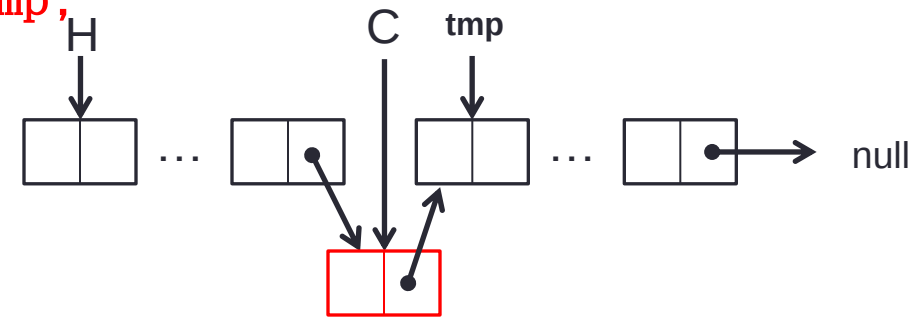
Example #4



ADT List (Linked List): Implementation

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

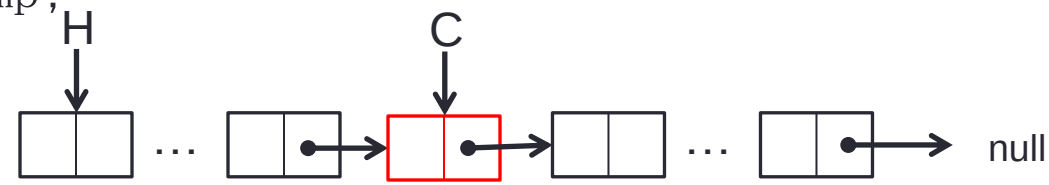
Example #4



ADT List (Linked List): Implementation

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

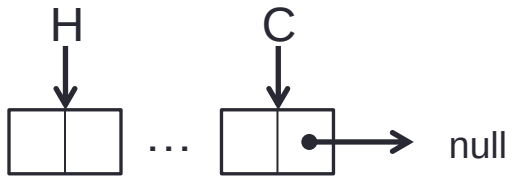
Example #4



ADT List (Linked List): Implementation

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

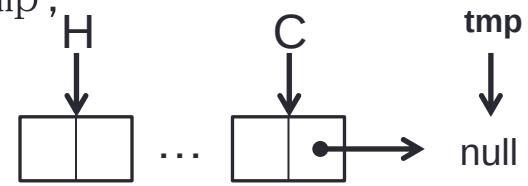
Example #5



ADT List (Linked List): Implementation

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

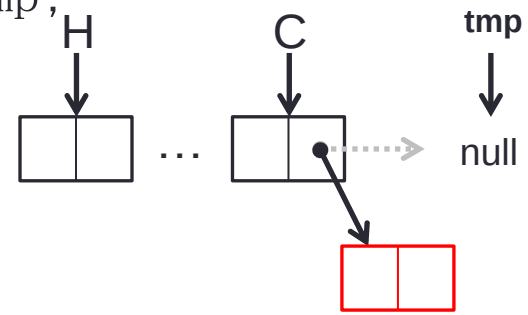
Example #5



ADT List (Linked List): Implementation

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

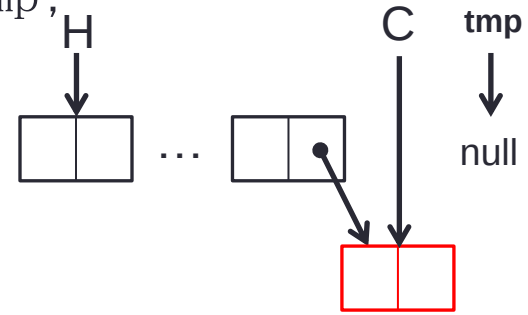
Example #5



ADT List (Linked List): Implementation

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

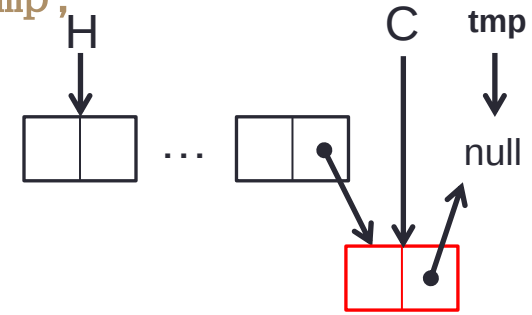
Example #5



ADT List (Linked List): Implementation

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

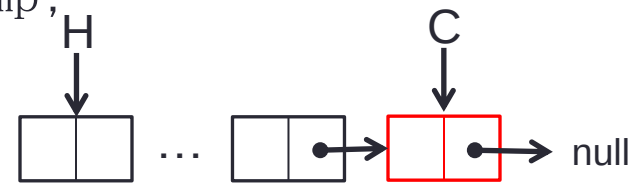
Example #5



ADT List (Linked List): Implementation

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

Example #5

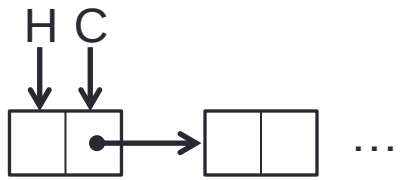


ADT List (Linked List): Implementation

```
public void remove () {
        if (current == head) {
                head = head.next; //removes the head
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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

ADT List (Linked List): Implementation

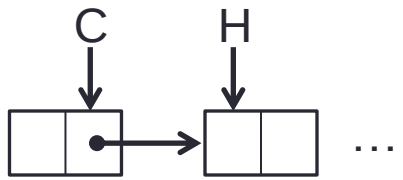
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #1

ADT List (Linked List): Implementation

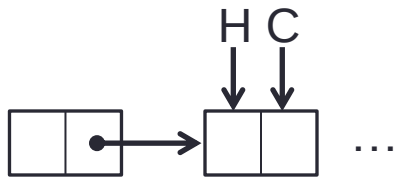
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #1

ADT List (Linked List): Implementation

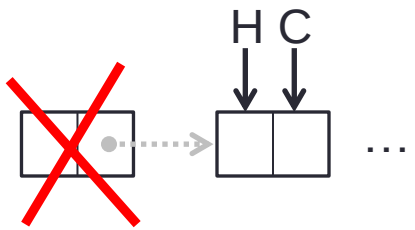
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #1

ADT List (Linked List): Implementation

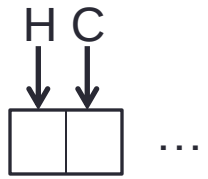
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #1

ADT List (Linked List): Implementation

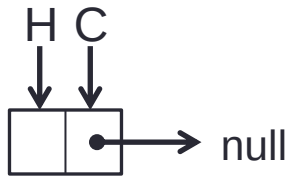
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #1

ADT List (Linked List): Implementation

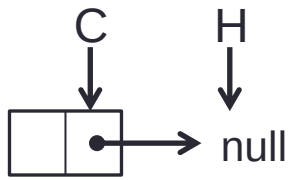
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #2

ADT List (Linked List): Implementation

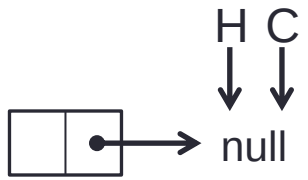
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #2

ADT List (Linked List): Implementation

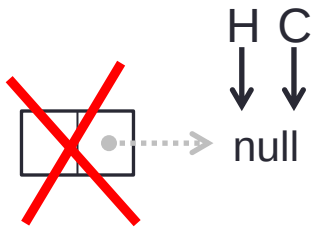
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #2

ADT List (Linked List): Implementation

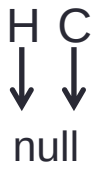
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #2

ADT List (Linked List): Implementation

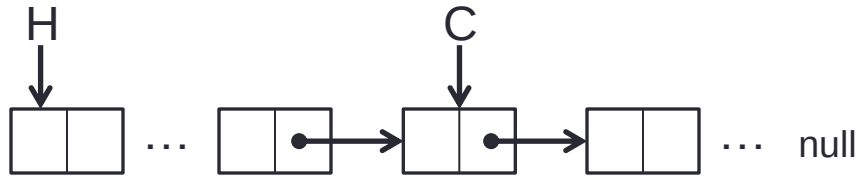
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #2

ADT List (Linked List): Implementation

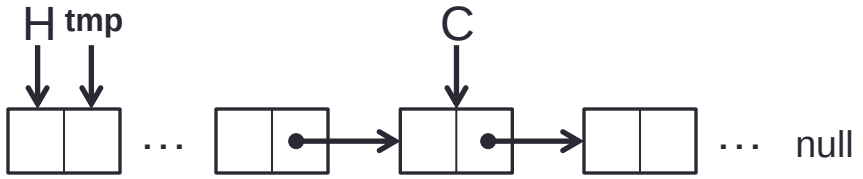
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

ADT List (Linked List): Implementation

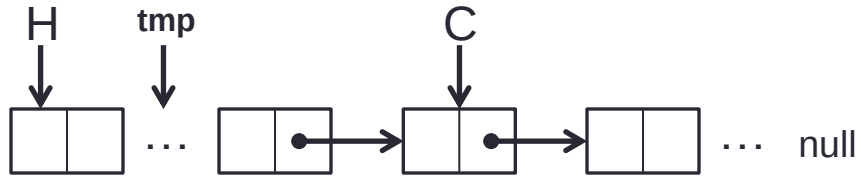
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

ADT List (Linked List): Implementation

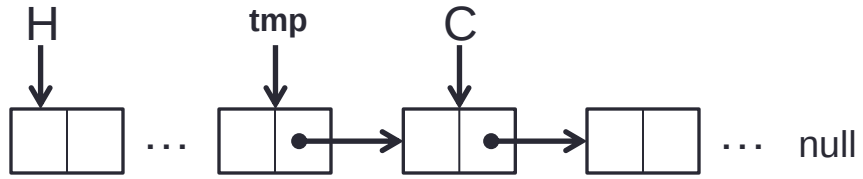
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

ADT List (Linked List): Implementation

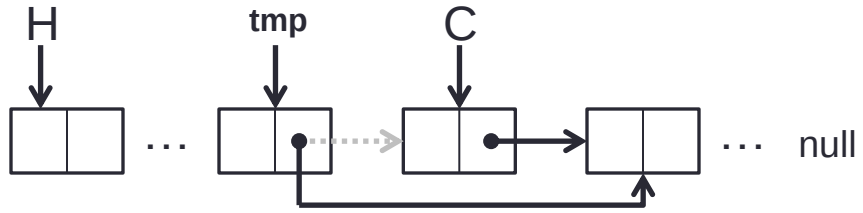
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

ADT List (Linked List): Implementation

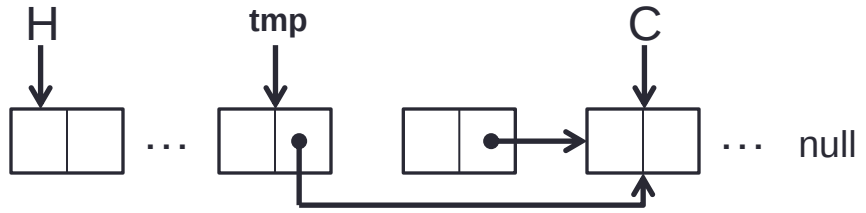
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

ADT List (Linked List): Implementation

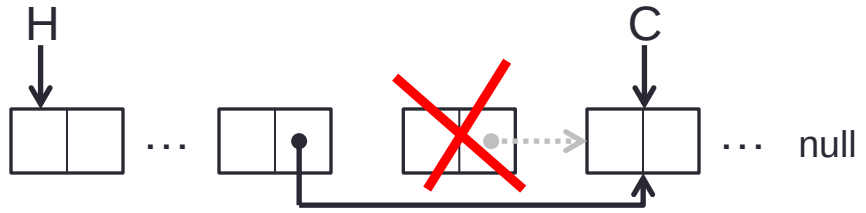
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

ADT List (Linked List): Implementation

```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #3

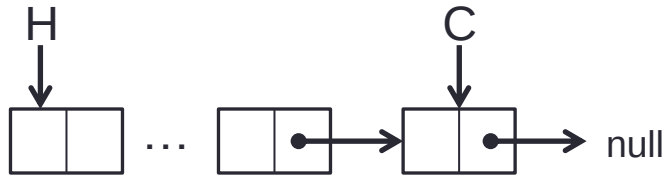
ADT List (Linked List): Implementation

```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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

Example #3

ADT List (Linked List): Implementation

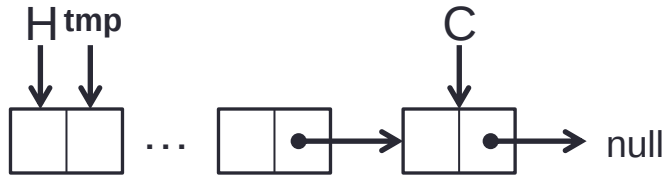
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

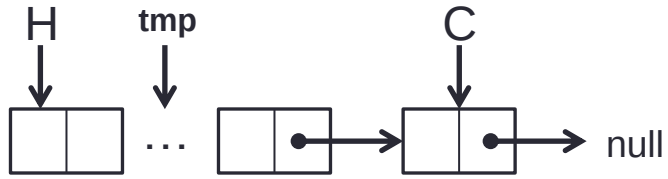
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

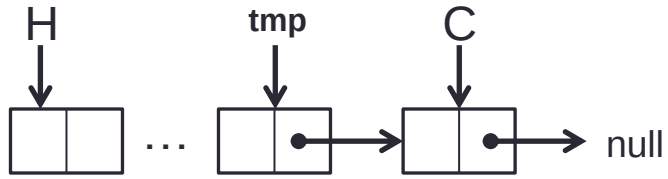
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

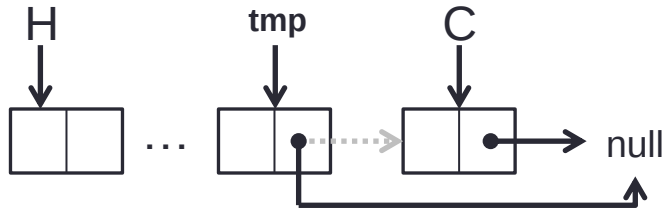
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

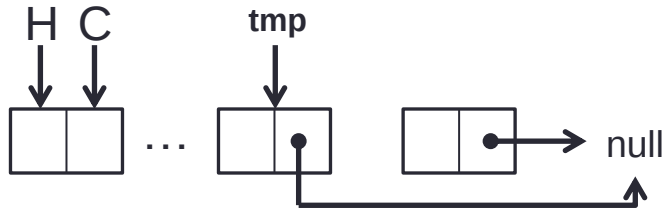
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

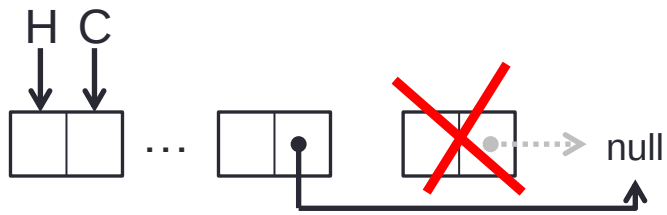
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

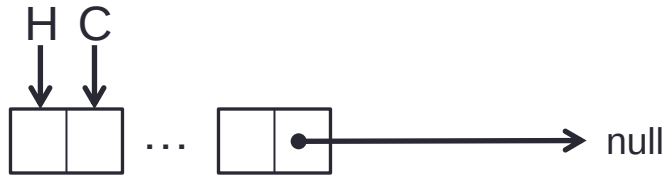
```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



Example #4

ADT List (Linked List): Implementation

```
public void remove () {
        if (current == head) {
                head = head.next;
        }
        else {
                Node<T> tmp = head;

                while (tmp.next != current)
                        tmp = tmp.next;

                tmp.next = current.next;
        }

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



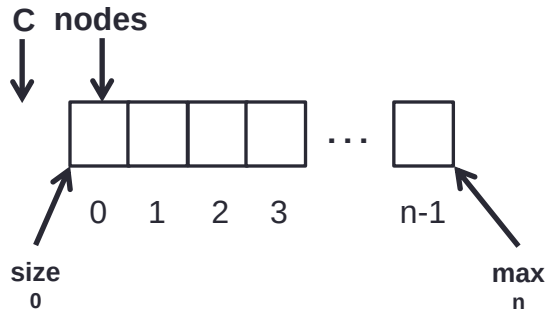
Example #4

ADT List (Array): Representation

```
public class ArrayList<T> implements List<T>
{
  private int maxsize;
  private int size;
  private int current;
  private T[] nodes;

  /** Creates a new instance of ArrayList */
  public ArrayList(int n) {
        maxsize = n;
        size = 0;
        current = -1;
        nodes = (T[]) new Object[n];
  }
```

ADT List (Array): Representation

```
public class ArrayList<T> implements List<T>
{
  private int maxsize;
  private int size;
  private int current;
  private T[] nodes;

  /** Creates a new instance of ArrayList */
  public ArrayList(int n) {
        maxsize = n;
        size = 0;
        current = -1;
        nodes = (T[]) new Object[n];
  }
```



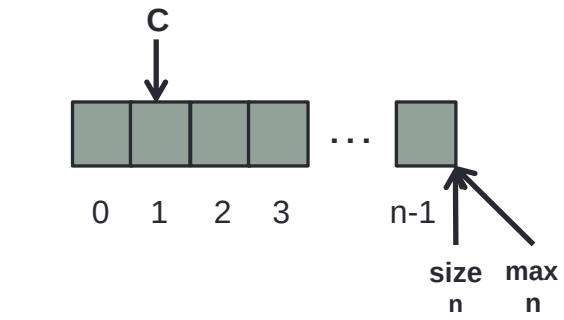
ADT List (Array): Implementation

```
public boolean full () {
      return size == maxsize;
}
public boolean empty () {
      return size == 0;
}
public boolean last () {
      return current == size - 1;
}
public void findFirst () {
      current = 0;
}
public void findNext () {
      current++;
}
```

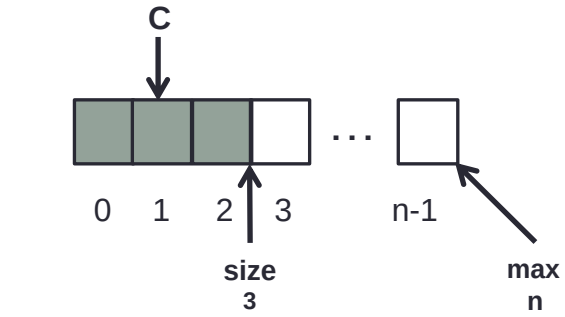
↳ last index

ADT List (Array): Implementation

```
public boolean full () {
      return size == maxsize;
}
public boolean empty () {
      return size == 0;
}
public boolean last () {
      return current == size - 1;
}
public void findFirst () {
      current = 0;
}
public void findNext () {
      current++;
}
```



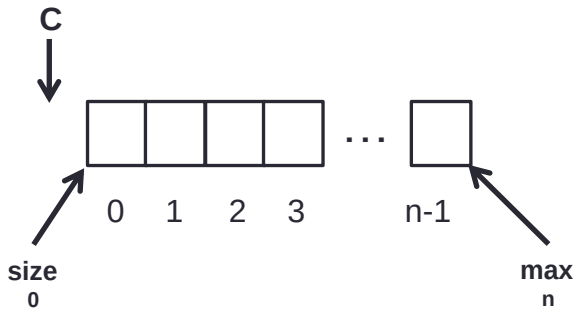
true



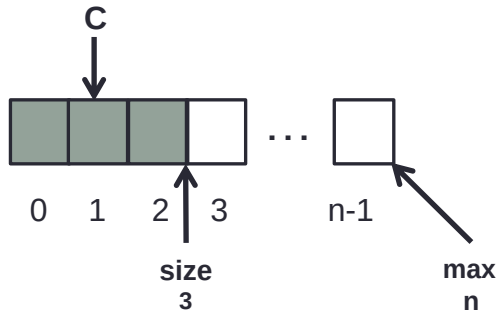
false

ADT List (Array): Implementation

```
public boolean full () {
      return size == maxsize;
}
public boolean empty () {
      return size == 0;
}
public boolean last () {
      return current == size - 1;
}
public void findFirst () {
      current = 0;
}
public void findNext () {
      current++;
}
```



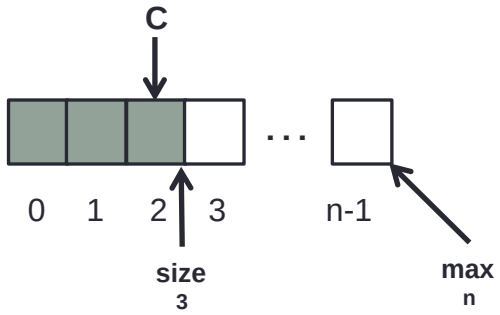
true



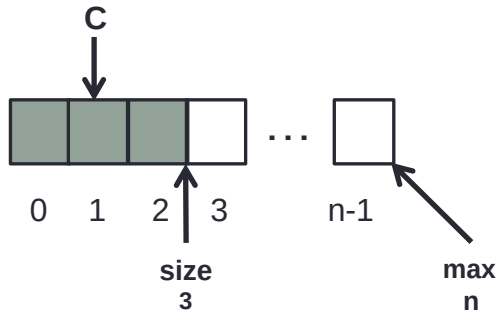
false

ADT List (Array): Implementation

```
public boolean full () {
      return size == maxsize;
}
public boolean empty () {
      return size == 0;
}
public boolean last () {
      return current == size - 1;
}
public void findFirst () {
      current = 0;
}
public void findNext () {
      current++;
}
```



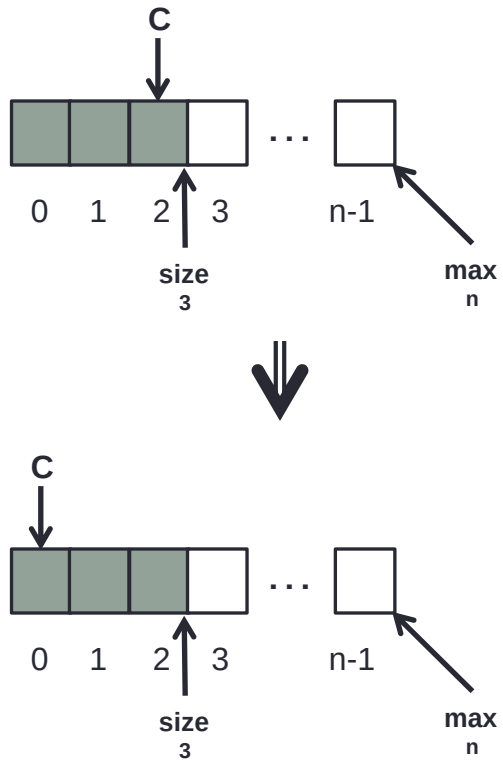
true



false

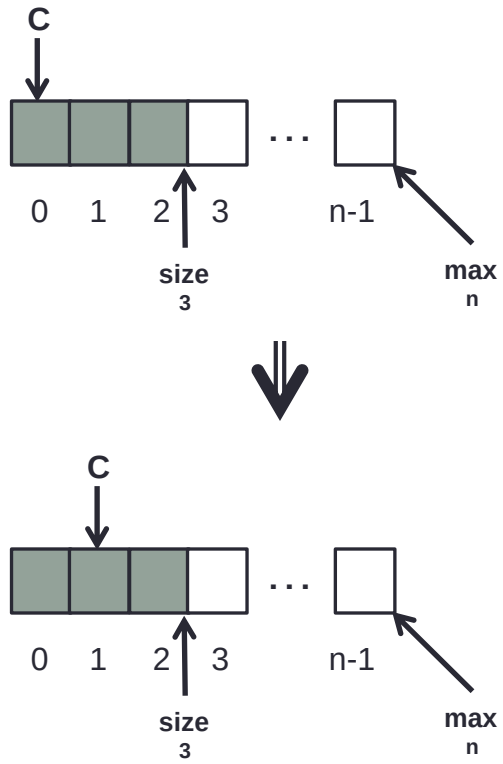
ADT List (Array): Implementation

```
public boolean full () {
      return size == maxsize;
}
public boolean empty () {
      return size == 0;
}
public boolean last () {
      return current == size - 1;
}
public void findFirst () {
      current = 0;
}
public void findNext () {
      current++;
}
```



ADT List (Array): Implementation

```
public boolean full () {
      return size == maxsize;
}
public boolean empty () {
      return size == 0;
}
public boolean last () {
      return current == size - 1;
}
public void findFirst () {
      current = 0;
}
public void findNext () {
      current++;
}
```

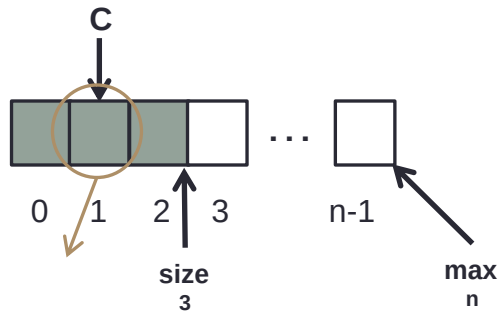


ADT List (Array): Implementation

```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```

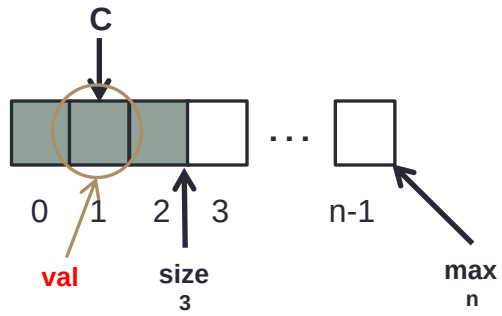
ADT List (Array): Implementation

```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



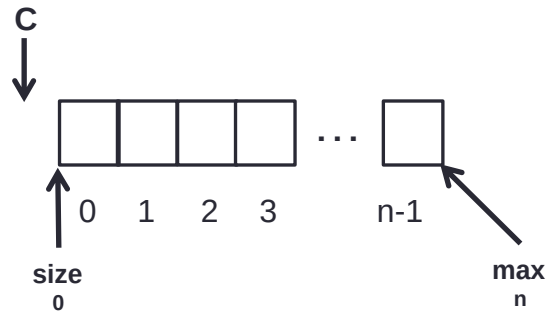
ADT List (Array): Implementation

```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



ADT List (Array): Implementation

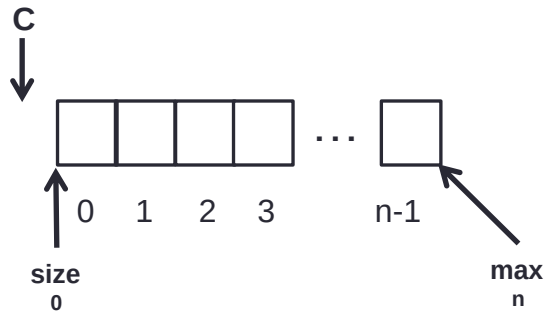
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #1

ADT List (Array): Implementation

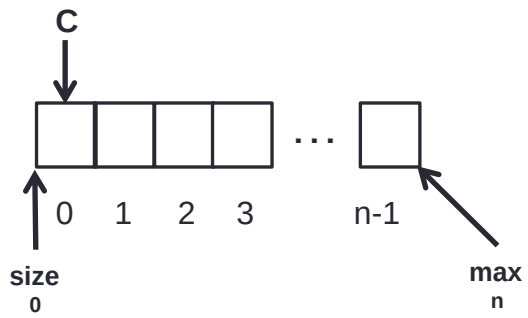
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #1

ADT List (Array): Implementation

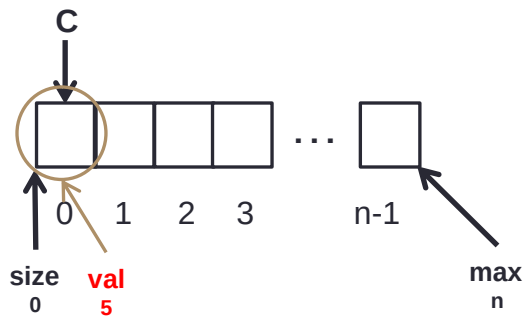
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #1

ADT List (Array): Implementation

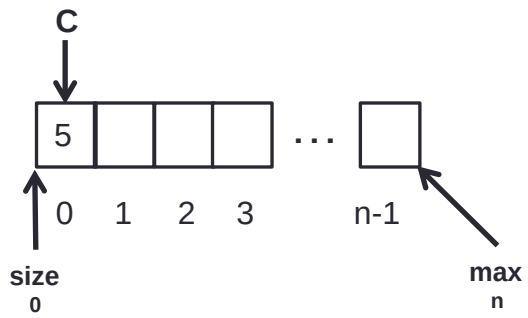
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #1

ADT List (Array): Implementation

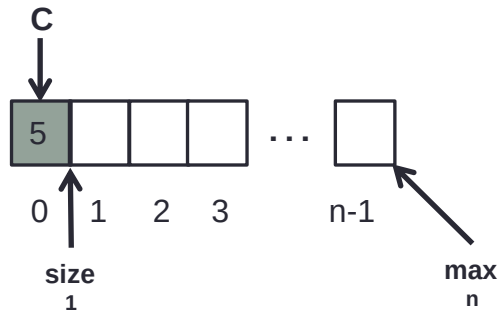
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #1

ADT List (Array): Implementation

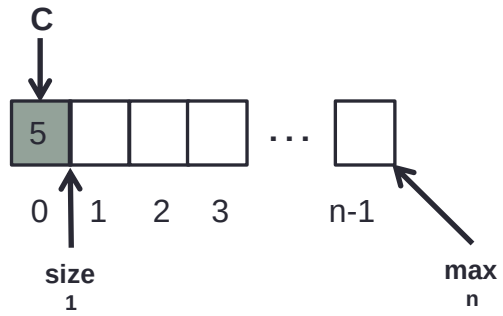
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #1

ADT List (Array): Implementation

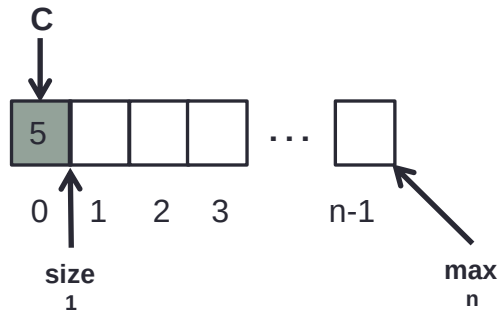
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #2

ADT List (Array): Implementation

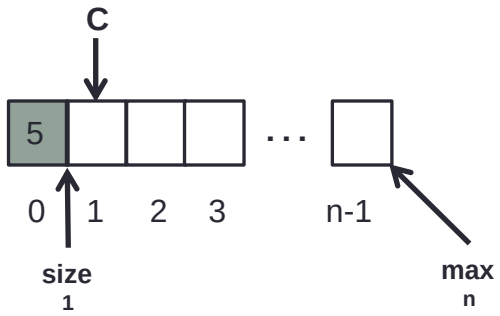
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #2

ADT List (Array): Implementation

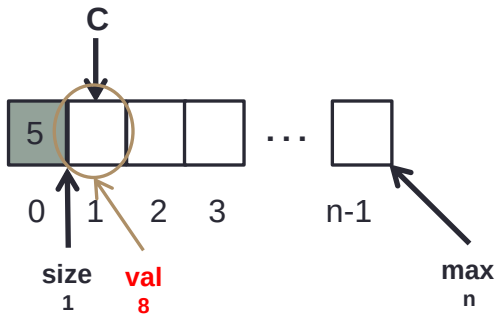
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #2

ADT List (Array): Implementation

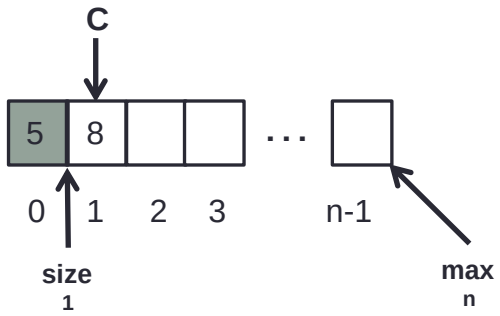
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #2

ADT List (Array): Implementation

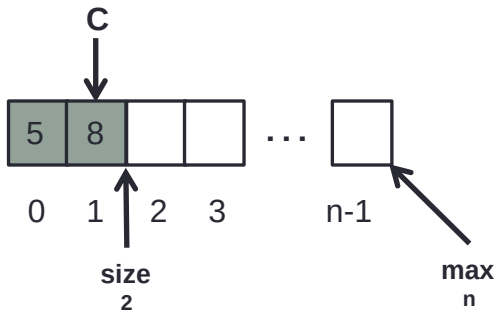
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #2

ADT List (Array): Implementation

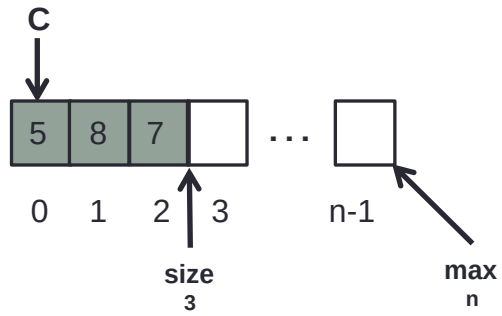
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #2

ADT List (Array): Implementation

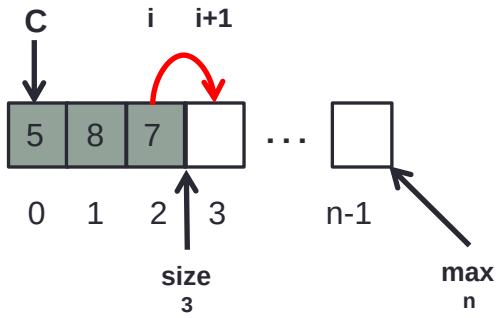
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

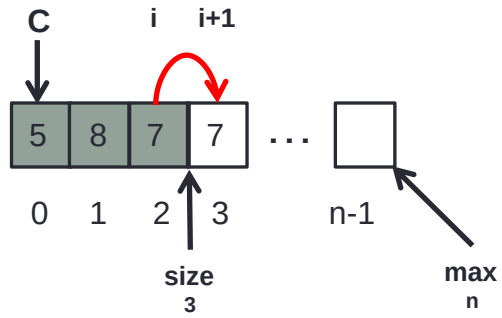
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

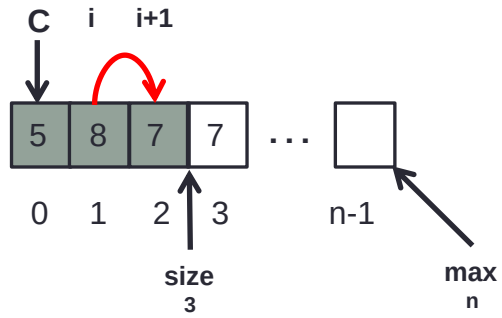
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

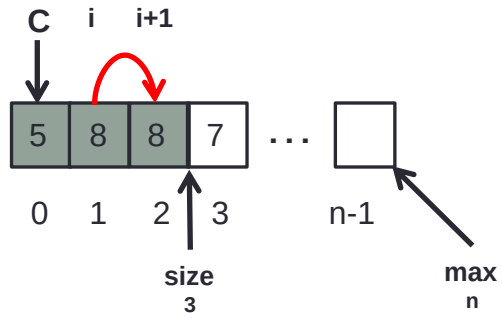
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

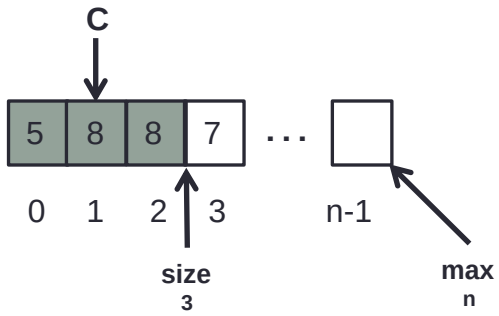
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

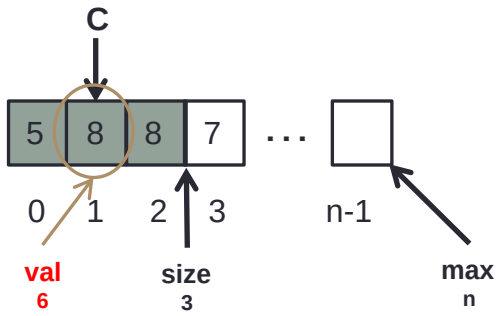
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

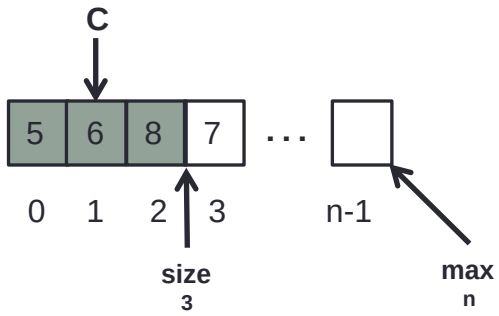
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

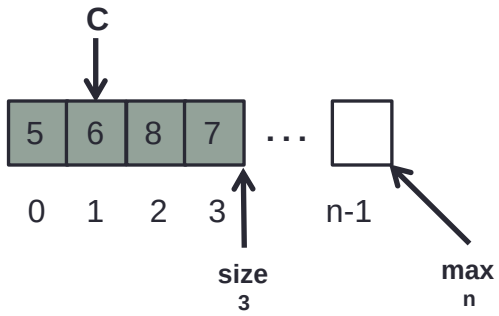
```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



Example #3

ADT List (Array): Implementation

```
public T retrieve () {
      return nodes[current];
}

public void update (T val) {
      nodes[current] = val;
}

public void insert (T val) {
      for (int i = size-1; i > current; --i) {
            nodes[i+1] = nodes[i];
      }
      current++;
      nodes[current] = val;
      size++;
}
```



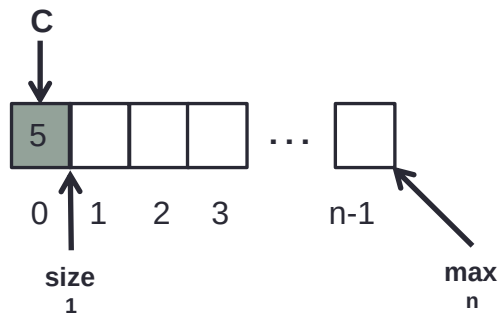
Example #3

ADT List (Array): Implementation

```
  public void remove () {
        for (int i = current + 1; i < size; i++) {
              nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
              current = -1;
        else if (current == size)
              current = 0;
  }
}
```

ADT List (Array): Implementation

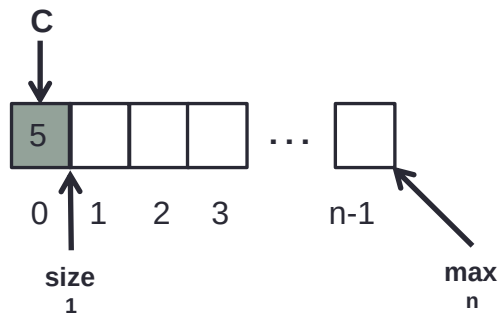
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #1

ADT List (Array): Implementation

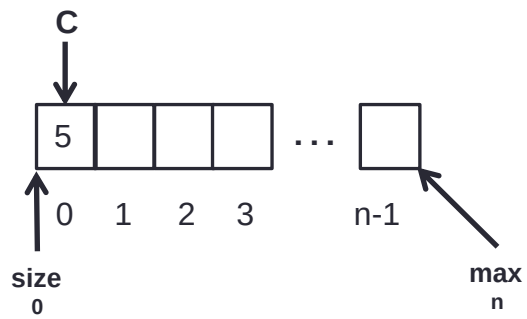
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #1

ADT List (Array): Implementation

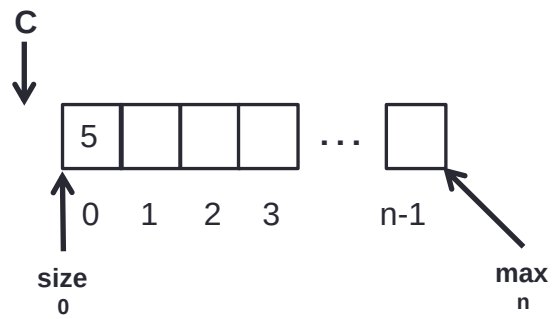
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #1

ADT List (Array): Implementation

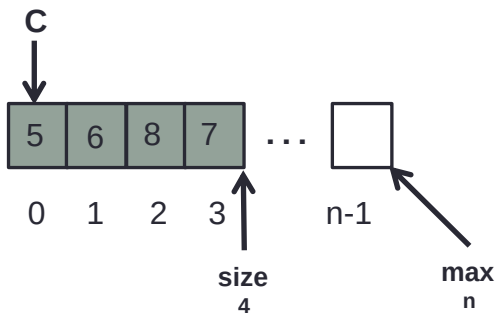
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #1

ADT List (Array): Implementation

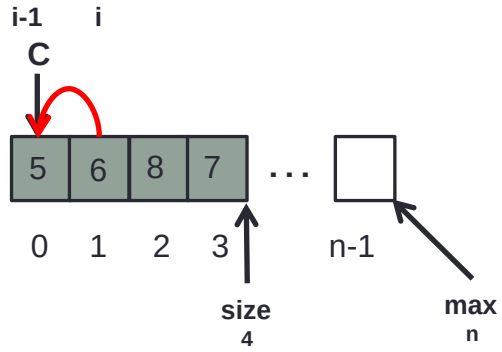
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

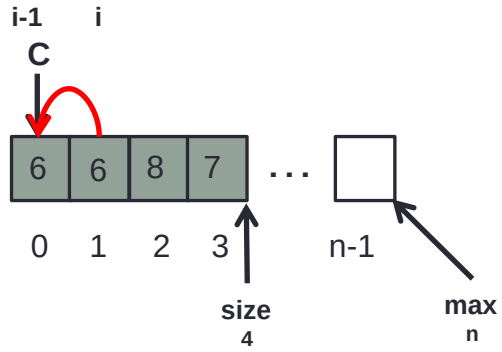
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

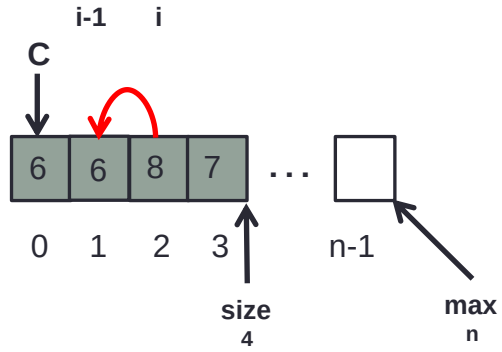
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

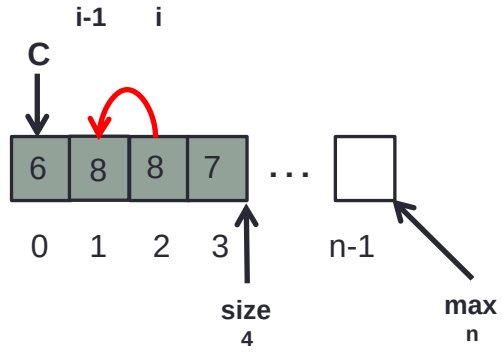
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

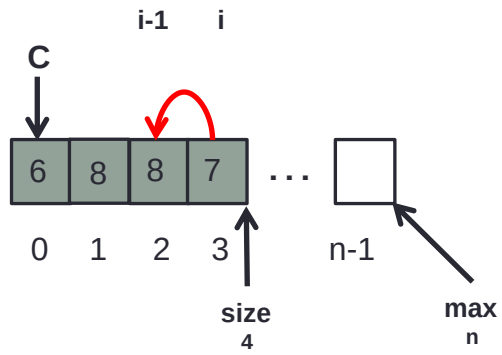
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

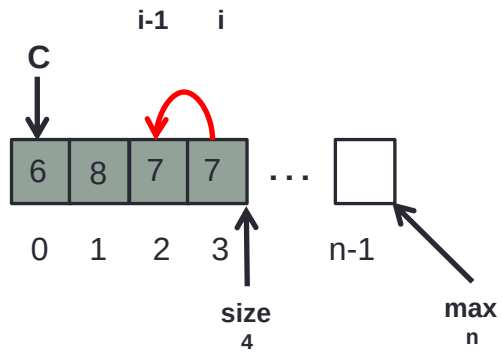
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

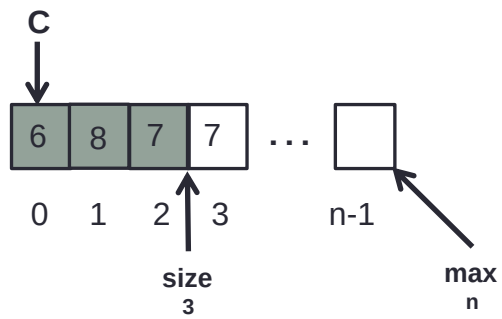
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

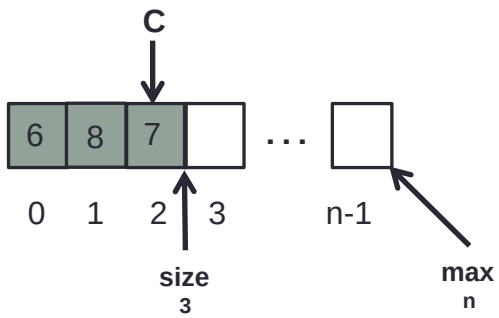
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #2

ADT List (Array): Implementation

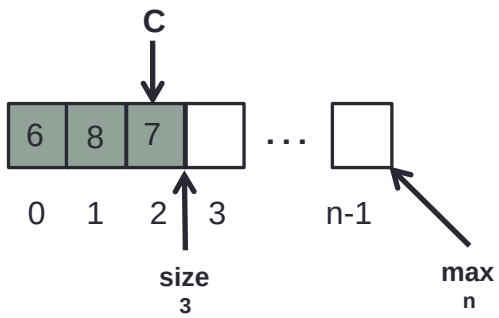
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #3

ADT List (Array): Implementation

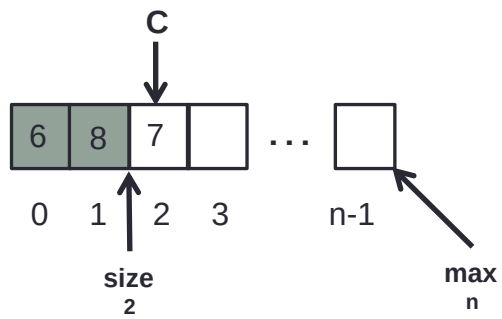
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #3

ADT List (Array): Implementation

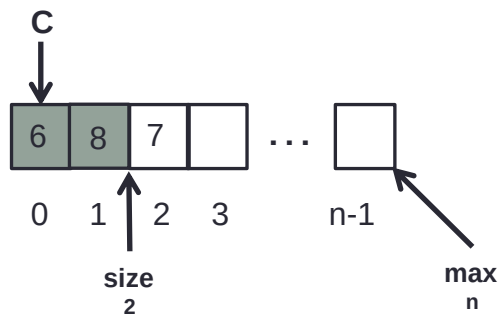
```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #3

ADT List (Array): Implementation

```
public void remove () {
        for (int i = current + 1; i < size; i++) {
                nodes[i-1] = nodes[i];
        }
        size--;
        if (size == 0)
                current = -1;
        else if (current == size)
                current = 0;
  }
}
```



Example #3

ADT List

- How to use the ADT List?
- The implementation of ADT is available to you as a Java class ready for use.

Example: You are required to implement a static method to get the length of a list.

Using ADT List

```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```

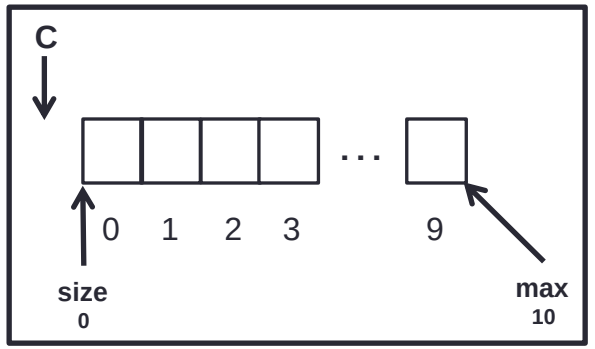
Using ADT List

```
public static <T> int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

A static method to find the length of a list.
Note: it has been implemented using the methods of ADT List

Using ADT List

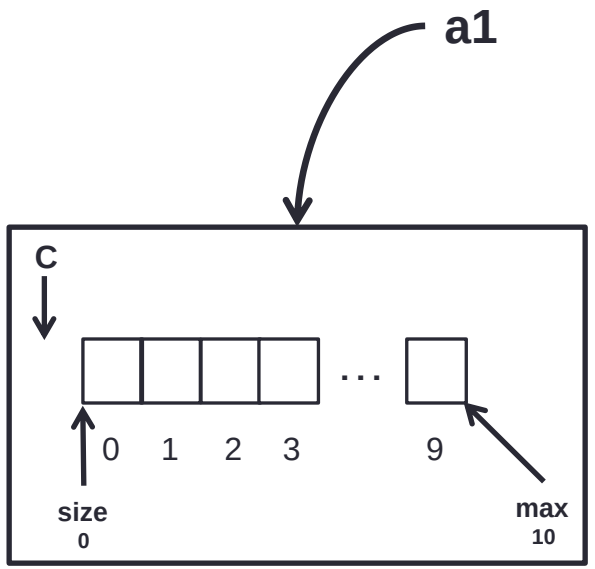
```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```



Output:

Using ADT List

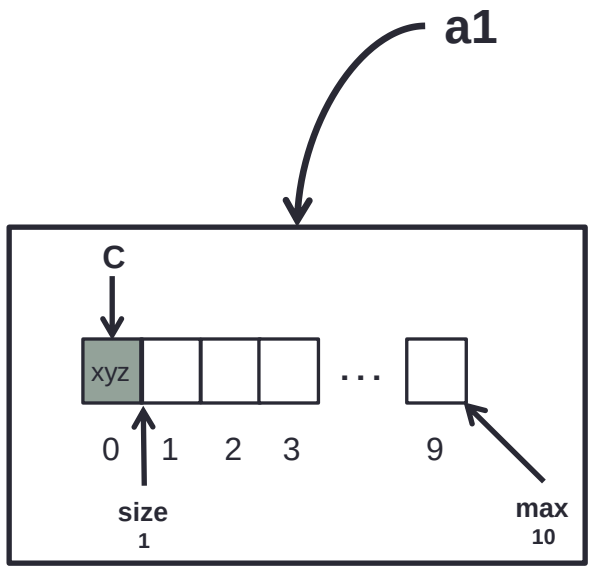
```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```



Output:

Using ADT List

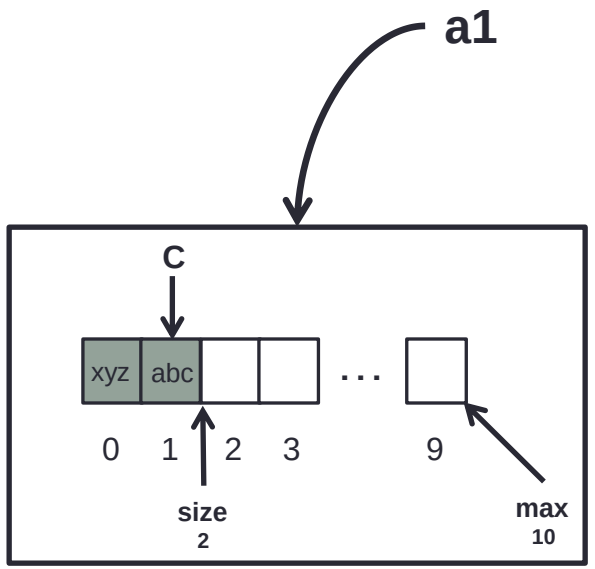
```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```



Output:

Using ADT List

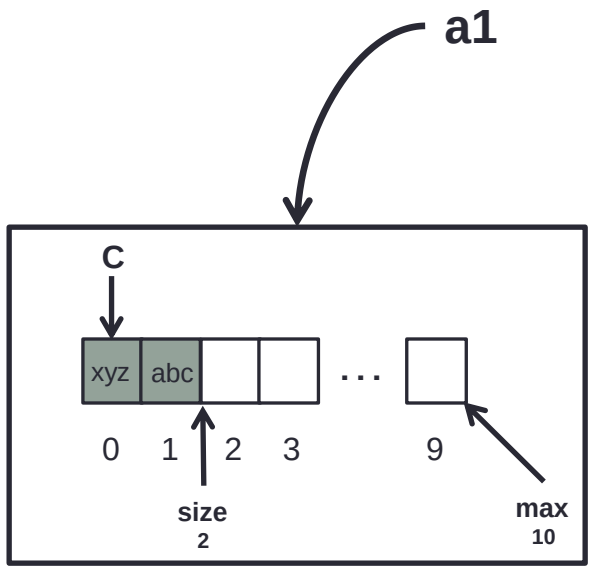
```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```



Output:

Using ADT List

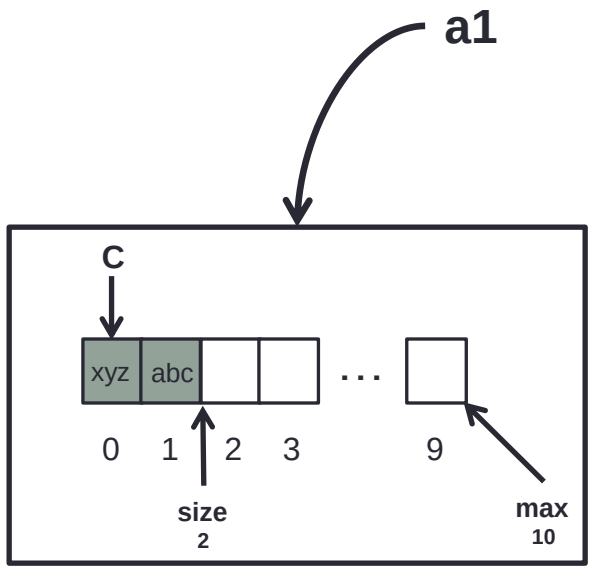
```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```



Output:

Using ADT List

```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```

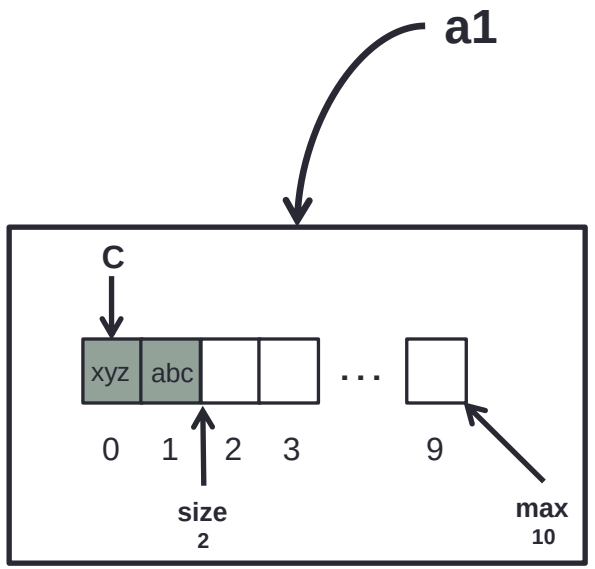


Output:

xyz
false

Using ADT List

```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```

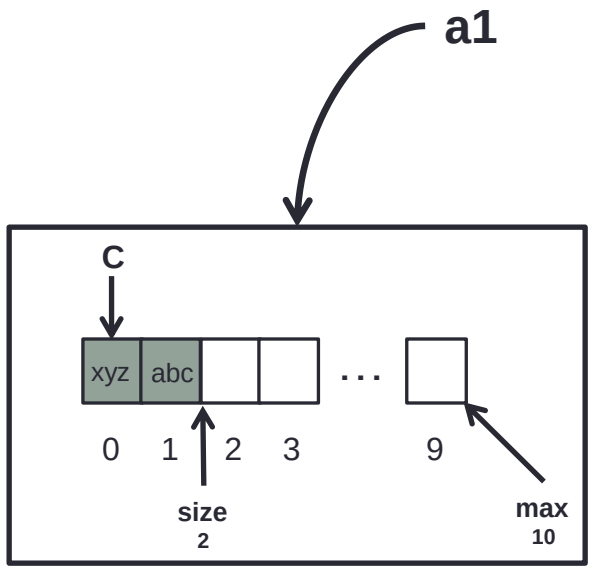


Output:

xyz
false

Using ADT List

```
public static <T> int length(ArrayList<T> l) {
        int count = 0;

        if (!l.empty()) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

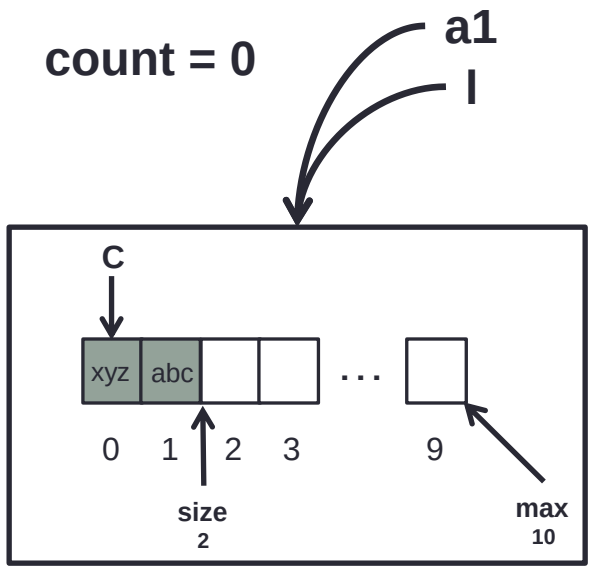


Output:

xyz
false

Using ADT List

```
public static <T>  int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

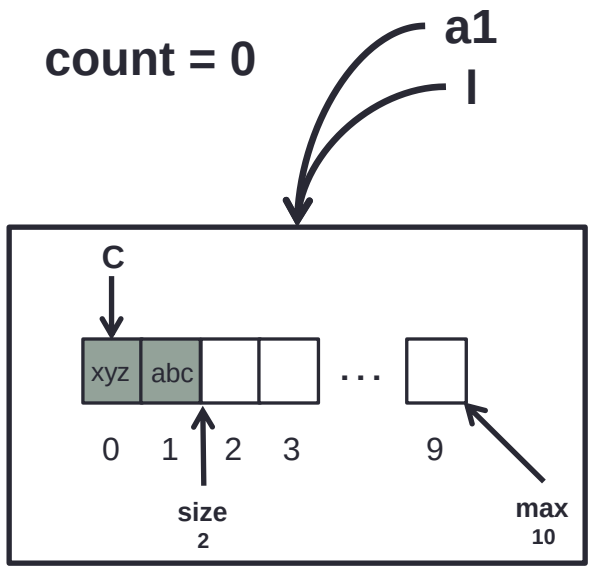


Output:

xyz
false

Using ADT List

```
public static <T> int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

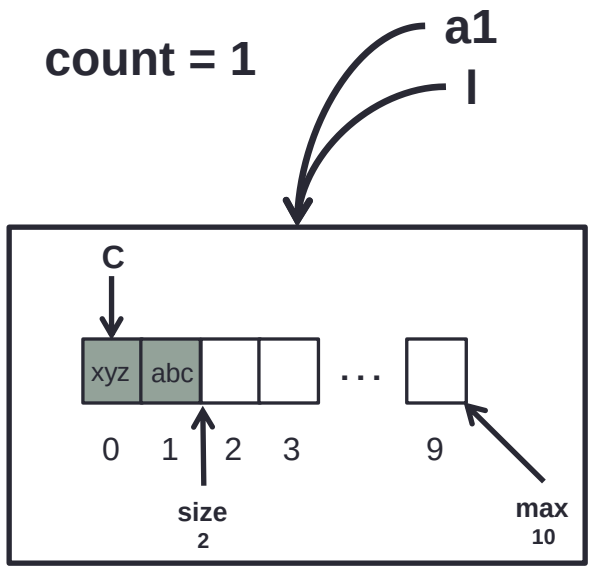


Output:

xyz
false

Using ADT List

```
public static <T> int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

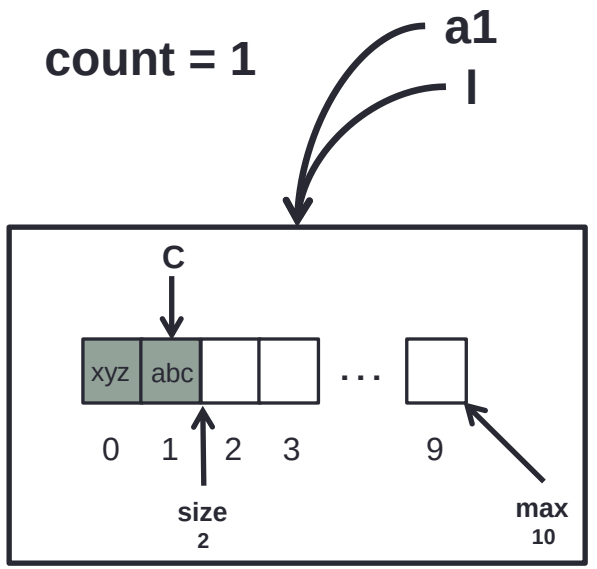


Output:

xyz
false

Using ADT List

```
public static<T>  int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

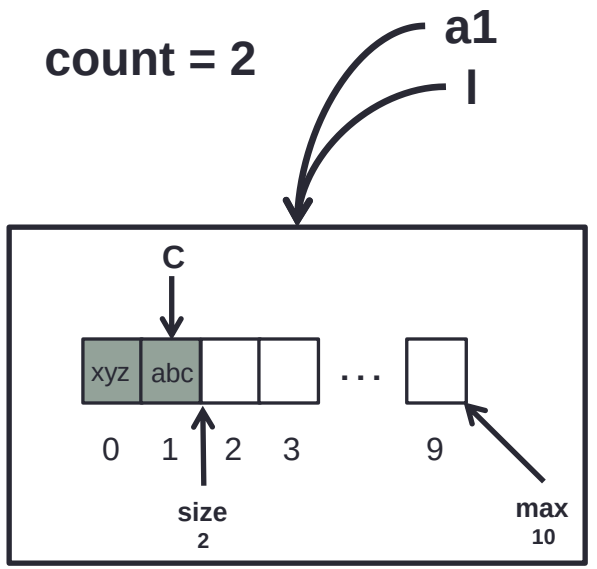


Output:

xyz
false

Using ADT List

```
public static <T>  int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

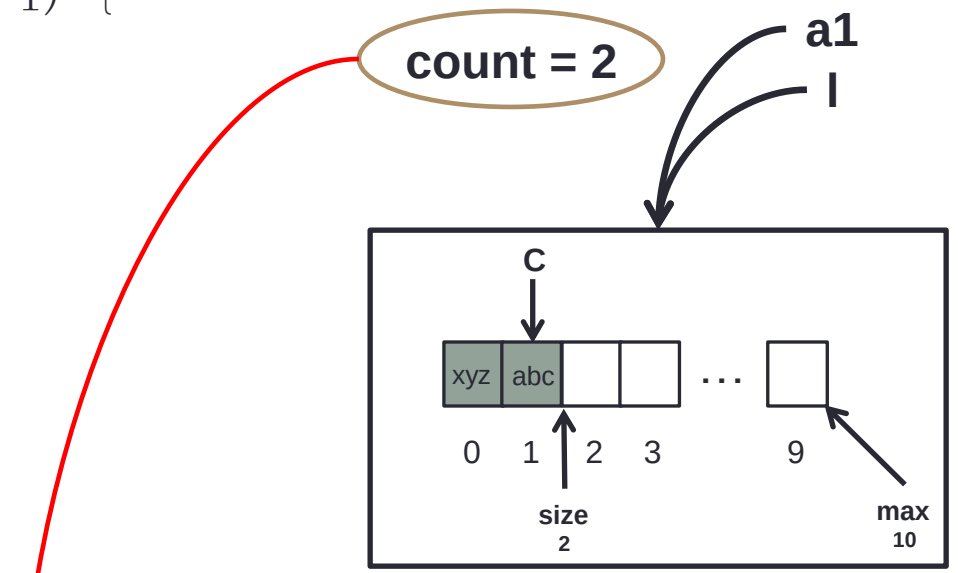


Output:

xyz
false

Using ADT List

```
public static <T> int length(ArrayList<T> l) {
        int count = 0;

        if (l.empty() == false) {
                l.findFirst();
                while (l.last() == false) {
                        count++;
                        l.findNext();
                }
                count++;
        }

        return count;
  }
}
```

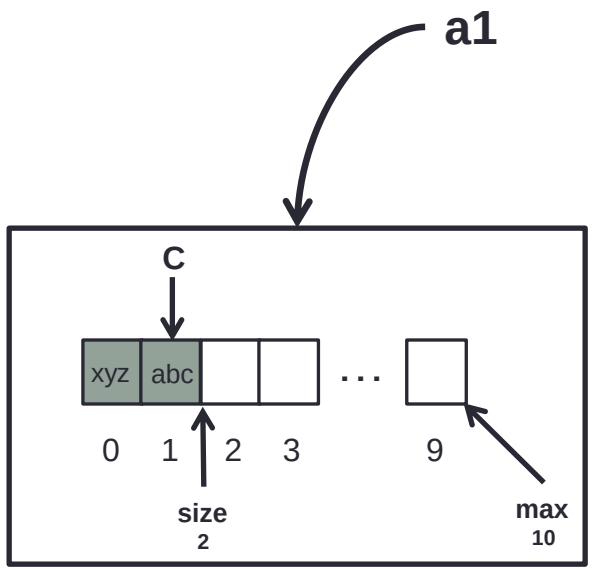


Output:

xyz
false

Using ADT List

```
public class TestArrayList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```



Output:

xyz
false
2
Hello, World

ADT List

What are the changes that need to be made to use List (Linked List implementation) instead of List (Array List implementation)?

Using ADT List

```
public class TestLinkedList {
  public static void main ( String[] args ) {
        ArrayList<String> al = new ArrayList<String>(10);
        LinkedList<String> al = new LinkedList<String>();
        String s1= "xyz", s2 = "abc";

        al.insert(s1);
        al.insert(s2);
        al.findFirst();

        System.out.println(al.retrieve());
        System.out.println(al.full());

        System.out.println(length(al));
        System.out.println("Hello, World");
    }
```

Only this line!

ADT List

You are required to implement a static method to search for an element e in a list L, and if e is present make current pointer point to e. Use operations of ADT List.

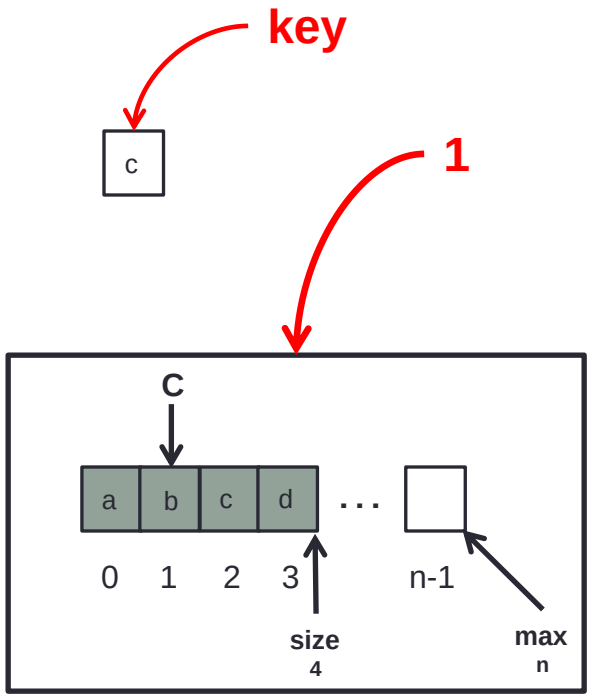
Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
                if(l.retrieve().equals(key))
                        return true;
                l.findNext();
        }
        if(l.retrieve().equals(key))
                return true;
  }

  return false;
}
...
```

Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

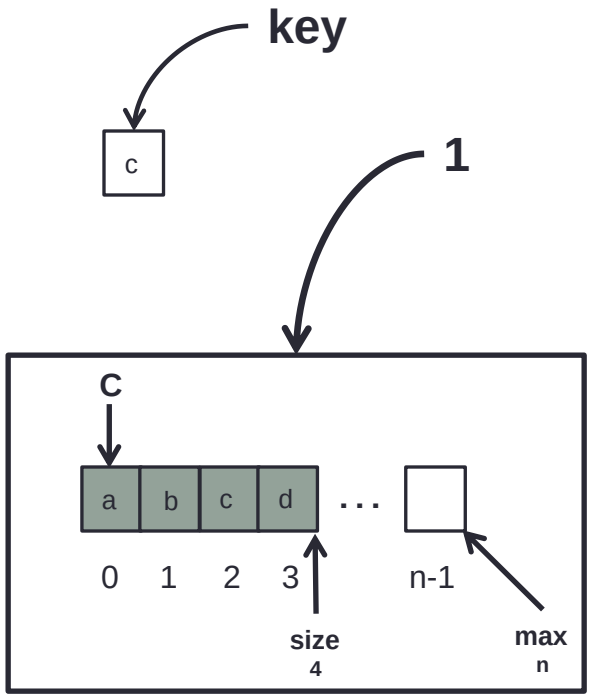
Example #1



Search Item (Static Method)

```
...
public static<T>  boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

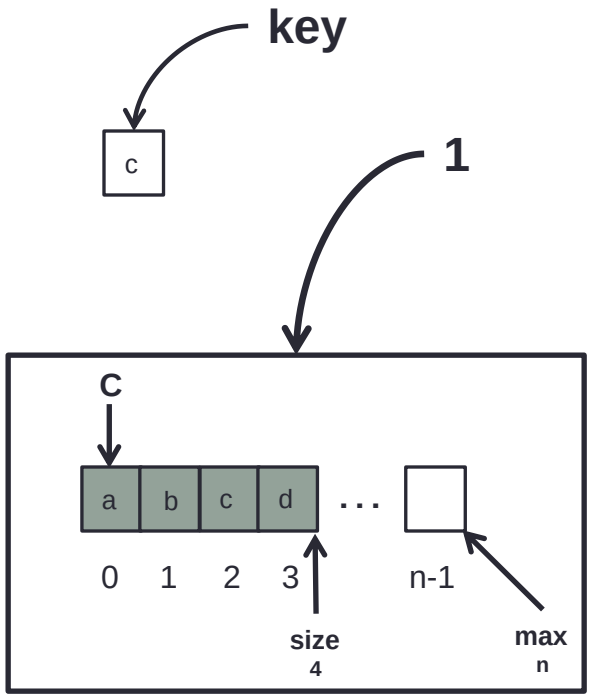
Example #1



Search Item (Static Method)

```
...
public static <T>  boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
                if(l.retrieve().equals(key))
                        return true;
                l.findNext();
        }
        if(l.retrieve().equals(key))
                return true;
  }

  return false;
}
...
```

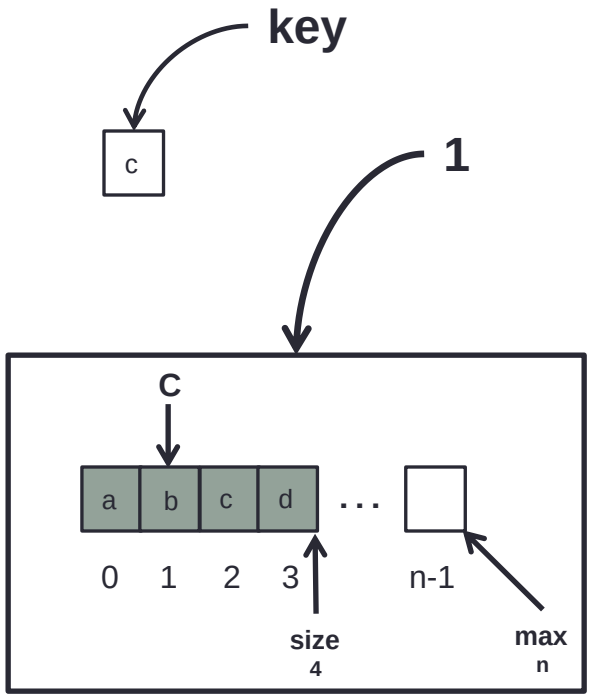
Example #1



Search Item (Static Method)

```
...
public static <T>  boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
                if(l.retrieve().equals(key))
                        return true;
                l.findNext();
        }
        if(l.retrieve().equals(key))
                return true;
  }

  return false;
}
...
```

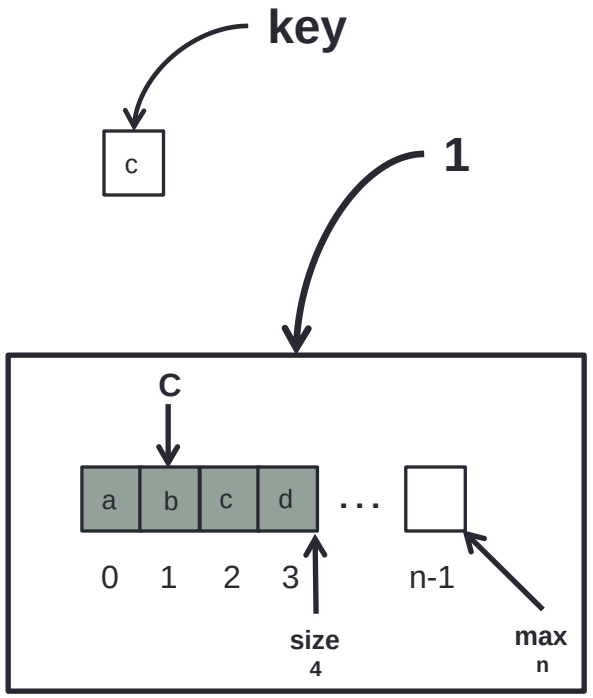
Example #1



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

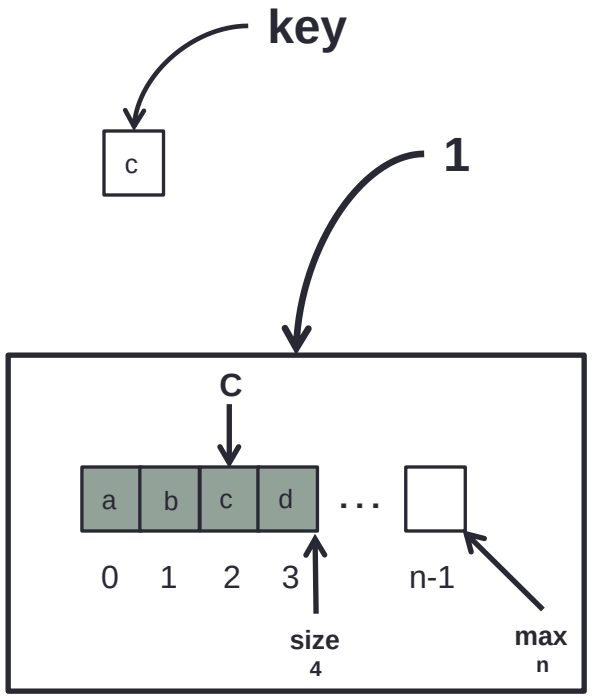
Example #1



Search Item (Static Method)

```
...
public static <T>  boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

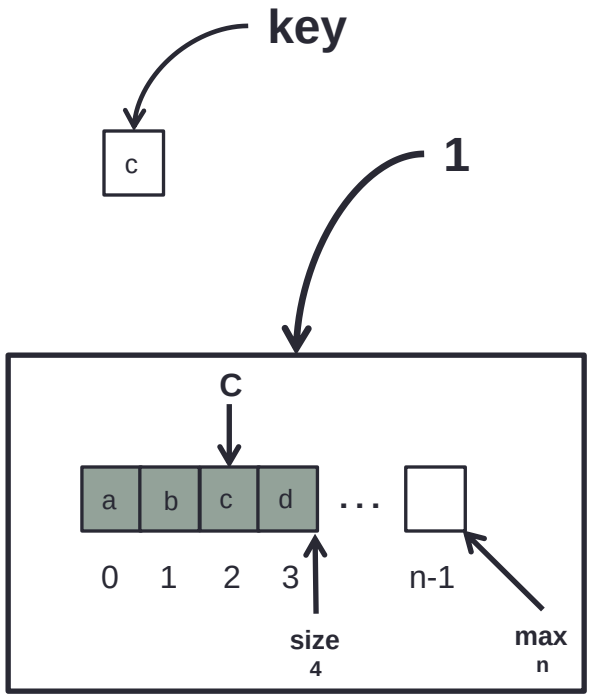
Example #1



Search Item (Static Method)

```
...
public static <T>  boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
                if(l.retrieve().equals(key))
                        return true;
                l.findNext();
        }
        if(l.retrieve().equals(key))
                return true;
  }

  return false;
}
...
```

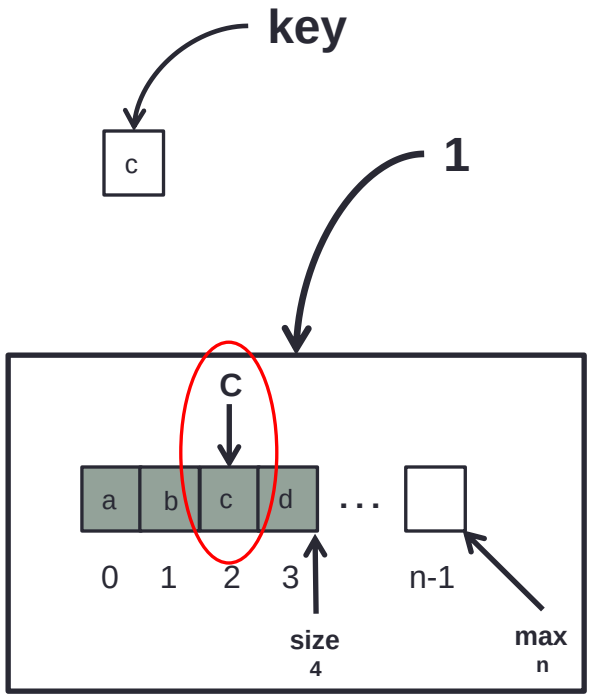
Example #1



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

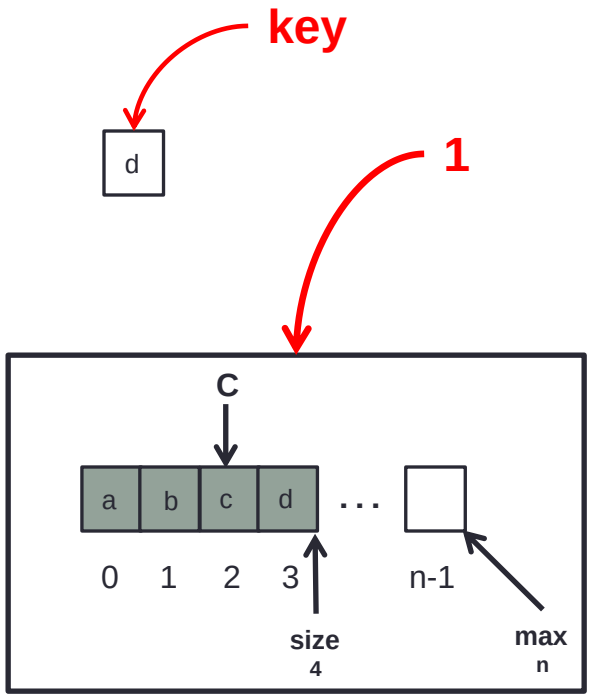
Example #1



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

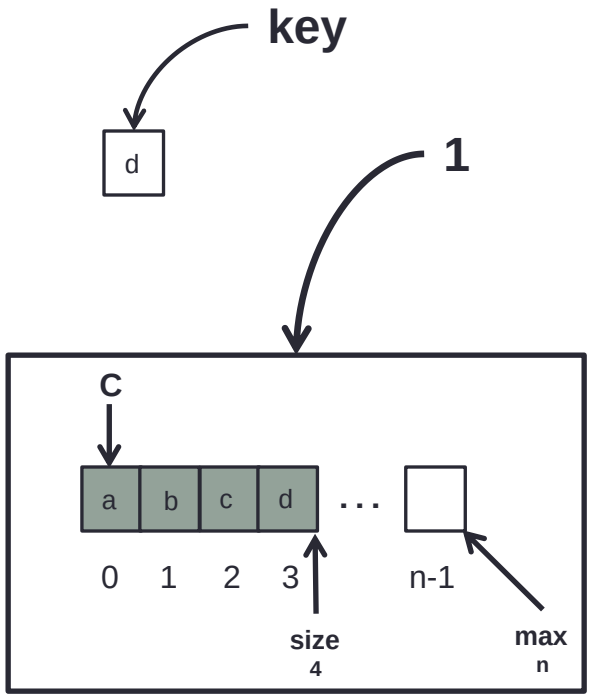
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

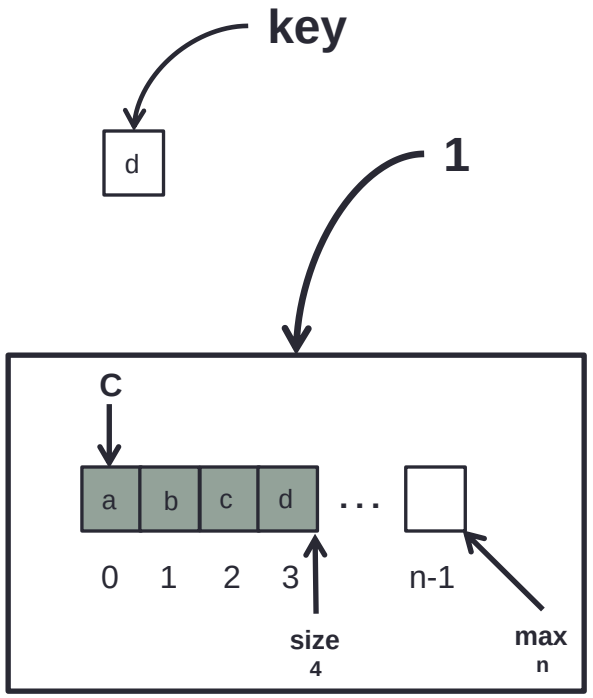
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

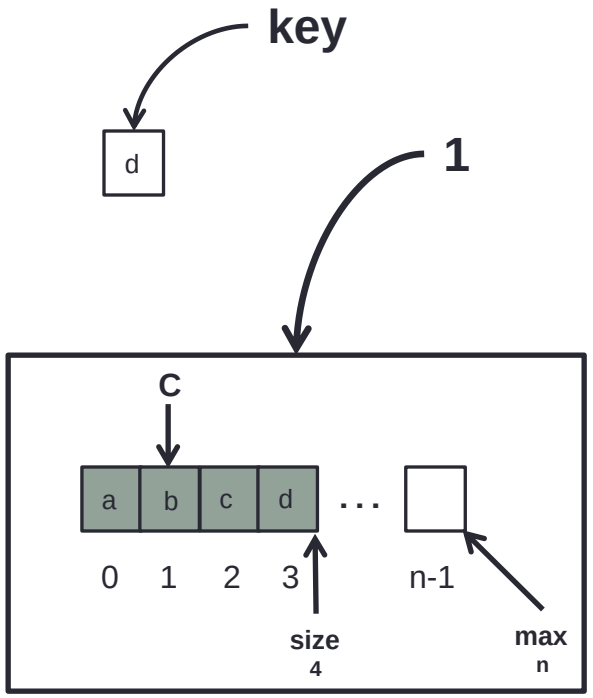
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

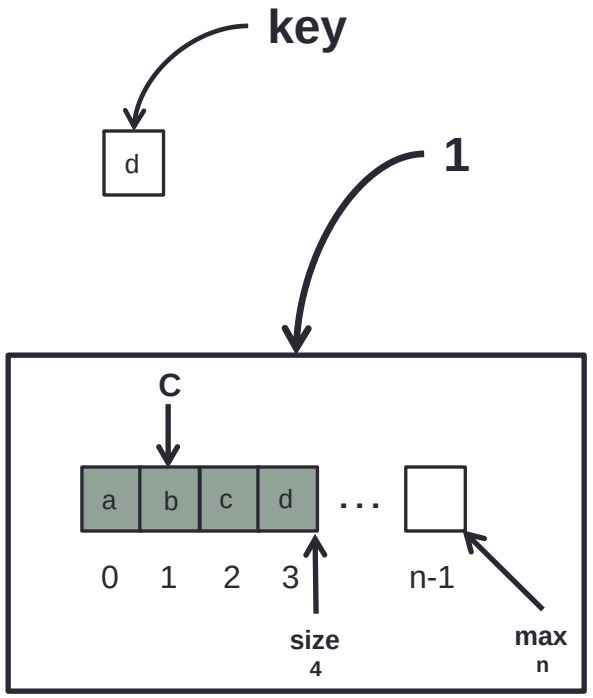
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

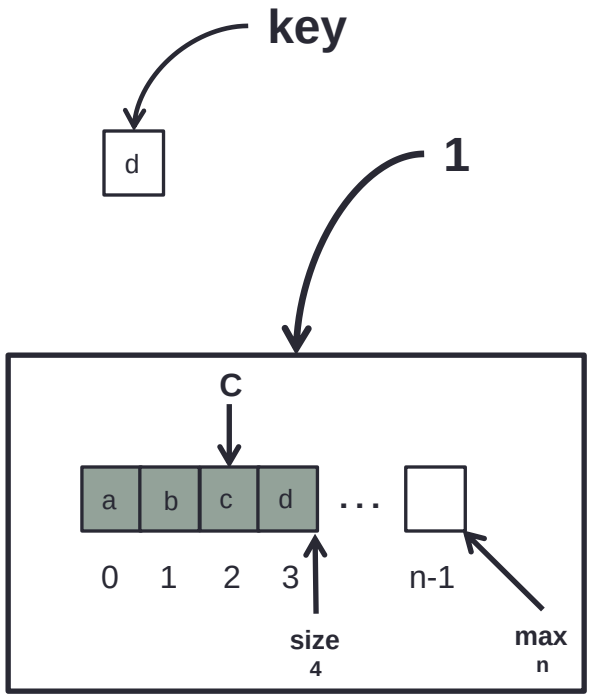
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

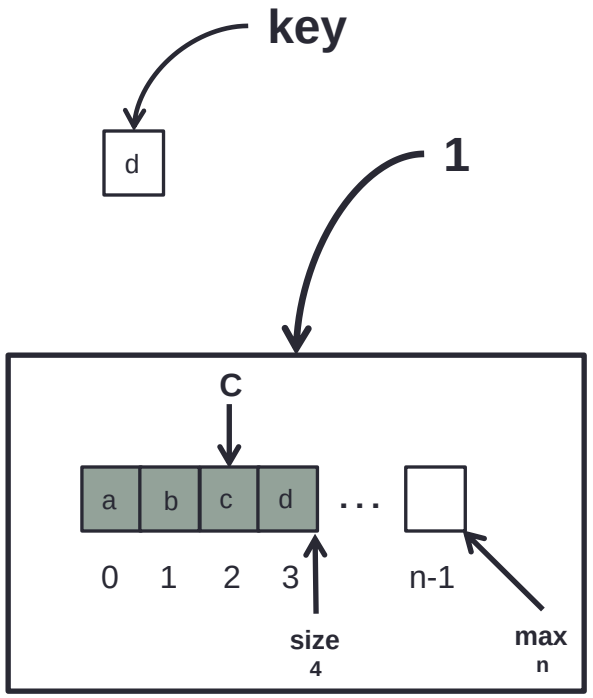
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

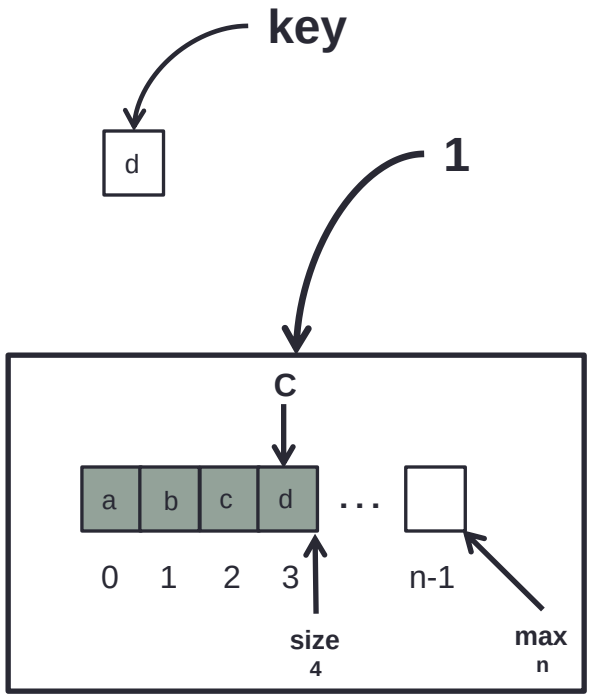
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

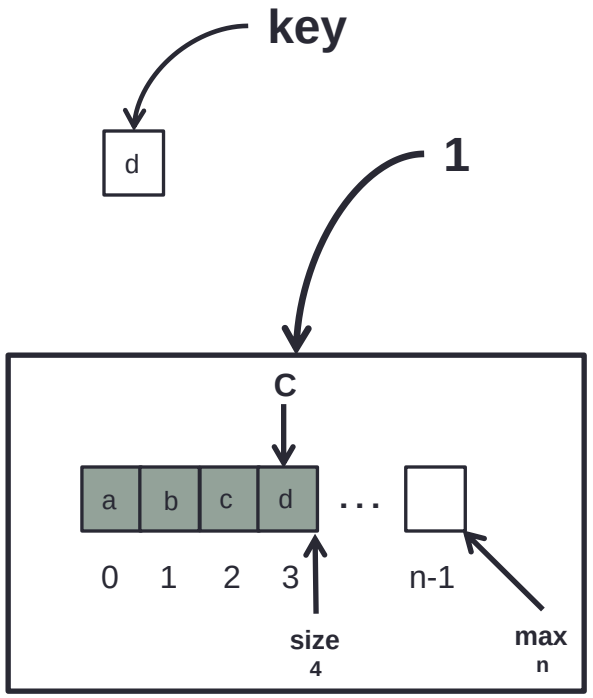
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

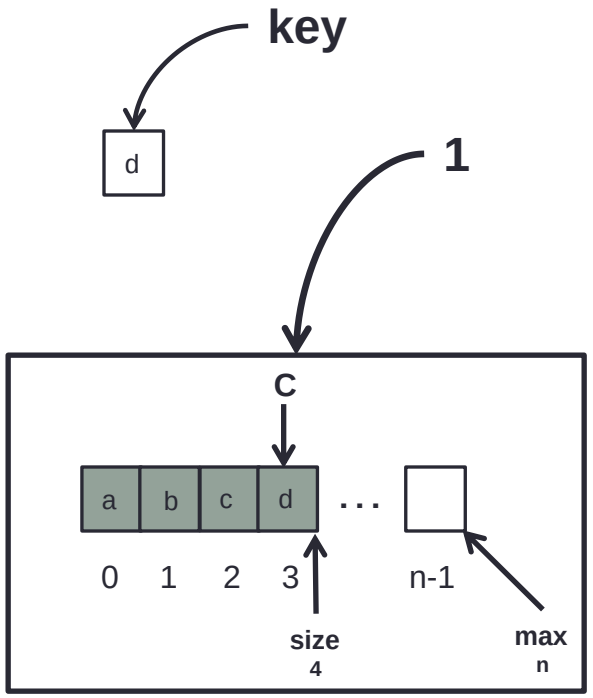
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

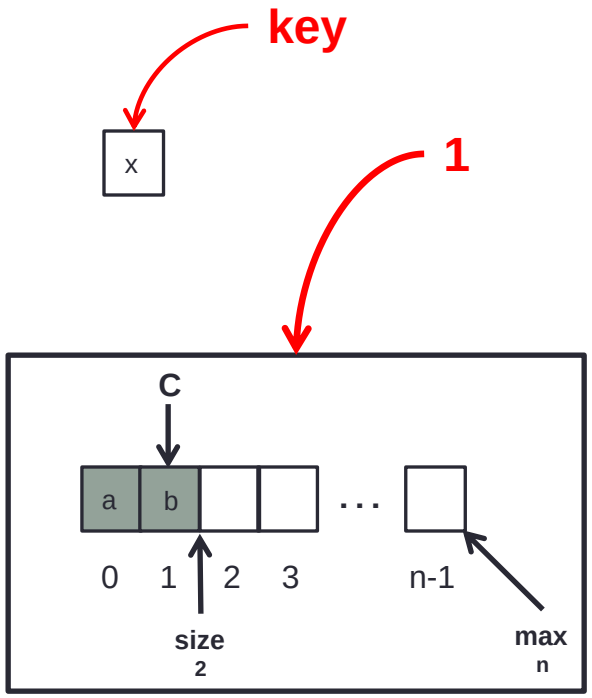
Example #2



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

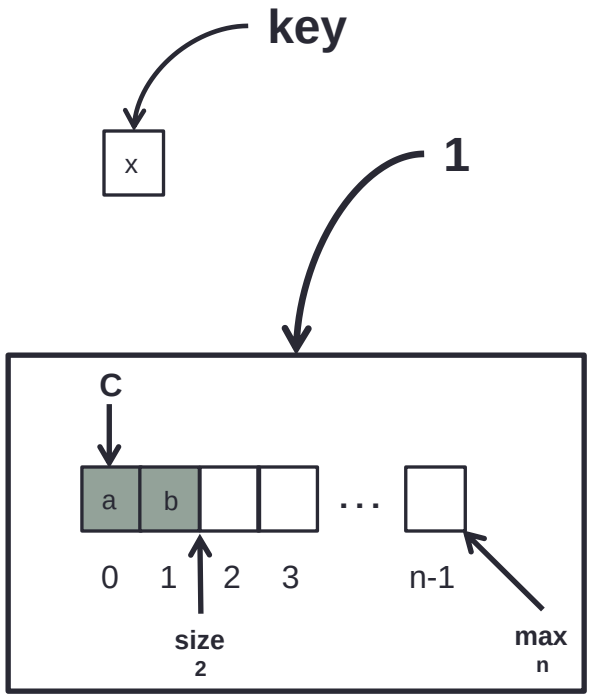
Example #3



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

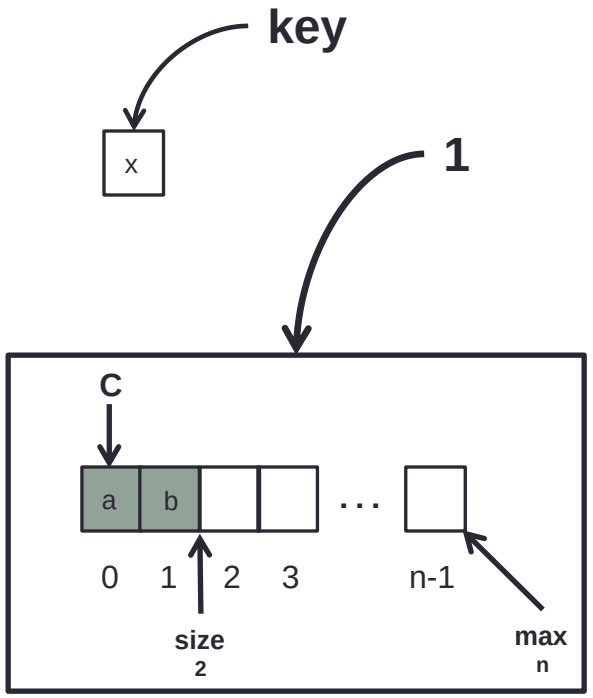
Example #3



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

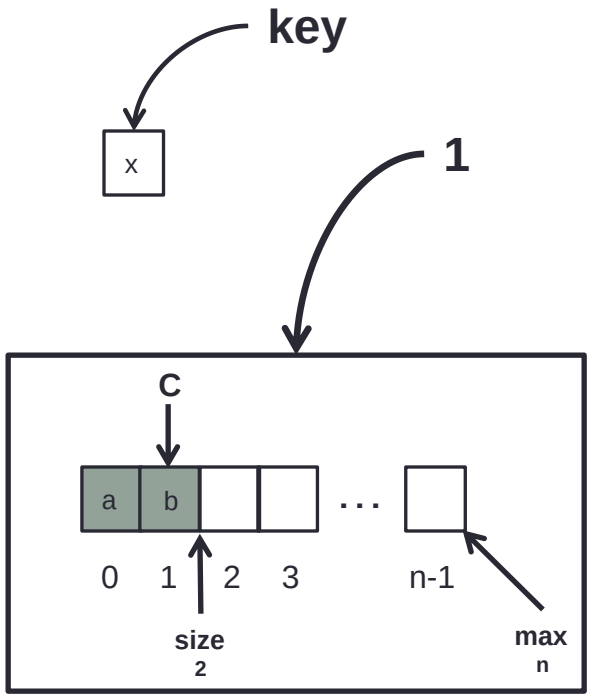
Example #3



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

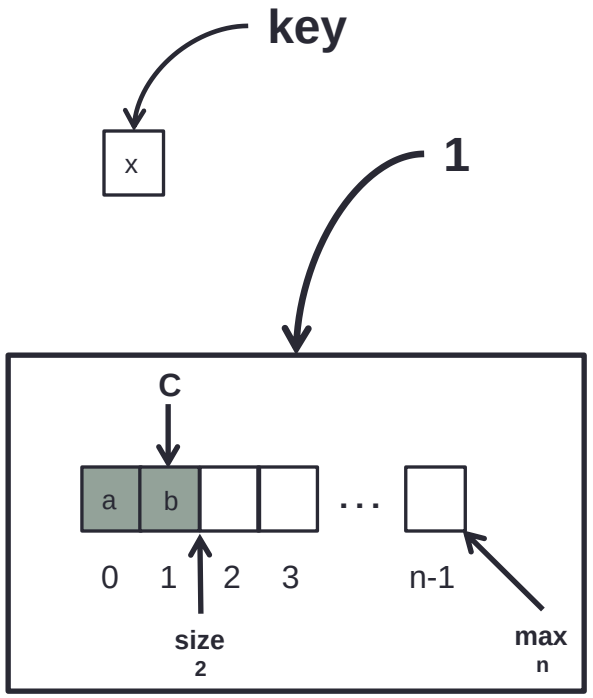
Example #3



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

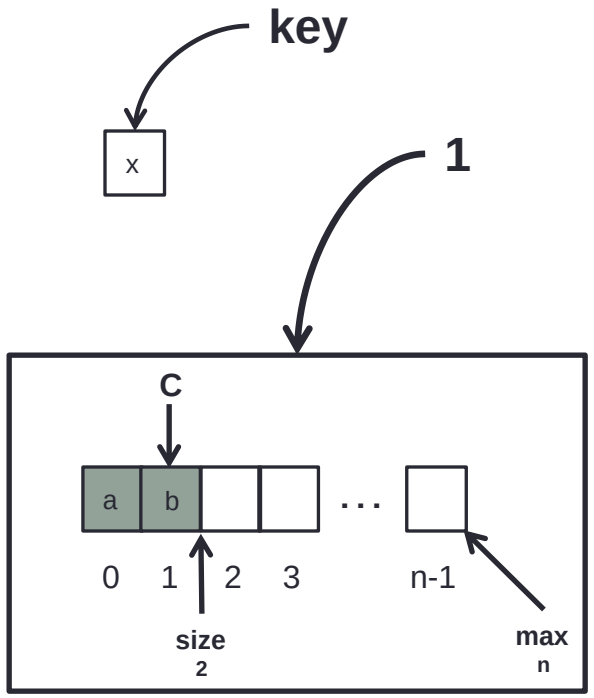
Example #3



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;

  }

  return false;
}
...
```

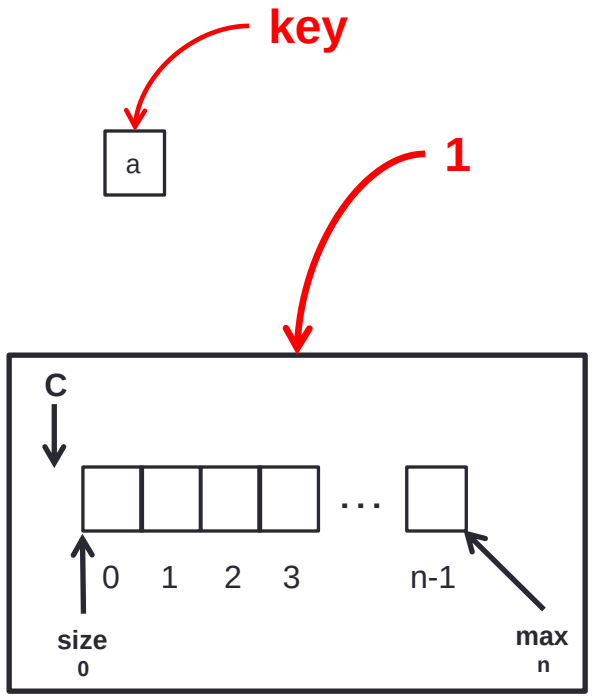
Example #3



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.Findfirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;
  }

  return false;
}
...
```

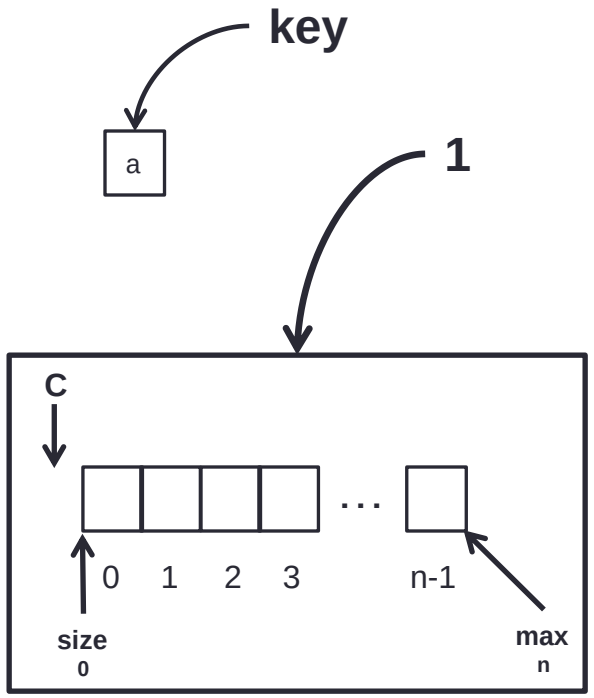
Example #4



Search Item (Static Method)

```
...
public static <T> boolean find(ArrayList<T> l, T key) {
  if(l.empty() == false) {
        l.findFirst();
        while(l.last() == false) {
              if(l.retrieve().equals(key))
                    return true;
              l.findNext();
        }
        if(l.retrieve().equals(key))
              return true;

  }

  return false;
}
...
```

Example #4



ADT List

You are required to implement the same search method but this time as a member method of the ADT List (Linked List implementation).

Search Item (Member Method) #1

```
...
public boolean find(T key) {
  Node<T> tmp = current;
  current = head;
  while(current != null) {
        if(current.data.equals(key))
                return true;
        current = current.next;
  }
  current = tmp;
  return false;
}
...
```

Search Item (Member Method) #2

```
...
public boolean find(T key) {
  Node<T> tmp = head;
  while(tmp != null) {
        if(tmp.data.equals(key)) {
                current = tmp;
                return true;
        }
        tmp = tmp.next;
  }
  return false;
}
...
```

Search Item (Member Method) #3

```
...
public boolean find(T key) {
  if(empty() == false) {
        findFirst();
        while(last() == false) {
                if(retrieve().equals(key))
                        return true;
                findNext();
        }
        if(retrieve().equals(key))
                return true;
  }
  return false;
}
...
```

Comparison: Linked & Array Based Lists

Comparison on the basis:

- *worst case* time complexity operations
  - Linked List: insert– **O(1)**; remove – **O(n)**.
  - Array List: insert – **O(n)**; remove – **O(n)**.
  - All other operations have time complexity **O(1)**.
- Best case time complexities?

Comparison: Linked & Array Based Lists

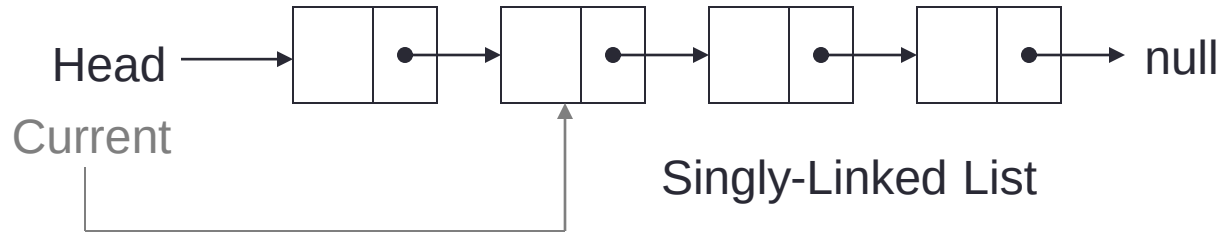
- Linked List.
  - Advantages: No need to know the size in advance. Fast "Insert" – O(1).
  - Disadvantage: a pointer at each node (more memory needed). For traversal, pointer hoping is required.
- Array Based List.
  - Advantages: No pointers to be stored (less memory). For traversal, no pointer hoping is required (array faster in traversal).
  - Disadvantage: list size must be known in advance. Slow "Insert" – O(n).

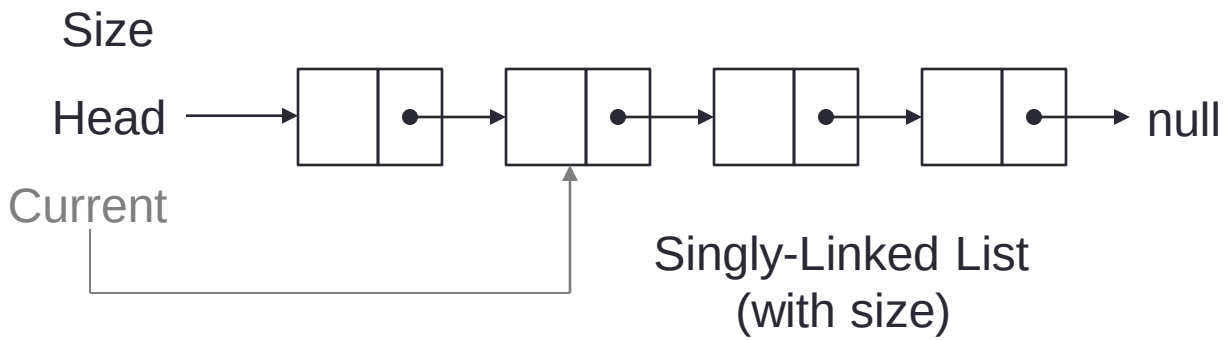
List: Other Implementations

- Singly-Linked List.
  - Design Variations: (i) Count of elements may be kept i.e. *size*. **Why?** (ii) pointer to tail may be kept i.e. *last*. **Why?**
- Doubly-Linked List.
  - each node has two pointers: next node and previous node.
  - Advantages: it is efficient to go from a node to its previous node or move back-to-front in the list
  - operations insert – O(1);
  - remove – O(1).

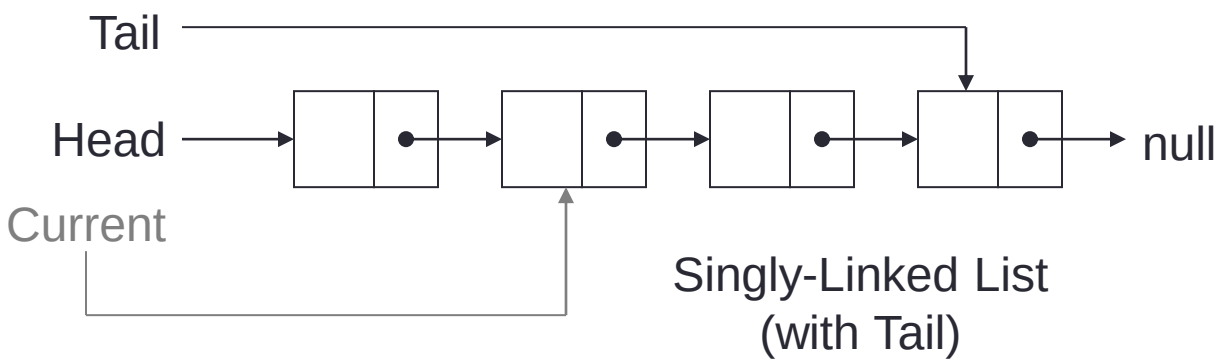
(We will cover this topic in detail later)

List: Singly Linked

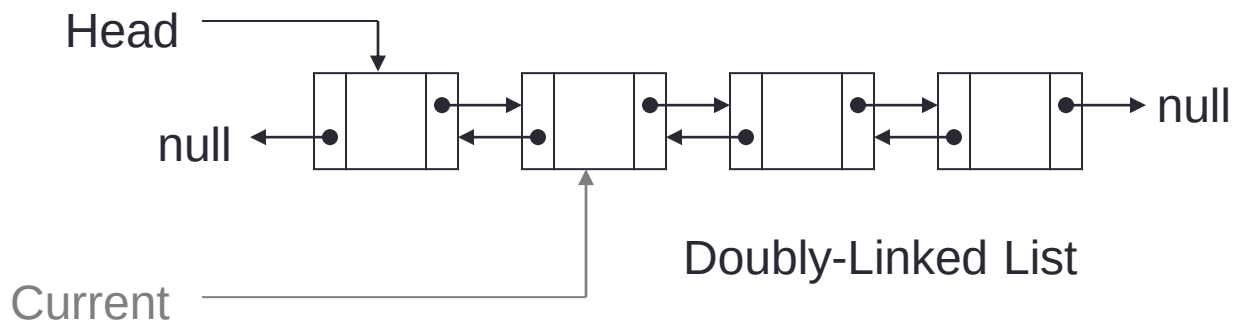




List: Singly Linked with Tail



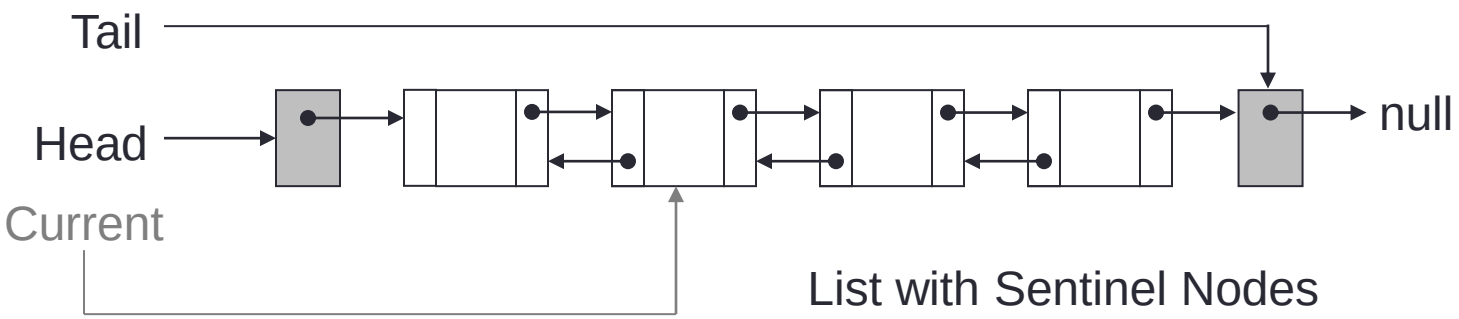
List: Doubly-Linked



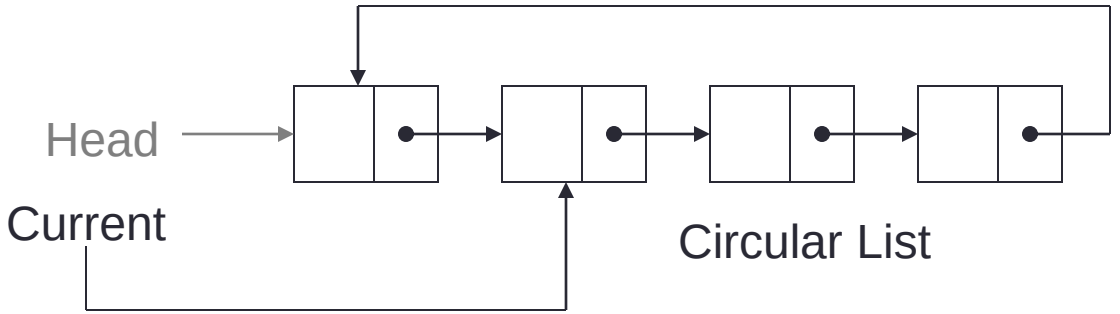
List: Other Implementations

- List with Sentinel Nodes.
  - Has special header & trailer nodes that do not store data
  - All nodes that store data have previous & next nodes – so no special cases for insert & remove.
  - Advantage– simplifies code
- Circular List.
  - tail pointer made to point to the head → tail has next node. Advantage: simpler code.

List: with Sentinel Nodes



List: Circular List



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.

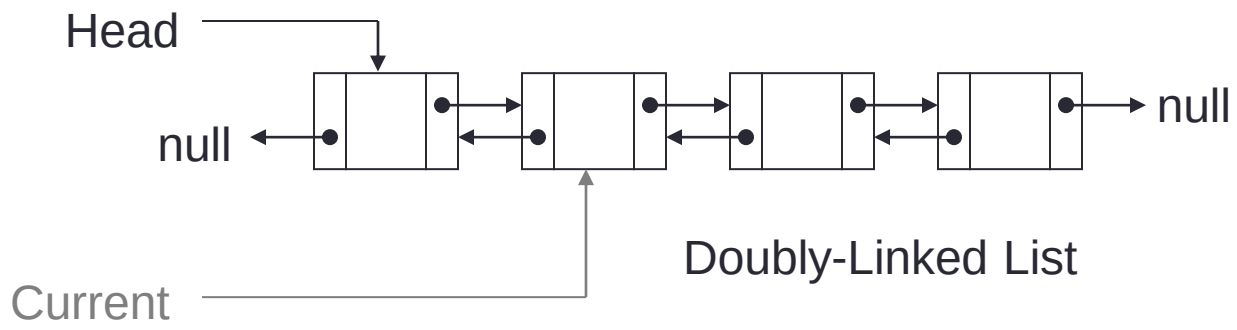
9. **Method f**irst (boolean flag).

   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

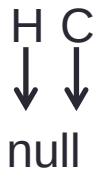
```
public class Node<T> {
  public T data;
  public Node<T> next;
  public Node<T> previous;

  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

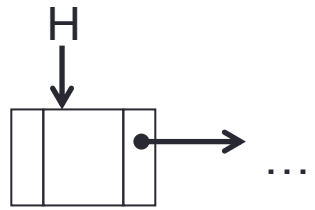
```
public class DoubleLinkedList<T> {
  private Node<T> head;
  private Node<T> current;

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

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

  public boolean last() {
          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;
  }
```

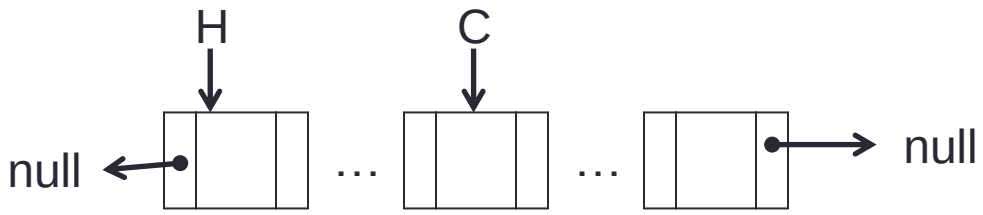
true



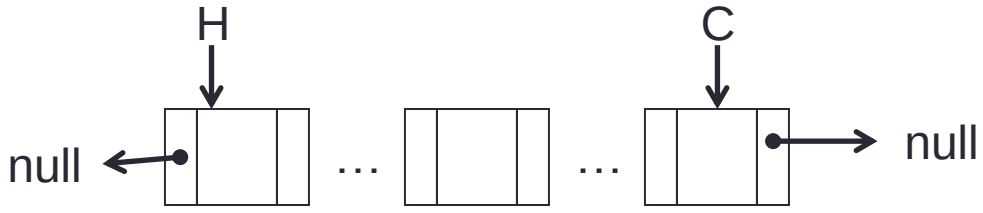
false

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;
  }
```



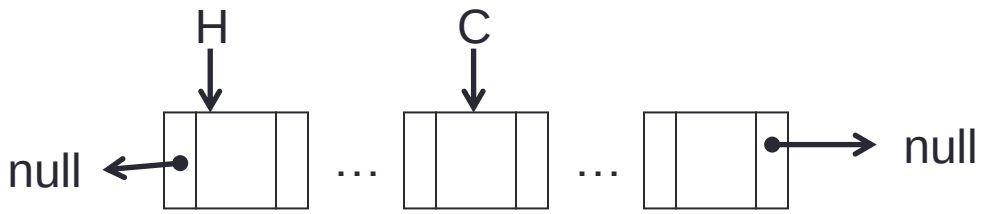
false



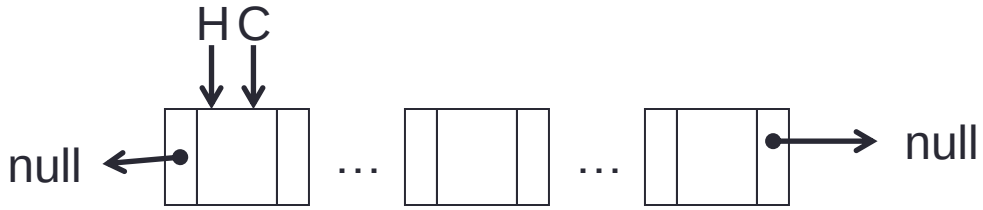
true

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;
  }
```



false



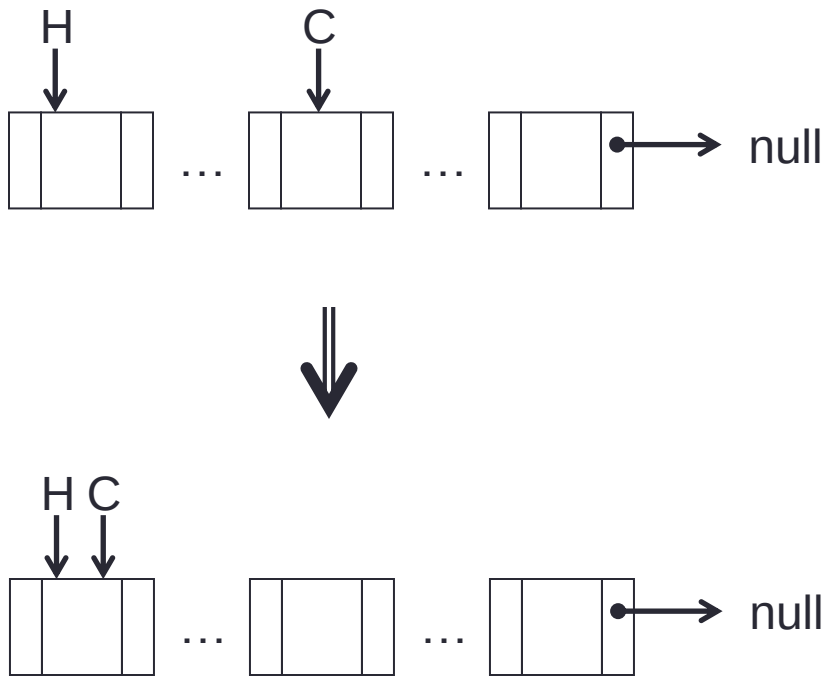
true

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;
}
```

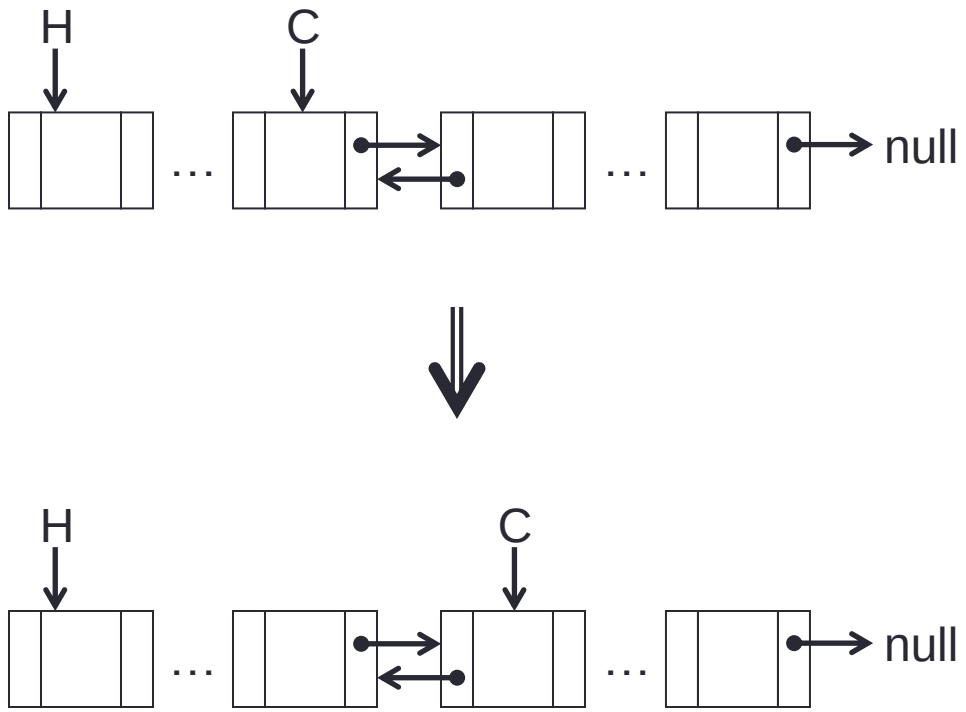
ADT List (Double-Linked List): Implementation

```
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;
}
```



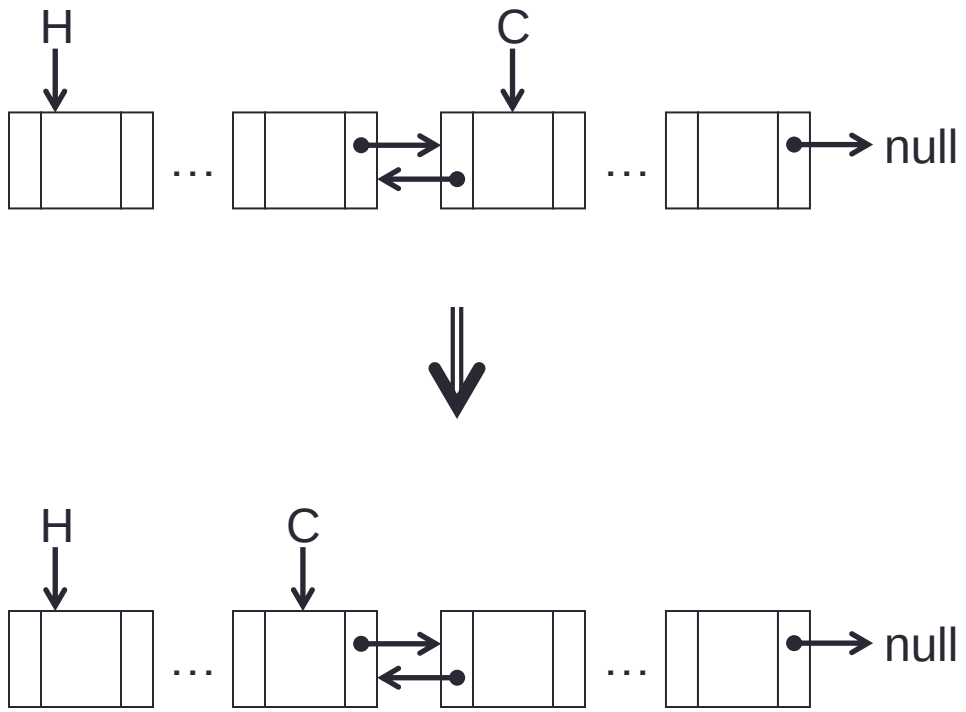
ADT List (Double-Linked List): Implementation

```
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;
}
```



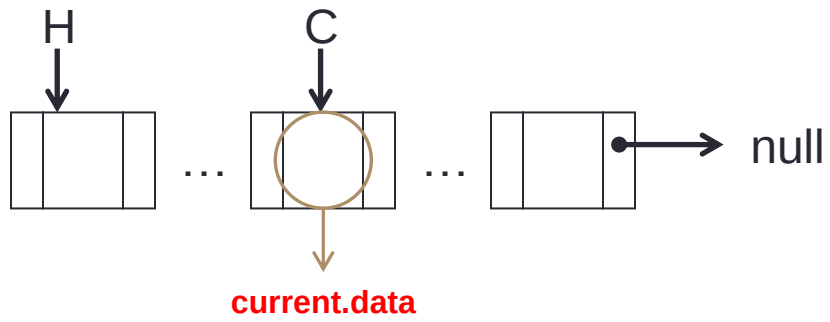
ADT List (Double-Linked List): Implementation

```
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;
}
```



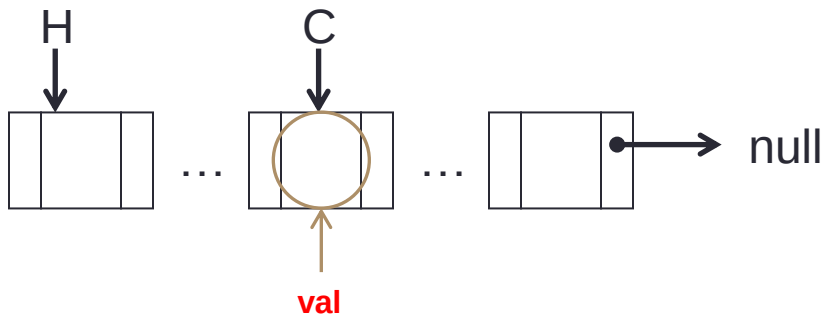
ADT List (Double-Linked List): Implementation

```
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;
}
```



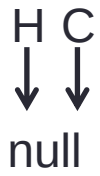
ADT List (Double-Linked List): Implementation

```
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

```
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;
        }
}
```

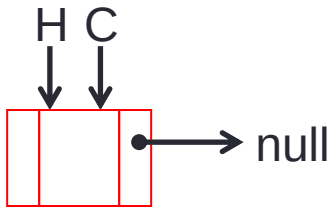
Example #1



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;
        }
}
```

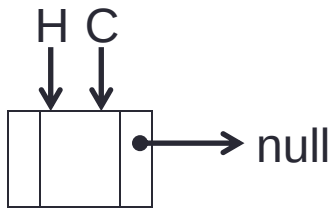
Example #1



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;
        }
}
```

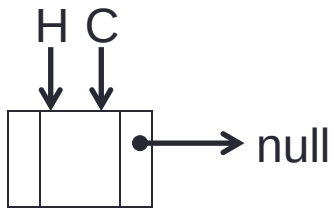
Example #2

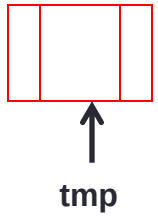


ADT List (Double-Linked List): Implementation

```
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;
        }
}
```

Example #2

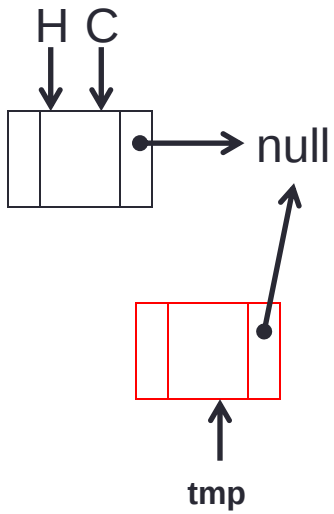




ADT List (Double-Linked List): Implementation

```
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;
        }
}
```

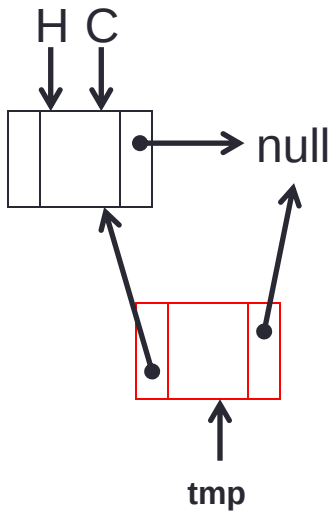
Example #2



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;
        }
}
```

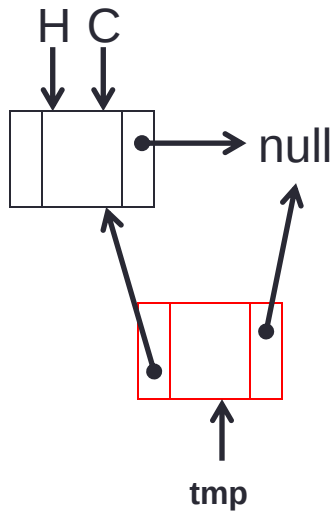
Example #2



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;
        }
}
```

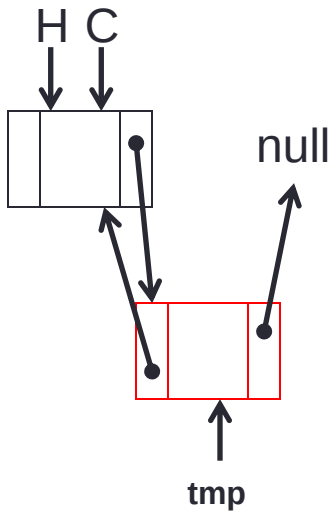
Example #2



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;
        }
}
```

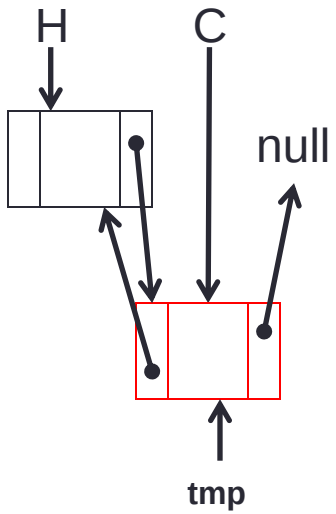
Example #2



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;
        }
}
```

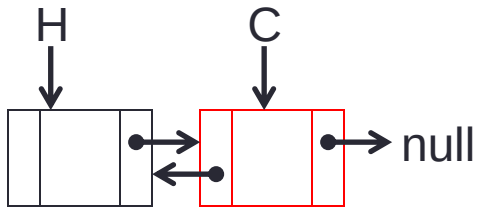
Example #2



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;
        }
}
```

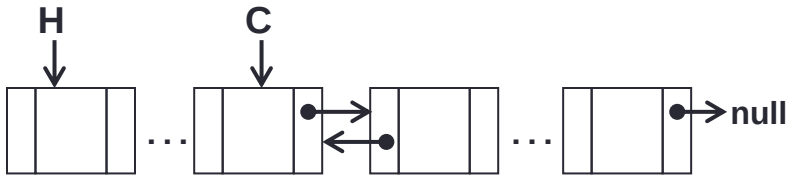
Example #2



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;
        }
}
```

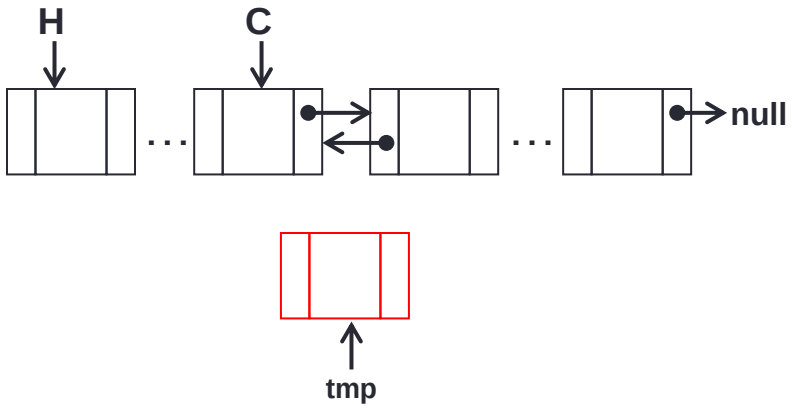
Example #3



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;
        }
}
```

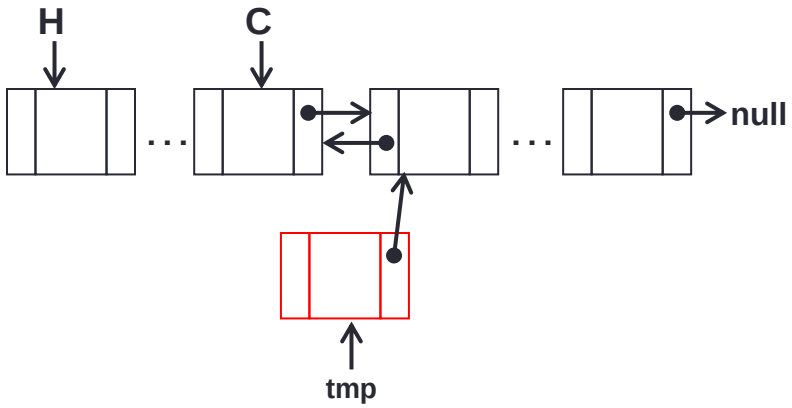
Example #3



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;
        }
}
```

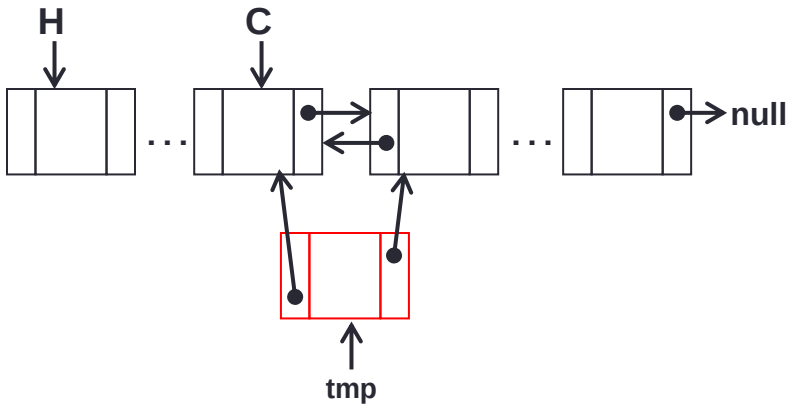
Example #3



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;
        }
}
```

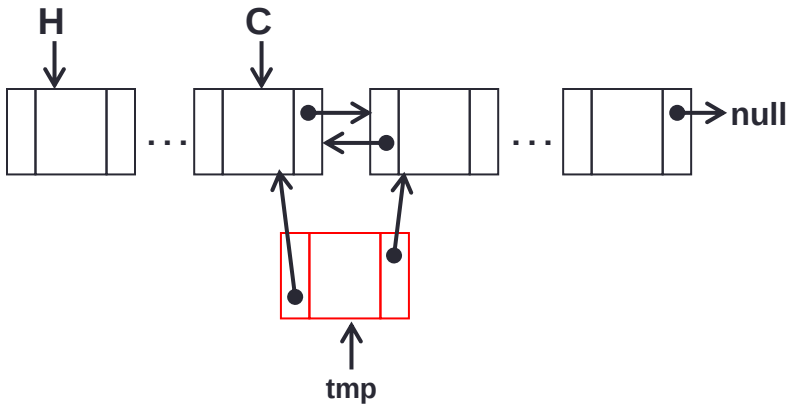
Example #3



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;
        }
}
```

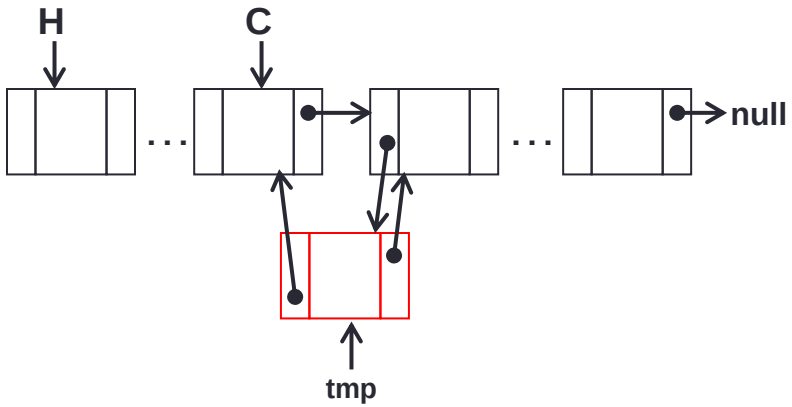
Example #3



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;
        }
}
```

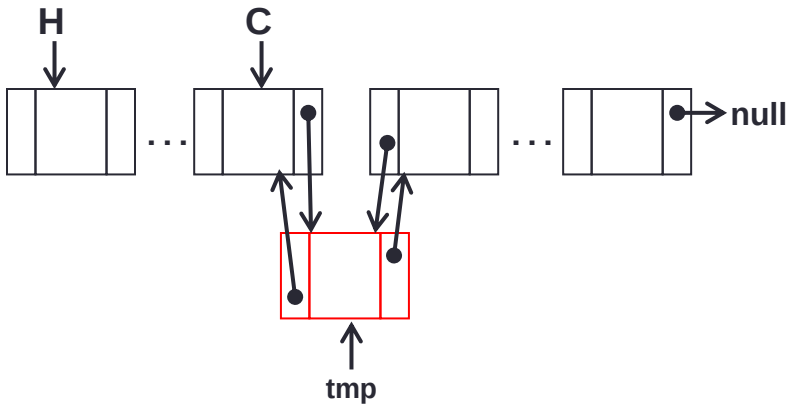
Example #3



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;
        }
}
```

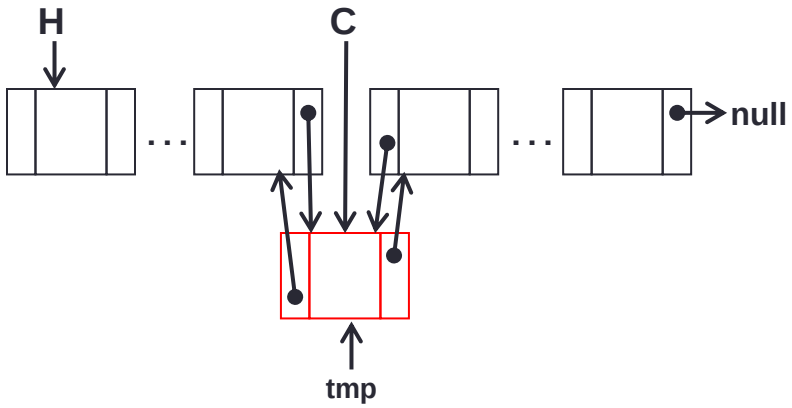
Example #3



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;
        }
}
```

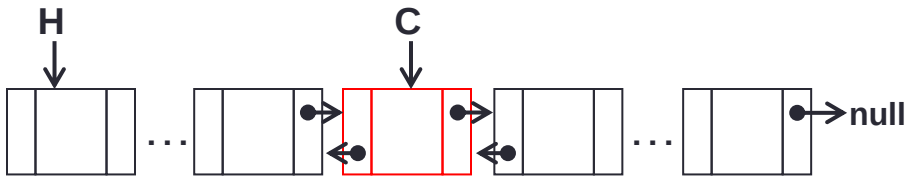
Example #3



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;
        }
}
```

Example #3



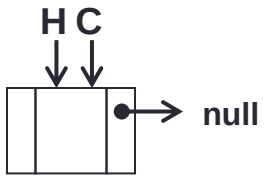
ADT List (Double-Linked List): Implementation

```
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

```
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;
  }
}
```

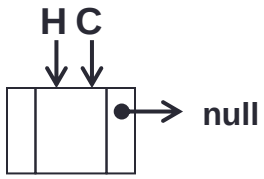
Example #1



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

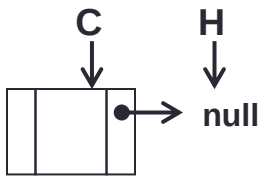
Example #1



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

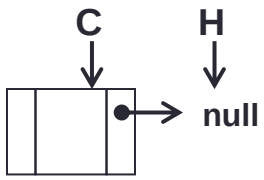
Example #1



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

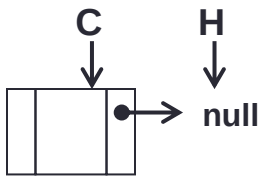
Example #1



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

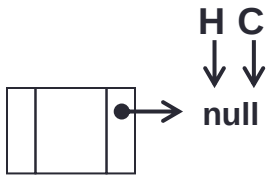
Example #1



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

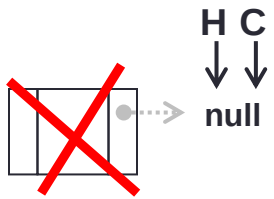
Example #1



ADT List (Double-Linked List): Implementation

```
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

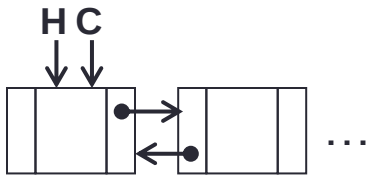
```
public void remove() {
        if(current == head) {
                head = head.next;
                if(head != null)
                   head.previous = null;
        }
        else {
                current.previous.next = current.next;
                if(current.next != null)
               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

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

Example #1

ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

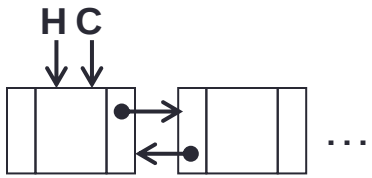
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

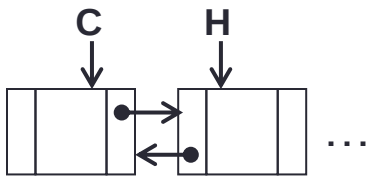
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

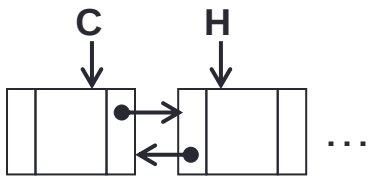
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

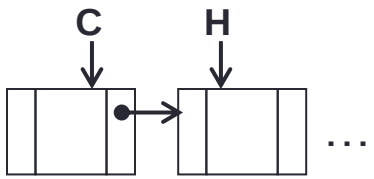
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

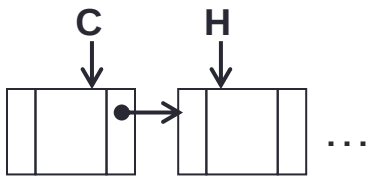
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

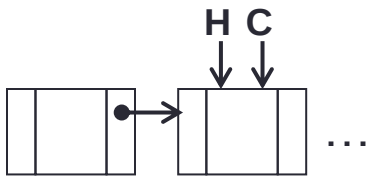
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

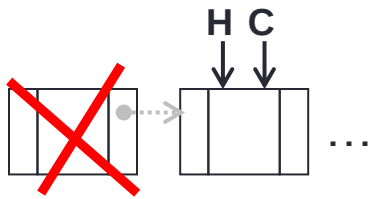
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

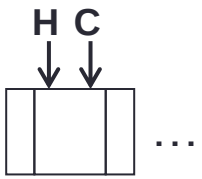
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

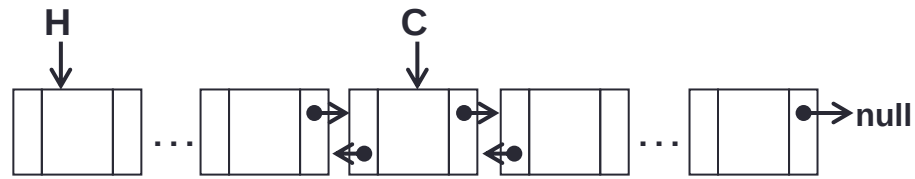
Example #2



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

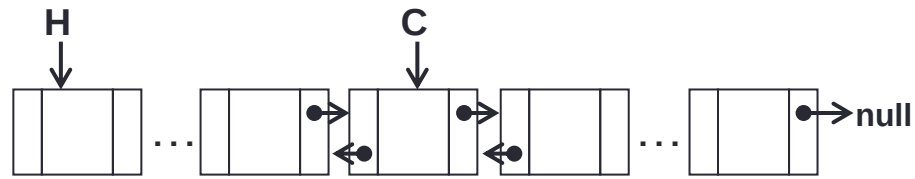
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

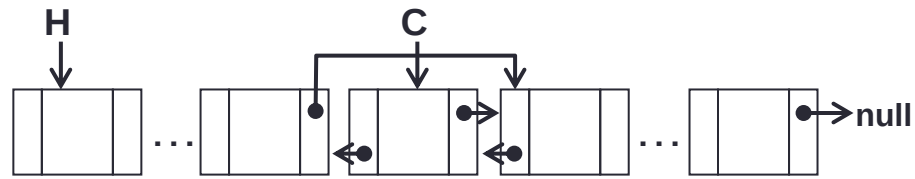
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

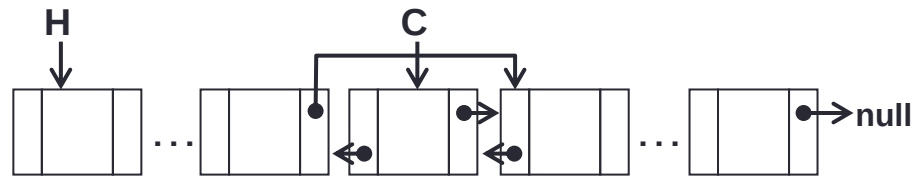
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

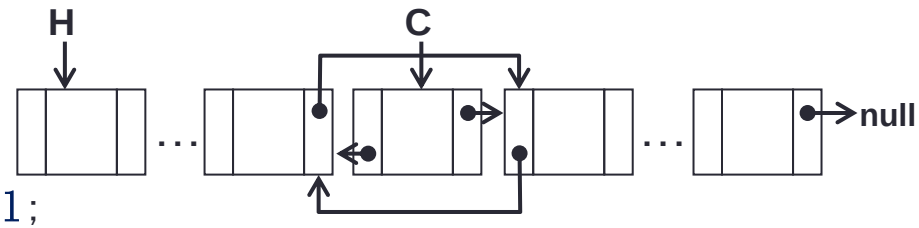
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

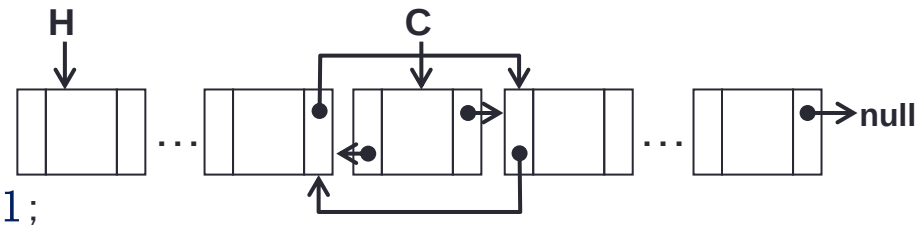
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

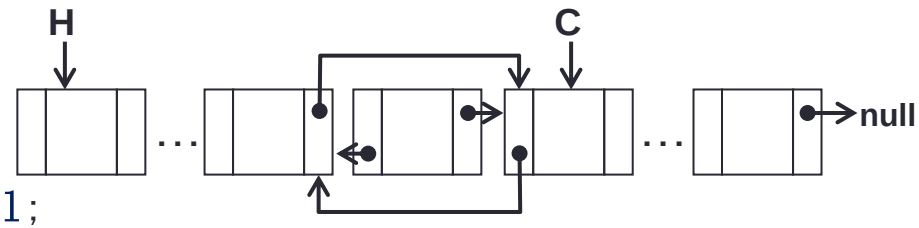
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

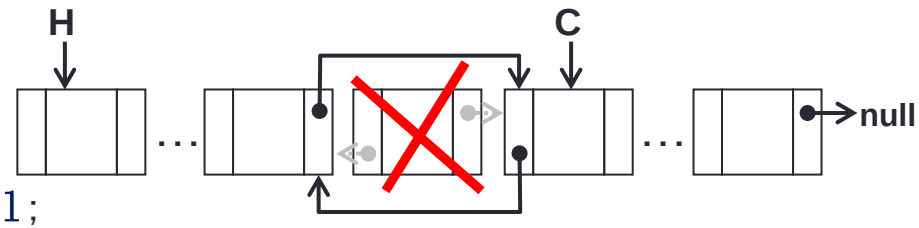
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

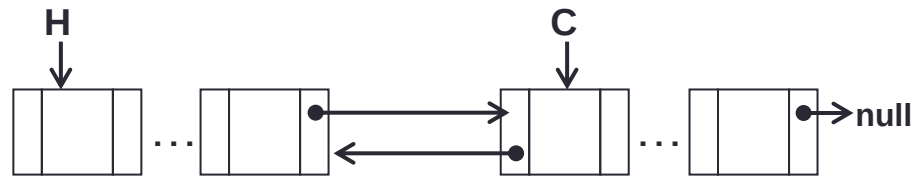
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

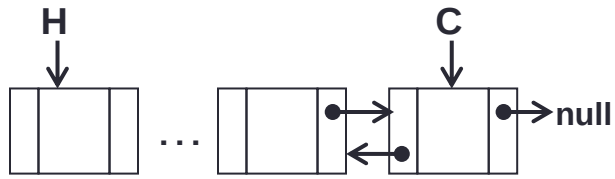
Example #3



ADT List (Double-Linked List): Implementation

```
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;
  }
}
```

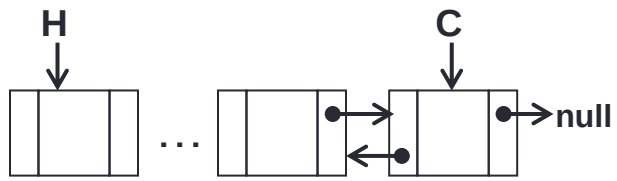
Example #4



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;
  }
}
```

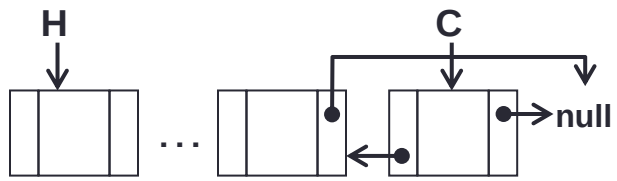
Example #4



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;
  }
}
```

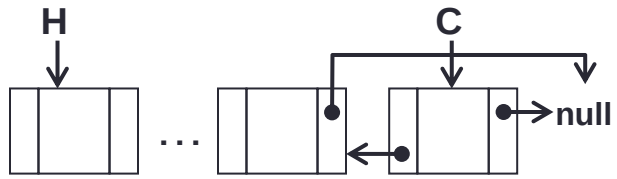
Example #4



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;
  }
}
```

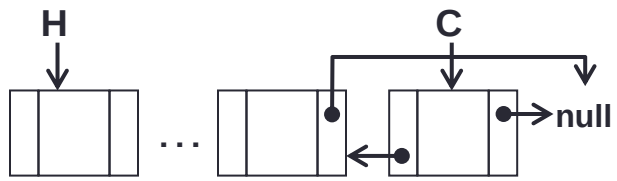
Example #4



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;
  }
}
```

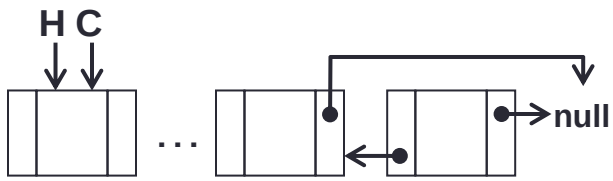
Example #4



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;
  }
}
```

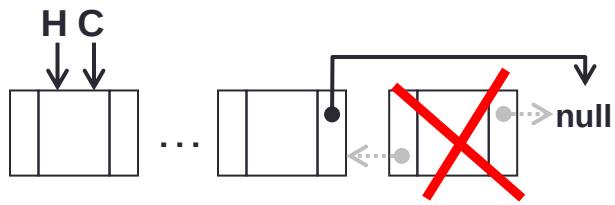
Example #4



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;
  }
}
```

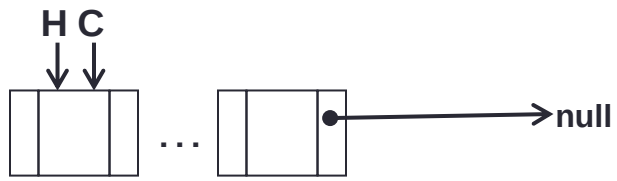
Example #4



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 #4



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	-	-	?
Retrieve	O(1)	O(1)	?
Update	O(1)	O(1)	?
Insert	O(n)	O(1)	?
Remove	O(n)	O(n)	?

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)