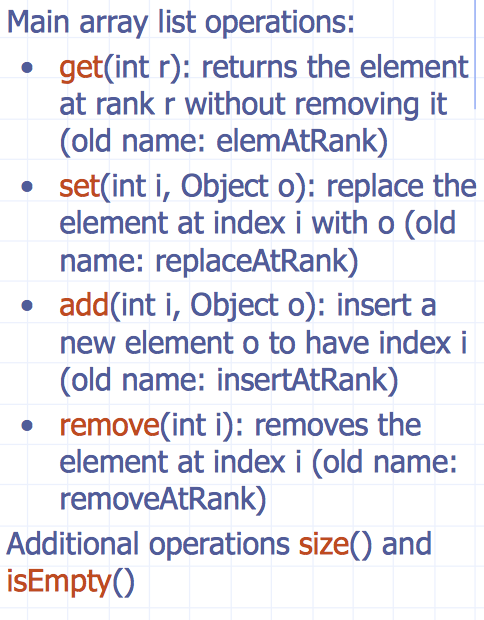
# Ch 7: List and Iterator

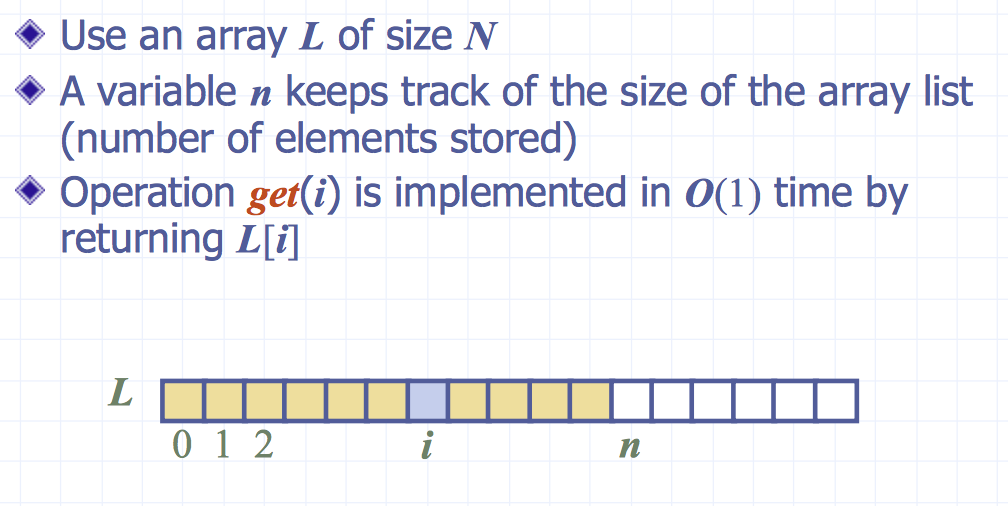
# 7.1 Array List ADT

Array List ADT extends the notion of array by storing a sequence of arbitrary object.

Element can be accessed, inserted or removed by its rank (#of elements preceding it)

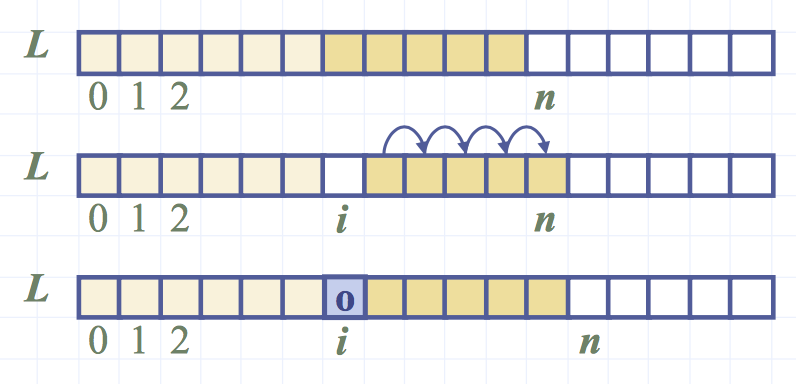
Exception thrown if incorrect index given





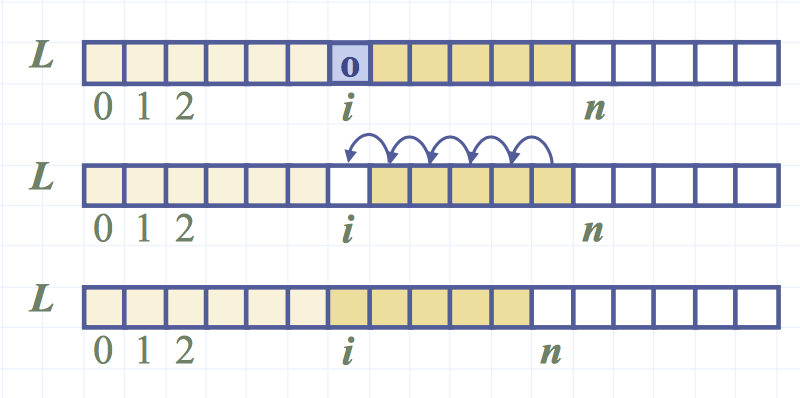
## 7.1.1 Insertion

operation add(i, o) , need to make room for new element by shifting forward the n-i elements. Worst case (i=0) takes O(n) time



## 7.1.2 Deletion

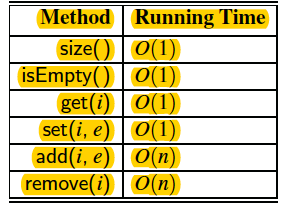
operation remove(i), need to fill hole left by removed element by shifting backward n − i − 1 elements L[i + 1], ..., L[n − 1]. worst case (i = 0) takes O(n) time



## 7.2.3 Performance

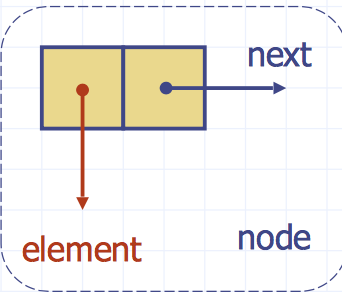
Space used is O(n)

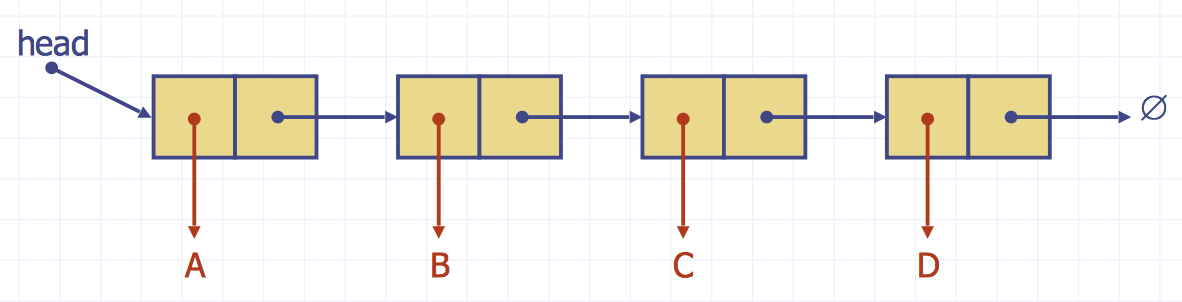
If use array in circular fashion: add(0,o) and remove(0) run in O(1) time. IF from beginning, if not O(n).

In add(), when array full, can replace array with larger one.

# 7.2 Singly Linked List

A singly linked list is a concrete data structure next consisting of a sequence of nodes.

Each ***node*** stores: element and link to next node.

To find end, need to traverse, if last pointer == null, then can add element. To add element, add to the head.

# 7.3 Positional List ADT

ADT that provides a user a way to refer to elements anywhere in a sequence, and to perform arbitrary insertions and deletions.

Can be implemented by an array OR dynamic list.

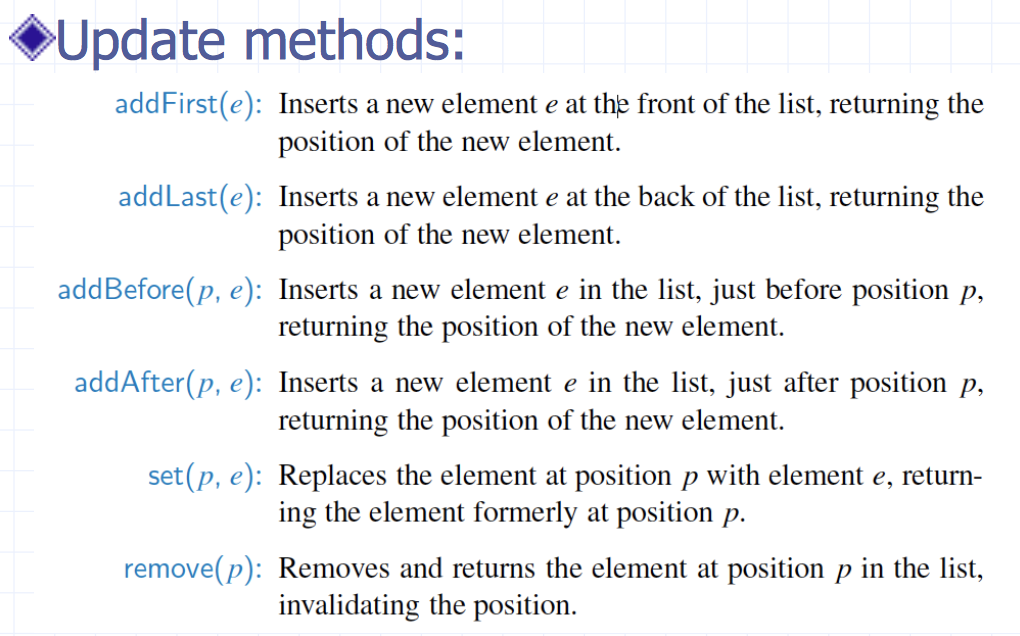
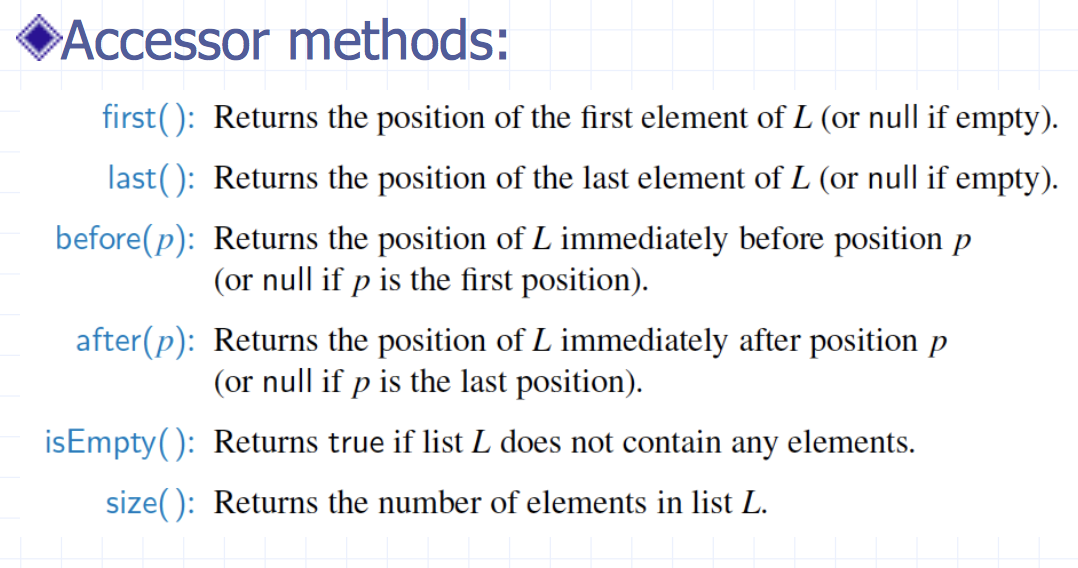
***Position*** acts as marker within the broader positional list.

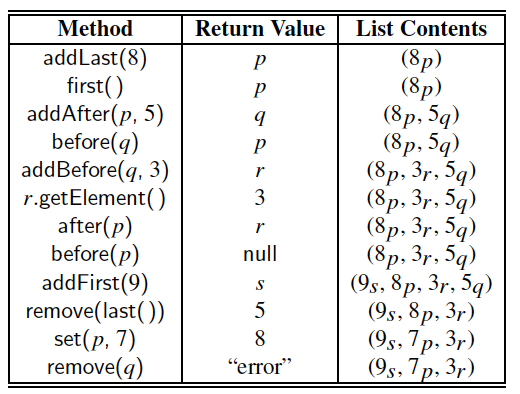
Position p is ***unaffected by changes*** elsewhere in a list; only way in which a position becomes invalid is if an explicit command is issued to delete it.

***Position instance is a simple object***, methods:

• ***P.getElement( ):*** Return the element stored at position p.

• ***P.setElement(e):*** Store element e at position p



Example:

# 7.4 Node List ADT

It’s a sequence of positions storing arbitrary objects. It establishes a before/ after relation between positions.

Generic methods: • size(), isEmpty()

Accessor methods: • first(), last() • prev(p), next(p)

Update Methods: • set(p, e) • addBefore(p, e), addAfter(p, e) • addFirst(e), addLast(e) • remove(p)

## Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 7.53.30 PM.png7.4.1 Inserting at the head

* Allocate new node
* Insert new element
* Have new node point to old head
* Update head to point to new node

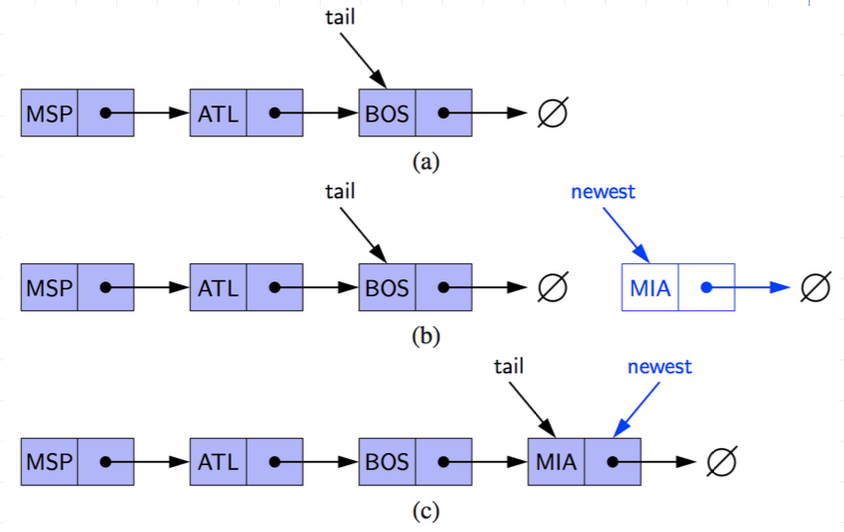
## Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 7.56.12 PM.png7.4.2 Removing at the head

* Update head to point to next node in

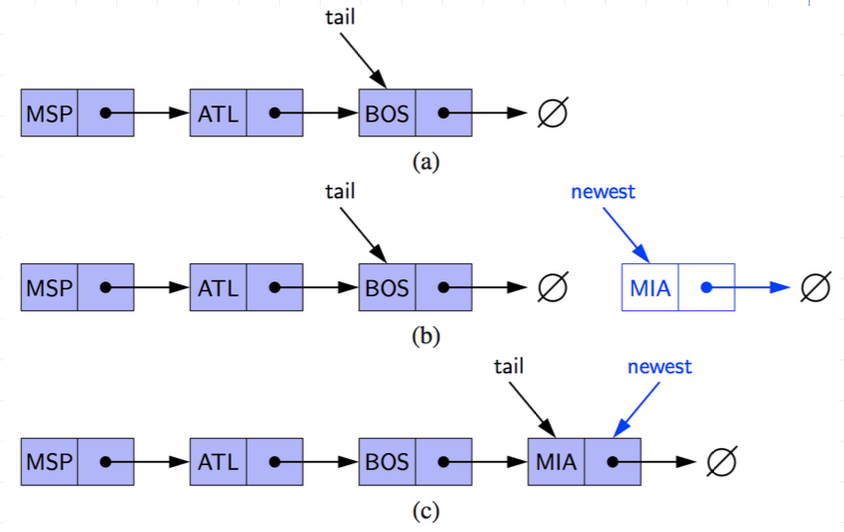
the list

* Allow garbage collector to reclaim the

former first node

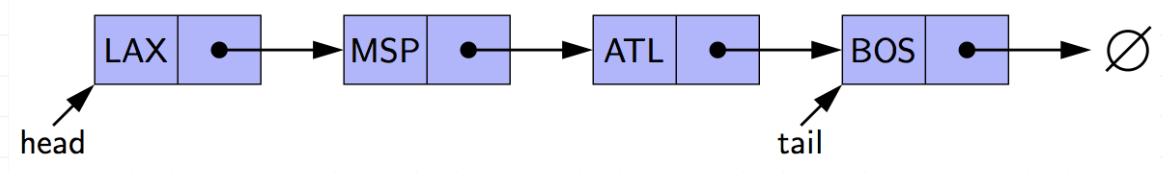


## 7.4.3 Inserting at the tail

* Allocate a new node
* Insert new element
* Have new node point to null
* Have old last node point to new node
* Update tail to point to new node

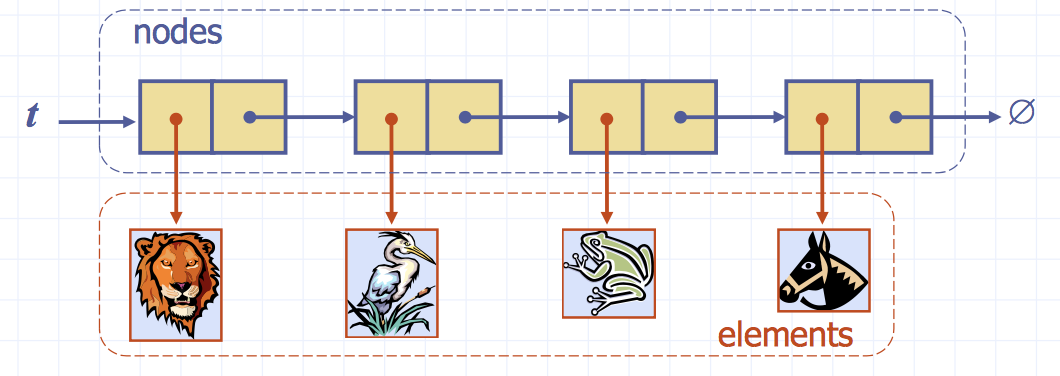
## 7.4.4 Removing at the tail

* Removing at the tail of a singly linked list is not efficient!
* There is no constant-time way to update the tail to point to the previous node



# 7.5 Stack as Single Linked List

* We can implement a stack with a singly linked list The top element is stored at the first node of the list
* The space used is O(n) and each operation of the Stack ADT takes O(1) time

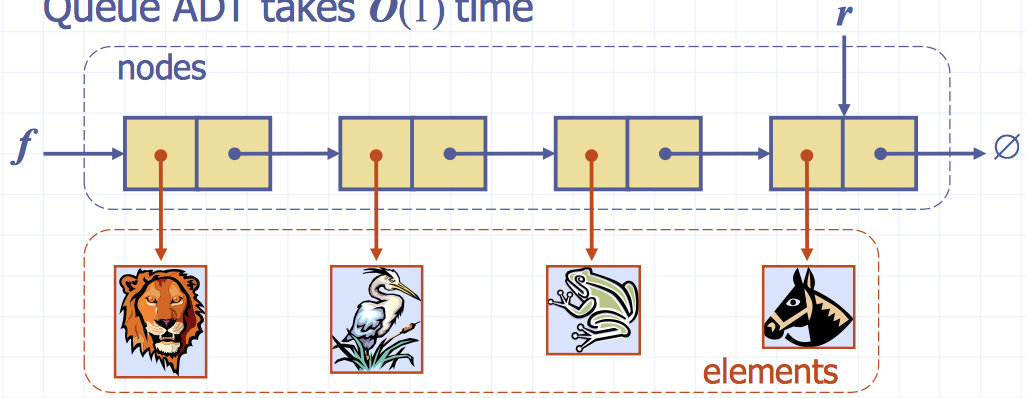


# 7.6 Queue as Single Linked List

We can implement a queue with a singly linked list

• front element is stored at the first node

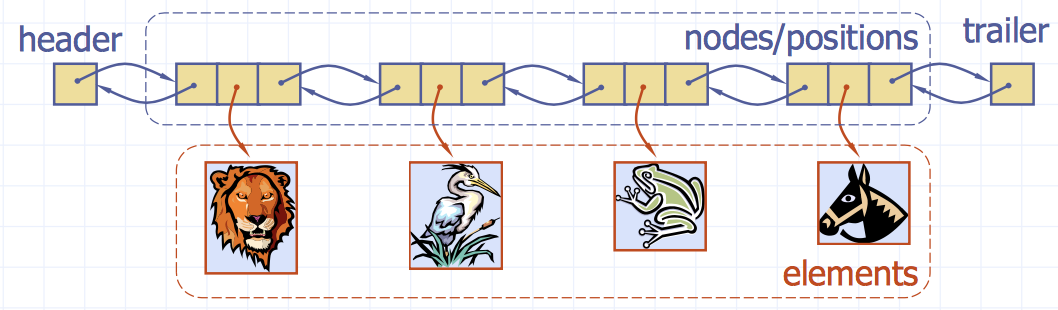
• rear element is stored at the last node

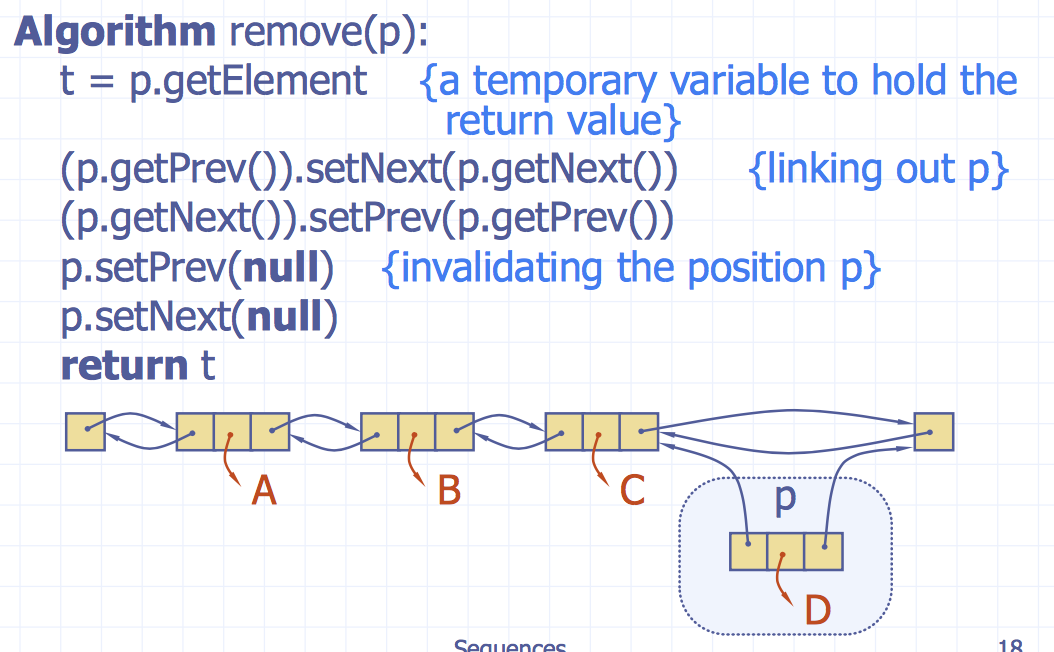
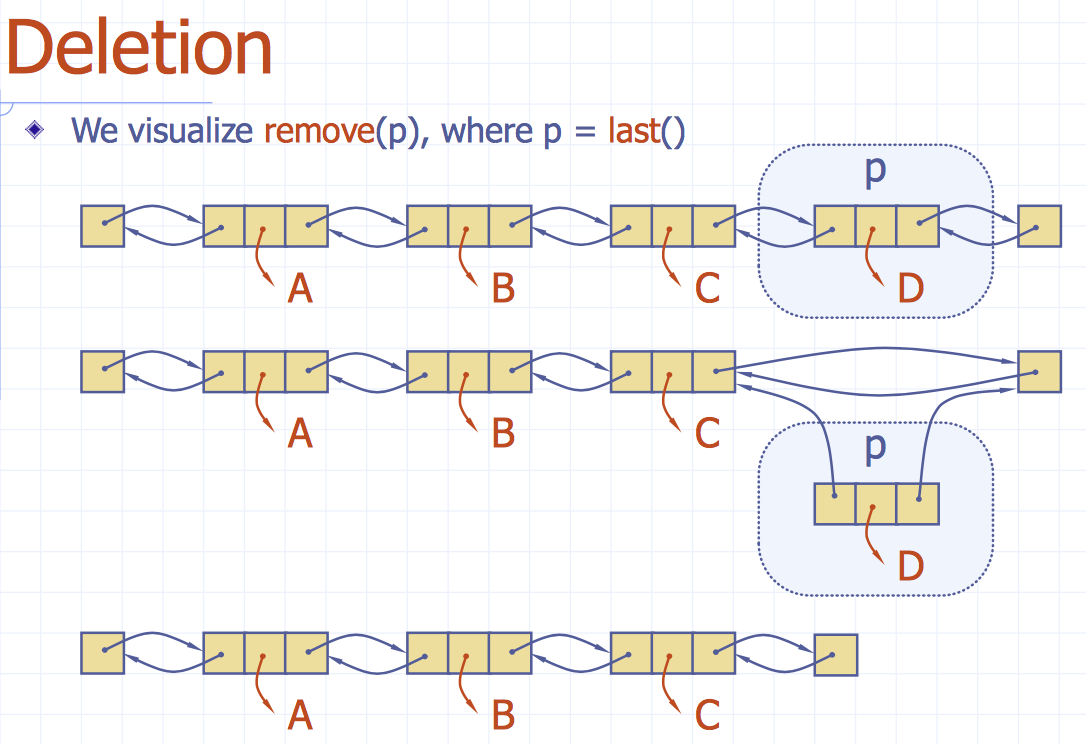
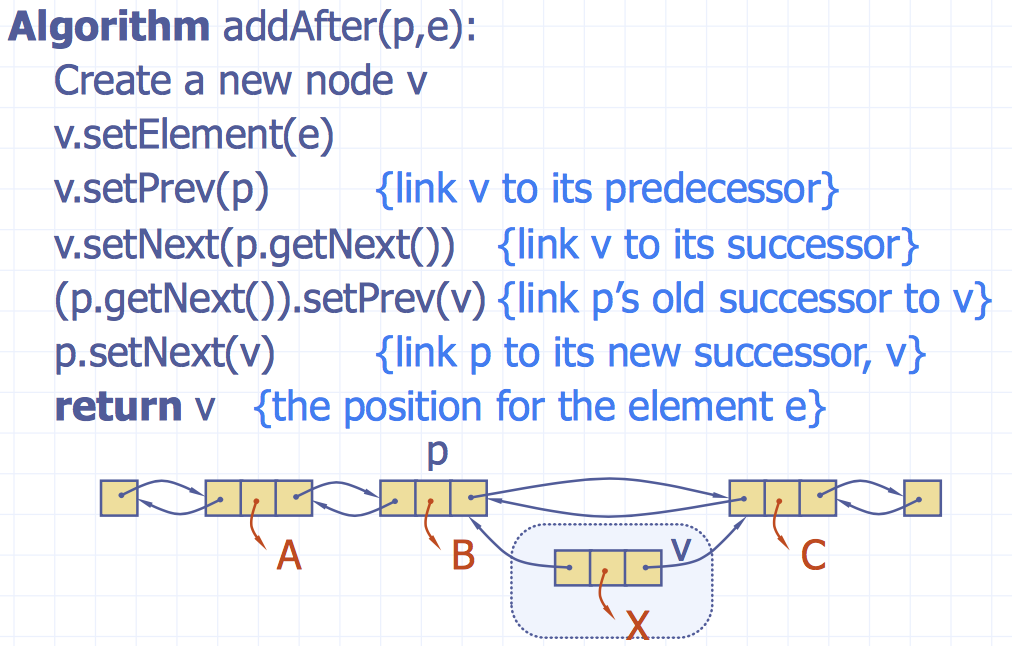
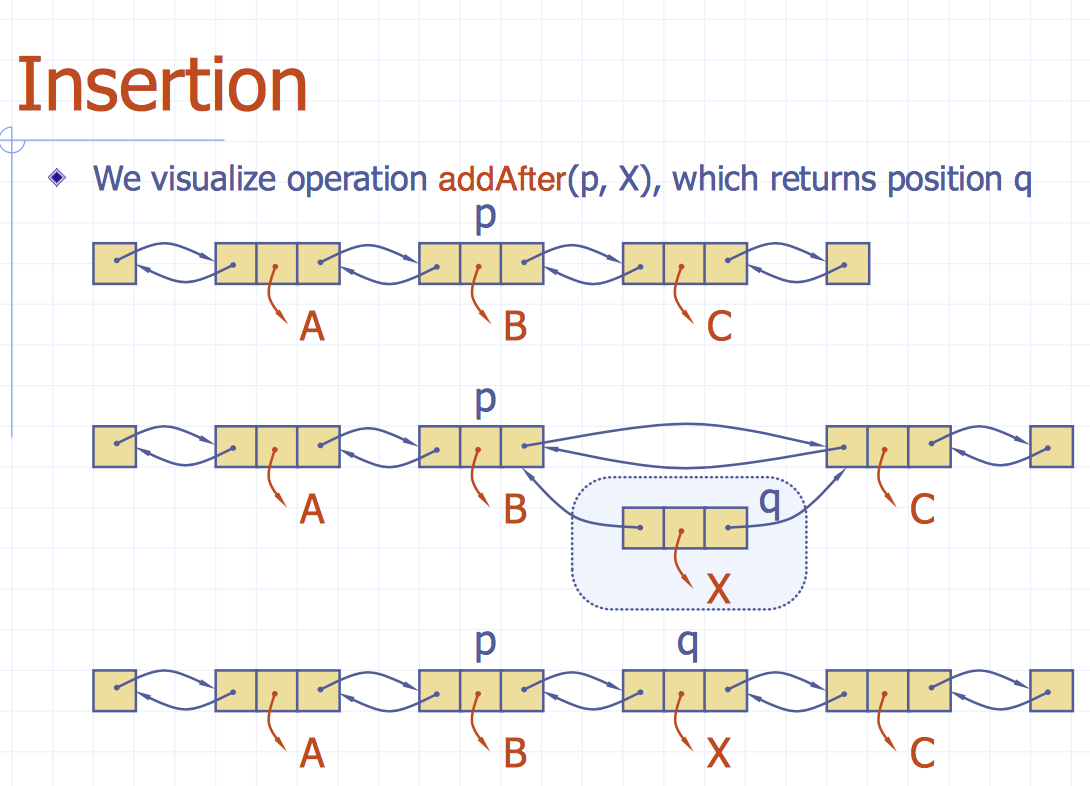
Space used is O(n) and each operation of the Queue ADT takes O(1) time

# Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 8.20.45 PM.png7.7 Double Linked-List

It provides a natural implementation of the Positional List ADT

A node ***implement*** and ***store***:

* element
* link to the previous node
* link to the next nod



## 7.7.1 Performance

***Space*** used by a ***list*** with n elements is O(n)

by each ***position*** of the list is O(1)

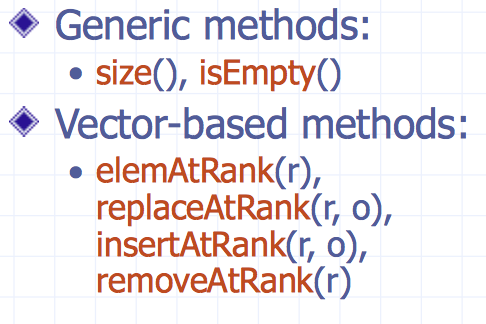
All operations of the List ADT run in O(1) time

Operations setElement() and getElement() of the Position ADT runs in O(1) time

# Macintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 8.42.48 PM.png7.8 Sequence ADT

It’s the union of the Vector and List ADTs

Elements accessed by Rank or Position



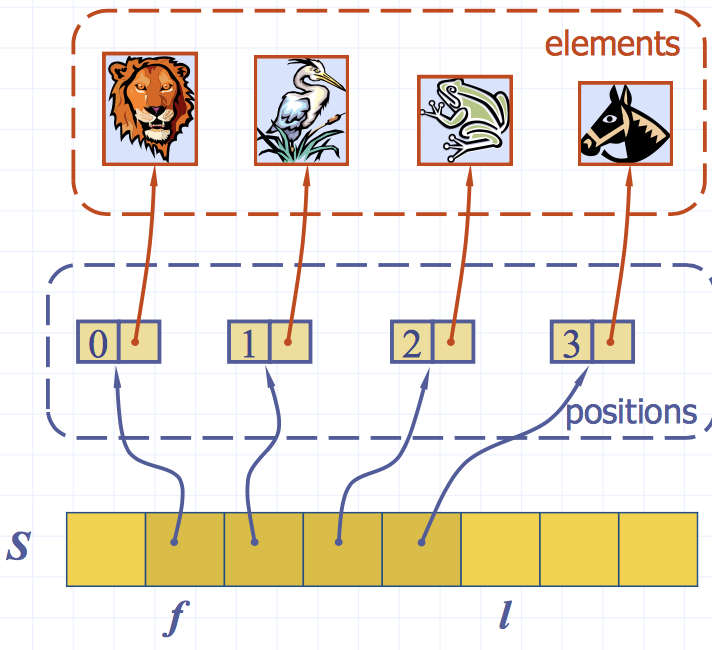
## 7.8.1 array-based implementation

We use a circular array storing positions

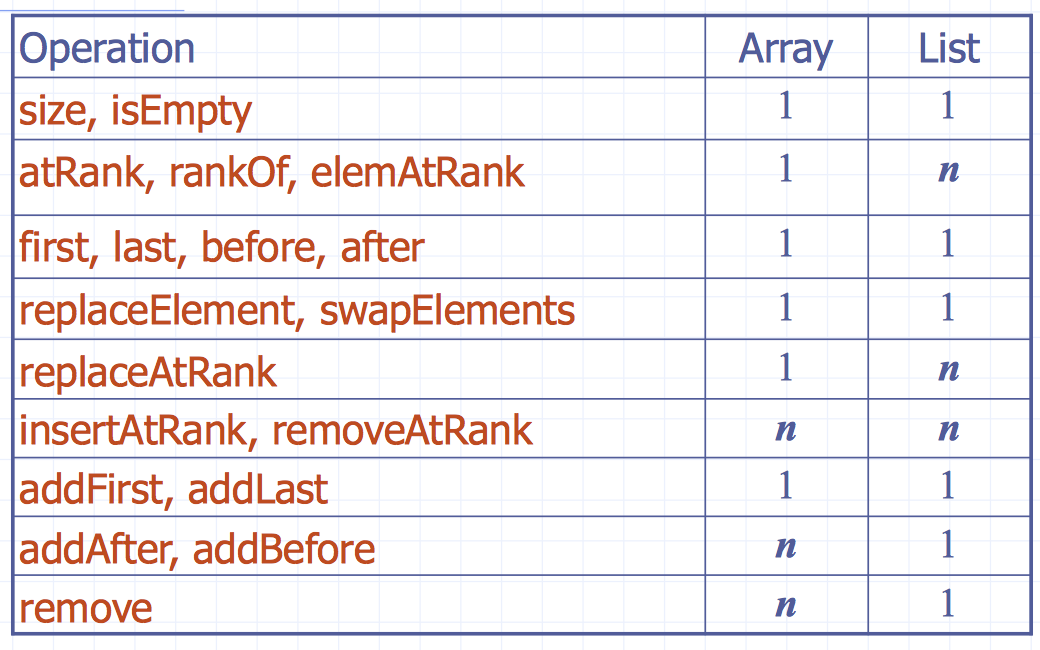
A position object stores:

• Element • Rank

Indices f and l keep track of first and last positions



## 7.8.2 Sequence Implementation



addFirst, removeFirst: array (1) if cyclic array. Else(n)

# 7.9 Favourites List ADT

It models a collection of elements while keeping track of the number of times each element is accessed.

The access counts allow us to know which elements are most frequently accessed.

Additional Methods:

• access(e): accessed the element e while incrementing its access count.

• remove(e): removes the element e from list.

• top(k): returns list of k most accessed elements.

# 7.10 Sequential search

O(n)

# 7.11 IteratorsMacintosh HD:Users:noemilemonnier:Desktop:Screen Shot 2017-10-20 at 8.52.27 PM.png