



CH7.

LIST AND ITERATOR ADTS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN JAVA, GOODRICH, TAMASSIA AND GOLDWASSER (WILEY 2016)

LIST ADT

- `size()`: Returns the number of elements in the list.
- `isEmpty()`: Returns a boolean indicating whether the list is empty.
- `get(i)`: Returns the element of the list having index *i*; an error condition occurs if *i* is not in range $[0, \text{size}() - 1]$.
- `set(i, e)`: Replaces the element at index *i* with *e*, and returns the old element that was replaced; an error condition occurs if *i* is not in range $[0, \text{size}() - 1]$.
- `add(i, e)`: Inserts a new element *e* into the list so that it has index *i*, moving all subsequent elements one index later in the list; an error condition occurs if *i* is not in range $[0, \text{size}()]$.
- `remove(i)`: Removes and returns the element at index *i*, moving all subsequent elements one index earlier in the list; an error condition occurs if *i* is not in range $[0, \text{size}() - 1]$.

EXAMPLE

- A sequence of List operations:

Method	Return Value	List Contents
add(0, A)	–	(A)
add(0, B)	–	(B, A)
get(1)	A	(B, A)
set(2, C)	“error”	(B, A)
add(2, C)	–	(B, A, C)
add(4, D)	“error”	(B, A, C)
remove(1)	A	(B, C)
add(1, D)	–	(B, D, C)
add(1, E)	–	(B, E, D, C)
get(4)	“error”	(B, E, D, C)
add(4, F)	–	(B, E, D, C, F)
set(2, G)	D	(B, E, G, C, F)
get(2)	G	(B, E, G, C, F)

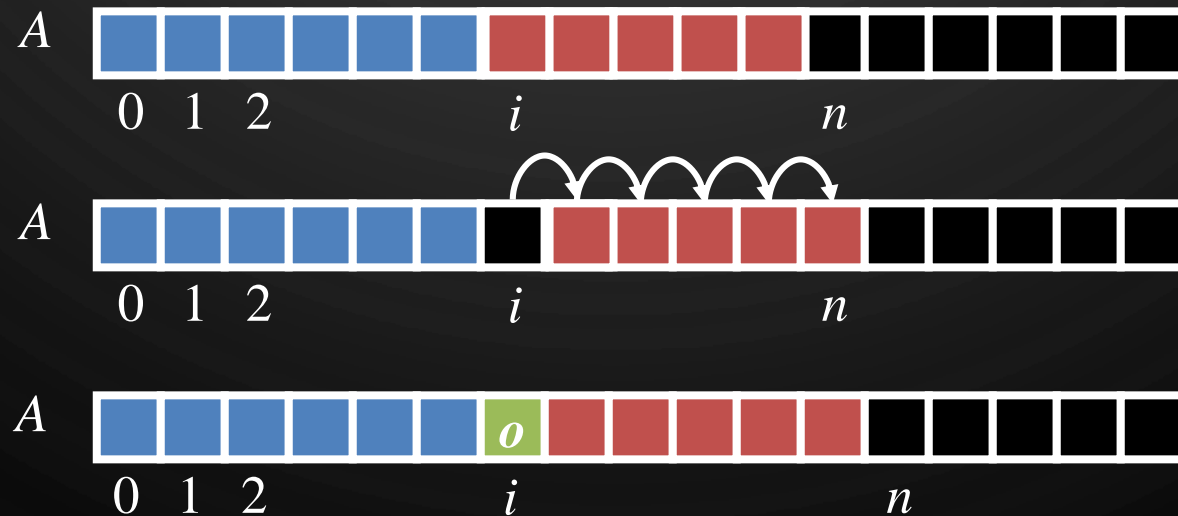
ARRAY LISTS

- An obvious choice for implementing the list ADT is to use an array, A , where $A[i]$ stores (a reference to) the element with index i .
- With a representation based on an array A , the $\text{get}(i)$ and $\text{set}(i, e)$ methods are easy to implement by accessing $A[i]$ (assuming i is a legitimate index).



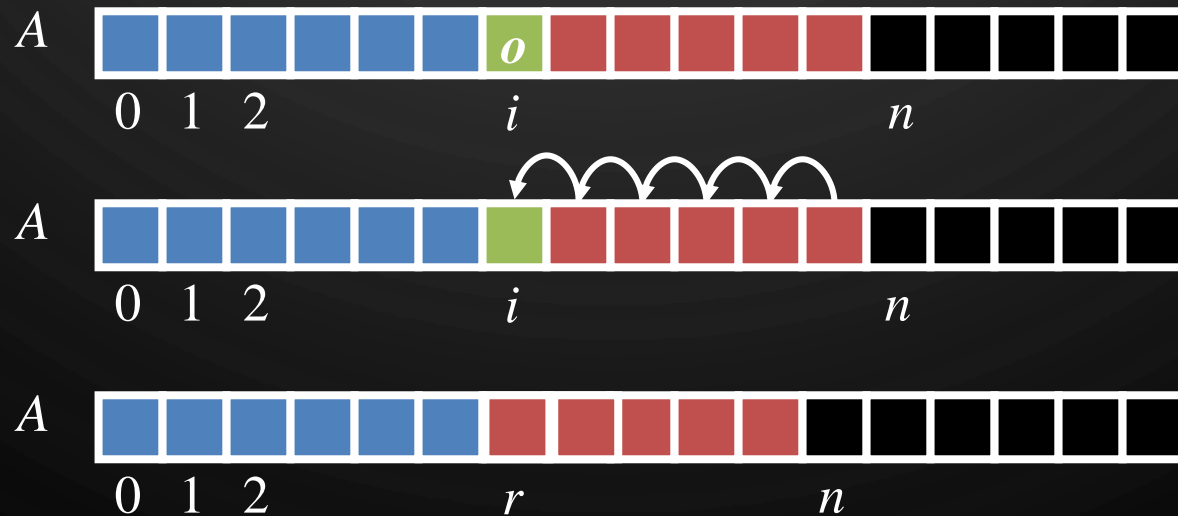
INSERTION

- In an operation $\text{add}(i, o)$, we need to make room for the new element by shifting forward the $n - i$ elements $A[i], \dots, A[n - 1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



ELEMENT REMOVAL

- In an operation `remove(i)`, we need to fill the hole left by the removed element by shifting backward the $n - i - 1$ elements $A[i + 1], \dots, A[n - 1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



PERFORMANCE

- In an array-based implementation of a dynamic list:
 - The space used by the data structure is $O(n)$
 - Indexing the element (`get/set`) at i takes $O(1)$ time
 - `add` and `remove` run in $O(n)$ time
- In an `add` operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one ...

EXERCISE:

- Implement the Deque ADT with the List ADT
 - Deque ADT:
 - `first()`, `last()`, `addFirst(e)`, `addLast(e)`,
`removeFirst()`, `removeLast()`, `size()`, `isEmpty()`
 - List functions:
 - `get(i)`, `set(i, e)`, `add(i, e)`, `remove(i)`, `size()`,
`isEmpty()`


LIST SUMMARY

	Array Fixed-Size or Expandable	List Singly or Doubly Linked
<code>add(i, e), remove(i)</code>	$O(1)$ Best Case ($i = n$) $O(n)$ Worst Case $O(n)$ Average Case	?
<code>get(i), set(i, e)</code>	$O(1)$?
<code>size(), isEmpty()</code>	$O(1)$?

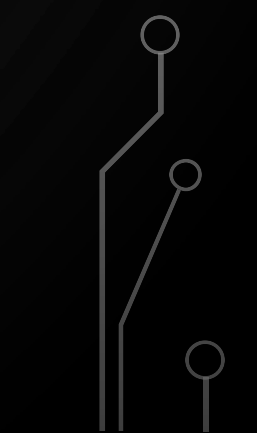


INTERVIEW QUESTION 1

- Write code to partition a list around a value x , such that all nodes less than x come before all nodes greater than or equal to x .




GAYLE LAAKMANN MCDOWELL, "CRACKING THE CODE INTERVIEW: 150 PROGRAMMING QUESTIONS AND SOLUTIONS", 5TH EDITION, CAREERCUP PUBLISHING, 2011.



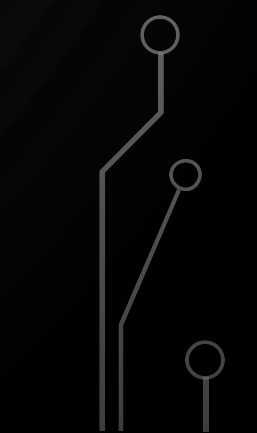


INTERVIEW QUESTION 2

- Implement a function to check if a list is a palindrome.



GAYLE LAAKMANN MCDOWELL, "CRACKING THE CODE INTERVIEW: 150 PROGRAMMING QUESTIONS AND SOLUTIONS", 5TH EDITION, CAREERCUP PUBLISHING, 2011.



POSITIONAL LISTS

- To provide for a general abstraction of a sequence of elements with the ability to identify the location of an element, we define a positional list ADT.
- A **position** acts as a marker or token within the broader positional list.
- A position p is unaffected by changes elsewhere in a list; the only way in which a position becomes invalid is if an explicit command is issued to delete it.
- A position instance is a simple object, supporting only the following method:
 - `p.getElement()`: Return the element stored at position p .

POSITIONAL LIST ADT

- Accessor methods:

`first()`: Returns the position of the first element of L (or null if empty).

`last()`: Returns the position of the last element of L (or null if empty).

`before(p)`: Returns the position of L immediately before position p (or null if p is the first position).

`after(p)`: Returns the position of L immediately after position p (or null if p is the last position).

`isEmpty()`: Returns true if list L does not contain any elements.

`size()`: Returns the number of elements in list L .

POSITIONAL LIST ADT, 2

- Update methods:

addFirst(e): Inserts a new element e at the front of the list, returning the position of the new element.

addLast(e): Inserts a new element e at the back of the list, returning the position of the new element.

addBefore(p, e): Inserts a new element e in the list, just before position p , returning the position of the new element.

addAfter(p, e): Inserts a new element e in the list, just after position p , returning the position of the new element.

set(p, e): Replaces the element at position p with element e , returning the element formerly at position p .

remove(p): Removes and returns the element at position p in the list, invalidating the position.

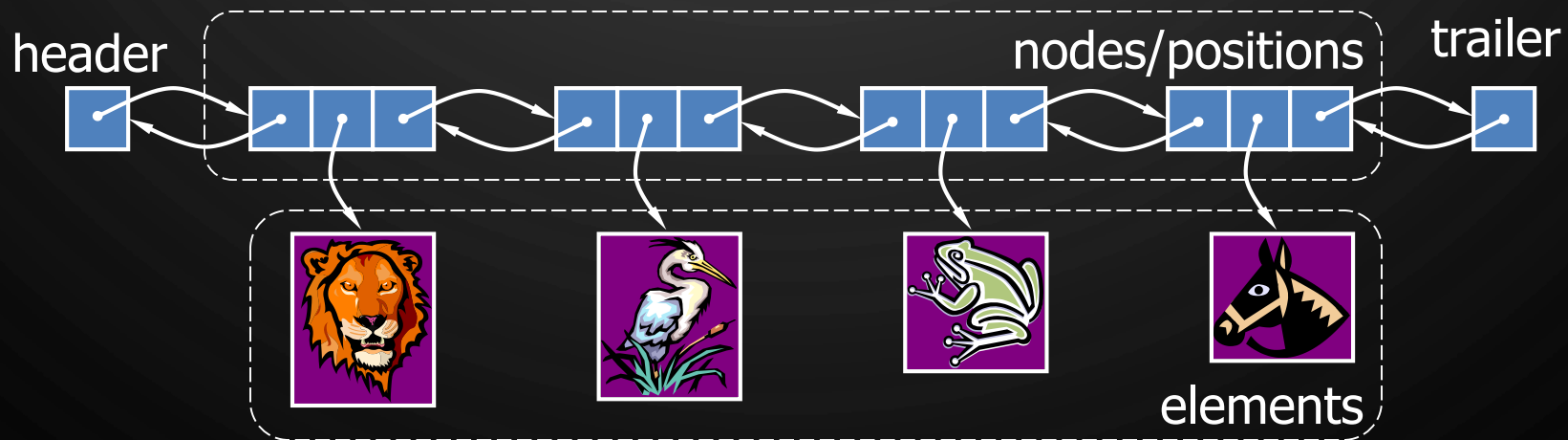
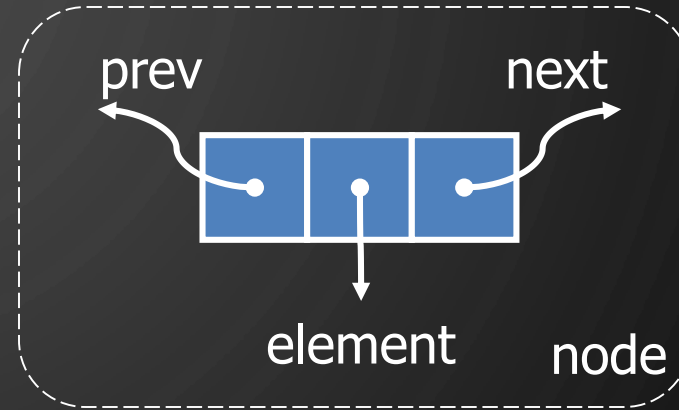
EXAMPLE

- A sequence of Positional List operations:

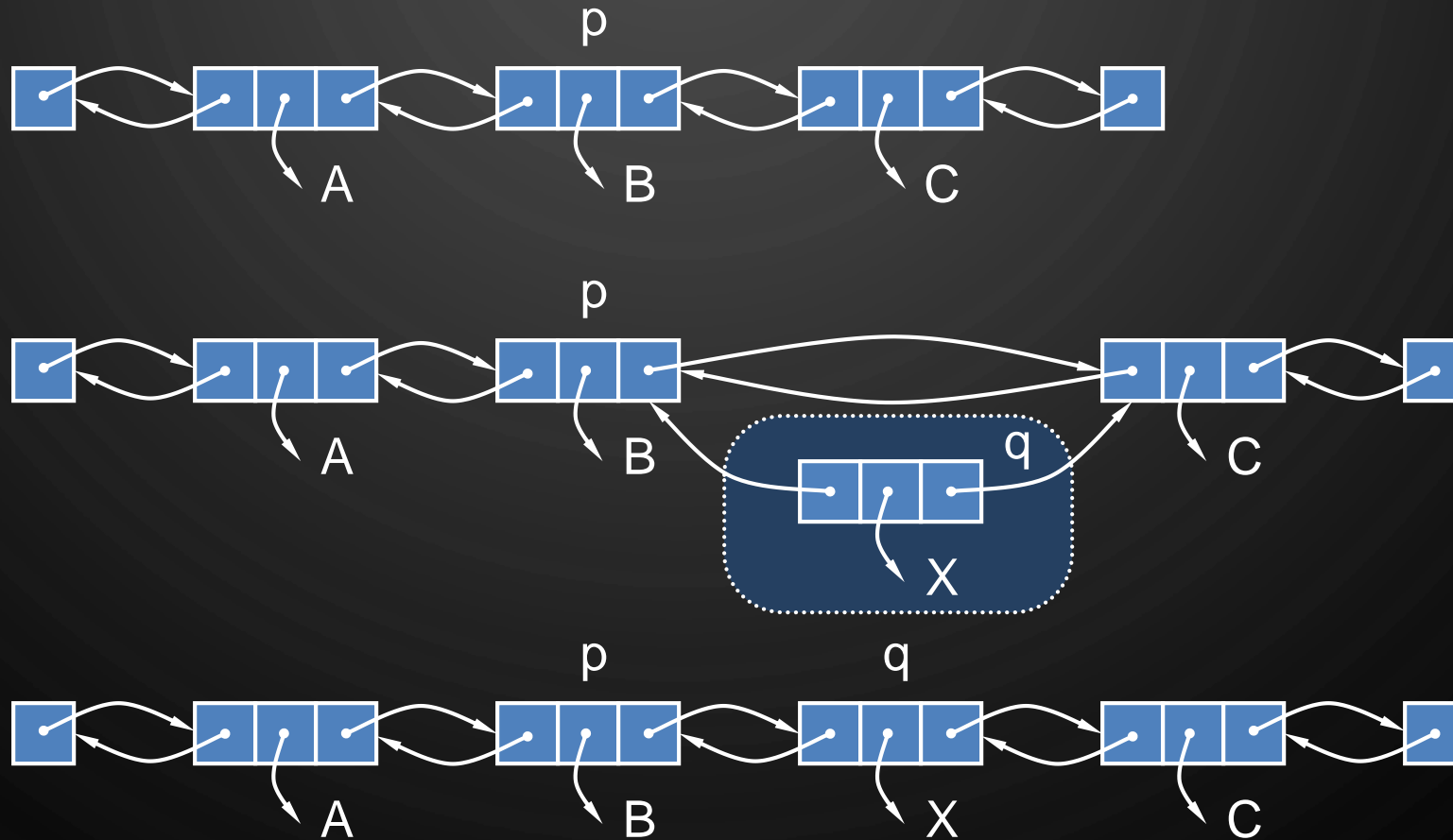
Method	Return Value	List Contents
addLast(8)	p	(8 p)
first()	p	(8 p)
addAfter(p , 5)	q	(8 p , 5 q)
before(q)	p	(8 p , 5 q)
addBefore(q , 3)	r	(8 p , 3 r , 5 q)
r .getElement()	3	(8 p , 3 r , 5 q)
after(p)	r	(8 p , 3 r , 5 q)
before(p)	null	(8 p , 3 r , 5 q)
addFirst(9)	s	(9 s , 8 p , 3 r , 5 q)
remove(last())	5	(9 s , 8 p , 3 r)
set(p , 7)	8	(9 s , 7 p , 3 r)
remove(q)	"error"	(9 s , 7 p , 3 r)

POSITIONAL LIST IMPLEMENTATION

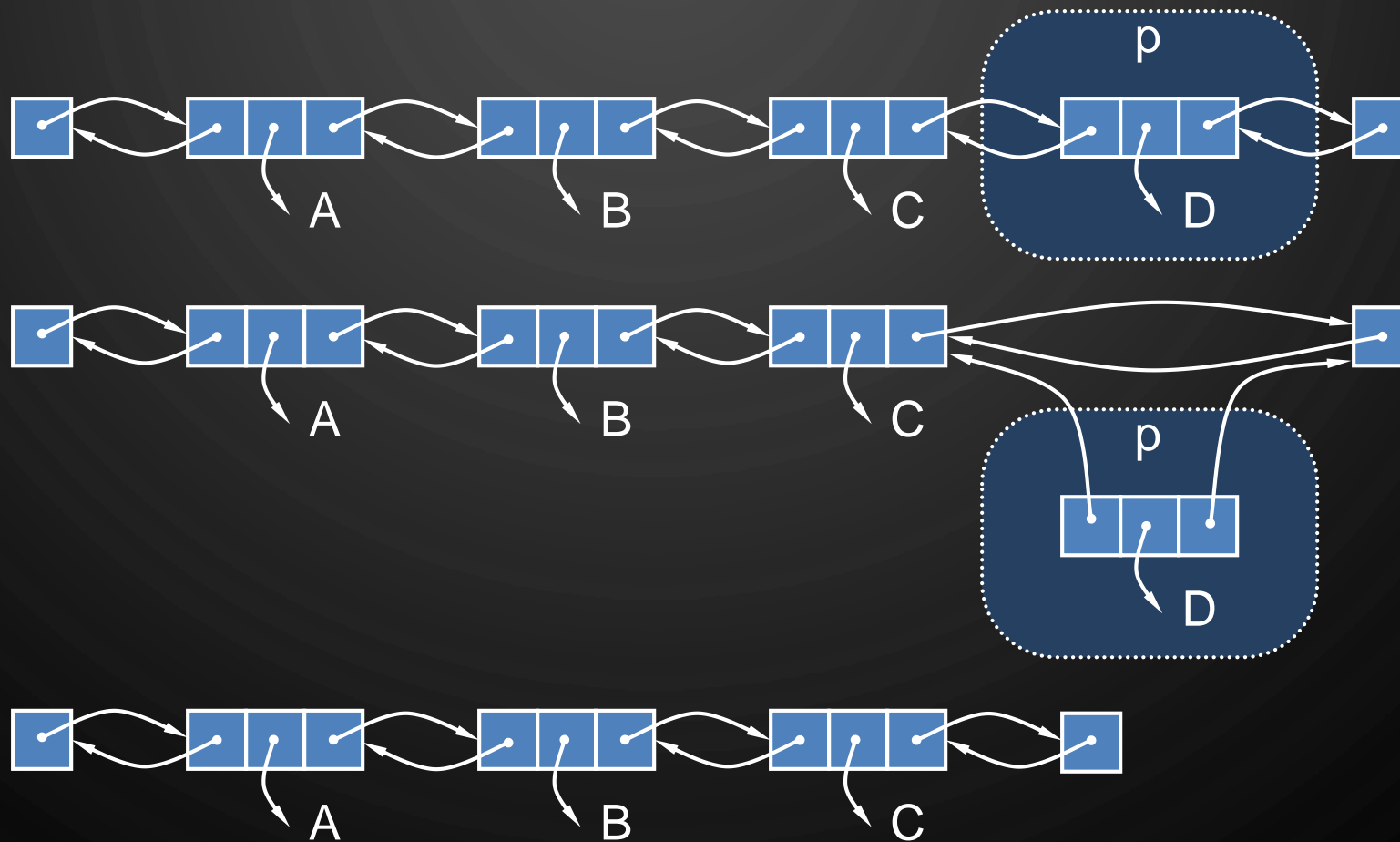
- The most natural way to implement a positional list is with a doubly-linked list.



INSERTION, E.G., ADDAFTER (P, E)



REMOVE (P)



PERFORMANCE

- Assume doubly-linked list implementation of Positional List ADT
 - The space used by a list with n elements is $O(n)$
 - The space used by each position of the list is $O(1)$
 - All the operations of the List ADT run in $O(1)$ time

POSITIONAL LIST SUMMARY

	List Singly-Linked	List Doubly- Linked
<code>first()</code> , <code>last()</code> , <code>addFirst()</code> , <code>addLast()</code> , <code>addAfter()</code>	$O(1)$	$O(1)$
<code>addBefore(p, e)</code> , <code>erase()</code>	$O(n)$ Worst and Average case $O(1)$ Best case	$O(1)$
<code>size()</code> , <code>isEmpty()</code>	$O(1)$	$O(1)$

INTERVIEW QUESTION 3

- When Bob wants to send Alice a message M on the internet, he breaks M into n data packets, numbers the packets consecutively, and injects them into the network. When the packets arrive at Alice's computer, they may be out of order, so Alice must assemble the sequence of n packets in order before she can be sure she has the entire message. Using Positional Lists describe and analyze an algorithm for Alice to do this.
 - Can you do better with a regular List?