ساختمان داده و الگوریتم ها

مبحث هشتم:
لیست و پشته

سجاد شیرعلی شهرضا

پاییز 1402

دوشنبه، 1 آبان 1402

Based on Washington University CSE 373 (Autumn 2020 by Hunter Schafer), University of North Florida COP2220 (Chip Klostermeyer), and University of Illinois Chicago CS 201 (Spring 2014 by Bing Liu) slides

اطلاع رسانی

- بخش مرتبط کتاب برای این جلسه: 10
- نظرسنجی دوم: امروز در طی کلاس!

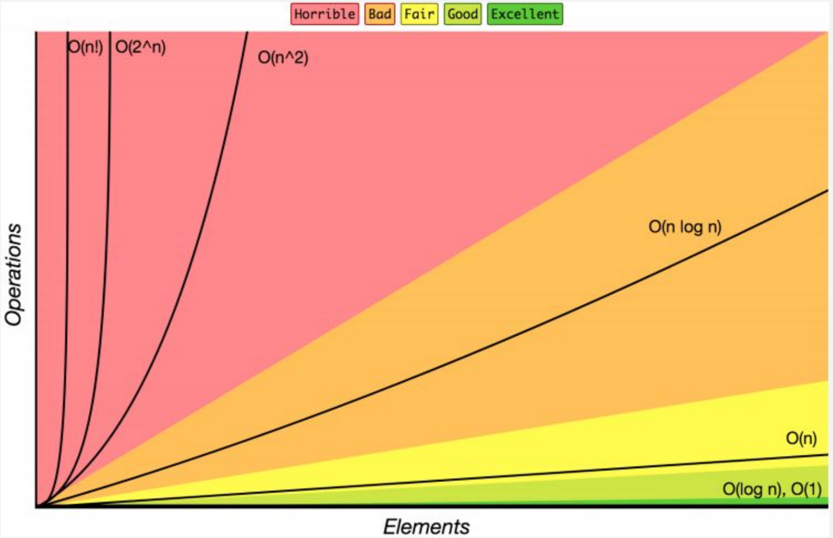
لیست

مجموعه ای ترتیب دار از اشیاء

Complexity Class Reminder

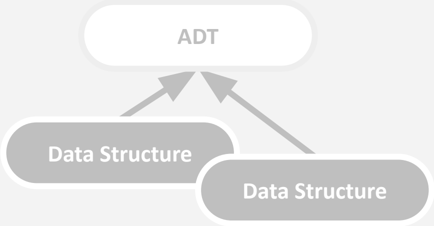
- **Complexity Class**: a category of algorithm efficiency based on the algorithm's relationship to the input size N

Complexity Class	Big-O	Runtime if you double N
constant	**O(1)**	**unchanged**
logarithmic	$O(\log_2 N)$	increases slightly
linear	**O(N)**	**doubles**
log-linear	$O(N \log_2 N)$	slightly more than doubles
quadratic	$\mathbf{O(N^2)}$	**quadruples**
...	...	...
exponential	$O(2^N)$	multiplies drastically



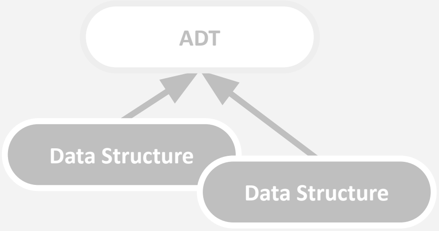
ADTs: Abstract Data Types

- Abstract Data Type (ADT): a data type that does not specify any one implementation.
  - An agreement about what is provided, but not how

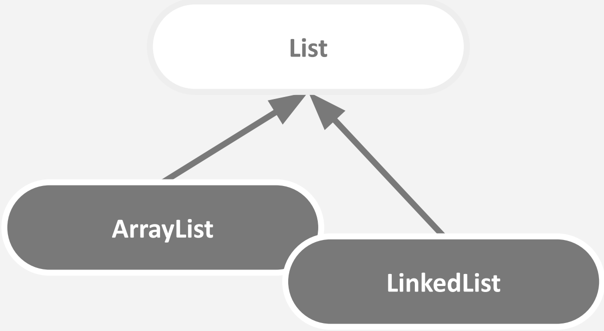


ADTs: Abstract Data Types

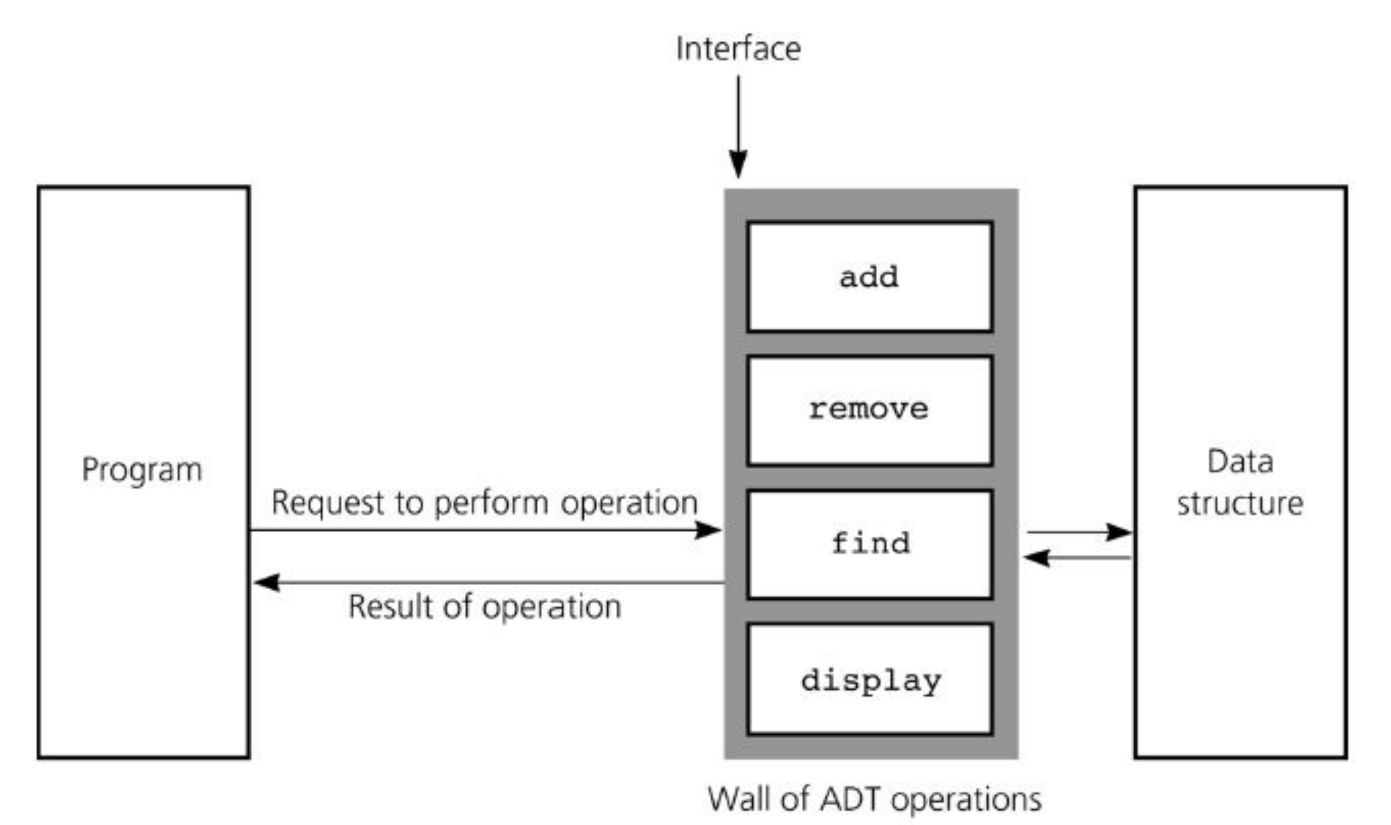
- Abstract Data Type (ADT): a data type that does not specify any one implementation.
  - An agreement about what is provided, but not how
- Data Structures implement ADTs
  - Resizable array can implement List, Stack, Queue, Deque, PQ, etc.
  - Linked nodes can implement List, Stack, Queue, Deque, PQ, etc.



For Example:



Another View of ADT



List ADT

- **List**: a collection storing an ordered sequence of elements.
  - Has a variable size defined as the number of elements in it
  - Items:
    - Are accessible by an index
    - Can be added to or removed from any position in the list
- Relation to code/mental image of a list:

```
List<String> names = new ArrayList<>(); // []
names.size();                           // evaluates to 0
names.add(“Reza”);                      // [“Reza”]
names.add(“Zahra”);                     // [“Reza, Zahra”]
names.insert(“Ali”, 0);                 // [“Ali”, “Reza”, “Zahra”]
names.size();                           // evaluates to 3
```

List Implementations

LIST ADT

State

- Set of ordered items
- Count of items

Behavior

- get(index) return item at index
- set(item, index) replace item at index
- add(item) add item to end of list
- insert(item, index) add item at index
- delete(index) delete item at index
- size() count of items

```
[88.6, 26.1, 94.4]
```

List Implementations

LIST ADT

State

Set of ordered items
Count of items

Behavior

get(index) return item at index
set(item, index) replace item at index
add(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items

ArrayList<E>

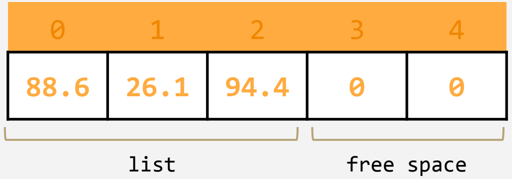
State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

```
[88.6, 26.1, 94.4]
```



0	1	2	3	4
88.6	26.1	94.4	0	0

list

free space

List Implementations

LIST ADT

State

Set of ordered items
Count of items

Behavior

get(index) return item at index
set(item, index) replace item at index
add(item) add item to end of list
insert(item, index) add item at index
delete(index) delete item at index
size() count of items

ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

LinkedList<E>

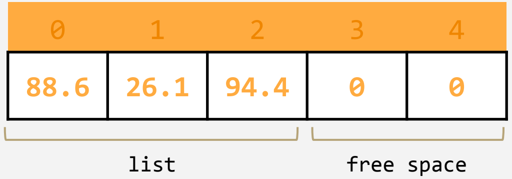
State

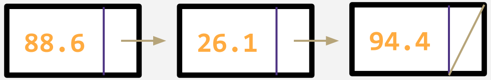
```
Node front;
size
```

Behavior

```
get loop until index, return node’s
value
set loop until index, update node’s
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```

[88.6, 26.1, 94.4]





پیاده سازی لیست با آرایه

ArrayList Storage

- Stores elements inside an array
  - Has a fixed capacity
  - Typically has more space than currently used
- Stores all of these elements at the front of the array
  - Keeps track of how many items added

List View

```
[“Ali”, “Reza”, “Zahra”]
```

ArrayList View

```
[“Ali”, “Reza”, “Zahra”, null, null, null]
```

ArrayList Insert

ArrayList<E>

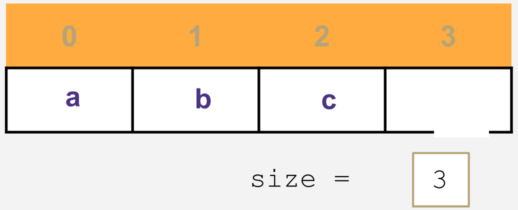
State

data[]
size

Behavior

get return data[index]
set data[index] = value
add data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size

`insert(element, index)` with shifting



0	1	2	3
a	b	c	

`size =` 3

ArrayList Insert

ArrayList<E>

State

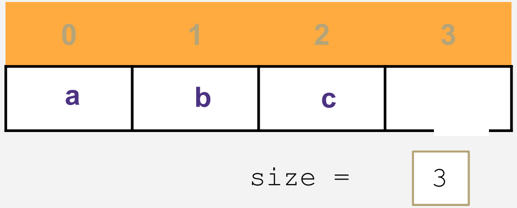
data[]
size

Behavior

get return data[index]
set data[index] = value
add data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size

`insert(element, index)` with shifting

`insert("d", 0)`



0	1	2	3
a	b	c	

size = 3

ArrayList Insert

ArrayList<E>

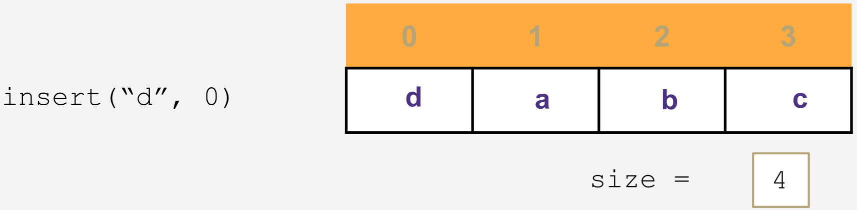
State

data[]
size

Behavior

<u>get</u> return data[index]
<u>set</u> data[index] = value
<u>add</u> data[size] = value, if out of space grow data
<u>insert</u> shift values to make hole at index, data[index] = value, if out of space grow data
<u>delete</u> shift following values forward
<u>size</u> return size

`insert(element, index)` with shifting



`insert("d", 0)`

0	1	2	3
d	a	b	c

size = 4

ArrayList Delete

ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

`insert(element, index)` with shifting

`insert("d", 0)`

0	1	2	3
d	a	b	c

size = 4

`delete(index)` with shifting

`delete(0)`

0	1	2	3
d	a	b	c

size = 4

ArrayList Delete

ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

`insert(element, index)` with shifting

`insert("d", 0)`

0	1	2	3
d	a	b	c

`size =` 4

`delete(index)` with shifting

`delete(0)`

0	1	2	3
a	b	c	c

`size =` 3

ArrayList Delete

ArrayList<E>

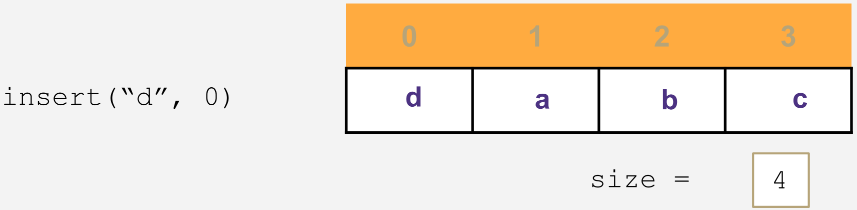
State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

`insert(element, index)` with shifting

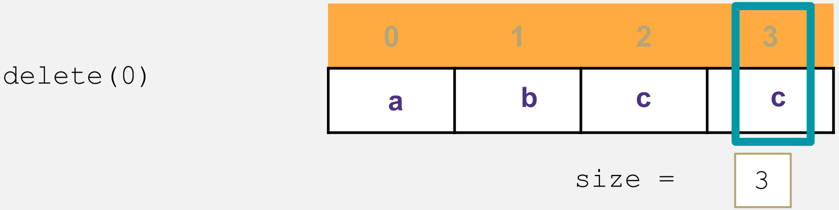


`insert("d", 0)`

0	1	2	3
d	a	b	c

size = 4

`delete(index)` with shifting



`delete(0)`

0	1	2	3
a	b	c	c

size = 3

What should we do with this?
Null? 0? -1?

ArrayList Insert - with Expansion

ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

`append(element)` with growth

0	1	2	3
10	3	4	5

`numberOfItems =` 4

ArrayList Insert - with Expansion

ArrayList<E>

State

data[]
size

Behavior

get return data[index]
set data[index] = value
add data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size

append(element) with growth

append(2)

0	1	2	3
10	3	4	5

numberOfItems = 4

What should we do?

ArrayList Insert - with Expansion

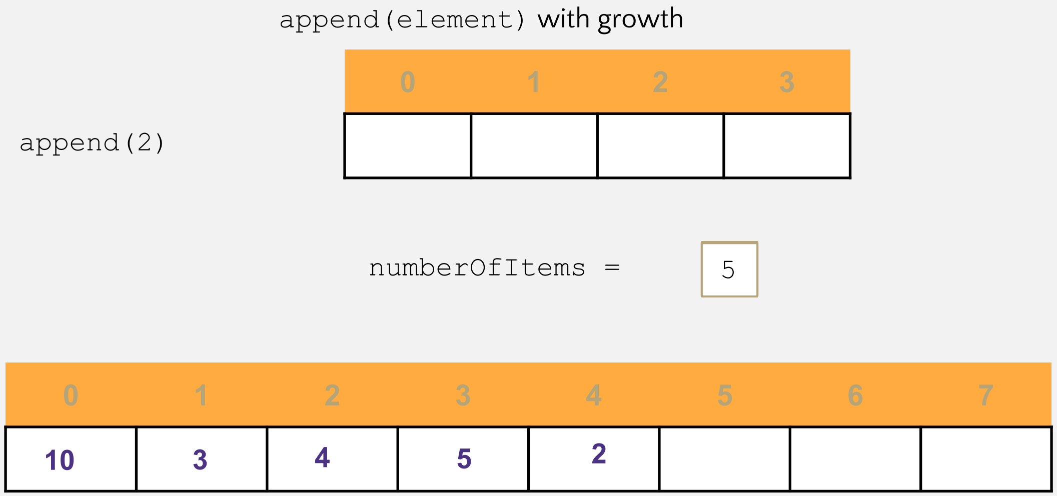
ArrayList<E>

State

```
data[]
size
```

Behavior

<u>get</u> return data[index]
<u>set</u> data[index] = value
<u>add</u> data[size] = value, if out of space grow data
<u>insert</u> shift values to make hole at index, data[index] = value, if out of space grow data
<u>delete</u> shift following values forward
<u>size</u> return size



`append(element)` with growth

`append(2)`

0	1	2	3

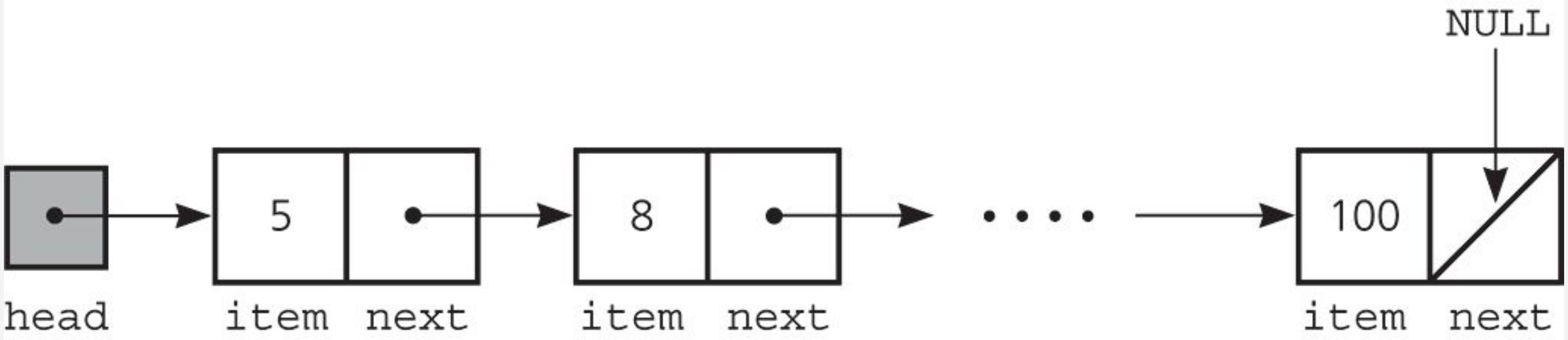
`numberOfItems =` 5

0	1	2	3	4	5	6	7
10	3	4	5	2			

سوال؟

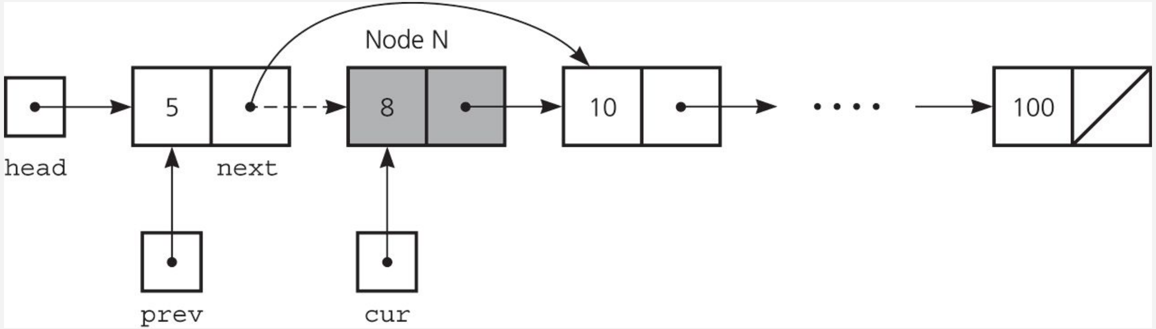
لیست پیوندی

Sample Linked List



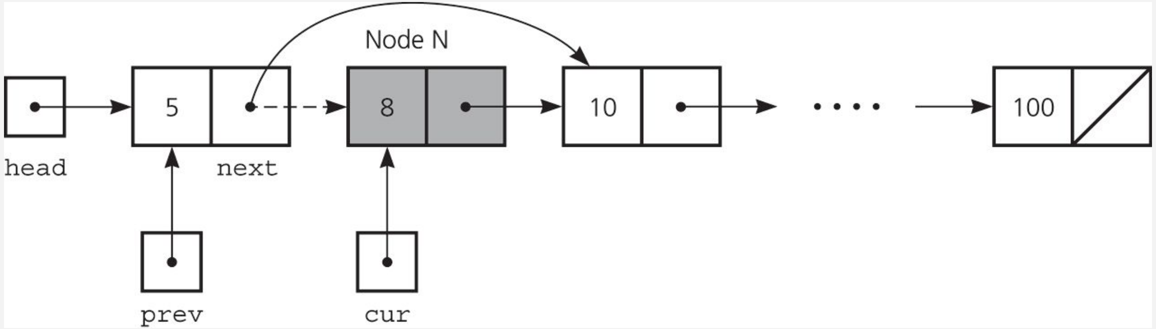
Delete from a Linked List

Deleting a node in the middle

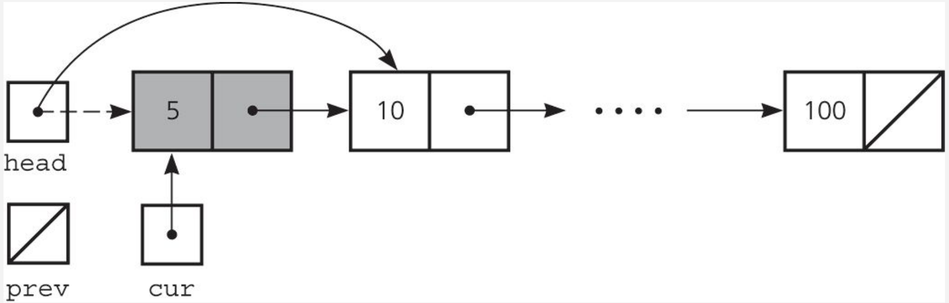


Delete from a Linked List

Deleting a node in the middle

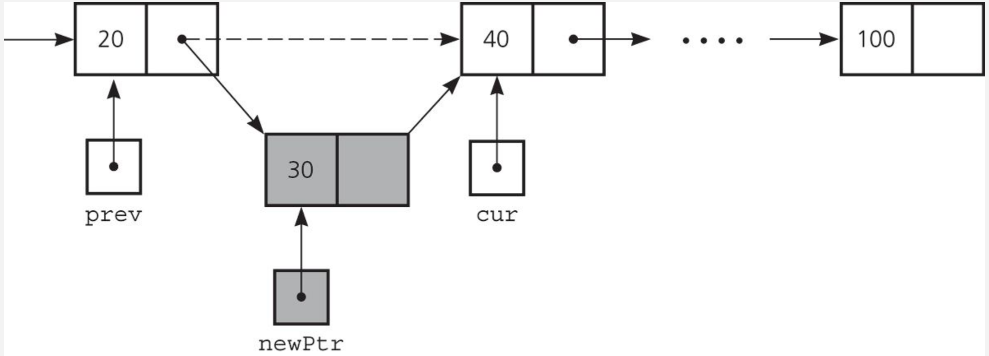


Deleting the first node



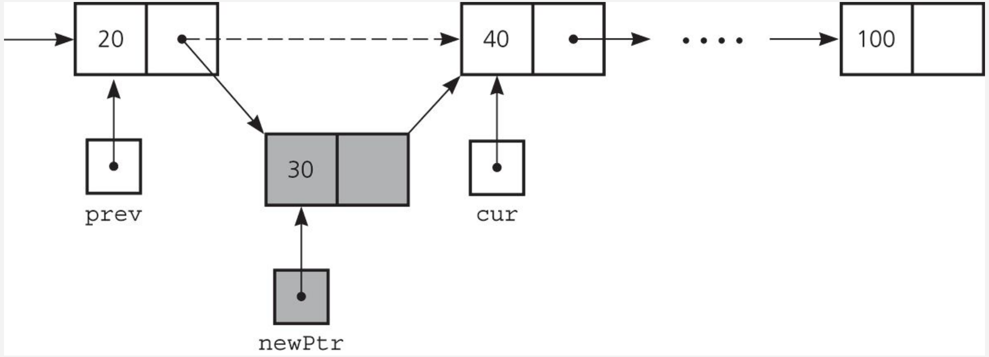
Insert into a Linked List

- Insert between two nodes

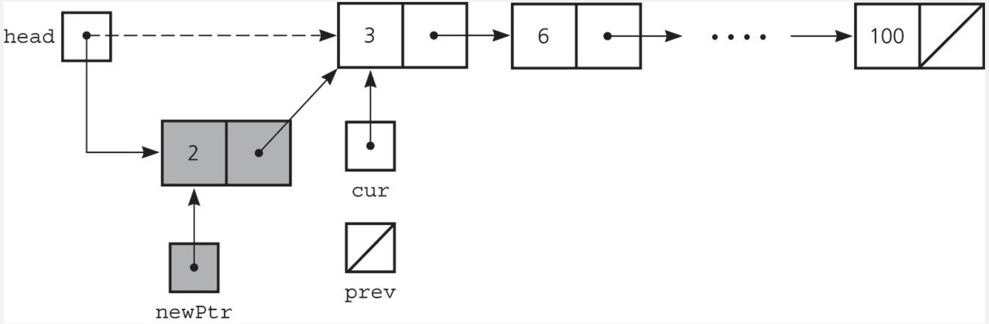


Insert into a Linked List

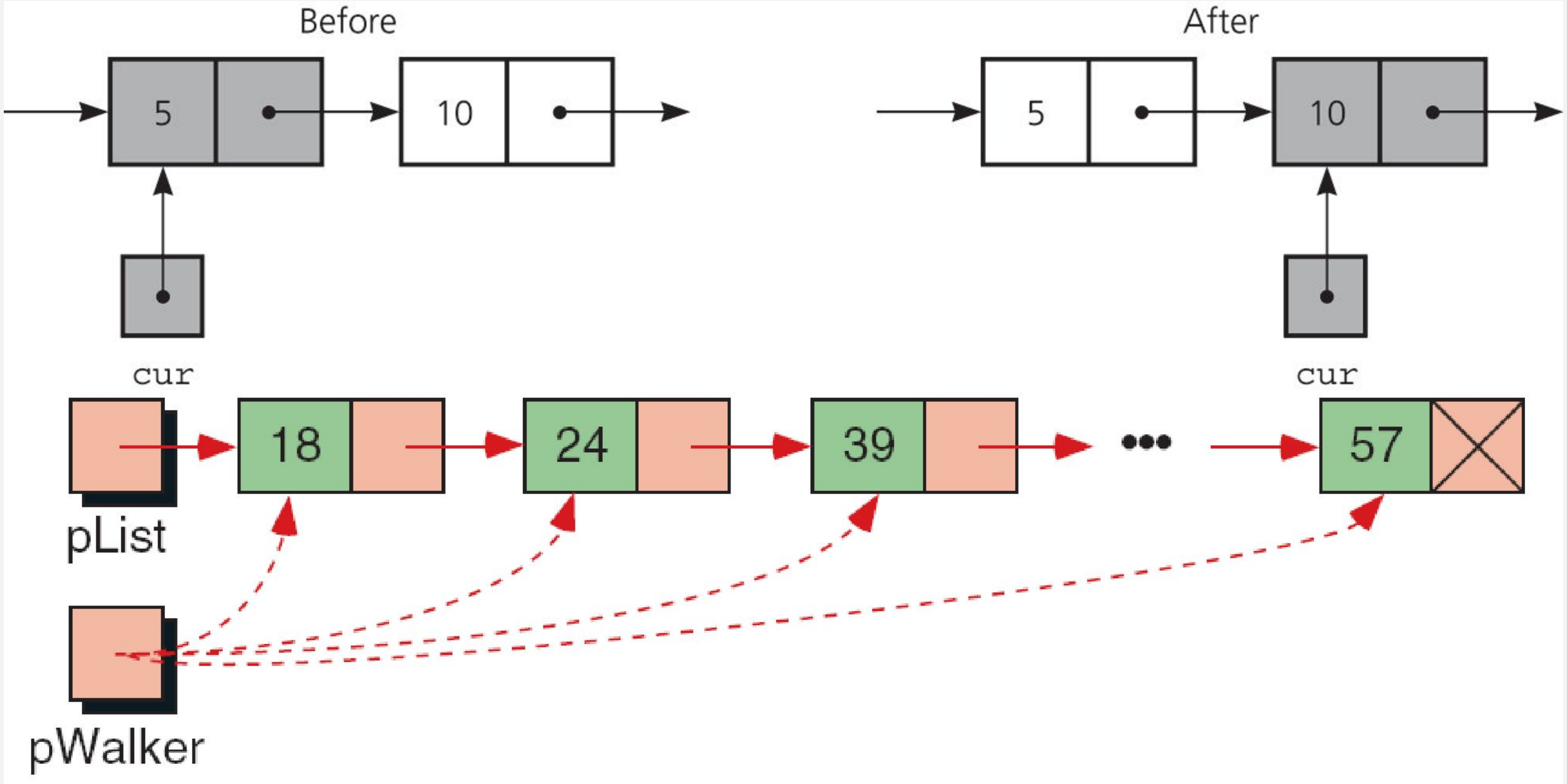
- Insert between two nodes



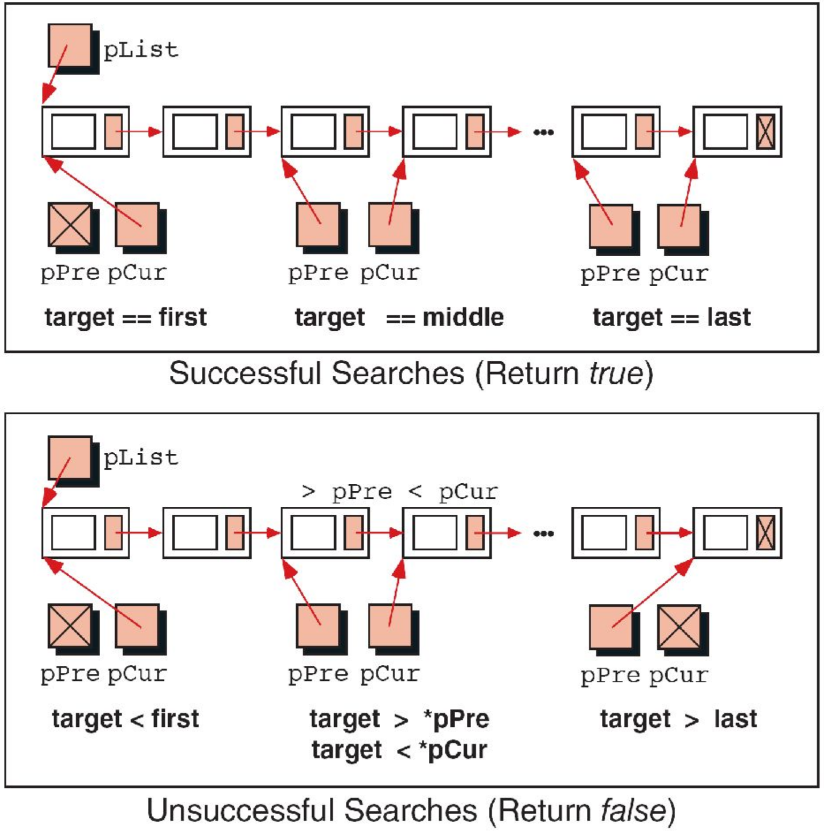
- Insert at the beginning



Traverse a Linked List



Search in Sorted Linked List

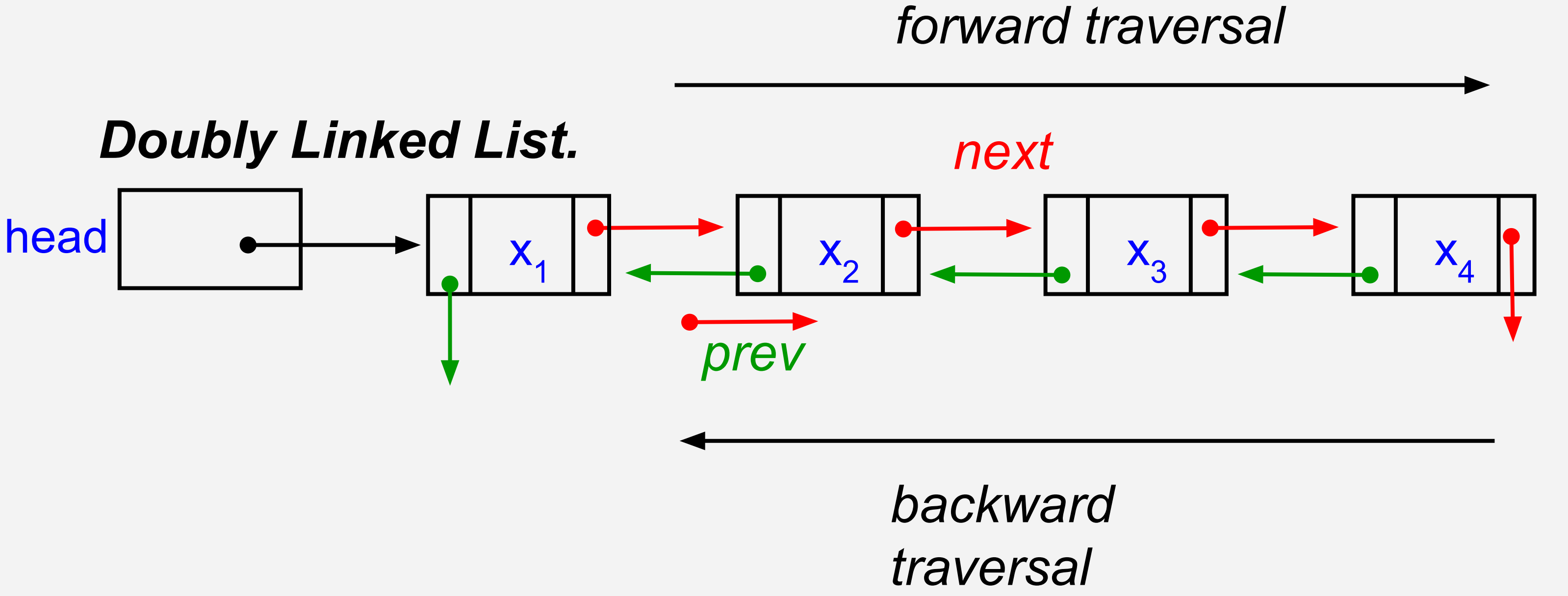


Successful Searches (Return *true*)

Unsuccessful Searches (Return *false*)

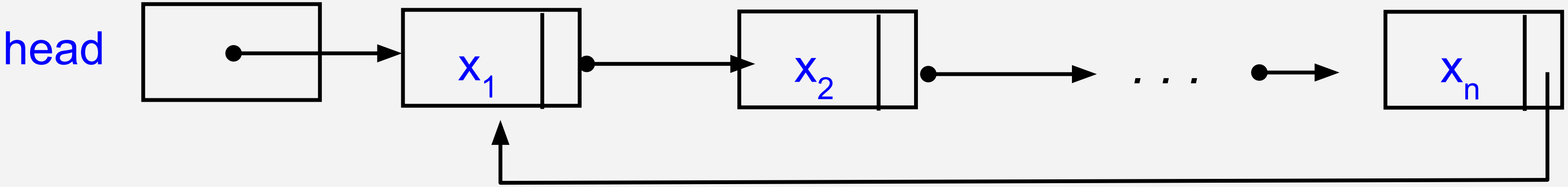
Doubly Linked Lists

- Sometimes we need to traverse a sequence in BOTH directions efficiently



Circular Linked List

- May need to cycle through a list repeatedly
    - E.g., round robin system for a shared resource



سوال؟

انتخاب ساختمان داده مناسب

Design Decisions

- For every ADT, many ways to implement
- Based on our situation we should consider:
  - Speed vs Memory Usage
  - Generic/Reusability vs Specific/Specialized
  - One Function vs Another
  - Robustness vs Performance
- Our job is selecting **the best** ADT implementation by making **the right design tradeoffs!**
  - A common topic in interview questions

Example Problem

- Akbar Joojeh is implementing a new system to manage orders
- A new order comes in -> place it at the end of orders
- Food is prepared in approximately the same order it was requested
  - Sometimes orders are fulfilled out of order

Example Problem

- Akbar Joojeh is implementing a new system to manage orders
- A new order comes in -> place it at the end of orders
- Food is prepared in approximately the same order it was requested
  - Sometimes orders are fulfilled out of order

- Let's represent tickets using the List ADT
  - What implementation should we use?
  - Why?

What implementation should we use? Why?

- ArrayList
  - Creating a new order is very fast (as long as we don't have to resize)
  - Cooks can see any given order easily
- LinkedList
  - Creating an order is slower (have to iterate through whole list)
  - We'll mostly be removing from the front of the list, which is fast because it requires no shifting

Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

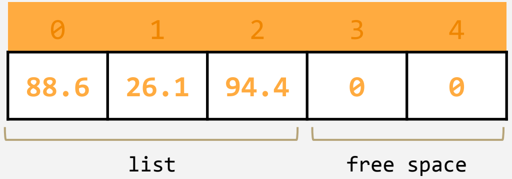
ArrayList<E>

State

```
data[]
size
```

Behavior

<u>get</u> return data[index]
<u>set</u> data[index] = value
<u>add</u> data[size] = value, if out of space grow data
<u>insert</u> shift values to make hole at index, data[index] = value, if out of space grow data
<u>delete</u> shift following values forward
<u>size</u> return size



0	1	2	3	4
88.6	26.1	94.4	0	0

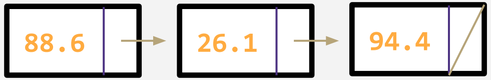
LinkedList<E>

State

```
Node front;
size
```

Behavior

<u>get</u> loop until index, return node's value
<u>set</u> loop until index, update node's value
<u>add</u> create new node, update next of last node
<u>insert</u> create new node, loop until index, update next fields
<u>delete</u> loop until index, skip node
<u>size</u> return size



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

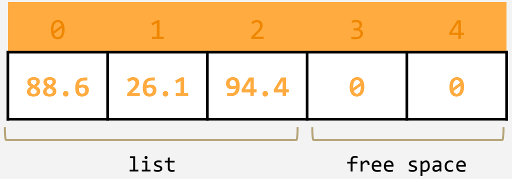
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



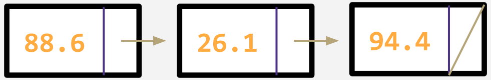
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node’s
value
set loop until index, update node’s
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

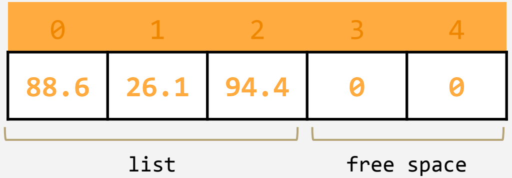
ArrayList<E>

State

data[]
size

Behavior

get return data[index]
set data[index] = value
add data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size



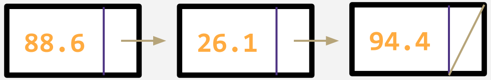
LinkedList<E>

State

Node front;
size

Behavior

get loop until index, return node’s value
set loop until index, update node’s value
add create new node, update next of last node
insert create new node, loop until index, update next fields
delete loop until index, skip node
size return size



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

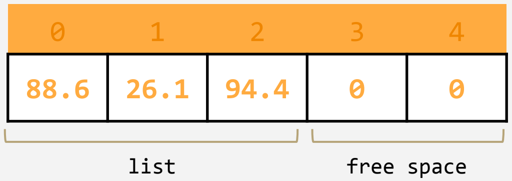
LinkedList<E>

State

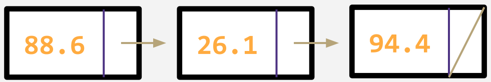
```
Node front;
size
```

Behavior

```
get loop until index, return node's
value
set loop until index, update node's
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



0	1	2	3	4
88.6	26.1	94.4	0	0



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

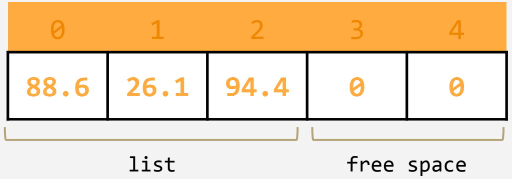
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



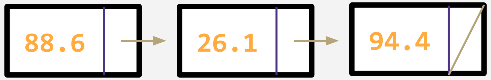
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node's
value
set loop until index, update node's
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

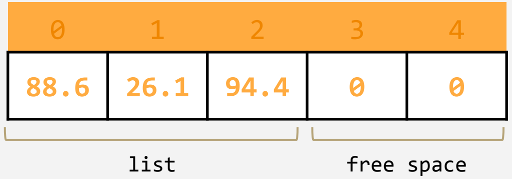
ArrayList<E>

State

```
data[]
size
```

Behavior

<u>get</u> return data[index]
<u>set</u> data[index] = value
<u>add</u> data[size] = value, if out of space grow data
<u>insert</u> shift values to make hole at index, data[index] = value, if out of space grow data
<u>delete</u> shift following values forward
<u>size</u> return size



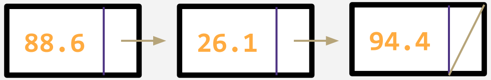
LinkedList<E>

State

```
Node front;
size
```

Behavior

<u>get</u> loop until index, return node’s value
<u>set</u> loop until index, update node’s value
<u>add</u> create new node, update next of last node
<u>insert</u> create new node, loop until index, update next fields
<u>delete</u> loop until index, skip node
<u>size</u> return size



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

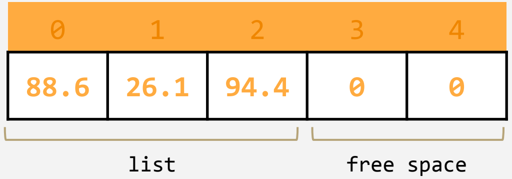
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



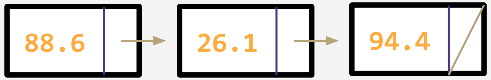
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node's
value
set loop until index, update node's
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

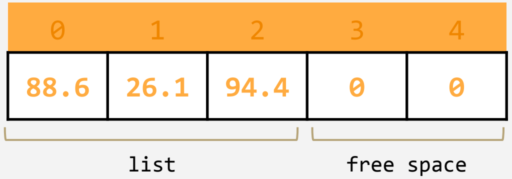
ArrayList<E>

State

data[]
size

Behavior

get return data[index]
set data[index] = value
add data[size] = value, if out of space grow data
insert shift values to make hole at index, data[index] = value, if out of space grow data
delete shift following values forward
size return size



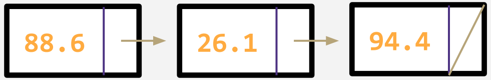
LinkedList<E>

State

Node front;
size

Behavior

get loop until index, return node’s value
set loop until index, update node’s value
add create new node, update next of last node
insert create new node, loop until index, update next fields
delete loop until index, skip node
size return size



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

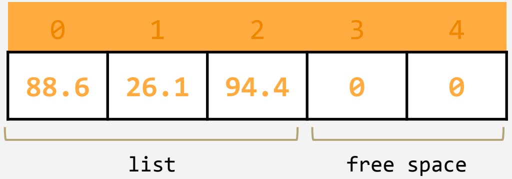
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



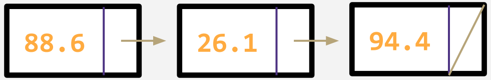
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node’s
value
set loop until index, update node’s
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

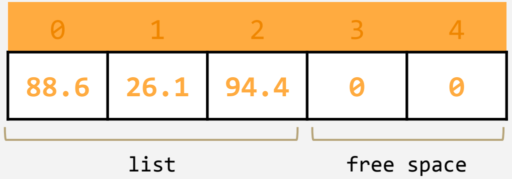
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



0	1	2	3	4
88.6	26.1	94.4	0	0

list

free space

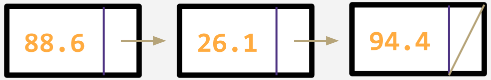
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node's
value
set loop until index, update node's
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

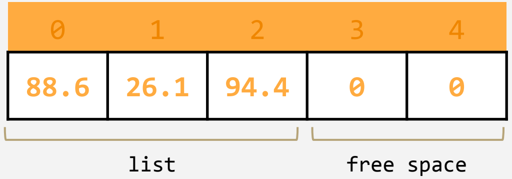
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



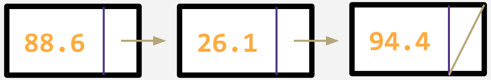
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node’s
value
set loop until index, update node’s
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```

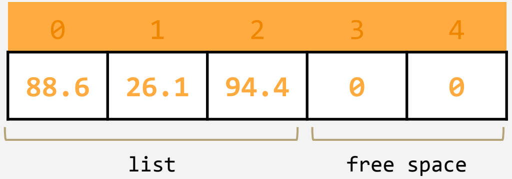
LinkedList<E>

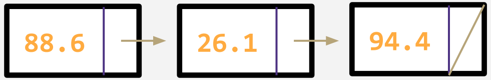
State

```
Node front;
size
```

Behavior

```
get loop until index, return node’s
value
set loop until index, update node’s
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```





Comparing ADT Implementations: List

	ArrayList	LinkedList
add (front)	linear	constant
remove (front)	linear	constant
add (back)	(usually) constant	linear
remove (back)	constant	linear
get	constant	linear
put	linear	linear

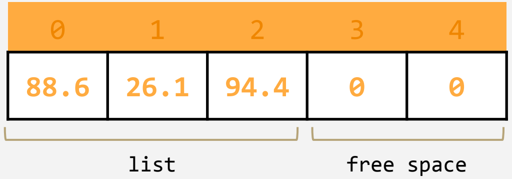
ArrayList<E>

State

```
data[]
size
```

Behavior

```
get return data[index]
set data[index] = value
add data[size] = value, if out
of space grow data
insert shift values to make hole
at index, data[index] = value,
if out of space grow data
delete shift following values
forward
size return size
```



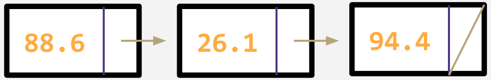
LinkedList<E>

State

```
Node front;
size
```

Behavior

```
get loop until index, return node's
value
set loop until index, update node's
value
add create new node, update next of
last node
insert create new node, loop until
index, update next fields
delete loop until index, skip node
size return size
```



Comparing ADT Implementations: List

	ArrayList	LinkedList
`add (front)`	linear	constant
`remove (front)`	linear	constant
`add (back)`	(usually) constant	linear
`remove (back)`	constant	linear
`get`	constant	linear
`put`	linear	linear

- Important to be able to come up with this, and understand why
- But only half the story: to be able to make a design decision, need the context to understand which of these we should prioritize

Conclusion

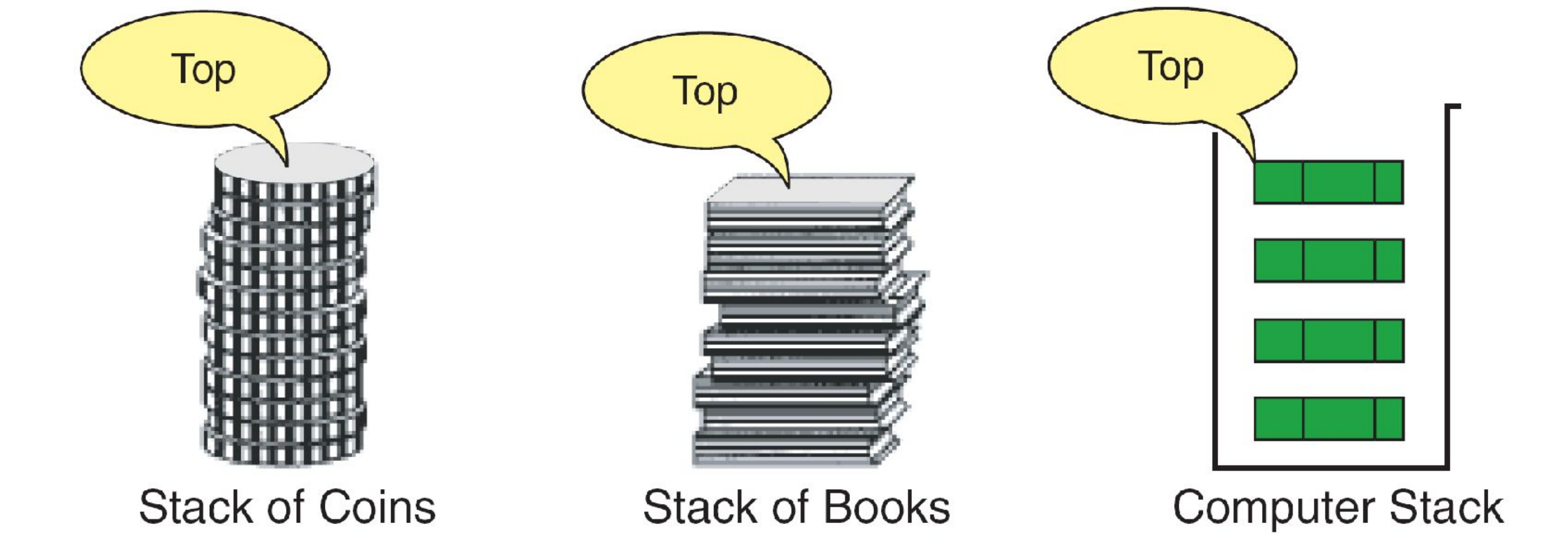
- Both ArrayList and LinkedList have pros and cons
  - Neither is strictly better than the other
- The Design Decision process:
  - Evaluate pros and cons
  - Decide on a design
  - Defend your design decision
- This is a major objective of the course!

سوال؟

پشته

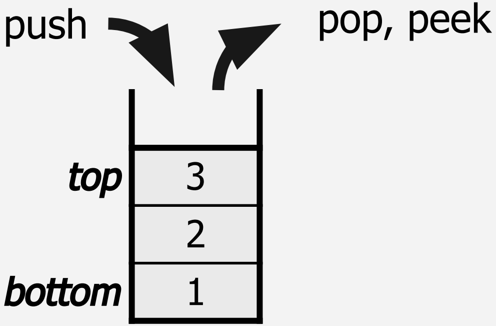
مجموعه ای از اشیاء روی هم قرار گرفته

Idea



Stack ADT

- An ADT representing an ordered sequence of elements
- Elements can only be added & removed from one end.
- Last-In, First-Out (LIFO)
- Elements stored in order of insertion
    - Don’t think of them as having indices
- Clients can only add/remove/examine the “top”



STACK ADT

State

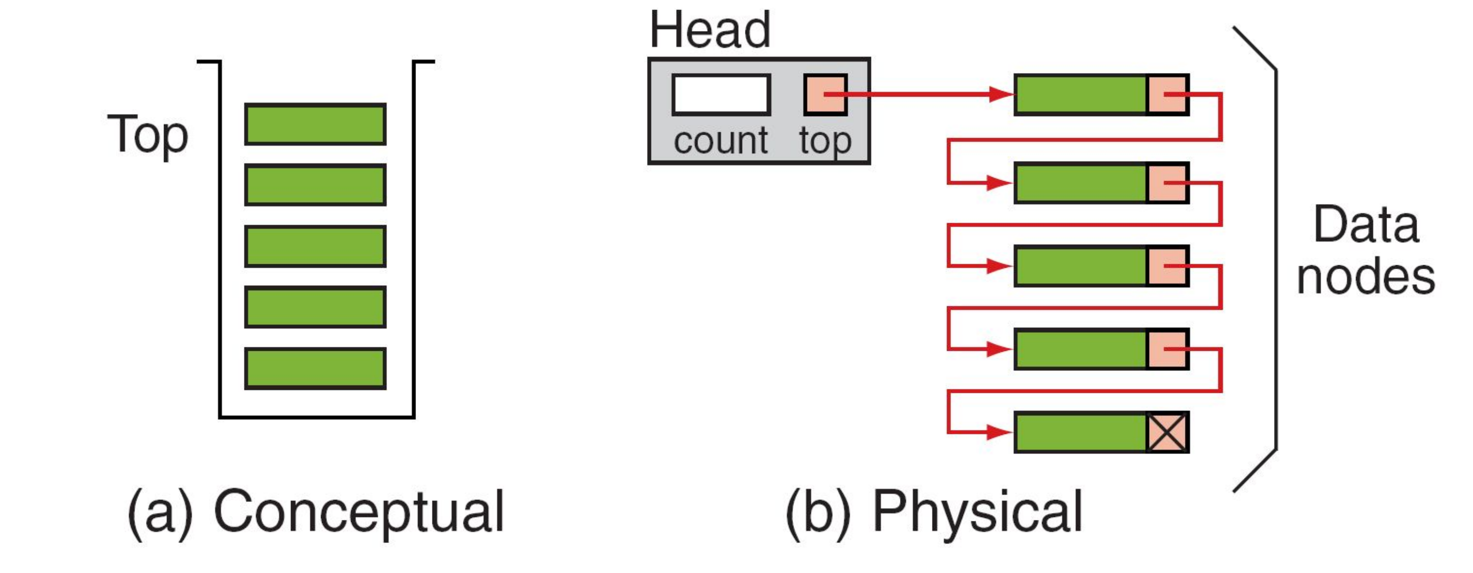
Collection of ordered items
Count of items

Behavior

`push(index)` add item to top
`pop()` return & remove item at top
`peek()` return item at top
`size()` count of items
`isEmpty()` is count 0?

پیاده سازی پشته با لیست پیوندی

Implementing Stack with Linked List



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

`push(index)` add item to top
`pop()` return & remove item at top
`peek()` return item at top
`size()` count of items
`isEmpty()` is count 0?

LinkedStack<E>

State

`Node top`
`size`

Behavior

`push` add new node at top
`pop` return & remove node at top
`peek` return node at top
`size` return size
`isEmpty` return size == 0

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

LinkedStack<E>

State

Node top
size

Behavior

push add new node at top
pop return & remove node at top
peek return node at top
size return size
isEmpty return size == 0

top

size = 0

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

LinkedStack<E>

State

```
Node top
size
```

Behavior

```
push add new node at top
pop return & remove node at
top
peek return node at top
size return size
isEmpty return size == 0
```

```
push(3)
```

top

size = 0

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

`push(index)` add item to top
`pop()` return & remove item at top
`peek()` return item at top
`size()` count of items
`isEmpty()` is count 0?

LinkedStack<E>

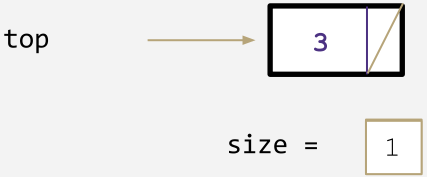
State

`Node top`
`size`

Behavior

`push` add new node at top
`pop` return & remove node at top
`peek` return node at top
`size` return size
`isEmpty` return size == 0

`push(3)`



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

`push(index)` add item to top
`pop()` return & remove item at top
`peek()` return item at top
`size()` count of items
`isEmpty()` is count 0?

LinkedStack<E>

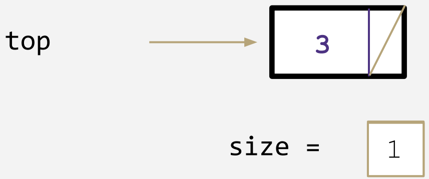
State

`Node top`
`size`

Behavior

`push` add new node at top
`pop` return & remove node at top
`peek` return node at top
`size` return size
`isEmpty` return size == 0

```
push(3)
push(4)
```



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

LinkedStack<E>

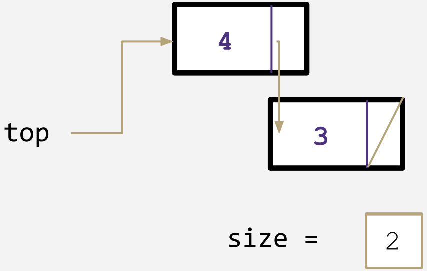
State

Node top
size

Behavior

push add new node at top
pop return & remove node at top
peek return node at top
size return size
isEmpty return size == 0

```
push(3)
push(4)
```



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

LinkedStack<E>

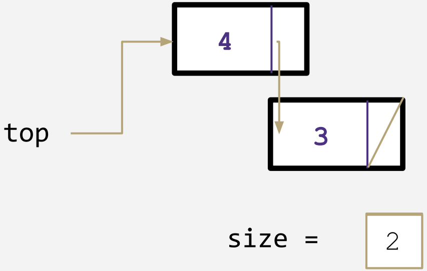
State

Node top
size

Behavior

push add new node at top
pop return & remove node at top
peek return node at top
size return size
isEmpty return size == 0

```
push(3)
push(4)
pop()
```



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

LinkedStack<E>

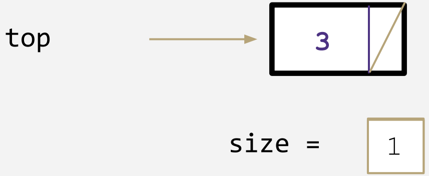
State

Node top
size

Behavior

push add new node at top
pop return & remove node at top
peek return node at top
size return size
isEmpty return size == 0

```
push(3)
push(4)
pop()
```



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

<u>push(index)</u> add item to top
<u>pop()</u> return & remove item at top
<u>peek()</u> return item at top
<u>size()</u> count of items
<u>isEmpty()</u> is count 0?

LinkedStack<E>

State

```
Node top
size
```

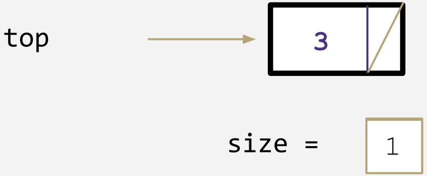
Behavior

<u>push</u> add new node at top
<u>pop</u> return & remove node at top
<u>peek</u> return node at top
<u>size</u> return size
<u>isEmpty</u> return size == 0

Big-Oh Analysis

```
pop()
peek()
size()
isEmpty()
push()
```

```
push(3)
push(4)
pop()
```



top

3

size = 1

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

LinkedStack<E>

State

Node top
size

Behavior

push add new node at top
pop return & remove node at top
peek return node at top
size return size
isEmpty return size == 0

Big-Oh Analysis

pop() O(1) Constant

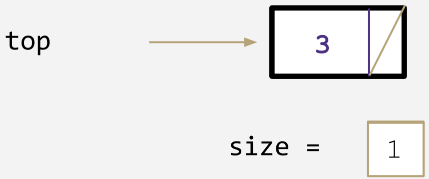
peek()

size()

isEmpty()

push()

push(3)
push(4)
pop()



top

3

size = 1

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

<u>push(index)</u> add item to top
<u>pop()</u> return & remove item at top
<u>peek()</u> return item at top
<u>size()</u> count of items
<u>isEmpty()</u> is count 0?

LinkedStack<E>

State

`Node top`
`size`

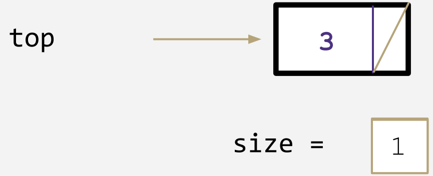
Behavior

<u>push</u> add new node at top
<u>pop</u> return & remove node at top
<u>peek</u> return node at top
<u>size</u> return size
<u>isEmpty</u> return size == 0

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	
isEmpty()	
push()	

```
push(3)
push(4)
pop()
```



Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

LinkedStack<E>

State

Node top
size

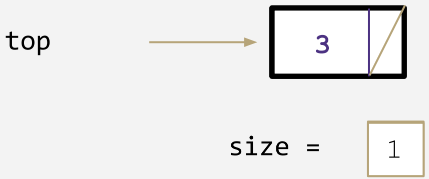
Behavior

push add new node at top
pop return & remove node at top
peek return node at top
size return size
isEmpty return size == 0

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	
push()	

```
push(3)
push(4)
pop()
```



top

3

size = 1

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

<u>push(index)</u> add item to top
<u>pop()</u> return & remove item at top
<u>peek()</u> return item at top
<u>size()</u> count of items
<u>isEmpty()</u> is count 0?

LinkedStack<E>

State

Node top
size

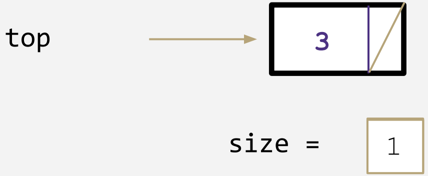
Behavior

<u>push</u> add new node at top
<u>pop</u> return & remove node at top
<u>peek</u> return node at top
<u>size</u> return size
<u>isEmpty</u> return size == 0

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	O(1) Constant
push()	

push(3)
push(4)
pop()



top

3

size = 1

Implementing Stack with Linked List

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

LinkedStack<E>

State

```
Node top
size
```

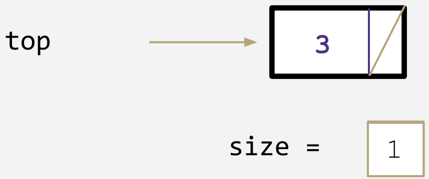
Behavior

```
push add new node at top
pop return & remove node at
top
peek return node at top
size return size
isEmpty return size == 0
```

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	O(1) Constant
push()	O(1) Constant

```
push(3)
push(4)
pop()
```



سوال؟

پیاده سازی پشته با آرایه

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

ArrayStack<E>

State

```
data[]
size
```

Behavior

```
push data[size] = value, if
out of room grow data
pop return data[size - 1],
size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

0	1	2	3

size = 0

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

ArrayStack<E>

State

```
data[]
size
```

Behavior

```
push data[size] = value, if
out of room grow data
pop return data[size - 1],
size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

`push(3)`

0	1	2	3

`size =` 0

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

push(3)

0	1	2	3
3			

size = 1

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

```
push(3)
push(4)
```

0	1	2	3
3			

size = 1

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

ArrayStack<E>

State

```
data[]
size
```

Behavior

```
push data[size] = value, if
out of room grow data
pop return data[size - 1],
size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

```
push(3)
push(4)
```

0	1	2	3
3	4		

```
size = 2
```

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

ArrayStack<E>

State

```
data[]
size
```

Behavior

```
push data[size] = value, if
out of room grow data
pop return data[size - 1],
size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

```
push(3)
push(4)
pop()
```

0	1	2	3
3	4		

`size =` 2

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

```
push(3)
push(4)
pop()
```

0	1	2	3
3			

size = 1

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3			

size = 1

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

ArrayStack<E>

State

```
data[]
size
```

Behavior

```
push data[size] = value, if
out of room grow data
pop return data[size - 1],
size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

Big-Oh Analysis

```
pop()
peek()
size()
isEmpty()
push()
```

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

Big-Oh Analysis

pop()	O(1) Constant
peek()	
size()	
isEmpty()	
push()	

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	
isEmpty()	
push()	

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	
push()	

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

`push(index)` add item to top
`pop()` return & remove item at top
`peek()` return item at top
`size()` count of items
`isEmpty()` is count 0?

ArrayStack<E>

State

`data[]`
`size`

Behavior

`push` data[size] = value, if out of room grow data
`pop` return data[size - 1], size -= 1
`peek` return data[size - 1]
`size` return size
`isEmpty` return size == 0

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	O(1) Constant
push()	

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

```
push(index) add item to top
pop() return & remove item
at top
peek() return item at top
size() count of items
isEmpty() is count 0?
```

ArrayStack<E>

State

```
data[]
size
```

Behavior

```
push data[size] = value, if
out of room grow data
pop return data[size - 1],
size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0
```

```
push(3)
push(4)
pop()
push(5)
```

0	1	2	3
3	5		

size = 2

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	O(1) Constant
push()	What is your guess?

Implementing a Stack with an Array

STACK ADT

State

Collection of ordered items
Count of items

Behavior

push(index) add item to top
pop() return & remove item at top
peek() return item at top
size() count of items
isEmpty() is count 0?

ArrayStack<E>

State

data[]
size

Behavior

push data[size] = value, if out of room grow data
pop return data[size - 1], size -= 1
peek return data[size - 1]
size return size
isEmpty return size == 0

Big-Oh Analysis

pop()	O(1) Constant
peek()	O(1) Constant
size()	O(1) Constant
isEmpty()	O(1) Constant
push()	O(n) linear if you have to resize, O(1) otherwise

سوال؟

کاربردهای پشته

نمونه هایی از حل مسئله با استفاده از پشته

Line Editing

- A line editor
- Place characters read into a buffer
  - May use a backspace symbol (denoted by ←) to do error correction
- Goal: Calculate the final text (corrected) and print it in reverse

Input : abc_defgh←2klpqr←←wxyz

Corrected Input : abc_defg2klpwxyz

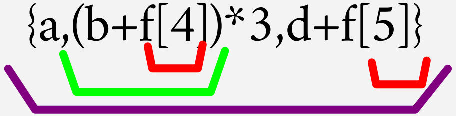
Reversed Output : zyxwplk2gfed_cba

Line Editing: Solution

- Initialize a new stack
- For each character read:
    - If it is a backspace, pop out last char entered
    - If not a backspace, push the char into stack
- To print in reverse, pop out each char for output

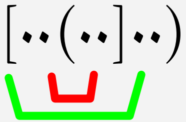
Bracket Matching Problem

- Ensures that pairs of brackets are properly matched
- An Example:



{a,(b+f[4])*3,d+f[5]}

- Bad Examples:
  - (..)..) // too many closing brackets
  - (..(..) // too many open brackets
  - [..(..]..) // mismatched brackets



Bracket Matching Problem: Solution

- Initialize the stack to empty
- For every char read
    - If open bracket then push onto stack
    - If close bracket, then
        - Return & remove most recent item from the stack
        - If doesn’t match then flag error
    - If non-bracket, skip the char read

سوال؟