CSE 373: Data Structures and Algorithms

Testing and Debugging

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Administrivia

Due dates

- Homework 1: Today 11:59pm
- Homework 2 Partner Form: Today 11:59pm

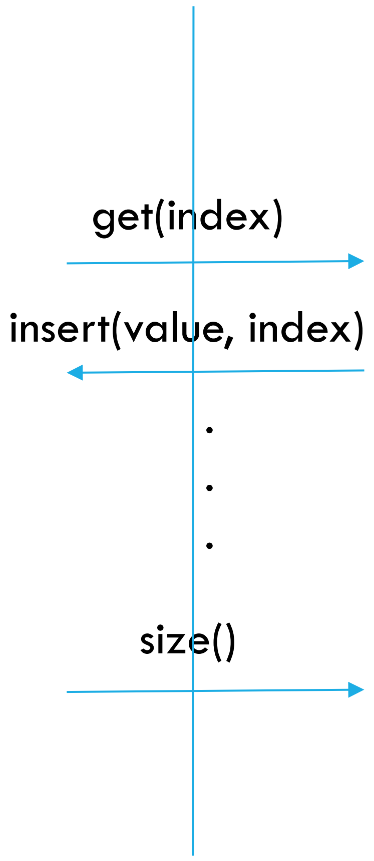
Notes:

- If you want us to assign you a partner, fill HW2 Partner Pool Form today by 8pm
- Homework 2 to go out today
- If you cannot login into gitlab.cs.washington.edu, email cse373-staff@cs.washington.edu ASAP

Case Study: The List ADT

Implementor

- get(index)
- set(value, index)
- append(value)
- insert(value, index)
- delete(index)
- size()



Client

AL LL

How do we print all the items in the list?

```java
for (int i = 0; i < myList.size(); i++) {
    System.out.println(myList.get(i));
}
```

N N

1 N

How efficient is the above code?

$O(N)$ $O(N^2)$

Case Study: The List ADT

list: stores an ordered sequence of information.

- Each item is accessible by an index.
- Lists have a variable size as items can be added and removed

Supported Operations:

- **get(index):** returns the item at the given index
- **set(value, index):** sets the item at the given index to the given value
- **append(value):** adds the given item to the end of the list
- **insert(value, index):** insert the given item at the given index maintaining order
- **delete(index):** removes the item at the given index maintaining order
- **size():** returns the number of elements in the list
- **Iterator()**: returns an iterator over the list

The Iterator ADT

An Iterator “wraps” some sequence.

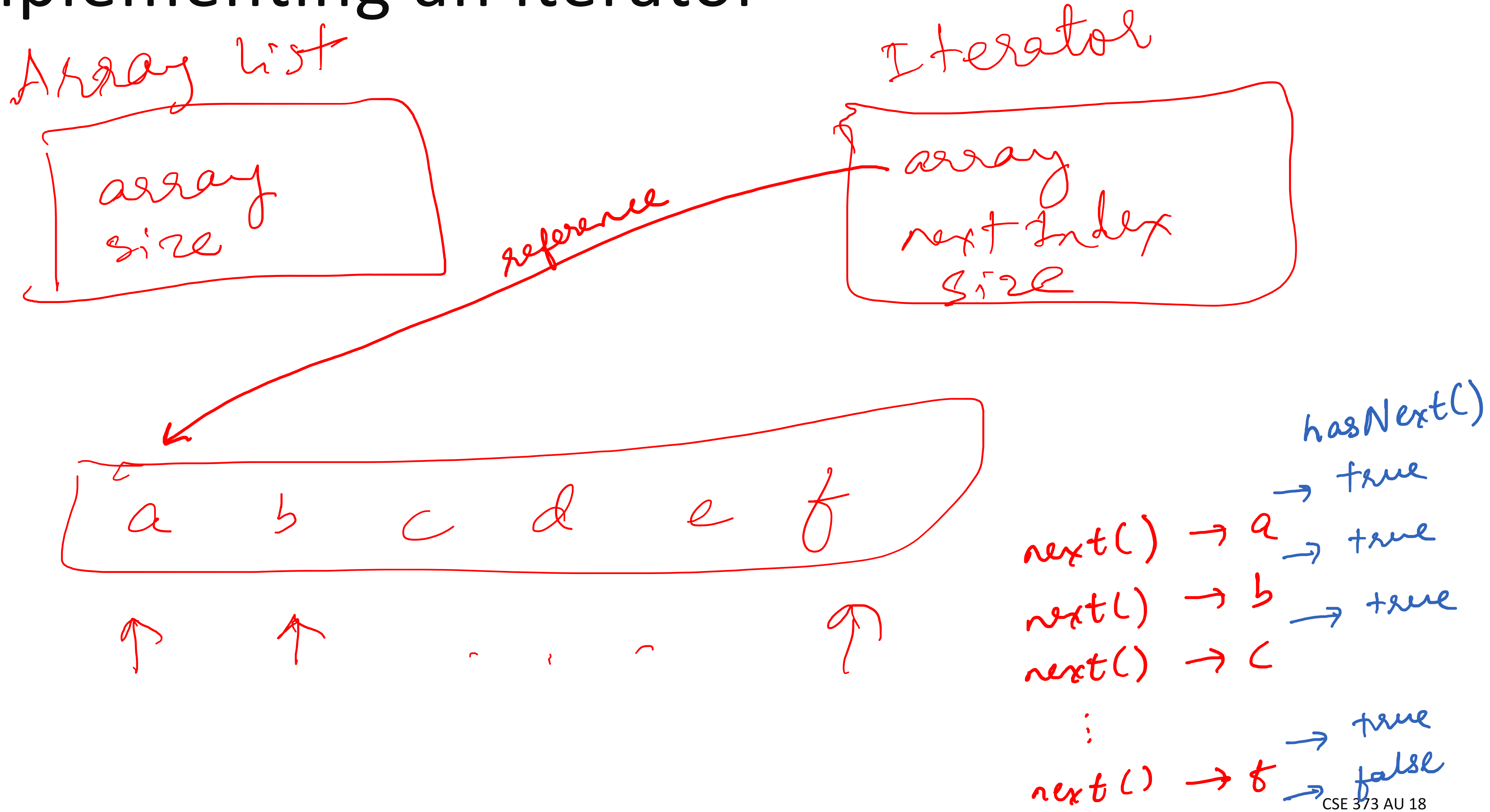
It yields each subsequent element one by one on request.

An iterator “remembers” what it needs to yield next.

Supported operations:

- **hasNext():** returns ‘true’ if there is another element left to yield and ‘false‘ otherwise
- **next():** returns the next element (if there is one)

Implementing an iterator



Testing

Testing

Computers don't make mistakes- people do!

"I'm almost done, I just need to make sure it works"
– Naive 14Xers

Software Test: a separate piece of code that exercises the code you are assessing by providing input to your code and finishes with an assertion of what the result should be.

1. Isolate

break your code into small modules

2. Build in increments

Make a plan from simplest to most complex cases

3. Test as you go

As your code grows, so should your tests

Types of Tests

Black Box

- Behavior only – ADT requirements
- From an outside point of view
- Does your code uphold its contracts with its users?
- Performance/efficiency

White Box

- Includes an understanding of the implementation
- Written by the author as they develop their code
- Break apart requirements into smaller steps
- “unit tests” break implementation into single assertions

What to test?

Expected behavior

- The main use case scenario
- Does your code do what it should given friendly conditions?

Forbidden Input

- What are all the ways the user can mess up?

Empty/Null

- Protect yourself!
- How do things get started?

Boundary/Edge Cases

- First
- last

Scale

- Is there a difference between 10, 100, 1000, 10000 items?

Tips for testing

You cannot test every possible input, parameter value, etc.

- Think of a limited set of tests likely to expose bugs.

Think about boundary cases

- Positive; zero; negative numbers
- Right at the edge of an array or collection's size

Think about empty cases and error cases

- 0, -1, null; an empty list or array

test behavior in combination

- Maybe `add` usually works, but fails after you call `remove`
- Make multiple calls; maybe `size` fails the second time only

Thought Experiment

Discuss with your neighbors: Imagine you are writing an implementation of the List interface that stores integers in an Array. What are some ways you can assess your program's correctness in the following cases:

Expected Behavior
- Create a new list
- Add some amount of items to it
- Remove a couple of them

Forbidden Input
- Add a negative number
- Add duplicates
- Add extra large numbers

Empty/Null
- Call remove on an empty list
- Add to a null list
- Call size on an null list

Boundary/Edge Cases
- Add 1 item to an empty list
- Set an item at the front of the list
- Set an item at the back of the list

Scale
- Add 1000 items to the list
- Remove 100 items in a row
- Set the value of the same item 50 times

JUnit

JUnit: a testing framework that works with IDEs to give you a special GUI experience when testing your code

other examples: @Override

```
@Test
```

Annotation

```
public void myTest() {
    Map<String, Integer> basicMap = new LinkedListDict<String, Integer>();
    basicMap.put("Kasey", 42);
    assertEquals(42, basicMap.get("Kasey"));
}
```

expected value

actual value

Assertions:

- assertEquals(item1, item2)
- assertTrue(Boolean expression)
- assertFalse(bollean expression)
- assertNotNull(item)

More: https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Write Tests for our Dictionary

Debugger