Data Structures

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Overview of Lists
- Ordered List
- Contiguous data representations (array-based list)

Lists

List

"Container". Holds other objects.

List (ordered is the default)

***A list in which the order of items matters*;**

[20, 30, 10, 40] is different from [30, 40, 20, 10].

Unordered list

***A list in which data items are placed in no particular order*;**
[20, 30, 10, 40] is equal to [30, 40, 20, 10].

Sorted list

A list that is sorted by the value in the key; there is a semantic relationship among the keys of the items in the list. For example, [10, 20, 30, 40]

Linear relationship

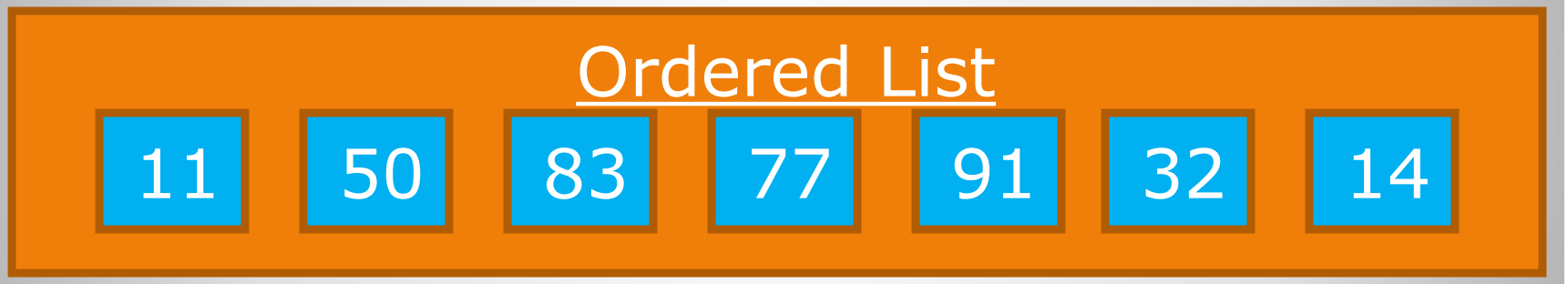
- Each element except the first has a **unique predecessor**
- Each element except the last has a **unique successor**

Length

The number of items in a list; the length can vary over time

Lists

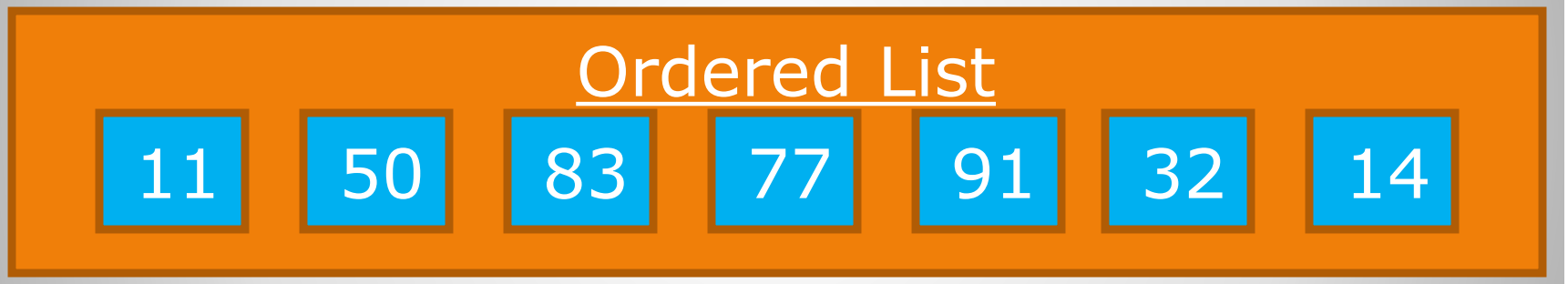
- Here is an ordered list.



1. How did the elements get into the list?
2. Why are they in this particular order?

Ordered List

- Inserted the items in the following order: 11, 50, 83, 77, 91, 32, 14



1. Which element is at the start of the list?
2. Which element is at the end of the list?

Ordered List

Abstract Data Type (ADT)

A data type whose properties (domain and operations) are specified independently of any particular implementation

What operations should we provide for our ordered list ADT???

Ordered List ADT

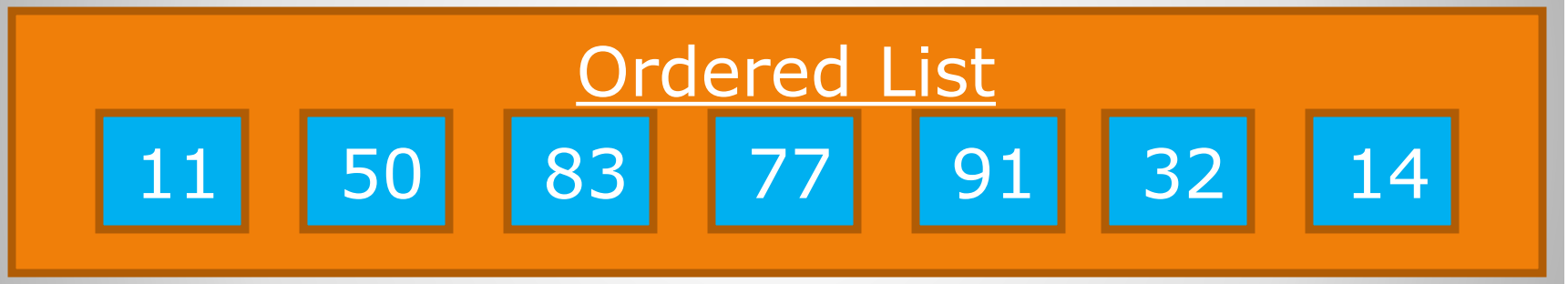
List Interface

- Here is the List Interface we will be using:

```
public interface List {
    public void insertItem(int item);
    public void deleteItem(int target);
    public boolean hasItem(int target);
    public int retrieveItem(int target) throws Exception;
    public void makeEmpty();
    public boolean isFull();
    public int getLength();
}
```

Note: Java has it own predefined List interface, but it is more complicated, so we are using our own version.

- How are the items ACTUALLY stored?



Ordered List Implementation

Contiguous Data Representation

- **Definition of *contiguous***
  - Being in actual contact : touching along a boundary or at a point.
  - Definition taken from: https://www.merriam-webster.com/dictionary/contiguous

- Data is allocated in a block (for example an array).
- The data are all stored next to each other in this block.

Contiguous Data Representation

- Elements can be stored in an array.
- For example:

What if we insert another?
What do we need to do?

Ordered List – Array Based

- You could resize the array every time you added an element.
- For example: insertItem(22)

What else could we do?

Ordered List

11	50	83	77	91	32	14	22
0	1	2	3	4	5	6	7

- Array can now store eight elements.
- 22 is in the new slot.

Ordered List – Array Based

- Hotel
- Are all rooms in a hotel always occupied by customers?
- Is the following always true:

# of rooms = # of rooms occupied by customers?

Hotel Room Usage

- The hotel does not build a new room when a customer arrives.
- The hotel does not destroy a room when a customer leaves.
- They just keep track of the rooms that are being used.
- Most of the time:

# of rooms != # of rooms occupied by customers

Hotel Room Usage

- Max Rooms – The total number of rooms in the hotel.
- Occupied Rooms – The number of rooms that are actually being used.

Hotel Room Usage

- The array elements are like rooms in a hotel.
- Make the array larger than the number of items being storing (allows for growth).
- We can just keep track of which ones are being used.
- Max = 10 (for this example)
- Occupied = 8 (first eight elements are being used)

What happens if we add an item?

Ordered List – Array Based

- Adding item
- Increment length
- Put data in that new element

44 was added to the list. It is put in the next unoccupied room.

Ordered List – Array Based

- Array-based **private** members

```
class OrderedList implements List {
    Declare int length
    Declare int[] info

    // Public members go here…
}
```

Ordered List

- **What should the OrderedList constructor do?**

Ordered List - Constructor

- **What should the OrderedList constructor do?**

OrderedList Constructor

Set length to 0

Set info to new array of int instance

Ordered List - Constructor

```
boolean isFull()
  return (length equals info.length);
```

```
makeEmpty()
   Set length to 0
```

Logically clear the data. No need to do anything with the array (do not destroy the "hotel" rooms)

Ordered List – isFull and makeEmpty

```
insertItem(int item)
   Set info[length] to item
   Increment length
```

Note: In practice, you should have code to make sure that list is not full before adding item (left of sample code to make it as simple as possible)

What is going on with this code?

Ordered List - insertItem

- Run the following code:
  Declare ol as OrderedList
  // Other code to fill the ordered list goes here…
  ol.insertItem(44)

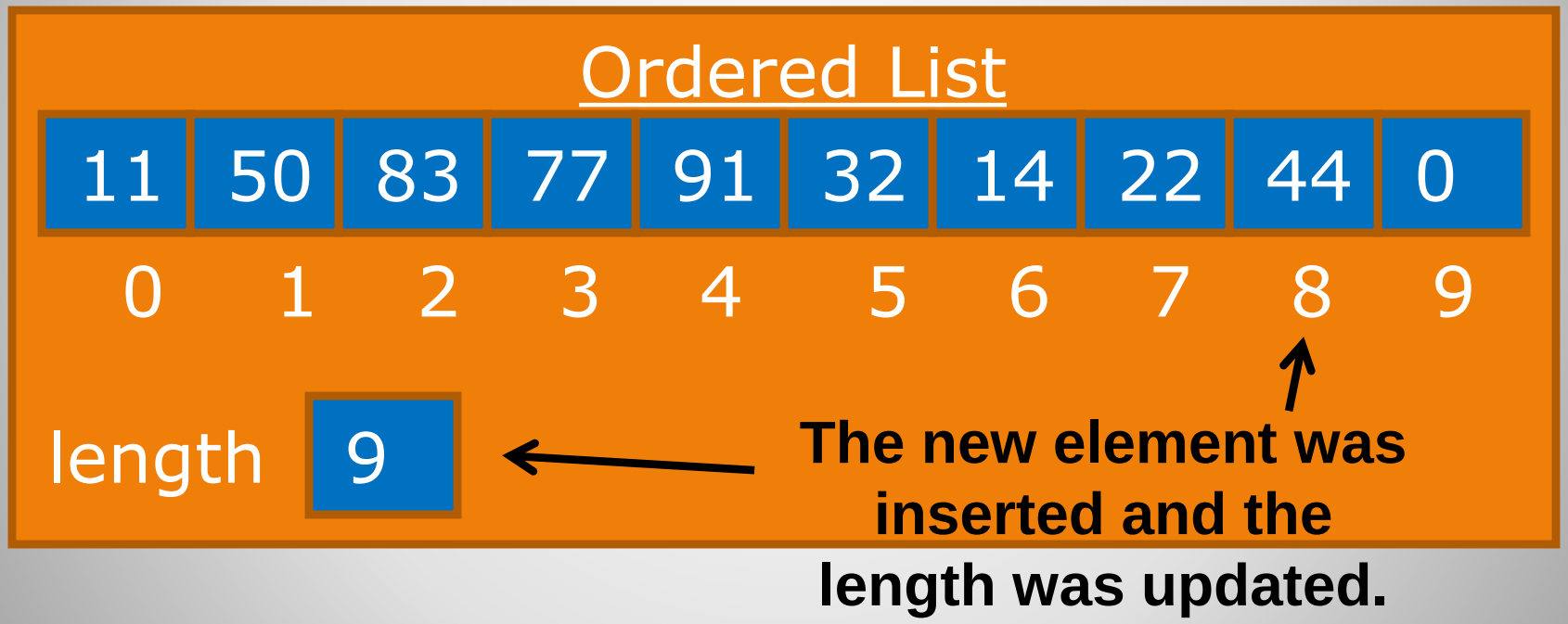
Ordered List

11	50	83	77	91	32	14	22	0	0
0	1	2	3	4	5	6	7	8	9

length 8

Ordered List - insertItem

- Here is what the array would look like:
  Declare ol as OrderedList
  // Other code to fill the ordered list goes here…
  ol.insertItem(44)



Ordered List - insertItem

- Now search for an item:
  Declare ol as OrderedList
  // Code to fill ol goes here…
  Set target to 83;
  Declare found as boolean
  Set found to ol.hasItem(target);

RetrieveItem checks if the a given item is in the list.

This code will return true in found since 83 is in the list.

Ordered List

11	50	83	77	91	32	14	22	44	0
0	1	2	3	4	5	6	7	8	9

length 9

Ordered List - hasItem

```
boolean hasItem(int target)
   Declare i as int
   Set i to 0
   Declare found as
   Set found to false
   while i less than length
      if info[i] equals target
         return true
      endIf

      Increment i
   endWhile

   return false
```

Ordered List - hasItem

- Now search for an item:
  Declare ol as OrderedList
  // Fill Ordered code here…
  Declare target as int and set to 50
  ol.deleteItem(target);

How does deleteItem work?

Ordered List - deleteItem

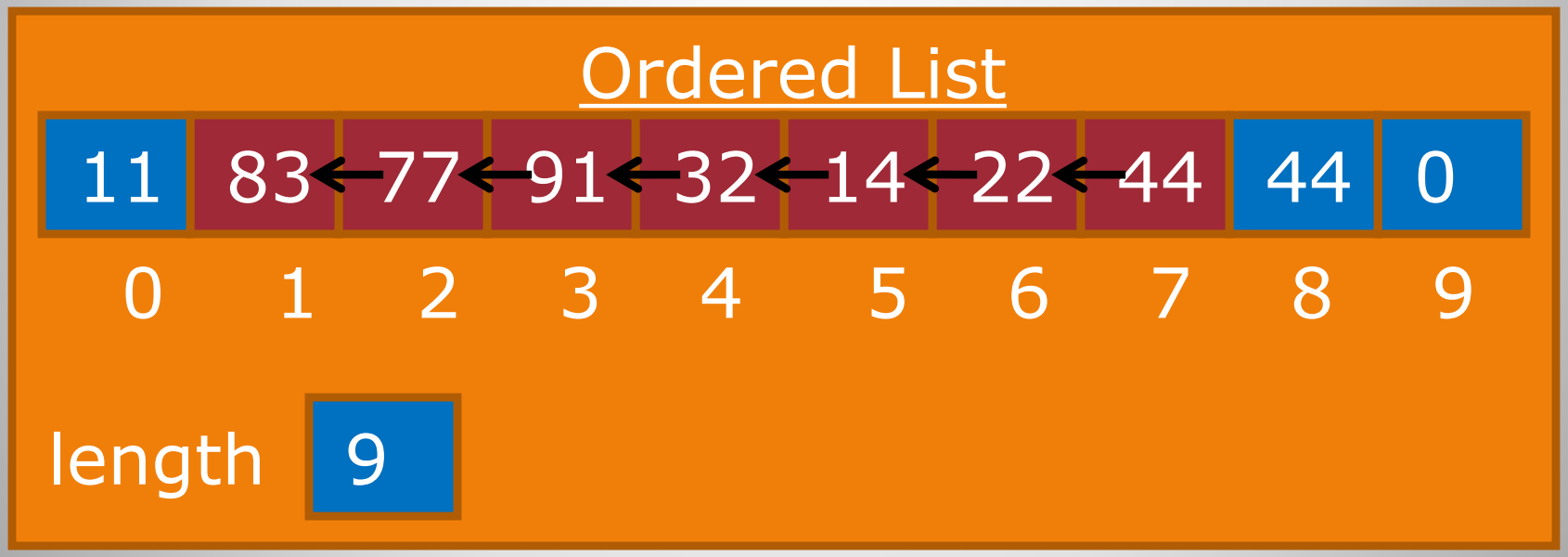
- Now search for an item:
  Declare ol as OrderedList
  // Fill Ordered code here…
  Declare target as int and set to 50
  ol.deleteItem(target);

1. Find the item location in the array

Ordered List - deleteItem

- Now search for an item:
  Declare ol as OrderedList
  // Fill Ordered code here…
  Declare target as int and set to 50
  ol.deleteItem(target);

1. ***Find the item location in the array***
2. ***Shift all elements that index to the end of the array***



Ordered List - deleteItem

- Now search for an item:
  Declare ol as OrderedList
  // Fill Ordered code here…
  Declare target as int and set to 50
  ol.deleteItem(target);

1. ***Find the item location in the array***
2. ***Shift all elements that index to the end of the array***
3. ***Decrement length***

Ordered List

11	83	77	91	32	14	22	44	44	0
0	1	2	3	4	5	6	7	8	9

length 8

Ordered List - deleteItem

Now we will finish with Big-O…

Big-O Comparison

- It is important to know the approximate runtime cost of the operation when you create a data structure.

- What are the Big-O runtimes for the list implementations?

Big-O Comparison

- Now analyze the ordered list operations for speed.
- What are the Big-O runtimes for the array-based list implementation?

Big-O Comparison

Operation	Cost
makeEmpty	???
isFull	???
getLength	???
retrieveItem	???
insertItem	???
deleteItem	???
resetList	???
getNextItem	???

Big-O Comparison – Ordered List (Array Based)

Operation	Cost
makeEmpty	O(1)
isFull	O(1)
getLength	O(1)
retrieveItem	O(n)
insertItem	O(1)
deleteItem	O(n)
resetList	O(1)
getNextItem	O(1)

Big-O Comparison – Ordered List (Array Based)

- **End of Slides**

End of Slides