NC STATE UNIVERSITY

Recursion and Lists

Zybooks 16.10

NC STATE UNIVERSITY

Review: Recursion and cases

- Recursion is all about breaking a big problem into smaller occurrences of that same problem.
- Every recursive algorithm involves at least 2 cases:
  - **Base Case**: A simple occurrence that can be answered directly.
  - **Recursive Case**: A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem.
- Some recursive algorithms have more than one base or recursive case, but all have at least one of each.
- A crucial part of recursive programming is identifying these cases.

NC STATE UNIVERSITY

Exercise

- Write a method `reverseLines` that accepts a file `Scanner` prints to `System.out` the lines of the file in reverse order.
  - Write the method recursively and without using loops.
  - Example input:

```
Schedule for next
semester:
CSC 230
CSC 316
```

Expected output:

```
CSC 316
CSC 230
semester:
Schedule for next
```

  - What are the cases to consider?
    - How can we solve a small part of the problem at a time?
    - What is a file that is very easy to reverse?

Slides are modified from those provided by Marty Stepp

4

NC STATE UNIVERSITY

With Loops

```
public static void reverseLines(Scanner fileScanner) {
  String rev = "";
  while(fileScanner.hasNextLine()){
     rev = fileScanner.nextLine() + "\n" + rev;
  }
  System.out.println(rev);
}
```


5

NC STATE UNIVERSITY

Possible Solution

```
//The method will print the lines in reverse?
public static void reverseLines(Scanner in) {
   if (in.hasNextLine()) { //recursive case
      String line = in.nextLine() //push
      System.out.println(line);  //pop
      reverseLines(in);  //recursive call
   } //implicit base case
}
```

Input	Expected	Actual
Schedule for next semester: CSC 230 CSC 316	CSC 316 CSC 230 semester: Schedule for next	Schedule for next semester: CSC 230 CSC 316

NC STATE UNIVERSITY

Reversal solution

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {          //recursive case
        String line = input.nextLine(); //push
        reverseLines(input);            //recursive call
        System.out.println(line);       //pop
    } //implicit base case
}
```

Input	Expected	Actual
Schedule for next semester: CSC 230 CSC 316	CSC 316 CSC 230 semester: Schedule for next	CSC 316 CSC 230 semester: Schedule for next

NC STATE UNIVERSITY

Tracing our algorithm

- **call stack**: The method invocations running at any one time.

```
reverseLines(new Scanner(new File("schedule.txt")));
```

input file:

```
Schedule for next
semester:
CSC 230
CSC 316
```

output:

Slides are modified from those provided by Marty Stepp

8

NC STATE UNIVERSITY

Tracing our algorithm

- **call stack**: The method invocations running at any one time.

```
reverseLi
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "Schedule for next"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "semester:"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "CSC 230"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "CSC 316"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {   // false
        ...
    }
}
```

```
Schedule for next
semester:
CSC 230
CSC 316
```

```
CSC 316
CSC 230
semester:
Schedule for next
```


Slides are modified from those provided by Marty Stepp

9

isPalindrome() Exercise

- Write a recursive method `isPalindrome` accepts a `String` and returns `true` if it reads the same forwards as backwards.
  - `isPalindrome("madam")` → `true`
  - `isPalindrome("racecar")` → `true`
  - `isPalindrome("step on no pets")` → `true`
  - `isPalindrome("able was I ere I saw elba")` → `true`
  - `isPalindrome("Java")` → `false`
  - `isPalindrome("rotater")` → `false`
  - `isPalindrome("byebye")` → `false`
  - `isPalindrome("notion")` → `false`

isPalindrome(String)

```
// Returns true if the given string reads the same
// forwards as backwards.
// Trivially true for empty or 1-letter strings.
public static boolean isPalindrome(String s) {
    if (s.length() < 2) {
        return true;   // base case
    } else {
        char first = s.charAt(0);
        char last  = s.charAt(s.length() - 1);
        if (first != last) {
            return false;
        }              // recursive case
        String middle = s.substring(1, s.length() - 1);
        return isPalindrome(middle);
    }
}
```

NC STATE UNIVERSITY

isPalindrome(String) Alternative Solution

```
// Returns true if the given string reads the same
// forwards as backwards.
// Trivially true for empty or 1-letter strings.
public static boolean isPalindrome(String s) {
    if (s.length() < 2) {
        return true;   // base case
    } else {
        return s.charAt(0) == s.charAt(s.length() - 1)
            && isPalindrome(s.substring(1, s.length() - 1));
    }
}
```


NC STATE UNIVERSITY

crawl() Exercise

- Write a method `crawl` accepts a `File` parameter and prints information about that file.
  - If the `File` object represents a normal file, just print its name.
  - If the `File` object represents a directory, print its name and information about every file/directory inside it, indented.

```
csc216
    handouts
        syllabus.doc
        lecture_schedule.xls
    homework
        1-sortedintlist
            ArrayIntList.java
            SortedIntList.java
            index.html
            style.css
```

  - **recursive data**: A directory can contain other directories.

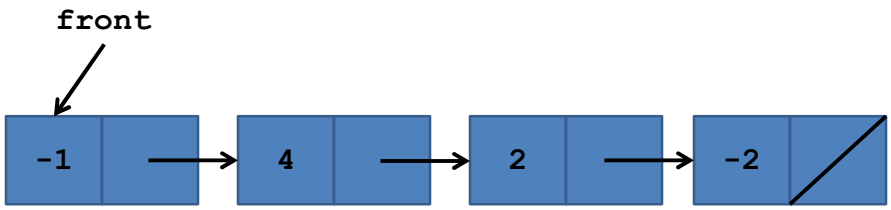
Exercise solution

```
// Prints information about this file,
// and (if it is a directory) any files inside it.
public static void crawl(File f) {
    crawl(f, "");   // call private recursive helper
}

// Recursive helper to implement crawl/indent
//behavior.
private static void crawl(File f, String indent) {
    System.out.println(indent + f.getName());
    if (f.isDirectory()) {
        // recursive case; print contained files/dirs
        for (File subFile : f.listFiles()) {
            crawl(subFile, indent + "    ");
        }
    } // else is the implicit base case
}
```


Recursion and Lists: The Idea

- We want to count the number of positive integers in a `LinkedIntList`

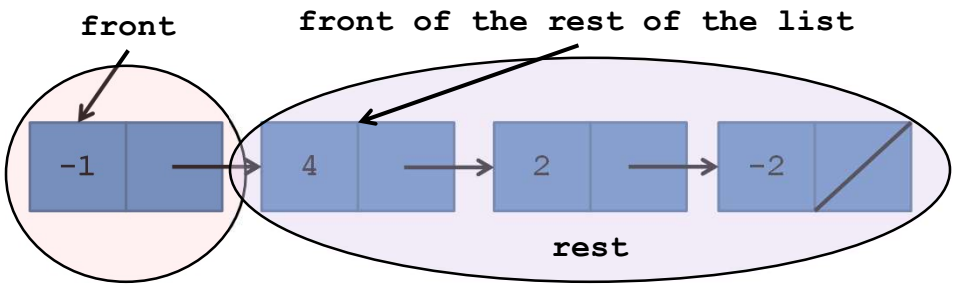


- How would we proceed?

NC STATE UNIVERSITY

Breaking a List into Parts

- Recursion is a technique that breaks a larger problem into smaller pieces
  - How can we break up a linked list?



- What is the smallest list?

NC STATE UNIVERSITY

Example: Counting Positive Numbers

- Write a method, **`countPositive()`**, which returns the number of positive numbers in a **`LinkedIntList`**.

- Test:

```
LinkedIntList l = new LinkedIntList();
l.add(4);
l.add(-7);
l.add(23);
l.add(-1);
assertEquals(2, l.countPositive());
```

NC STATE UNIVERSITY

Counting Positive Numbers

```
public int countPositive() {
 if (front == null) { //Base case – empty list
  return 0;
 }
 if (front.data > 0) { //Recursive case – positive
  return 1
       + (new LinkedIntList(front.next)).countPositive();
 } else { //Recursive case – negative or zero
  return (new LinkedIntList(front.next)).countPositive();
 }
}
```

The rest of the list as a new list object!

Recursive call

NC STATE UNIVERSITY

Counting Positive Numbers

- Constructing a whole new list to handle the recursion is a little clunky.
- Instead we can create a private helper method to help us with our recursive solution.
- The private helper method has a parameter, which is the new front of the list (or a current pointer into the list).

```
public int countPositive() { … }

private int countPositive(ListNode current) { … }
```

NC STATE UNIVERSITY

Counting Positive Numbers

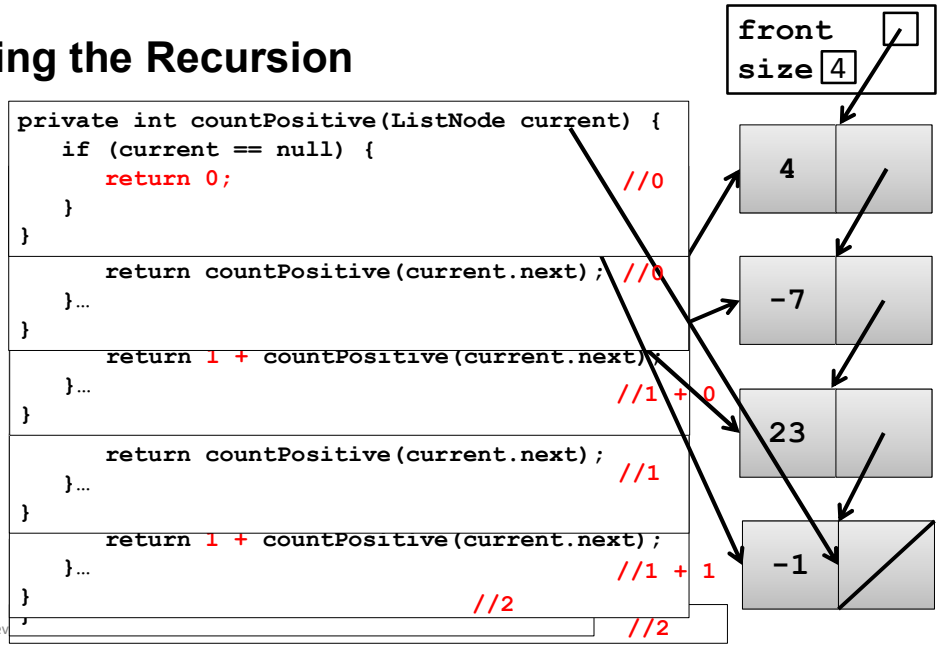
```
public int countPositive() {
   return countPositive(front); //Starts at front of list
}
private int countPositive(ListNode current) {
   if (current == null) { //Base case – empty list
      return 0;
   }
   if (current.data > 0) { //Recursive case - positive
      return 1 + countPositive(current.next);
   } else {  //Recursive case – negative or zero
      return countPositive(current.next);
   }
}
```

Making the list smaller

Recursive call

NC STATE UNIVERSITY

Exploring the Recursion



Contains Method

- Write the LinkedIntList.contains() method

```
public boolean contains(int v) { … }
private boolean contains(ListNode current, int v) { … }
```

- What is(are) the base case(s) (when do we terminate)?
- How do we know we have reached the base case? What is the test?
- What is the recursive case?
  - How do we break apart the list?
- What do we do when we find the value?
- What do we do when we can't find the value?

Contains Method

```
public boolean contains(int v) {

}

private boolean contains(ListNode current, int v) {









}
```

NC STATE UNIVERSITY

Adding to End of List

- Write the **`add(int)`** method that adds the given value to the *end* of the list.

```
public boolean recursiveAdd(int value) { … }
private boolean recurisveAdd(ListNode current, int value) { … }
```

- Think about:
  - Are there special cases to consider (e.g., front of the list)?
  - What is the base case?
  - What is the recursive case?
- Hint: Think about peeking ahead!
- Hint: Don't forget to increment size!

NC STATE UNIVERSITY

Recursive Add Method

```
public boolean recursiveAdd(int value) {




}
private boolean recursiveAdd(ListNode current, int value) {
```

We have to peek ahead.
Otherwise, we'll create a node that is not attached to the list!

```
}
```

Recursive Add Method

```
public boolean recursiveAdd(int value) {
    if (front == null) {
        front = new ListNode(value);
        size++;
        return true;
    } else {
        return front.add(value);
    }
}
```



Recursive List Helper Methods

- Some recursive list methods benefit from helper methods
  - Similar to public/private method pairs
  - Allows for a more elegant solution
- These helper methods are in the **`ListNode`** class
  - The helper method is the recursive method
- Why do we need a helper method to implement our operation recursively?
- Implement the helper method for **`recursiveAdd`**

Recursive ListNode Helper Methods

```
private class ListNode {              //ListNode!
   … //Other ListNode state, constructors, and methods
   public boolean add(int value) {
      if (next == null) {             //Base case – peeking ahead
         next = new ListNode(value); //Add new value
         size++;                      //Increment size
         return true;                 //Return true
      }
      return next.add(value);         //Recursive case – try
                                      //to add to the next node
  }
}
```

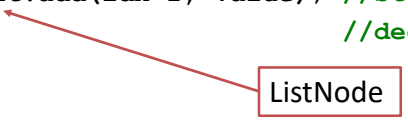

Adding Value at Index

- Write the **`add(int idx, int value)`** method that adds the value at the specified index
- What does the method look like in **`LinkedIntList`**?
  - What are the base cases?
- What does the helper method in **`ListNode`** look like?

NC STATE UNIVERSITY

Adding Value at Index

```
//Method is in LinkedIntList
public void recursiveAdd(int idx, int value) {
   if (idx == 0) { //Special case of front
      //Update global reference to front
      front = new ListNode(value, front);
   } else if (front == null) { //Index OOBs
      throw new IndexOutOfBoundsException();
   } else {
      front.add(idx-1, value); //Start recursion
                               //decrement i
   }
}
```



NC STATE UNIVERSITY

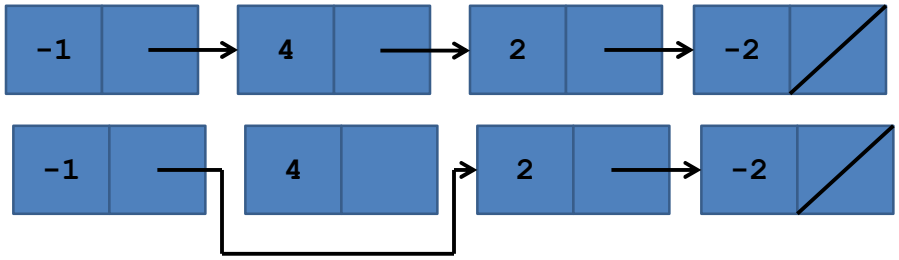
Adding Value at Index Helper

```
//Method is in ListNode
public void add(int idx, int value) {
   if (idx < 0) { //Base Case – out of lower bound
      throw new IndexOutOfBoundsException();
   } else if (idx == 0) { //Base Case – reached index
      next = new ListNode(value, next); //Add a new node that
                                //points to what next points to
   } else if (next == null) { //Base Case – out of upper bound
      throw new IndexOutOfBoundsException();
   } else { //Recursive case
         next.add(idx-1, value); //See if the next node is the
             //right location to add the value – decrement idx
   }
}
```

NC STATE UNIVERSITY

Remove Item from List

- Use a helper method
- Look at data in the next node
  - If the data is what we want to remove, then "link around" the next node



NC STATE UNIVERSITY

Patterns of Recursion

- What is the base case?
  - Empty list?
  - Reached a specific index?
  - Special case for the front of the list?
- What are the parameters?
  - Are the same parameters passed for each recursive call?
  - Do the parameters' values vary on each recursive call?
  - Are any passed?

Patterns of Recursion

- What is the return value?
  - Is the same value returned for each recursive call?
  - Does the value returned from each recursive call vary?
  - Is any value returned?

Patterns of Recursion

Method	Base Case			Parameter Values			Return Value		
	Empty	Front	Idx	Same	Vary	None	Same	Vary	None
`count(val)`	X								
`contains(val)`									
`indexOf(val)`									
`size()`									
`get(idx)`									
`set(idx, val)`									
`add(val)`									
`add(idx, val)`									
`remove(val)`									