

CSE 143

Lecture 10

Recursion

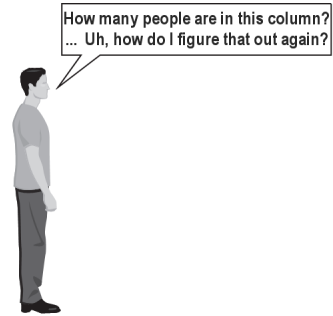
slides created by Marty Stepp and Alyssa Harding
<http://www.cs.washington.edu/143/>

Recursion

- **Iteration:** a programming technique in which you describe actions to be repeated using a loop
- **Recursion:** a programming technique in which you describe actions to be repeated using a method that calls itself
- Both approaches can be used to solve the same problems
 - Some problems are easier solved **iteratively**
 - Some problems are easier solved **recursively**
 - Sometimes the recursive solution is a LOT simpler than the iterative solution!

Exercise

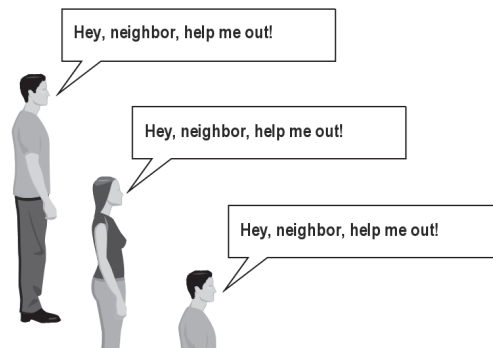
- How many students total are directly behind you in your "column" of the classroom?
 - You have poor vision, so you can see only the people right next to you. So you can't just look back and count.
 - But you are allowed to ask questions of the person next to you.
 - How can we solve this problem? (*recursively*)



3

The idea

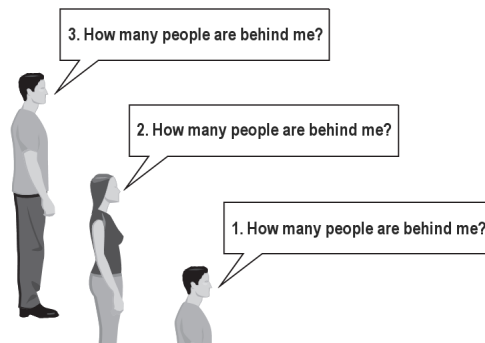
- Recursion is all about breaking a big problem into smaller occurrences of that same problem.
 - Each person can solve a small part of the problem.
 - What is a small version of the problem that would be easy to answer?
 - What information from a neighbor might help me?



4

Recursive algorithm

- Number of people behind me:
 - If there is someone behind me, ask him/her how many people are behind him/her.
 - When they respond with a value N , then I will answer $N + 1$.
 - If there is nobody behind me, I will answer 0 .



5

Recursion and cases

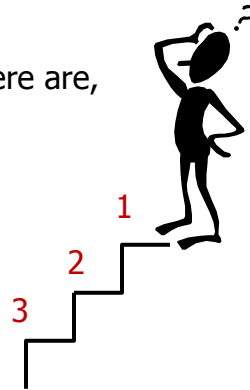
- Every recursive algorithm involves at least 2 cases:
 - **base case:** A simple occurrence that can be answered directly.
 - **recursive case:** A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem.
 - 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.

6

Example: stairs

- You want to walk down a flight of stairs.
- Iterative approach:

“Let me count the number of stairs there are,
and then take that many steps!”



7

Example: stairs

- You want to walk down a flight of stairs.
- Recursive approach:

“If I’m at the bottom, I stop.
Otherwise, I take a step down and repeat.”



8

Recursion in Java

- Consider the following method to print a line of * characters:

```
// Prints a line containing the given number of stars.  
// Precondition: n >= 0  
public static void printStars(int n) {  
    for (int i = 0; i < n; i++) {  
        System.out.print("*");  
    }  
    System.out.println();    // end the line of output  
}
```

- Write a recursive version of this method (that calls itself).
 - Solve the problem without using any loops.
 - Hint: Your solution should print just one star at a time.

9

A basic case

- What are the cases to consider?
 - What is a very easy number of stars to print without a loop?

```
public static void printStars(int n) {  
    if (n == 1) {  
        // base case; just print one star  
        System.out.println("*");  
    } else {  
        ...  
    }  
}
```

10

Handling more cases

- Handling additional cases, with no loops (in a bad way):

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else if (n == 2) {
        System.out.print("*");
        System.out.println("*");
    } else if (n == 3) {
        System.out.print("*");
        System.out.print("*");
        System.out.println("*");
    } else if (n == 4) {
        System.out.print("*");
        System.out.print("*");
        System.out.print("*");
        System.out.println("*");
    } else ...
}
```

11

Handling more cases 2

- Taking advantage of the repeated pattern (somewhat better):

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else if (n == 2) {
        System.out.print("*");
        printStars(1); // prints "*"
    } else if (n == 3) {
        System.out.print("*");
        printStars(2); // prints "***"
    } else if (n == 4) {
        System.out.print("*");
        printStars(3); // prints "****"
    } else ...
}
```

12

Using recursion properly

- Condensing the recursive cases into a single case:

```
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else {
        // recursive case; print one more star
        System.out.print("*");
        printStars(n - 1);
    }
}
```

13

"Recursion Zen"

- The real, even simpler, base case is an n of 0, not 1:

```
public static void printStars(int n) {
    if (n == 0) {
        // base case; just end the line of output
        System.out.println();
    } else {
        // recursive case; print one more star
        System.out.print("*");
        printStars(n - 1);
    }
}
```

- **Recursion Zen:** The art of properly identifying the best set of cases for a recursive algorithm and expressing them elegantly.
(A CSE 143 informal term)

14

Recursive tracing

- Consider the following recursive method:

```
public static int mystery(int n) {
    if (n < 10) {
        return n;
    } else {
        int a = n / 10;
        int b = n % 10;
        return mystery(a + b);
    }
}
```

- What is the result of the following call?

```
mystery(648)
```

15

A recursive trace

mystery(648):

- int a = 648 / 10; // 64
- int b = 648 % 10; // 8
- return mystery(a + b); // **mystery(72)**

mystery(72):

- int a = 72 / 10; // 7
- int b = 72 % 10; // 2
- return mystery(a + b); // **mystery(9)**

mystery(9):

- return 9;

16

Recursive tracing 2

- Consider the following recursive method:

```
public static int mystery(int n) {
    if (n < 10) {
        return (10 * n) + n;
    } else {
        int a = mystery(n / 10);
        int b = mystery(n % 10);
        return (100 * a) + b;
    }
}
```

- What is the result of the following call?

```
mystery(348)
```

17

A recursive trace 2

mystery(348)

```
▪ int a = mystery(34);
  • int a = mystery(3);
    return (10 * 3) + 3; // 33
  • int b = mystery(4);
    return (10 * 4) + 4; // 44
  • return (100 * 33) + 44; // 3344

▪ int b = mystery(8);
  return (10 * 8) + 8; // 88

- return (100 * 3344) + 88; // 334488
```

- What is this method really doing?

18

Example: reverse

- Now we'll look at a problem that's hard to solve iteratively, but easier with recursion
- Given a `Scanner` as input, print the lines in reverse
- How would you solve this iteratively?
 - Loop while there are more lines
 - Requires additional storage, like a `List` or a `Stack`

19

Example: reverse

- Writing `reverse` recursively:
- What is the base case?

```
public static void reverse(Scanner input) {  
    // base case: no more lines  
    if ( !input.hasNextLine() ) {  
        // do nothing  
    } else {  
        ...  
    }  
}
```

This is a good base case,
but we don't need to
do anything in this case

20

Example: reverse

- Writing `reverse` recursively:
- What is the base case?

```
public static void reverse(Scanner input) {  
    // base case: no more lines  
    // recursive case  
    if ( input.hasNextLine() ) {  
        ...  
    }  
}
```

It's better style not to have an empty if statement.

21

Example: reverse

- Writing `reverse` recursively:
- What is the recursive case's work?

```
public static void reverse(Scanner input) {  
    // base case: no more lines  
    // recursive case  
    if ( input.hasNextLine() ) {  
        String line = input.nextLine();  
        // reverse the rest of the input  
        System.out.println(line);  
    }  
}
```

We made a little progress, how do we do the rest?

22

Example: reverse

- Writing `reverse` recursively:
- What is the recursive case's work?

```
public static void reverse(Scanner input) {  
    // base case: no more lines  
    // recursive case  
    if ( input.hasNextLine() ) {  
        String line = input.nextLine();  
        reverse(input);  
        System.out.println(line);  
    }  
}
```

} We recursively call the method with the easier problem!

23

Example: reverse

```
public static void main (String[] args) {  
    public static void reverse(Scanner input) {  
        public static void reverse(Scanner input) {  
            public static void reverse(Scanner input) {  
                public static void reverse(Scanner input) {  
                    public static void reverse(Scanner input) {  
                        if ( input.hasNextLine() ) { // false!  
                            String line = input.nextLine();  
                            reverse(input);  
                            System.out.println(line);  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

Input:

student
that
loves
recursion

recursion
loves
that
student

24