Recursion

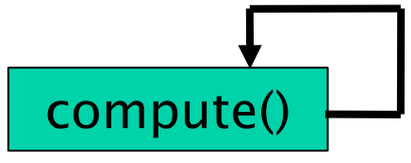
Garfield AP Computer Science

As usual, significant borrowings from Stuart Reges and Marty Stepp at UW -- thanks!!

Definitions

- **recursion**: The definition of an operation in terms of itself.
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem.

- **recursive programming**: Writing methods that call themselves to solve problems recursively.
  - An equally powerful substitute for iteration (loops)
  - Particularly well-suited to solving certain types of problems



Why recursion

- "cultural experience" - A different way of thinking of problems
- Can solve some kinds of problems better than iteration
- Leads to elegant, simplistic, short code (when used well)
- Many programming languages ("functional" languages such as Scheme, ML, and Haskell) use recursion exclusively (no loops)
- The AP test will include questions that require you to read and interpret recursive code

Cases

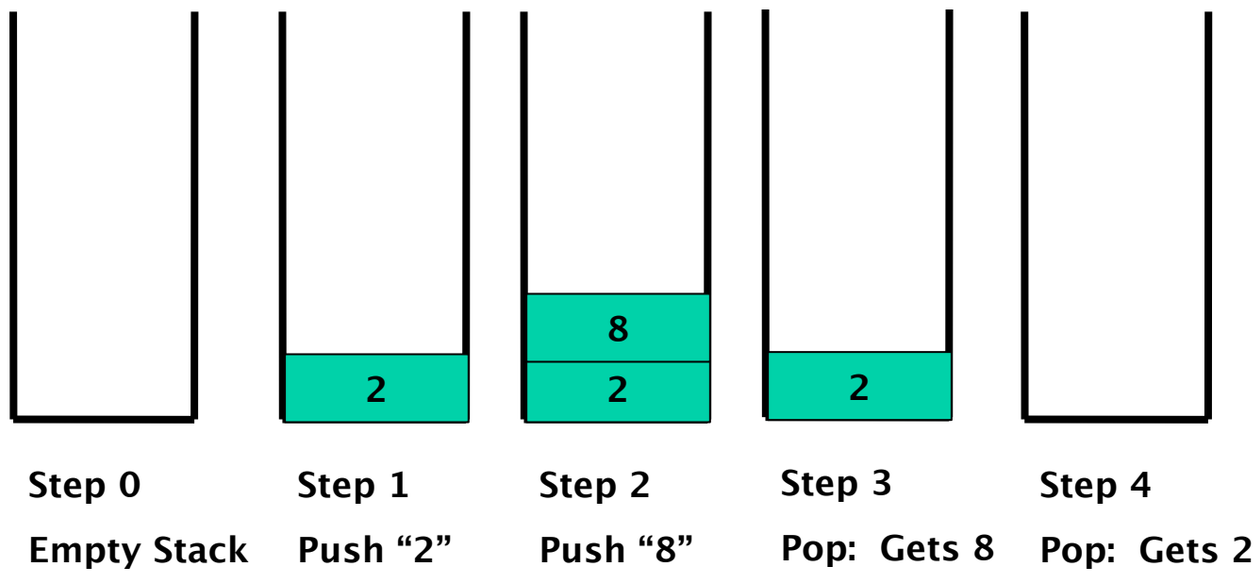
- Every recursive algorithm involves at least 2 cases:
  - **base case**: A simple occurrence that can be answered directly.
  - **recursive case**: A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem.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

Call stack

stack: Data structure from which elements are retrieved in the reverse of the order they were added. Think of a stack of papers.

Computers have a call stack -- method calls are pushed on a stack and the methods are popped when they return

Visualization of a stack holding ints:



Doubling method

```
// Recursive method to double a value
public static int times2(int value) {
    if(value == 0) { // base case
        return 0;
    } else {
        return times2(value - 1) + 2; // recursive
case
        }
    }
```

The best way to get a feel for how this is working is to draw out the call stack as we go!

Line of stars

- 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.

Base case

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

Base 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 {
        ...
    }
}
```

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 ...
}
```

Improvement

- 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 ...
}
```

Recursive case

- 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);
    }
}
```

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)
```

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;`

Reverse file

- Write a recursive method `reverseLines` that accepts a file `Scanner` and prints the lines of the file in reverse order.
  - Example input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

Expected console output:

```
Are belong to you.
All my base
Violets are blue.
Roses are red,
```

  - 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?

Pseudocode

- Reversing the lines of a file:
  - Read a line L from the file.
  - Print the rest of the lines in reverse order.
  - Print the line L.

- If only we had a way to reverse the rest of the lines of the file....

Tracing our algorithm

- Use the call stack to visualize how the code works

```
reverseLines(new Scanner("poem.txt"));
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Violets are blue."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Violets are blue."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "All my base"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "All my base"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Are belong to you."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Are belong to you."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Are belong to you."
```

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

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Are belong to you."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Are belong to you."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "All my base"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "All my base"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Violets are blue."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Violets are blue."
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
Violets are blue.
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
Violets are blue.
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
    reverseLines(new Scanner("poem.txt"));
```

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();  // "Roses are red,"
        reverseLines(input);
        System.out.println(line);
    }
}
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
Violets are blue.
Roses are red,
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
reverseLines(new Scanner("poem.txt"));
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
Violets are blue.
Roses are red,
```

Tracing our algorithm

- Use the call stack to visualize how the code works

```
reverseLines(new Scanner("poem.txt"));
```

input file:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

output:

```
Are belong to you.
All my base
Violets are blue.
Roses are red,
```