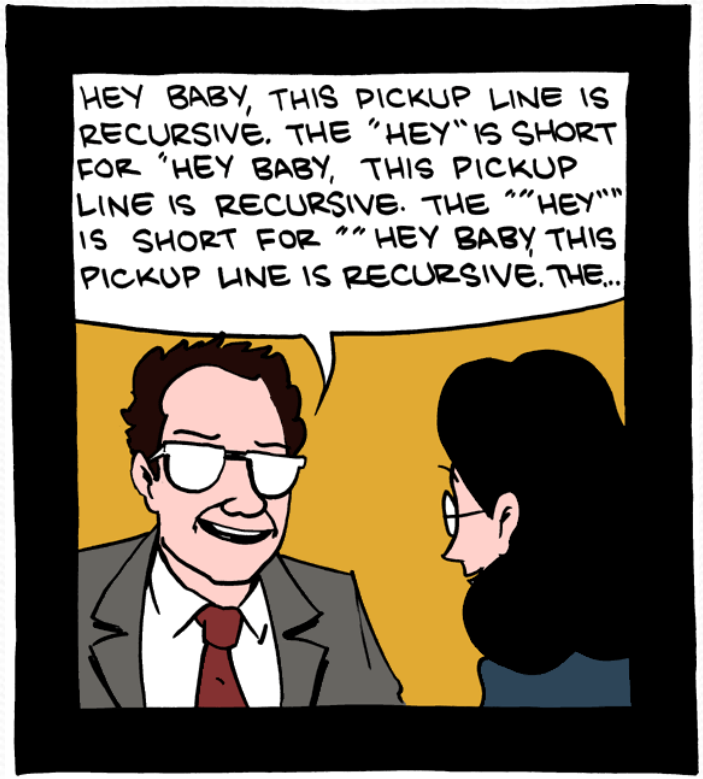
CS 142

Lecture 16: recursion



Benoit Mandelbrot: Master of seduction.

Thanks to Marty Stepp and Stuart Reges for parts of these slides.

Another recursive task

- How can we remove exactly half of the M&M's in a large bowl, without dumping them all out or being able to count them?
  - What if multiple people help out with solving the problem? Can each person do a small part of the work?
  - What is a number of M&M's that it is easy to double, even if you can't count?

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.

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);
    }
}
```

"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);
    }
}
```

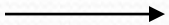
  - **Recursion Zen**: The art of properly identifying the best set of cases for a recursive algorithm and expressing them elegantly.

    (A CS 142 informal term)

Exercise

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

```
I have eaten
the plums
that were in
the icebox
```



Expected console output:

```
the icebox
that were in
the plums
I have eaten
```

  - 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?

Reversal 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....

Reversal solution

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

- Where is the base case?

Tracing our algorithm

- **call stack**: The method invocations currently running

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

```
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "I have eaten"
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "the plums"
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "that were in"
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        String line = input.nextLine();   // "the icebox"
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {    // false
        ...
    }
}
```

```
I have eaten
the plums
that were in
the icebox
```

```
the icebox
that were in
the plums
I have eaten
```

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)
```

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;`

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)
```

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?

Exercise

- Write a recursive method `pow` accepts an integer base and exponent and returns the base raised to that exponent.
  - Example: `pow(3, 4)` returns 81
  - Solve the problem recursively and without using loops.

An optimization

- Notice the following mathematical property:

$3^{12} = 531441 = 9^6$

$= (3^2)^6$

$531441 = (9^2)^3$

$= ((3^2)^2)^3$

  - When does this "trick" work?
  - How can we incorporate this optimization into our `pow` method?
  - What is the benefit of this trick if the method already works?


There are only 10 types
of people in the world:
Those who understand binary
and those who don't

Exercise

- Write a recursive method `printBinary` that accepts an integer and prints that number's representation in binary (base 2).
  - Example: `printBinary(7)` prints 111
  - Example: `printBinary(12)` prints 1100
  - Example: `printBinary(42)` prints 101010

place	10	1
value	**4**	**2**

32	16	8	4	2	1
1	**0**	**1**	**0**	**1**	**0**

  - Write the method recursively and without using any loops.

Stutter

- How did we break the number apart?

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

Case analysis

- Recursion is about solving a small piece of a large problem.
  - What is 69743 in binary?
    - Do we know *anything* about its representation in binary?
  - Case analysis:
    - What is/are easy numbers to print in binary?
    - Can we express a larger number in terms of a smaller number(s)?

printBinary solution

```
// Prints the given integer's binary representation.
// Precondition: n >= 0
public static void printBinary(int n) {
    if (n < 2) {
        // base case; same as base 10
        System.out.println(n);
    } else {
        // recursive case; break number apart
        printBinary(n / 2);
        printBinary(n % 2);
    }
}
```

- Can we eliminate the precondition and deal with negatives?