Introduction to Recursion

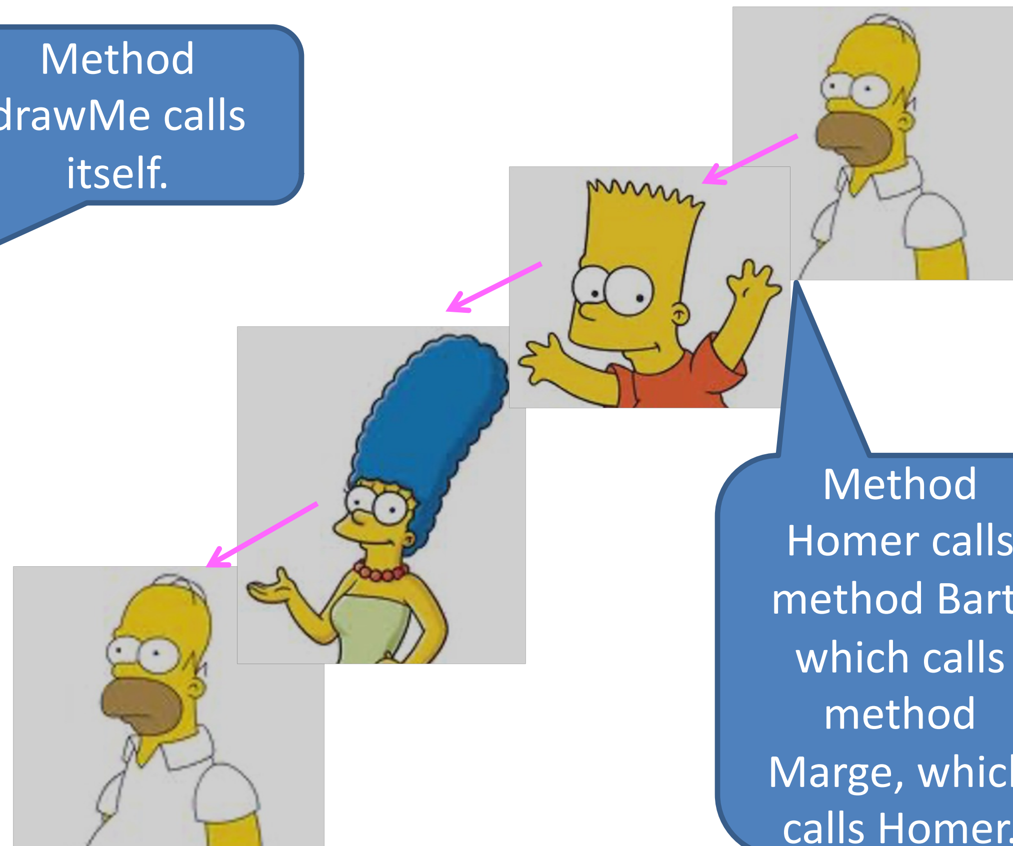
Goals

- What is recursion?
- Why recursion?
- Examples of recursion
  - Calculate factorial
- Any recursive code can be rewritten by iteration.

What is recursion?

- A method calls itself directly or indirectly

Method drawMe calls itself.



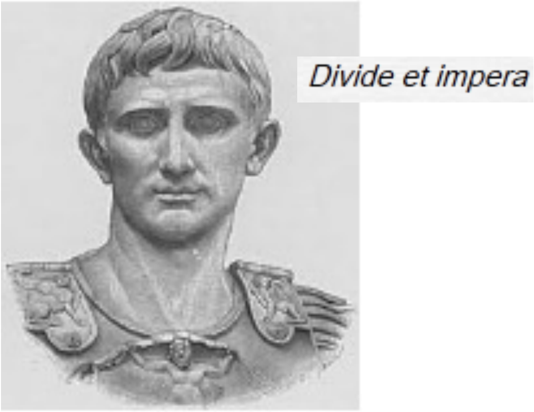
Method Homer calls method Bart, which calls method Marge, which calls Homer.

Motivation

WHY RECURSION

Why recursion?

- New mode of thinking.
- Powerful programming tool
- Divide-and-conquer paradigm
  – Divide
  – Conquer
  – Combine
  – Originated from Julius Caesar



Solve problems using recursion

- **Divide** the problem into pieces and assume all problems with **smaller** sizes can be solved.
- **Combine** answers to **smaller** size problems to obtain answer to the original problem.
  - Each recursive call must involve smaller values of the arguments.
- **Explicitly Conquer** (assign a direct answer to) problem with **smallest** size (base case).
  - cannot delivery responsibility to subordinates indefinitely

Example

FACTORIAL

Factorial: non-recursive definition

- Factorial $n!$, when n is positive, is the products of n and all the positive integers below it, ie,

  $$n! = n * (n-1) * (n-2) * \ldots *$$

- Define $0! = 1$.
- What can $n!$ represent?
  - Number of all possible ways to line up n objects.
  - For the above five suspects, there are a total of 5! = 120 ways to line them up.

Factorial: recursive definition

- By definition

$$n! = n * \underline{(n-1) * (n-2) * \ldots * 1}$$

and

$$(n-1)! = \underline{(n-1) * (n-2) * \ldots * 1}$$

- Represent $n!$ in terms of $(n-1)!$

$$n! = n * (n-1)!$$

Components of recursive definition

- A recursive definition is made up of two parts.
  - a base case tells us directly what the answer is.

    $0! = 1$

  - a recursive case defines the answer in terms of the answer to some other related problem.
    - When $n >= 1$, construct $n!$ on the basis of $(n-1)!$

$$n! = \begin{cases} 1 & if\ n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

Calculate factorial n!

Use recursive definition of factorial.

```
//return type is long since
//factorial grows very fast
long factorial(int n) {
    if (n == 0)
        return 1;
    else return n * factorial(n-1);
}
```

$$n! = \begin{cases} 1 & if\, n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

Illustration: calculate factorial(3)

$$n! = \begin{cases} 1 & if\, n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1

Illustration: calculate factorial(3)

$$n! = \begin{cases} 1 & if\, n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

factorial(3) = 3 * factorial(2) = 3 * 2 = 6

factorial(2) = 2 * factorial(1) = 2 * 1 = 2

factorial(1) = 1 * factorial(0) = 1 * 1 =1

factorial(0) = 1

Implement function calls

- When a caller function calls a callee,
  - Caller function saves its local variables.
  - Pass parameters to callee function,
  - Switch control from caller to callee,
  - Run codes in callee functions,
  - Get return if applicable,
  - Switch control back from callee to callee, all local variables in callee will be released.
  - Restore local variables of caller.

Function calls illustration

- What if Function A calls Function B, which in turn calls Function C?
- What happens to the local variables of Functions A and B when Function C is called?
- Recursive calls is also function calls.
  - A recursive function just calls itself with **smaller** size.

Exercises

- Print a string in backward order
- Search for a word in an array of sorted strings

Summary

- In recursive function, find out base case.
- Assume that all smaller case can be solved, combine those smaller solutions to get a solution to the original problem.
- When you call a function itself, the size must be reduced, or system needs to allocate infinite spaces to hold the setting of previous running instances, and the system will crash.