Recursion Revisited

David Greenstein
Monta Vista High School

Recursion

- A recursive solution describes a procedure for a particular task in terms of applying the same procedure to a similar but smaller task.

- Must have a ***base case*** when the task is so simple that no recursion is needed.

- Recursive calls must eventually converge to a base case.

Recursive Method

- A recursive method calls itself.

- A recursive method has at least one *base case* and at least one *recursive case*.
  - In the base case, there are no recursive calls
  - In the recursive case, the method calls itself, but for a “smaller” task

- A recursive method must have a conditional statement to decide *base case* vs. *recursive case*.

- Recursive calls must eventually converge to a base case, when recursion stops.

Recursive Method (cont)

Conditional

Base case

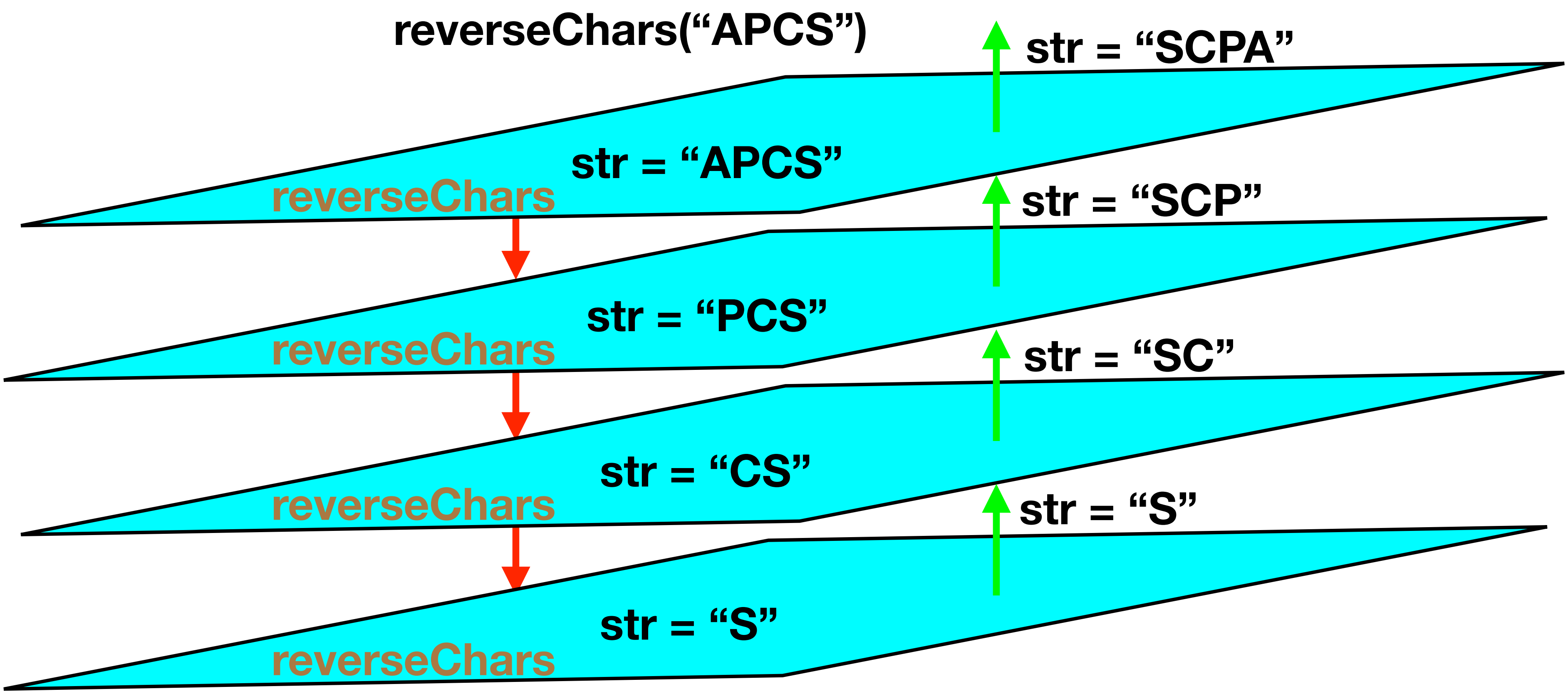
```
public String reverseChars (String str)
{
    if (str.length() == 0) return “”;

    return reverseChars(str.substring(1))
                                  + str.charAt(0);
}
```

Calls itself
(recursive case)

Converges
(e.g. smaller task)

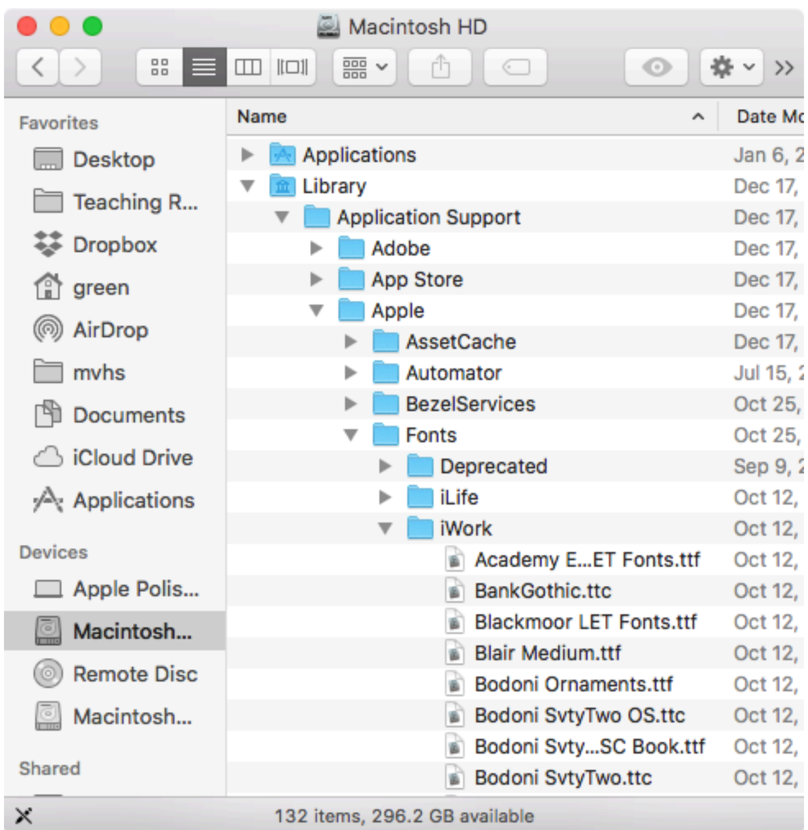
How Recursion Works

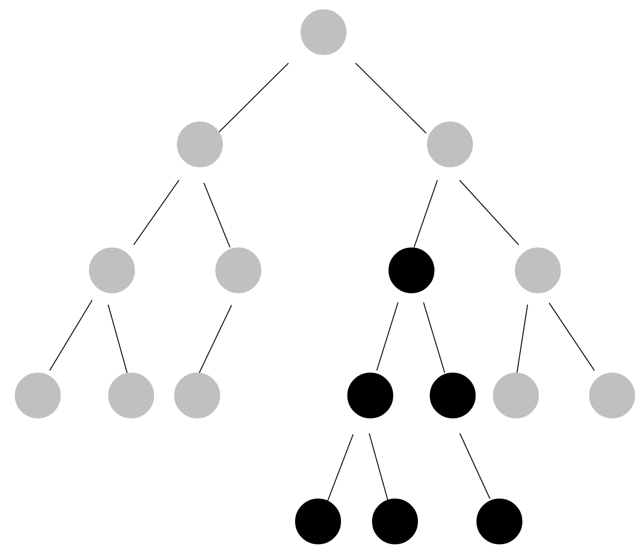
- Implemented on a computer as <u>a form of iterations</u>, but hidden from the programmer.
- Assisted by the *<u>system stack</u>*.



When to Use Recursion

- Recursion is especially useful for handling <u>nested structures</u> and <u>branching processes</u>.





When to Use Recursion

- Recursion is especially useful for handling nested structures and branching processes.
- When it significantly simplifies code without significant performance penalty.

Do Not Use Recursion

- When a method allocates large local arrays.
- When a method unpredictably changes object fields.
- When an iterative solution is just as simple.

Recursion and Math Induction

- Recursive methods are hard to trace in a conventional way.
- A recursive method can be proven correct using mathematical induction.
- Other properties of a recursive method (running time, required space, etc.) can be obtained by using mathematical induction.

Mathematical Induction Basics

- You have a sequence of statements to prove true (hypothesis)

  $$P_1, P_2, P_3, \ldots P_{n-1}, P_n, \ldots$$

- First, show P_1 (n = 1) is true ("base case").

- Next, assume that for any $n > 1$, that $P_1, \ldots P_{n-1}$ are all true ("inductive hypothesis").

- Using deductive reasoning, show P_n must be true, too ("inductive step").

- Finally, you can conclude ("by mathematical induction") that P_n is true for any $n \geq 1$.

Mathematical Induction Example:

Prove that for any integer $n \geq 1$

$$1 + 2 + 4 + ... + 2^n = 2^{n+1} - 1$$

Proof:

1. Base case: If $n = 1$, then $1 + 2^1 = 2^2 - 1$
2. **Suppose** (inductive hypothesis)

   for $n = k\text{-}1$ that $1 + 2 + 4 + ... + 2^{k-1} = 2^k - 1$ is true
3. **Then** (inductive step) for $n = k$

$$1 + 2 + 4 + ... + 2^k =$$

$$(1 + 2 + 4 + ... 2^{k-1}) + 2^k = (2^k - 1) + 2^k =$$

$$2 \cdot (2^k) - 1 = 2^{k+1} - 1$$

By math induction, the equality is true for any $n \geq 1$, q.e.d.

Mathematical Induction Problem:

Prove that for any n $\geq$ 1

$$1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Proof:

1. Base case: when n = 1 then
2. **Suppose** (Inductive hypothesis)
3. **Then** (inductive step) show

Mathematical Induction Problem:

Prove that for any n ≥ 1

$$1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Proof:

1. Base case: when $n = 1$ then $1^2 = \frac{1(1+1)(2+1)}{6} = 1$

2. **Suppose** (Inductive hypothesis) n = k - 1 and

$$1^2 + 2^2 + \ldots + (k-1)^2 = \frac{(k-1)k(2k-1)}{6}$$

3. **Then** (inductive step) show $1^2 + 2^2 + \ldots + (k-1)^2 + k^2 = \frac{k(k+1)(2k+1)}{6}$

Using substitution:

$$1^2 + 2^2 + \ldots + (k-1)^2 + k^2 = \frac{(k-1)k(2k-1)}{6} + k^2 = \frac{k(k+1)(2k+1)}{6}$$

Induction and Recursion

```
public String reverseChars (String str)
{
   if (str.length() == 0) return "";

   return reverseChars(str.substring(1))
                                 + str.charAt(0);
}
```

Let us verify that this method works, that is, **reverseChars(s)** indeed returns the reverse of **s**. We will use math induction “over the length of **s**.”

Proof

```java
public String reverseChars (String str)
{
    if (str.length() == 0) return "";

    return reverseChars(str.substring(1))
                                      + str.charAt(0);
}
```

Let n = **s.length()**; Prove that **reverseChars(s)** returns a string with the characters of **s** reversed for n ≥ 0.

1. Base case: If n = 0 or n = 1 then **reverseChars()** works because the string remains unchanged.
2. **Suppose** (inductive hypothesis) assume for **s.length()** = k-1 that **reverseChars(s)** works.
3. **Then** (inductive step) Add a k'th character **c** to the front of string **s** to make the string **c + s**. **reverseChars(c + s)** returns **reverseChars(s) + c** which is the reverse of **c + s**.

By math induction, **reverseChars** works for a string of length n ≥ 0, q.e.d.

Questions?