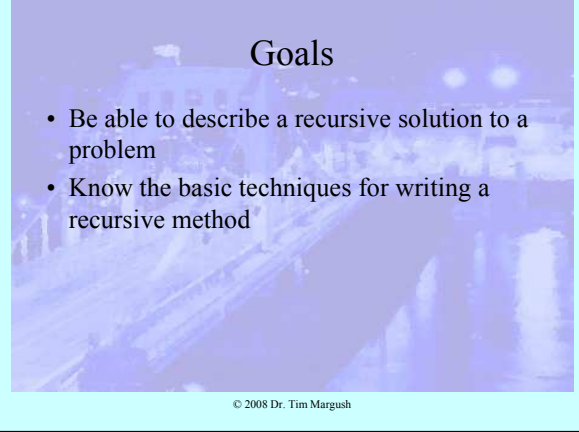


Data Structures
and Algorithms I

Recursion

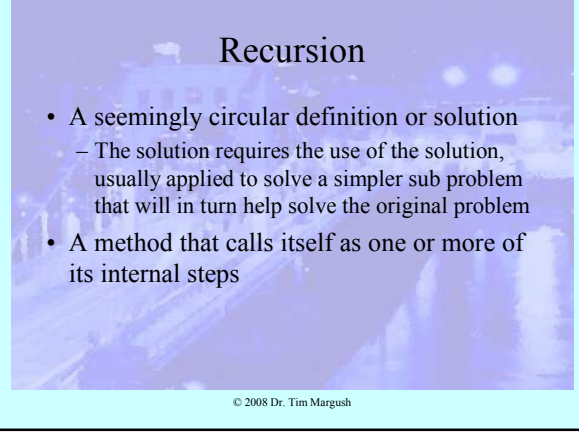
Dr. Tim Margush
University of Akron
© 2008



Goals

- Be able to describe a recursive solution to a problem
- Know the basic techniques for writing a recursive method

© 2008 Dr. Tim Margush



Recursion

- A seemingly circular definition or solution
 - The solution requires the use of the solution, usually applied to solve a simpler sub problem that will in turn help solve the original problem
- A method that calls itself as one or more of its internal steps

© 2008 Dr. Tim Margush

A Bad Example, Bad Example, Bad Example, Bad...

```
public void myBad(){
    System.out.print("Bad ");
    myBad();
    System.out.println("Example");
}
...
myBad();
```

© 2008 Dr. Tim Margush

State

```
public void myBad(int n){
    System.out.print("Bad ");
    myBad(n-1);
    System.out.println("Example");
}
...
myBad(4);
```

© 2008 Dr. Tim Margush

State

```
public void myBad(int n){
    if (n<1)
        System.out.println("Example");
    else{
        System.out.print("Bad ");
        myBad(n-1);
    }
}
...
myBad(4);
```

© 2008 Dr. Tim Margush

State

```
public void myBad(int n){
  if (n<1)
    System.out.println("Example");
  else{
    System.out.print("Bad ");
    myBad(n-1);
  }
}
...
myBad(4);
```

© 2008 Dr. Tim Margush

Recursive Definitions

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

What are the special or base cases?

$$2^n = 2^{n/2} * 2^{n/2}$$

$$\sum_{i=0}^n i = n + \sum_{i=0}^{n-1} i$$

© 2008 Dr. Tim Margush

Recursive Definitions

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

$$0! = 1$$

$$2^n = 2^{n/2} * 2^{n/2}$$

$$2^1 = 2$$

$$\sum_{i=0}^n i = n + \sum_{i=0}^{n-1} i$$

what if n is odd?

When n=0, sum is 0

© 2008 Dr. Tim Margush

Summation Without Loops

```
int[] nums = {3, 5, 2, 8, 9, 10, 4};  
int sumOfFirst(int n){  
    if (n==0)  
        return 0;  
    else  
        return sumOfFirst(n-1) + nums[n-1];  
}
```

© 2008 Dr. Tim Margush

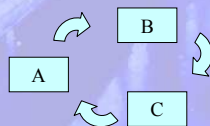
Summation Without Loops

```
int[] nums = {3, 5, 2, 8, 9, 10, 4};  
int sumFrom_n_toEnd(int n){  
    if ( _____ )  
        return 0;  
    else  
        return _____ ;  
}
```

© 2008 Dr. Tim Margush

Indirect Recursion

- This occurs when method A
 - calls method B, which
 - calls method C, which
 - calls method A, which
 - is recursion!



© 2008 Dr. Tim Margush

Using Stamps

- I have an unlimited supply of 2, 5, 9, 32, and 41 cent stamps
 - What stamps should I use to mail a package that requires x cents?
- $\text{bestStamps}(x) =$
 - If a stamp has value x , done! Use it
 - Use the largest stamp $< x$, with value y and include whatever $\text{bestStamps}(x-y)$ suggests

© 2008 Dr. Tim Margush

Using Stamps

- I have an unlimited supply of 2, 5, 9, 32, and 41 cent stamps
 - 87 cents?
- Use 41 plus $\text{bestStamps}(46)$
- Use 41 plus $\text{bestStamps}(5)$
- Use 5

© 2008 Dr. Tim Margush

Using Stamps

- I have an unlimited supply of 2, 5, 9, 32, and 41 cent stamps
 - 15 cents?
- Use 9 plus $\text{bestStamps}(6)$
- Use 5 plus $\text{bestStamps}(1)$
- Use ????
 - Should we pick the next bigger stamp to mail the package?

© 2008 Dr. Tim Margush

Recursion and Helpers

```
private void sort(int [] x, int from_i, int to_j){
    if (from_i >= to_j) return;
    int smallest_loc = from_i;
    for (int k = from_i+1; k<to_j; k++)
        if (x[k] < x[smallest_loc]) smallest_loc = k;
    int temp = x[from_i];
    x[from_i] = x[smallest_loc];
    x[smallest_loc] = temp;
    sort(x, from_i+1, to_i);
}
```

© 2008 Dr. Tim Margush

Helper

- Recursion is sometimes hidden behind a simpler interface

```
public void sort(int x[]){
    //use recursive helper
    sort(x, 0, x.length);
}
```

© 2008 Dr. Tim Margush

GCD

- $\text{gcd}(x,y) = y$ if y divides x
- $= \text{gcd}(y, x\%y)$ Otherwise
- $\text{gcd}(24,18) = \text{gcd}(18, 6) = 6$
- $\text{gcd}(1155, 84) = \text{gcd}(84, 63)$
- $= \text{gcd}(63, 21) = 21$

© 2008 Dr. Tim Margush

String Reverse

```
String sReverse(String s){  
    if (s.length()<2) return s;  
    return sReverse(s.substring(1)  
                    +s.charAt(0))
```

abcdefg

© 2008 Dr. Tim Margush

Summary

- A recursive method calls itself (directly or indirectly)
- Recursive solutions must have a valid base case
- Recursive calls must work toward simpler cases
- The solution must be obtainable by solving simpler sub problems

© 2008 Dr. Tim Margush
