

CSE1301  
Computer Programming  
Lecture 28  
*Recursion (Part 2)*

1

Topics

- Finding recursive solutions to problems
- Towers of Hanoi
- Sierpinski Triangle

Reading: D&D Chapter 5  
Exercises 5.37 to 5.40, and 5.42

2

Towers of Hanoi

- A classic problem
- Three pegs
- $N$  discs, arranged bottom to top by decreasing size
- Objective: Move the discs from peg 1 to peg 3
- Two constraints:
  - One disk is moved at a time
  - No larger disc can be placed above a smaller disk
- Write a program which will print the precise sequence of peg-to-peg disc transfers.

3

Towers of Hanoi

Base case:  $N = 1$     Peg 1  $\rightarrow$  Peg 3

4

Towers of Hanoi

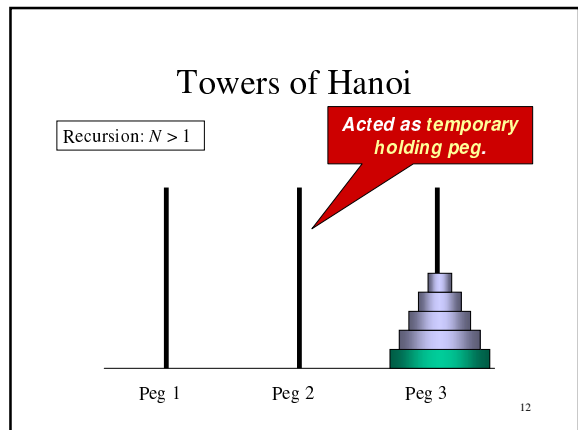
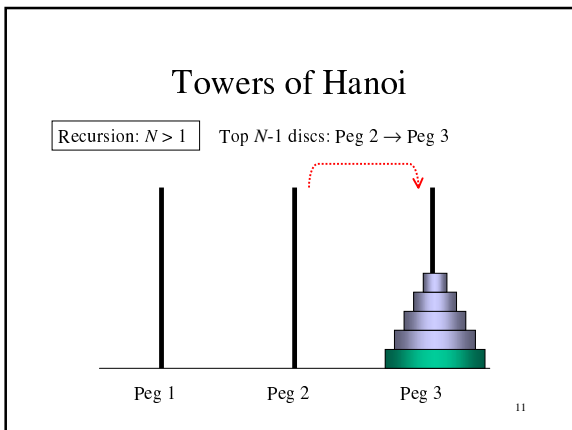
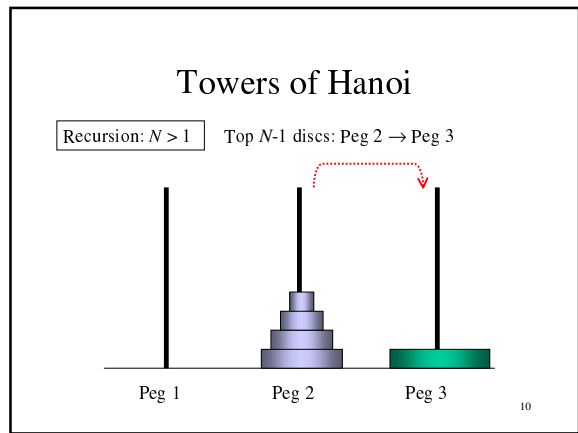
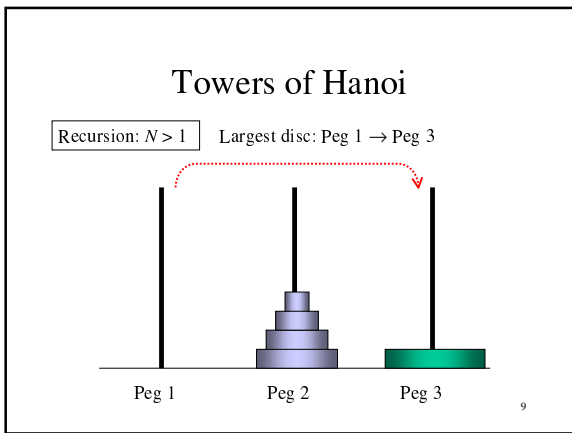
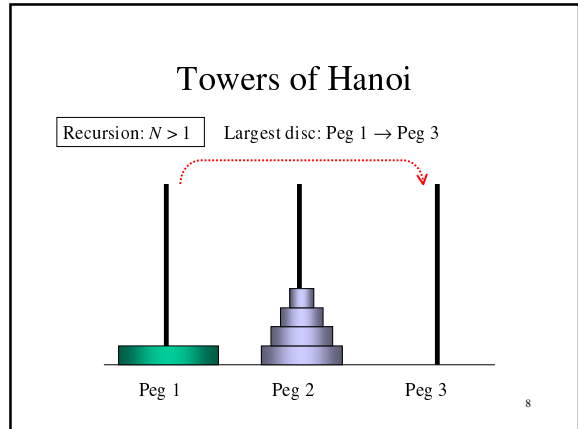
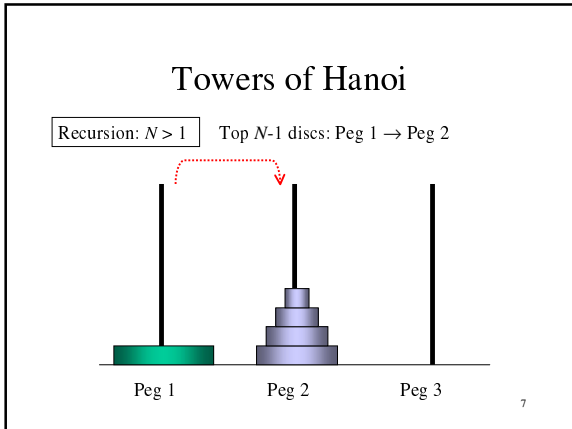
Base case:  $N = 1$     Peg 1  $\rightarrow$  Peg 3

5

Towers of Hanoi

Recursion:  $N > 1$     Top  $N-1$  discs: Peg 1  $\rightarrow$  Peg 2

6



### Towers of Hanoi

- Q: But how do you move those  $N-1$  discs from Peg 1 to the temporary holding peg?
- A: Recursively, of course!
  - E.g., "move  $N-1$  discs from Peg 1 to Peg 2 using Peg 3 as temporarily holding area," and so on.
- Denote:
  - *fromPeg, toPeg*
  - Temporary holding peg: *otherPeg*

13

### Towers of Hanoi

```

procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = /* determine otherPeg */
    ToH ( numDiscs - 1, fromPeg, otherPeg )
    output fromPeg, "->", toPeg
    ToH ( numDiscs - 1, otherPeg, toPeg )
  }
}
    
```

*Base case*

14

### Towers of Hanoi

```

procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = /* determine temporary holding peg here */
    ToH ( numDiscs - 1, fromPeg, otherPeg )
    output fromPeg, "->", toPeg
    ToH ( numDiscs - 1, otherPeg, toPeg )
  }
}
    
```

*Recursion*

15

### Towers of Hanoi

```

procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = theOtherPeg(fromPeg, toPeg)
    ToH(numDiscs - 1, fromPeg, otherPeg)
    output fromPeg, "->", toPeg
    ToH(numDiscs - 1, otherPeg, toPeg)
  }
}
    
```

16

### Towers of Hanoi

<pre> function theOtherPeg ( pegA, pegB ) {   if ( pegA is 1 ) then   {     if ( pegB is 2 ) then       return 3     else       return 2   }   else if ( pegA is 2 ) then   {     if ( pegB is 1 ) then       return 3     else       return 1;   } }                 </pre>	<pre> else if ( pegA is 3 ) then {   if ( pegB is 2 ) then     return 1   else     return 2; }                 </pre>
--	---

*Solution 1*

17

### Towers of Hanoi

<i>fromPeg</i>	<i>toPeg</i>	<i>otherPeg</i>
1	2	3
1	3	2
2	1	3
2	3	1
3	2	1
3	1	2

*otherPeg* is always  $6 - (\text{fromPeg} + \text{toPeg})$

18

### Towers of Hanoi

```
function theOtherPeg (fromPeg, toPeg)
{
    return ( 6 - fromPeg - toPeg )
}
```

*Solution 2*

19

### Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
    if ( numDiscs is 1 ) then
        output fromPeg, "-->", toPeg
    else
    {
        otherPeg = theOtherPeg(fromPeg, toPeg)
        ToH(numDiscs - 1, fromPeg, otherPeg)
        output fromPeg, "-->", toPeg
        ToH(numDiscs - 1, otherPeg, toPeg)
    }
}
```

20

### Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
    if ( numDiscs is 1 ) then
        output fromPeg, "-->", toPeg
    else
    {
        otherPeg = theOtherPeg(fromPeg, toPeg)
        ToH(numDiscs - 1, fromPeg, otherPeg)
        output fromPeg, "-->", toPeg
        ToH(numDiscs - 1, otherPeg, toPeg)
    }
}
```

*Convergence*

21

```
/* Given two pegs, this function determines the other peg. */
int theOtherPeg ( int fromPeg, int toPeg )
{
    return ( 6 - fromPeg - toPeg );
}

/* This functions prints out the precise sequence of peg-to-peg disc
 * transfers to solve the Towers of Hanoi problem. */
void ToH ( int numDiscs, int fromPeg, int toPeg )
{
    int otherPeg;

    if ( numDiscs == 1 )
    {
        printf("%d -> %d\n", fromPeg, toPeg);
    }
    else
    {
        otherPeg = theOtherPeg(fromPeg, toPeg);
        ToH(numDiscs - 1, fromPeg, otherPeg);
        printf("%d -> %d\n", fromPeg, toPeg);
        ToH(numDiscs - 1, otherPeg, toPeg);
    }
}
```

*hanoi.c*

22

```
#include <stdio.h>
#include "hanoi.c"

/* This is a test program for the recursive
 * function for the Towers of Hanoi.
 */

int main(void)
{
    int n;

    printf("Enter number of discs: ");
    scanf("%d", &n);

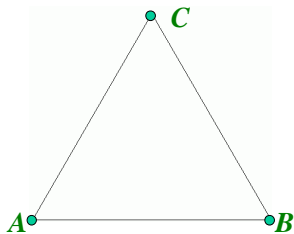
    if ( n > 0 )
        ToH(n, 1, 3);
    else
        printf("Invalid n value.\n");

    return 0;
}
```

*testprog.c*

23

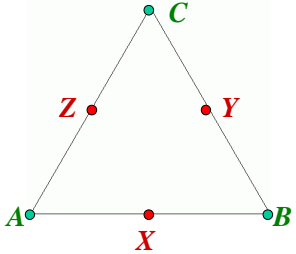
### Example: A Sierpinski Triangle



Level 0: The initial triangle (equilateral)

24

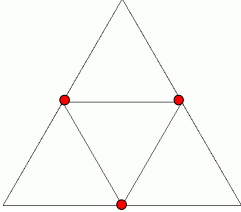
Example: A Sierpinski Triangle



Determine mid-point of each side.

25

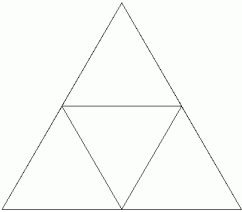
Example: A Sierpinski Triangle



Form smaller triangles.

26

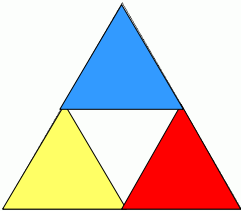
Example: A Sierpinski Triangle



Level 1: The length of each side of the smaller triangles is half that of the initial triangle.

27

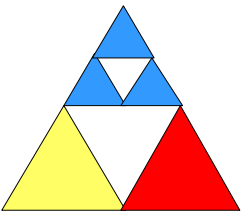
Example: A Sierpinski Triangle



Apply same procedure to each shaded triangle.

28

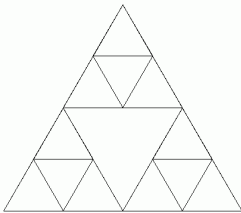
Example: A Sierpinski Triangle



Apply same procedure to each shaded triangle.

29

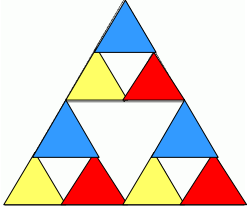
Example: A Sierpinski Triangle



Level 2

30

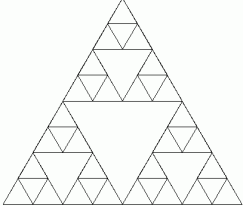
### Example: A Sierpinski Triangle



Apply same procedure again to smaller triangles.

31

### Example: A Sierpinski Triangle

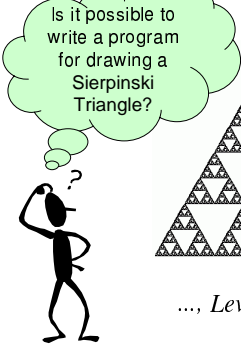
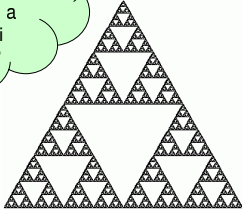


Level 3, and so on...

32

### Example: A Sierpinski Triangle

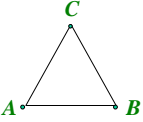
Is it possible to write a program for drawing a Sierpinski Triangle?

..., Level 7, etc...

33

### Algorithm: A Sierpinski Triangle

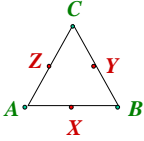


```

procedure Sierp ( A, B, C )
{
  drawTriangle ( A, B, C )
}
    
```

34

### Algorithm: A Sierpinski Triangle

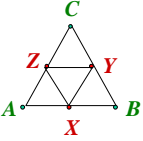


```

procedure Sierp ( A, B, C )
{
  drawTriangle ( A, B, C )
  let X be midPoint ( A, B )
  let Y be midPoint ( B, C )
  let Z be midPoint ( C, A )
}
    
```

35

### Algorithm: A Sierpinski Triangle



```

procedure Sierp ( A, B, C )
{
  drawTriangle ( A, B, C )
  let X be midPoint ( A, B )
  let Y be midPoint ( B, C )
  let Z be midPoint ( C, A )
  ????
}
    
```

36

Algorithm: A Sierpinski Triangle

37

Algorithm: A Sierpinski Triangle

Perform: Sierp ( A, X, Z )

38

Algorithm: A Sierpinski Triangle

Perform: Sierp ( X, B, Y )

39

Algorithm: A Sierpinski Triangle

Perform: Sierp ( Z, Y, C )

40

Algorithm: A Sierpinski Triangle

```

    procedure Sierp ( A, B, C )
    {
        drawTriangle ( A, B, C )
        let X be midPoint ( A, B )
        let Y be midPoint ( B, C )
        let Z be midPoint ( C, A )

        Sierp ( A, X, Z )
        Sierp ( X, B, Y )
        Sierp ( Z, Y, C )
    }
    
```

41

Algorithm: A Sierpinski Triangle

**Function  
calls itself:  
Recursion**

```

    procedure Sierp ( A, B, C )
    {
        drawTriangle ( A, B, C )
        let X be midPoint ( A, B )
        let Y be midPoint ( B, C )
        let Z be midPoint ( C, A )

        Sierp ( A, X, Z )
        Sierp ( X, B, Y )
        Sierp ( Z, Y, C )
    }
    
```

42

### Algorithm: A Sierpinski Triangle

When and how will it stop?

```

procedure Sierp ( A, B, C )
{
  drawTriangle ( A, B, C )
  let X be midPoint ( A, B )
  let Y be midPoint ( B, C )
  let Z be midPoint ( C, A )

  Sierp ( A, X, Z )
  Sierp ( X, B, Y )
  Sierp ( Z, Y, C )
}
    
```

### Algorithm: A Sierpinski Triangle

```

procedure Sierp ( level, A, B, C )
{
  if ( level > MAXLEVEL ) then return
  else {
    drawTriangle ( A, B, C )
    let X be midPoint ( A, B )
    let Y be midPoint ( B, C )
    let Z be midPoint ( C, A )

    Sierp ( level + 1, A, X, Z )
    Sierp ( level + 1, X, B, Y )
    Sierp ( level + 1, Z, Y, C )
  }
}
    
```

### Algorithm: A Sierpinski Triangle

```

procedure Sierp ( level, A, B, C )
{
  if ( level > MAXLEVEL ) then return
  else {
    drawTriangle ( A, B, C )
    let X be midPoint ( A, B )
    let Y be midPoint ( B, C )
    let Z be midPoint ( C, A )

    Sierp ( level + 1, A, X, Z )
    Sierp ( level + 1, X, B, Y )
    Sierp ( level + 1, Z, Y, C )
  }
}
    
```

**Base Case:**  
No recursive call

### Algorithm: A Sierpinski Triangle

```

procedure Sierp ( level, A, B, C )
{
  if ( level > MAXLEVEL ) then return
  else {
    drawTriangle ( A, B, C )
    let X be midPoint ( A, B )
    let Y be midPoint ( B, C )
    let Z be midPoint ( C, A )

    Sierp ( level + 1, A, X, Z )
    Sierp ( level + 1, X, B, Y )
    Sierp ( level + 1, Z, Y, C )
  }
}
    
```

**General Case:**  
Recursive call

### Algorithm: A Sierpinski Triangle

```

procedure Sierp ( level, A, B, C )
{
  if ( level > MAXLEVEL ) then return
  else {
    drawTriangle ( A, B, C )
    let X be midPoint ( A, B )
    let Y be midPoint ( B, C )
    let Z be midPoint ( C, A )

    Sierp ( level + 1, A, X, Z )
    Sierp ( level + 1, X, B, Y )
    Sierp ( level + 1, Z, Y, C )
  }
}
    
```

**The general case converges towards the base case.**

### Reading

- King 9.6
- D&D Chapter 5
  - Exercises 5.37 to 5.40, and 5.42
- Kernighan and Ritchie 4.10