University of Guelph　　Prof. Pascal Matsakis

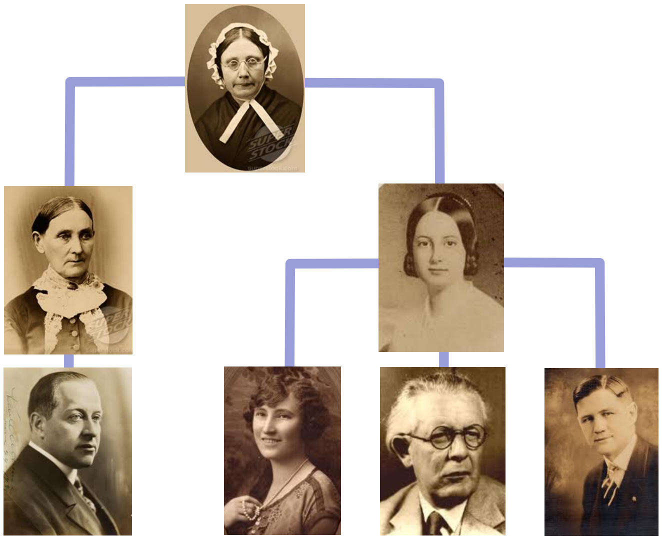
CIS2520

3. Recursion

Reading suggestion: Chapter 3 of the textbook

RECURSIVE DEFINITIONS: First Example　　3.2

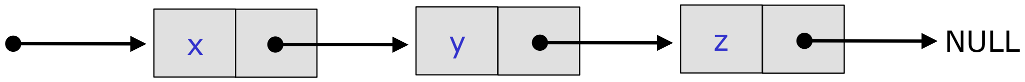
An **ancestor** is:
a) a parent, or
b) a parent's **ancestor**



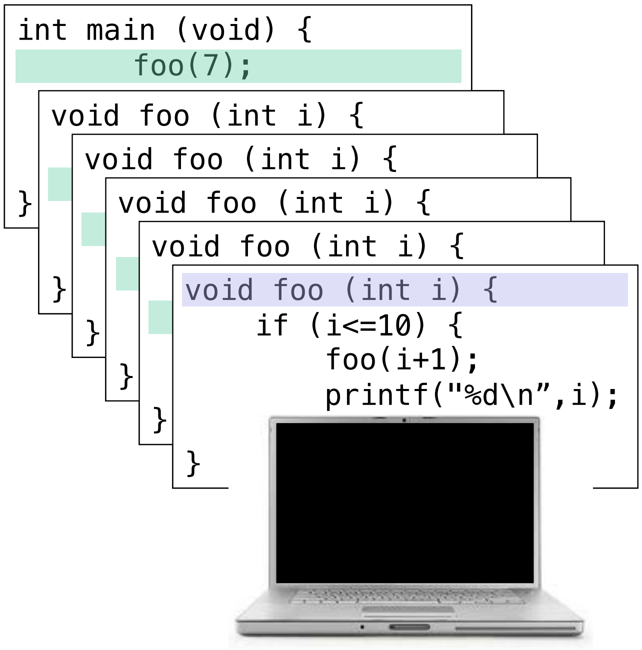
Reading suggestion: Chapter 3 of the textbook

RECURSIVE DEFINITIONS: Second Example 3.3

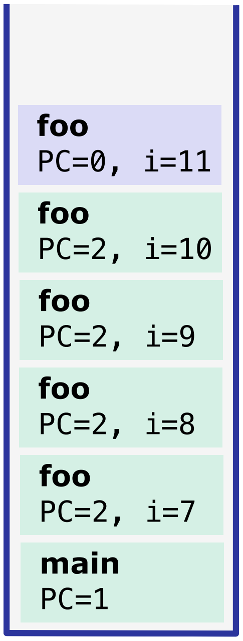
The **length** of a list is:
a) 0 if the list is empty
b) 1 + the **length** of the tail of the list if the list is not empty



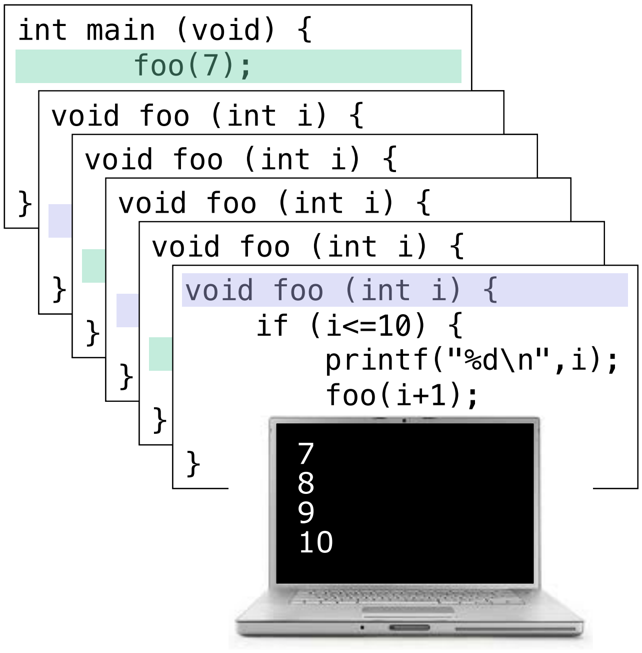
UNDERSTANDING RECURSION: First Example 3.4



```
int main (void) {
          foo(7);
}

void foo (int i) {
}

void foo (int i) {
}

void foo (int i) {
}

void foo (int i) {
}

void foo (int i) {
     if (i<=10) {
          foo(i+1);
          printf("%d\n”,i);
}
```



UNDERSTANDING RECURSION: Second Example 3.5



```
int main (void) {
        foo(7);
}
```

```
void foo (int i) {
}
```

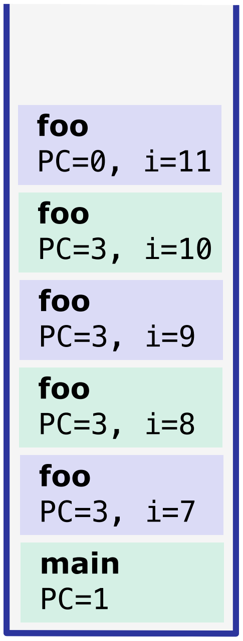
```
void foo (int i) {
}
```

```
void foo (int i) {
}
```

```
void foo (int i) {
}
```

```
void foo (int i) {
     if (i<=10) {
          printf("%d\n",i);
          foo(i+1);
}
```

```
7
8
9
10
```



```
foo
PC=0, i=11
foo
PC=3, i=10
foo
PC=3, i=9
foo
PC=3, i=8
foo
PC=3, i=7
main
PC=1
```

Reading suggestion: Chapter 3 of the textbook

CHARACTERISTICS OF A RECURSIVE FUNCTION 3.6

✧ calls itself
✧ has some terminating condition
✧ moves “closer” to the terminating condition

```
if (terminating condition) {
   do final actions
}  else {
   move one step closer to terminating condition
   recursive call(s)
}
```

OR

```
if (!(terminating condition)) {
  move one step closer to terminating condition
  recursive call(s)
}
```

Reading suggestion: Chapter 3 of the textbook

EXAMPLE: Sum of the first nonnegative integers 3.7

```
int bar (int n) {
        if (n==0) return 0;
        else return(n+bar(n-1));
}
```

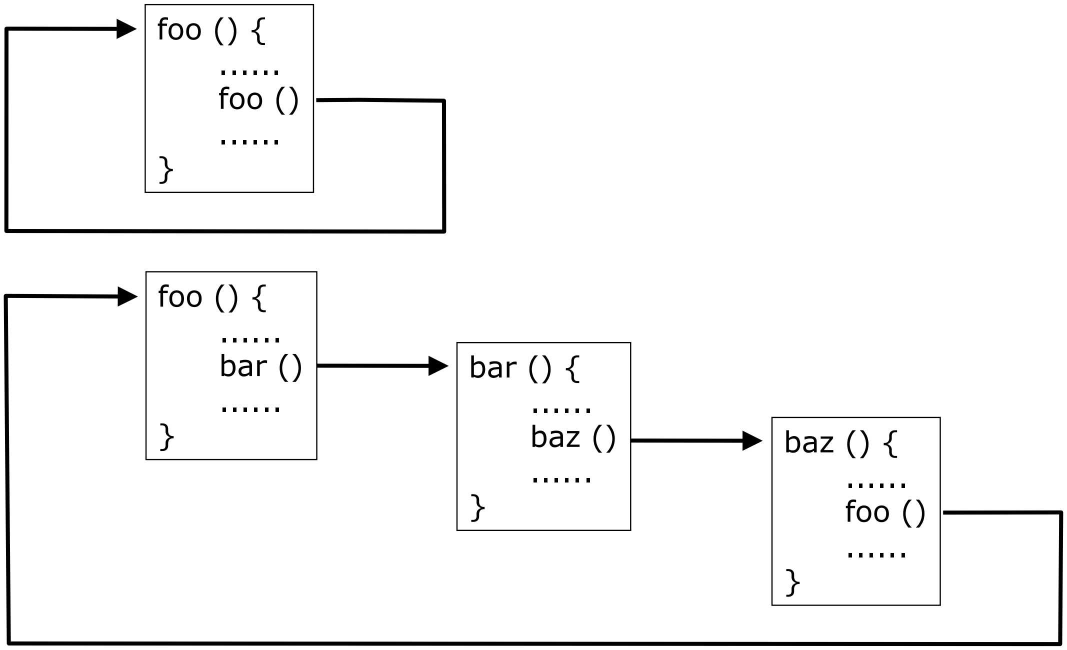
EXAMPLE: Factorial 3.8

```
int baz (int n) {
        if (n==0) return 1;
        else return(n*baz(n-1));
}
```

EXAMPLE: Story Telling 3.9

```
void TellStory() {
        printf(“%s”, “It was a dark and stormy night ”);
        printf(“%s”, “and the captain said to the mate ”);
        printf(“%s”, “`Tell us a story mate’ ”);
        printf(“%s”, “and this is the story he told: ”);
        TellStory();
}
```

DIRECT vs. INDIRECT 3.10



```
foo () {
    ......
    foo ()
    ......
}
```

```
foo () {
    ......
    bar ()
    ......
}
```

```
bar () {
    ......
    baz ()
    ......
}
```

```
baz () {
    ......
    foo ()
    ......
}
```

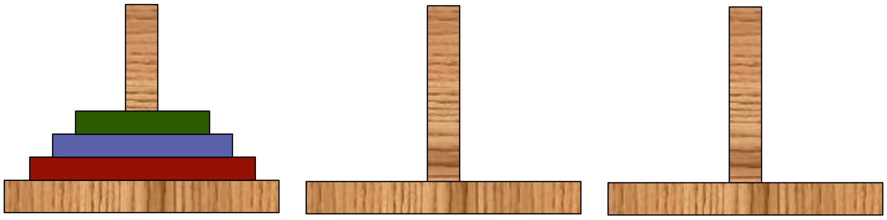
ITERATIVE vs. RECURSIVE 3.11

Problem to be solved: use **iterative** approach or **recursive** approach?
Two questions:

✧ how **easy** are they
to understand and implement?

✧ how **efficient** are they
in terms of computational time and memory usage?

TOWERS OF HANOI: Description and Example (1/2) 3.12

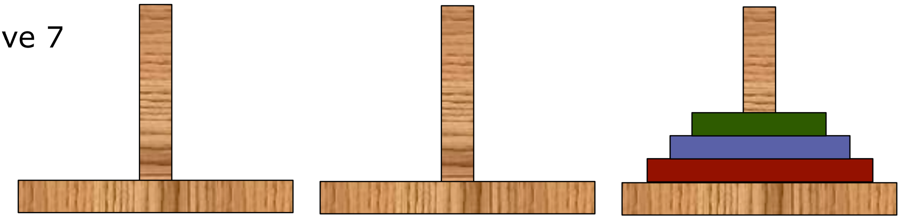


3 pegs and *n* disks

Initially: the disks are on one peg
Goal: move all disks to another peg
Rules: ✧ move one disk at a time
✧ never place a disk on a smaller one

TOWERS OF HANOI: Description and Example (2/2) 3.13

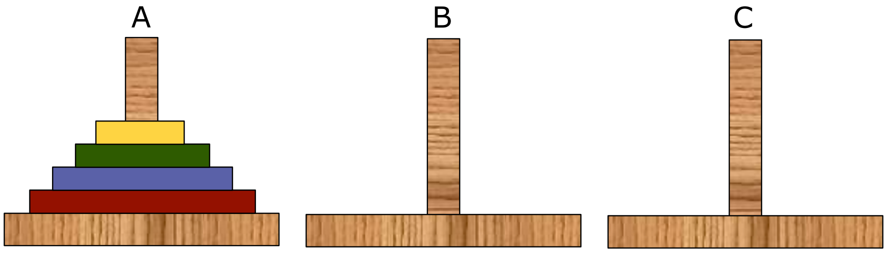
move 7



3 pegs and *n* disks

Initially: the disks are on one peg
Goal: move all disks to another peg
Rules: ✧ move one disk at a time
✧ never place a disk on a smaller one

TOWERS OF HANOI: Recursive Solution (1/4) 3.14

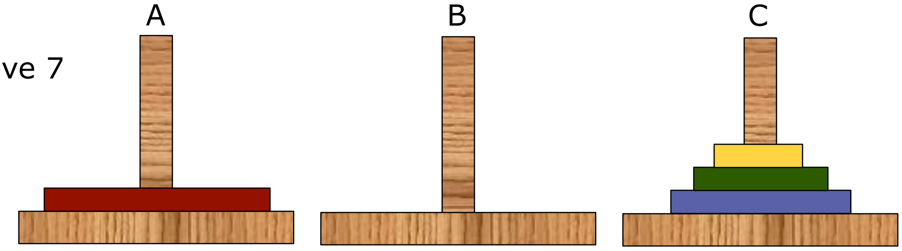


3 pegs and *n* disks

```
void MoveTower (int n, char A, char B, char C) {
        if (n > 0) {


        }
}
```

TOWERS OF HANOI: Recursive Solution (2/4) 3.15

move 7

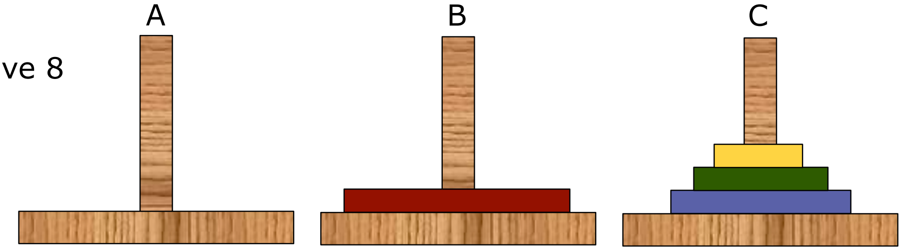


3 pegs and *n* disks

```
void MoveTower (int n, char A, char B, char C) {

        if (n > 0) {
                MoveTower(n-1,A,C,B);


        }
}
```

TOWERS OF HANOI: Recursive Solution (3/4) 3.16

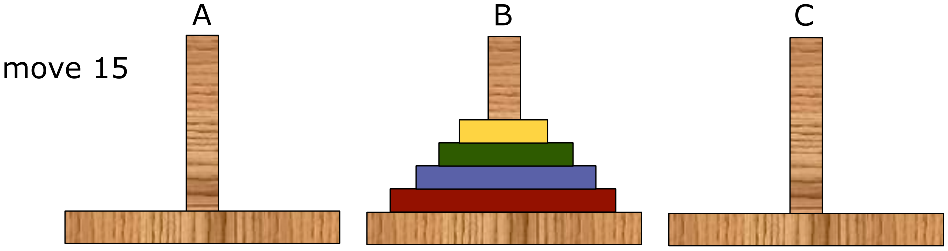
move 8



3 pegs and *n* disks

```
void MoveTower (int n, char A, char B, char C) {

        if (n > 0) {
                MoveTower(n-1,A,C,B);
                MoveDisk(A,B);

        }
}
```

TOWERS OF HANOI: Recursive Solution (4/4) 3.17



3 pegs and *n* disks

```
void MoveTower (int n, char A, char B, char C) {

        if (n > 0) {
                MoveTower(n-1,A,C,B);
                MoveDisk(A,B);
                MoveTower(n-1,C,B,A);
        }
}
```

Reading suggestion: Chapter 3 of the textbook

THE END