

# *Intro to Programming II*

## *Recursion*

Chris Brooks

Department of Computer Science

University of San Francisco

## 10-2: Recursion

---

- Recursion is a fundamental problem-solving technique
- Involves decomposing a problem into:
  - A base case that can be solved directly
  - A recursive step that indicates how to handle more complex cases.
- A common recursive example is factorial:

```
long factorial(int input)
    if (input == 1)
        return 1;
    else
        return input * factorial(input - 1);
```

## 10-3: Recursion

---

- A more interesting example is the Towers of Hanoi.
- It's hard to write an iterative program to solve this, but the recursive version is startlingly simple:

```
void towers(int ndisks, Tower startTower, Tower goalTower,
Tower tempTower)
    if (ndisks == 0)
        return;
    else
        towers(ndisks - 1, startTower, tempTower, goalTower);
        moveDisk(startTower, goalTower);
        towers(ndisks - 1, tempTower, goalTower, startTower);
```

## 10-4: Infinite recursion

---

- A common error in recursion is forgetting the base case.
- This can lead to *infinite recursion*

```
double factorial(int input)
    return input * factorial(input - 1);
```

- This will eventually have a stack overflow.

## 10-5: Exercise: Fibonacci numbers

---

- The Fibonacci numbers are defined as follows:

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n - 1) + f(n - 2)$$

- The first few numbers are 0,1,1,2,3,5,8,13,21,...
- Write a class called Fibonacci. It should have a method called `getFib(int n)` that recursively calculates the *n*th Fibonacci number, plus a main method to test it.

## 10-6: Exercise: Fibonacci numbers

---

- What is a problem with the naive way of implementing Fibonacci?
- Can you think of a way around this?
- How would you implement Fibonacci iteratively?

## 10-7: Recursion: Traversing a Maze

---

- Solving a maze is the sort of problem that requires trial-and-error.
- When you're stuck, back up and undo the last thing you did.
- This sort of approach works well with recursion.
- We'll represent the maze as a two-dimensional array.
  - 1 = clear, 0 = blocked.
- Start in the upper left, get to the lower right.

## 10-8: Exercise: Solving a maze

---

- Change the rules so that you always try to go left, then right, then up, then down.
- Write a Maze constructor that takes two arguments: row and col and generates a random maze of that size.

## 10-9: Recursion in graphics

---

- We can use recursion to easily create interesting graphical effects.
- For example, recursively tiling a surface.

## 10-10: Recursion in graphics: Exercise

---

- Add your own pictures to the applet.
- Change the applet so that the recursive part of the picture is in the lower right.

## 10-11: Fractals

---

- We can also use recursion to draw fractals
- Example: Koch snowflake
- Rule: Each line segment is replaced by a “wedge” with sides that are the same length as the replaced piece.
- As we increase the depth, it begins to look like a snowflake.

## 10-12: Koch Snowflake: exercise

---

- Change the color scheme of the applet.
- Change the default max and min values
- Change the initial triangle to have its “point” downward.