Introduction to Programming (CS 101)

Spring 2024

Lecture 11:

More about structs and recursion (recursion (recursion ...

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade and Prof. Manoj Prabhakaran

Recap (IA)

What is the output of the following program?

```
int alter(int &a) {
  return a+=2;
}

main_program {
  int a = 1;
  cout << alter(a) << endl;
}
```

1

2

3

`alter(a)` would work the same if `alter` returned `int` or `int&`. But `alter(alter(a))` would only work for the latter.

Recap (IB)

What is the output of the following program?

```
int& alter(int &a) {
  return a+=2;
}

main_program {
  int a = 1;
  alter(a) = 2;
  cout << a << endl;
}
```

1

2

3

4

A function returning a reference can act like an lvalue.

Recap (IC)

What is the output of the following program?

```
int& alter(const int &a) {
  return a+=2;
}

main_program {
  int a = 1;
  alter(a) = 2;
  cout << a << endl;
}
```

A 1

B 2

C 3

D Compiler error

Compiler error! Cannot do `a+=2` since the reference `a` is defined to be a const-qualified type in `alter`

Recap (ID)

What is the output of the following program?

```
const int& alter(int &b) {
  return (b+1);
}

main_program {
  int a = 1;
  cout << alter(a) << endl;
}
```

Compiler error

1

2

Can create a `const int&` to point to a temporary object in `b + 1`

Recap (IE)

What is the output of the following program?

```
const int& alter(const int &b) {
  return (b+1);
}

main_program {
  int a = 1;
  a = alter(a) + 1;
  cout << a << endl;
}
```

1

2

3

Compiler error

Cannot modify a `const int&` like `b`, but can directly modify the variable it points to i.e., `a`

Recap (IF)

What is the output of the following program?

```
main_program {
  float b = 4.1;
  const int &c = b;
  cout << c << endl;
}
```

1

4

Compiler error

Can assign `const int&` to a temporary object (that results from casting `4.1` to `int`)

struct variables

CS 101, 2025

Creating your own data-type using `struct`

- A structure (defined using `struct`) denotes a collection of variables
  - The variables in the collection are referred to as ***members*** of the structure

- Syntax:

```
struct structure-name {
  member-type1 member-name1 ;
  member-typen member-namen ;
};
```

  - `struct`: Predefined keyword used to define a structure
  - `structure-name`: User-defined structure name (naming rules same as for ordinary variables)
  - `member-type1 member-name1;`: Refers to a member variable's type and name

struct

- Example of a structure:

```
struct Movie {
  string title, genre;
  char rating;
  float IMDBscore, RTscore;
  bool isEnglish;
  ⋮
};
```

- Structure definition does not allocate space for the members
- `Movie m1, m2;` // this statement allocates memory for the variables `m1,m2` and their
  // respective members
- To access a structure's member, join the variable and the member name with a period. E.g., `m1.rating`, `m2.title`, etc.

```
m1.isEnglish = true; cout << m1.RTscore + m2.RTscore;
```

`struct` variables

- Rules for accessing (scope of) `struct` variables are the same as that for other primitive data types
- Lifetime of the `struct` variable is the block in which it's defined

```
int main() {
  struct Student { int age, year; string branch; bool checkrcd;};
  Student s1 = {.checkrcd = false};
  cin >> s1.age >> s1.year >> s1.branch;
  if(s1.year >= 2023 && s1.age >= 21) {
    s1.checkrcd = true;
    Student s1copy = {s1.age, s1.year, s1.branch, false};
  }
  cout << s1copy.age << endl; //Error; cannot find s1copy
}
```

Recursion

CS 101, 2025

Designing a recursive function

- **Recursion**: When a function calls itself as part of its execution
  - Carefully think through the *base cases* i.e., the simplest non-recursive parts of the problem
  - What is the recursive step? Break the problem down into simpler instances of the same problem that eventually lead to a base case.
  - Check: Are all possible cases handled?

- Example: Compute the factorial of a non-negative number, recursively (without any loops)
  - Base case?
    - `if(n == 0) return 1;` or a more general `if(n <= 1) return 1;`
  - Recursive part?
    - We know for $n \geq 1$, $n! = n \times (n-1)!$

Designing a recursive function

- **Recursion**: When a function calls itself as part of its execution
  - Carefully think through the *base cases* i.e., the simplest non-recursive parts of the problem
  - What is the recursive step? Break the problem down into simpler instances of the same problem that eventually lead to a base case.
  - Check: Are all possible cases handled?

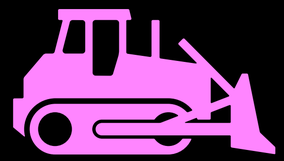
- Example: Compute the factorial of a non-negative number, recursively (without any loops)

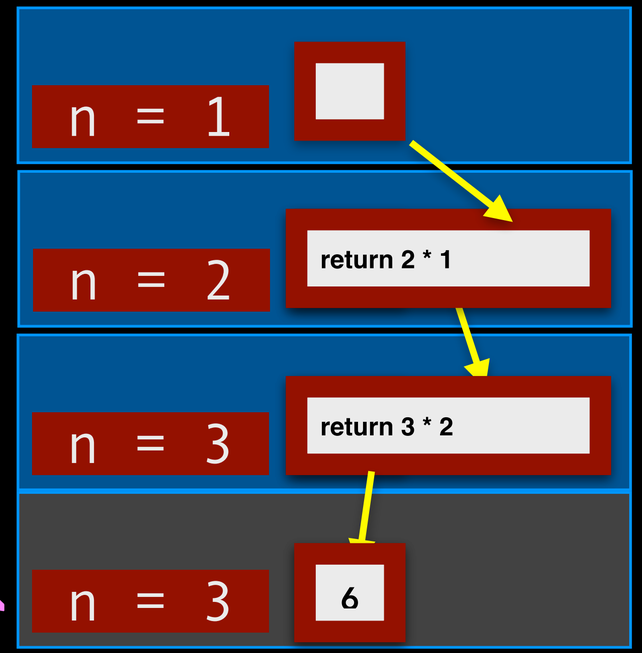
```
int factorial(int n) {
  if(n <= 1) return 1;
  else return n*factorial(n-1);
}
```

Visualizing a recursive function's calls on the stack

```
int factorial(int n) {
  if(n <= 1) return 1;
  else return n*factorial(n-1);
}
```

```
main_program {
  int n;
  cin >> n; //n = 3
  cout << factorial(n);
}
```

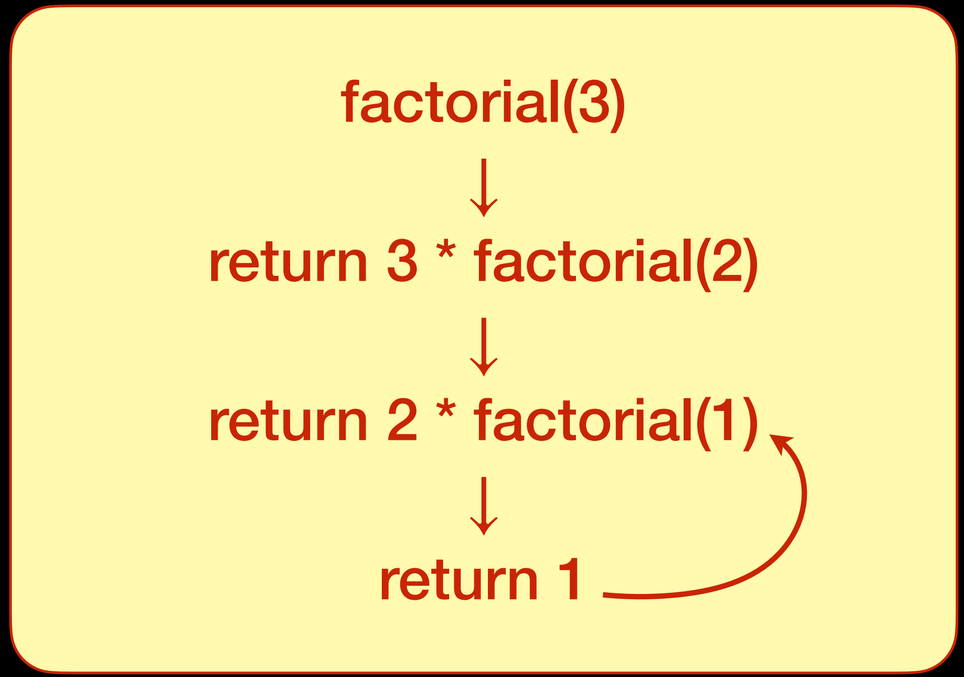




Demonstration of recursion

```
int factorial(int n) {
  if(n <= 1) return 1;
  else return n*factorial(n-1);
}
```

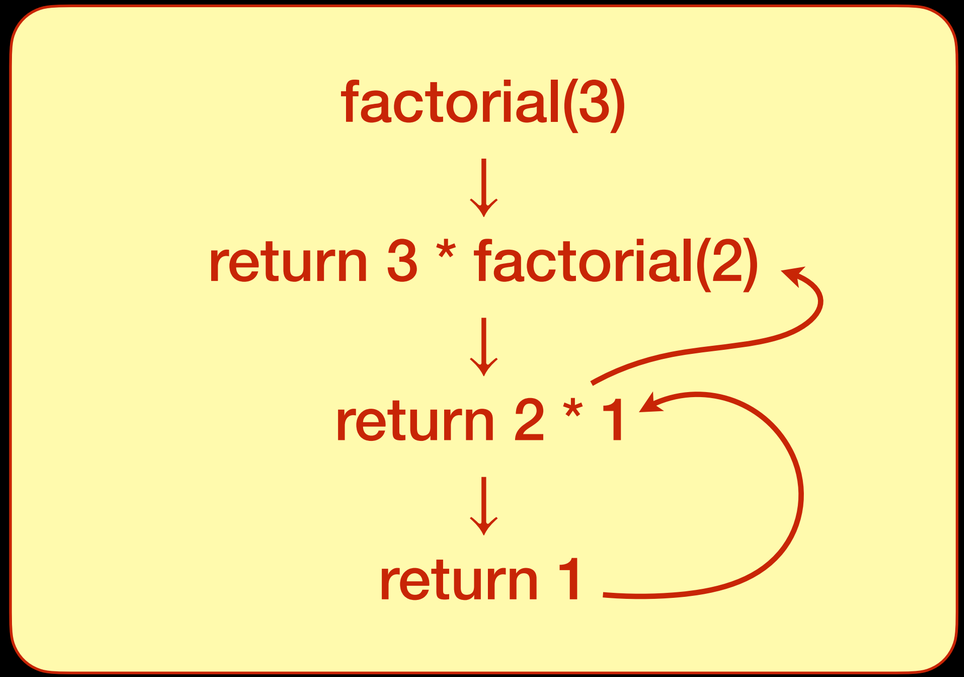
```
main_program {
  int n;
  cin >> n; //n = 3
  cout << factorial(n);
}
```



Demonstration of recursion

```
int factorial(int n) {
  if(n <= 1) return 1;
  else return n*factorial(n-1);
}
```

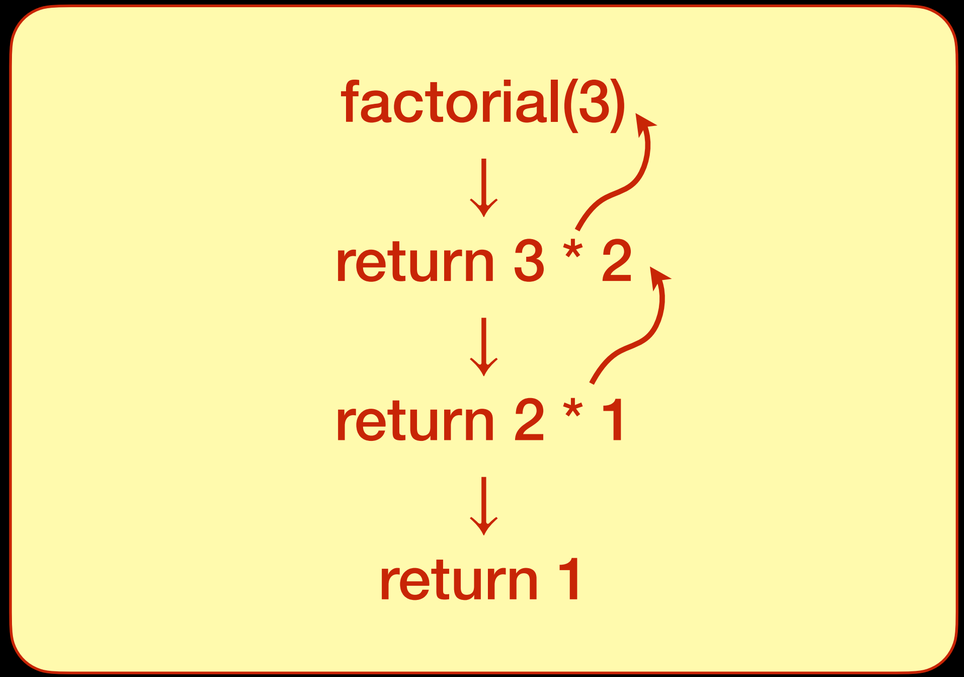
```
main_program {
  int n;
  cin >> n; //n = 3
  cout << factorial(n);
}
```



Demonstration of recursion

```
int factorial(int n) {
  if(n <= 1) return 1;
  else return n*factorial(n-1);
}
```

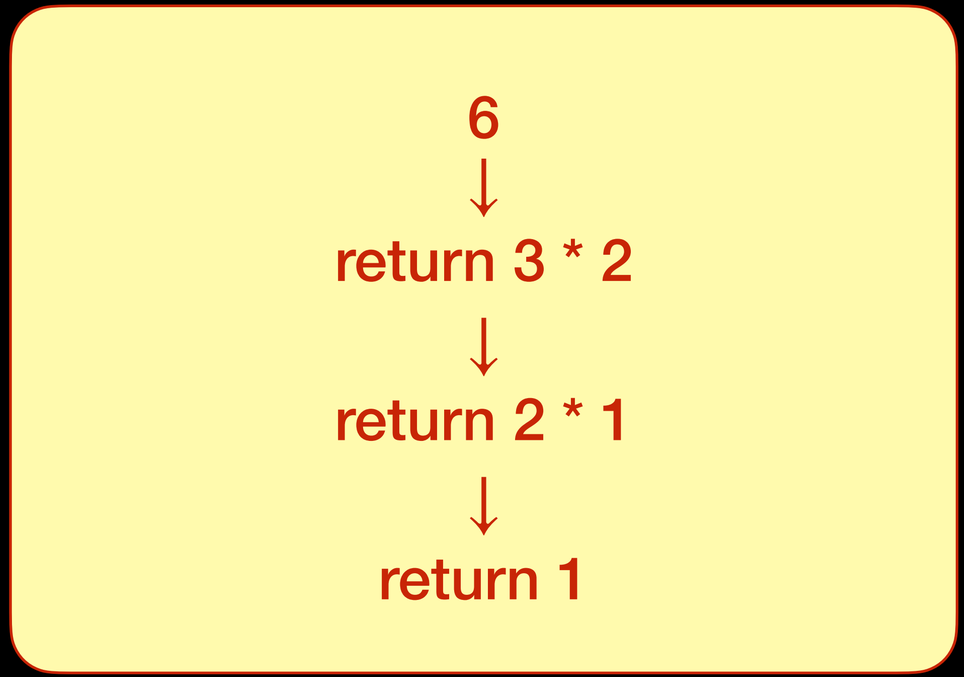
```
main_program {
  int n;
  cin >> n; //n = 3
  cout << factorial(n);
}
```



Demonstration of recursion

```
int factorial(int n) {
  if(n <= 1) return 1;
  else return n*factorial(n-1);
}
```

```
main_program {
  int n;
  cin >> n; //n = 3
  cout << factorial(n);
}
```

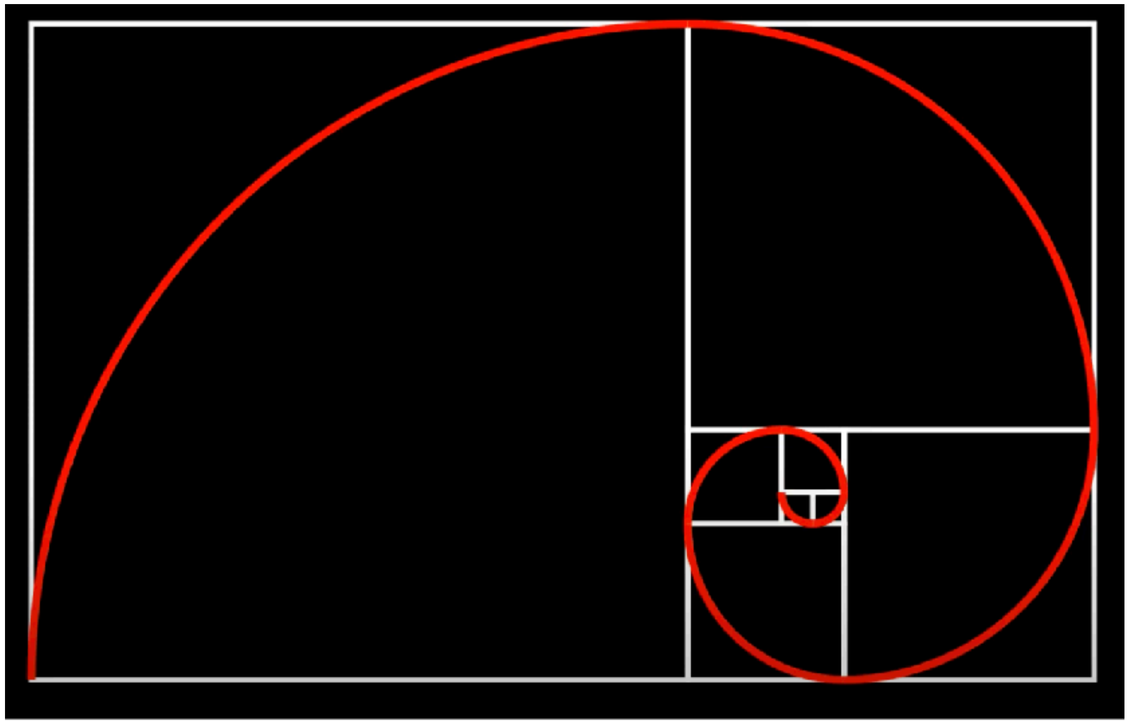


Fibonacci sequence

CS 101, 2025

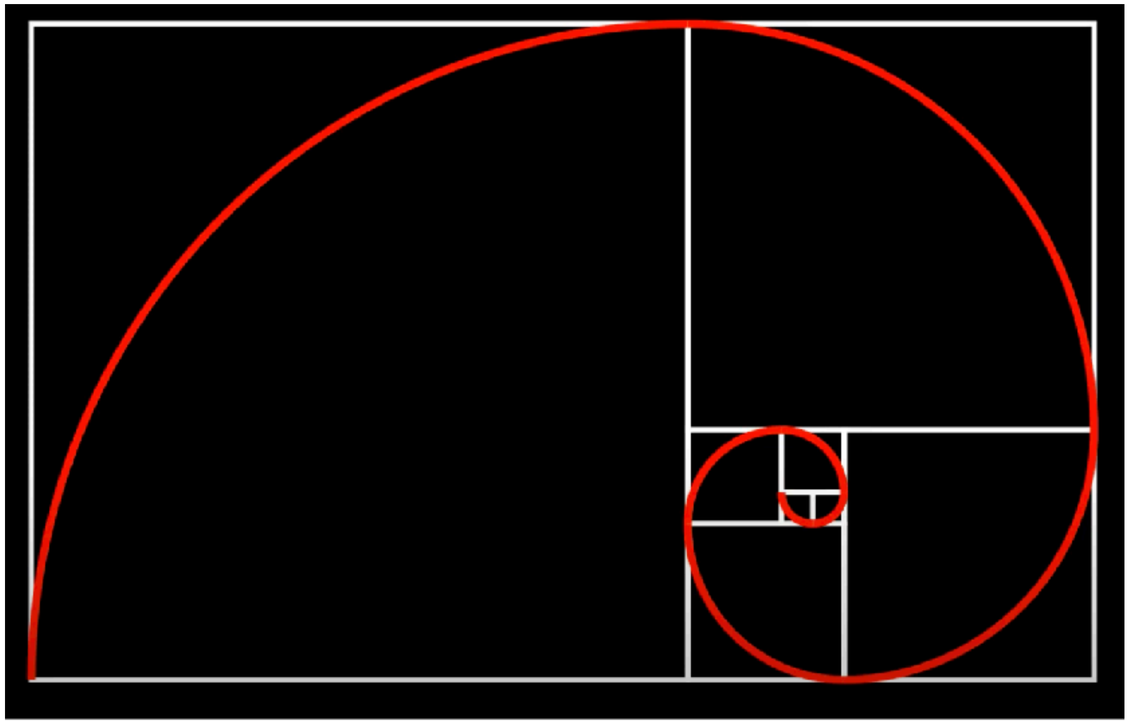
Fibonacci sequence

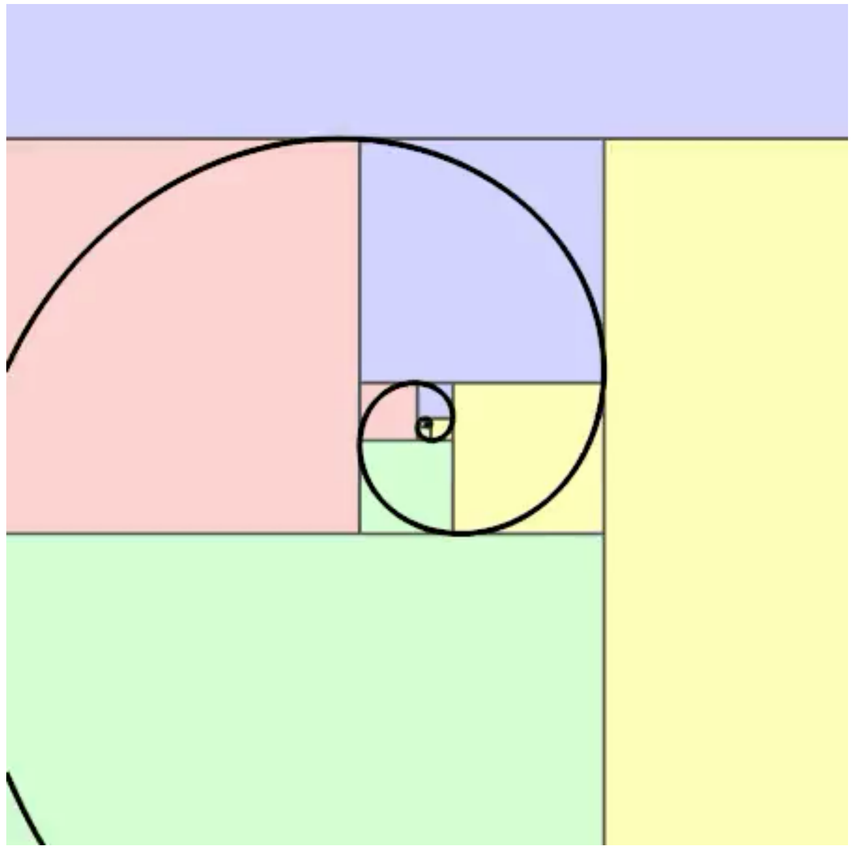
- Each element is the sum of two elements before it
- Starting from 0 and 1, the sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...



Fibonacci sequence

- Each element is the sum of two elements before it
- Starting from 0 and 1, the sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...





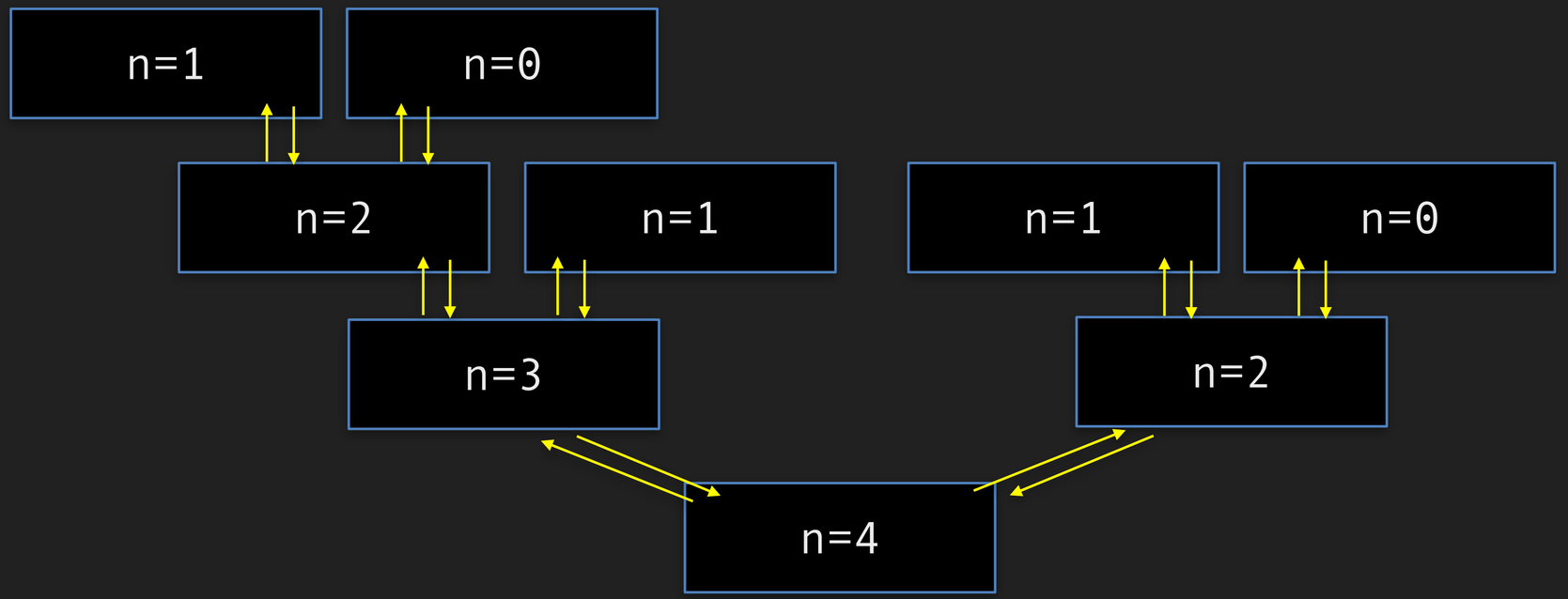
Video derived from https://en.wikipedia.org/wiki/Golden_spiral#/media/File:GoldenSpiralLogarithmic_color_in.gif

Example: Fibonacci Sequence

A very inefficient implementation!

```
int Fibonacci(unsigned int n) {
  if(n==0) return 0;
  if(n==1) return 1;
  return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Base cases



Combinations

CS 101, 2025

n choose k

- Write a program to recursively compute the number of (unordered) ways to choose k objects from a set of n distinct objects. Relevant formula: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

- Single out an object (say **X**). Now think about the number of ways in which you want to:
  - Mandatorily include **X** in your selection → $\binom{n-1}{k-1}$
  - Mandatorily exclude **X** from your selection → $\binom{n-1}{k}$

- Base cases: $\binom{n}{0} = 1$ and $\binom{n}{n} = 1$

n choose k

- Write a program to recursively compute the number of (unordered) ways to choose k objects from a set of n distinct objects. Relevant formula: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

```
int combination(int n, int k) {
  if(k == 0 || k == n) return 1;
  return (combination(n-1,k-1) + combination(n-1,k));
}
```

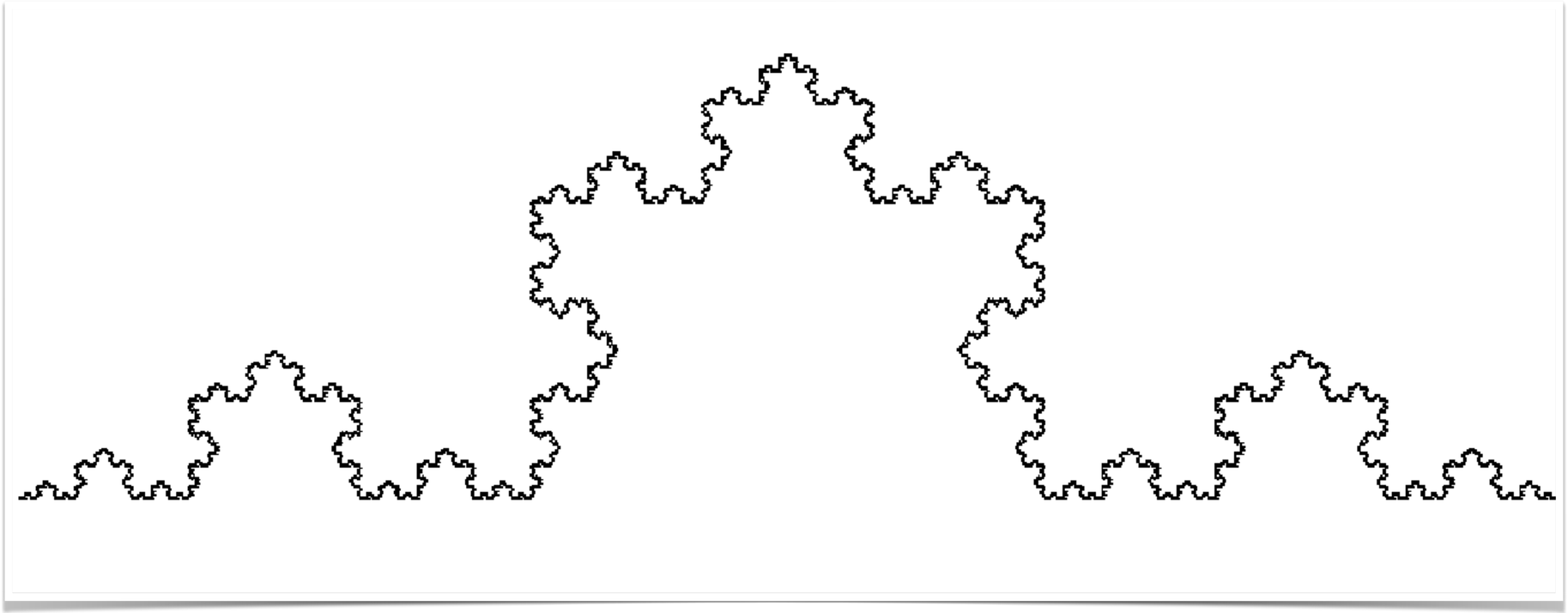
Fractal

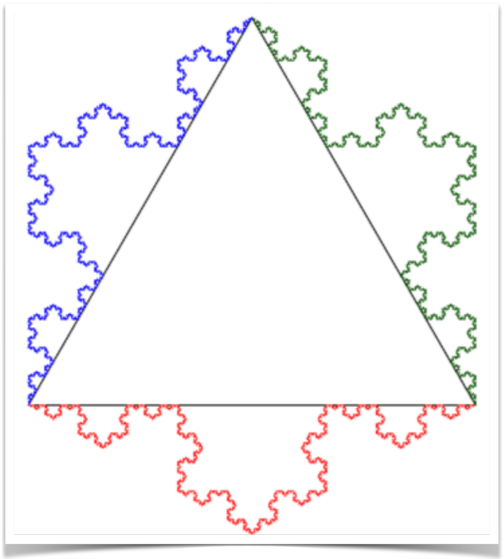
CS 101, 2025

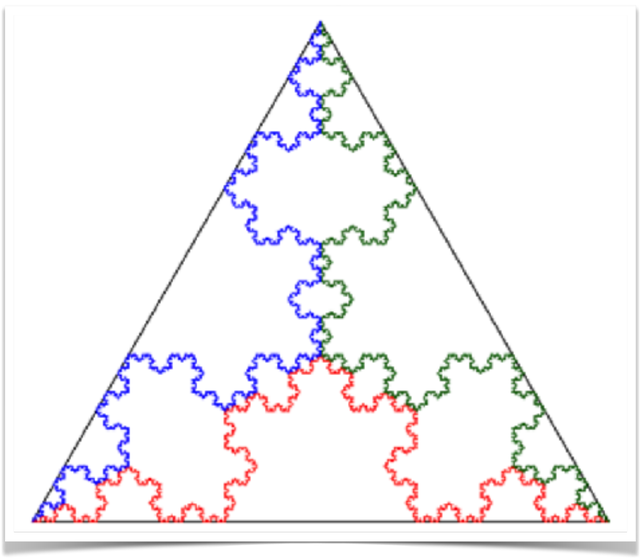
"Beautiful, damn hard, increasingly useful. That's fractals."

- Benoit Mandelbrot

Koch curve



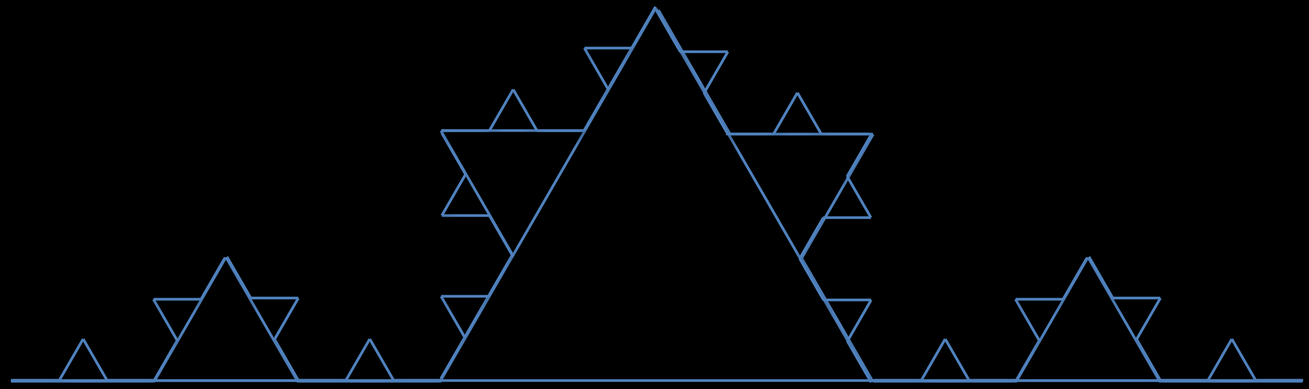




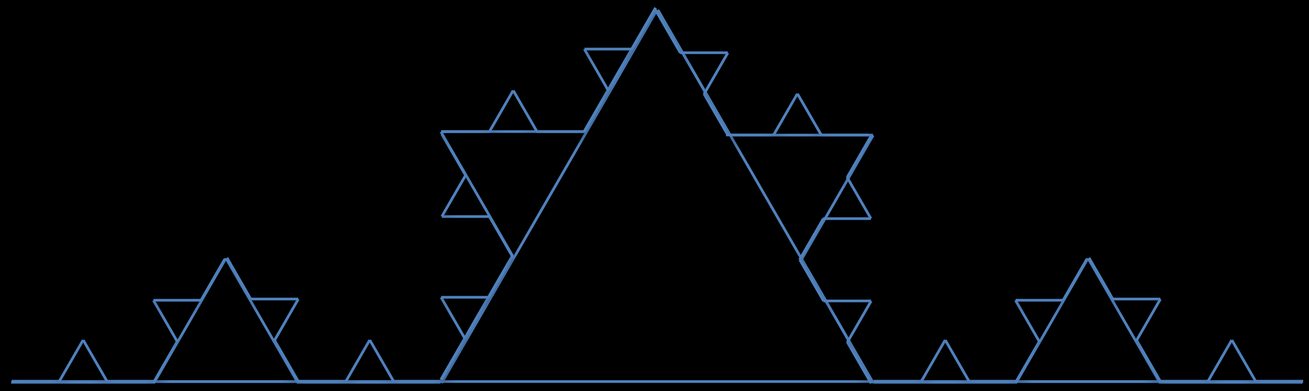
Koch snowflake and
Koch anti-snowflake

Last row images from: https://larryriddle.agnesscott.org/ifs/kcurve/kcurve.htm

Unpacking the Koch curve



Building a recursive function to draw a Koch curve

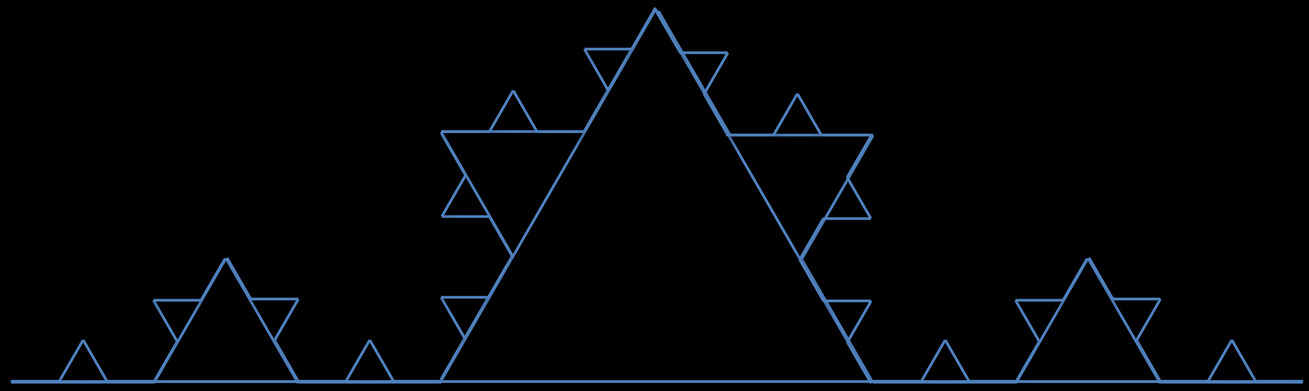


Recursive call

```
void draw(double L, int level) {
  if (level == 0) { forward(L); return; }
  if (level == 1) {
    draw(L/3,0);  left(60);
    draw(L/3,0);  right(120);
    draw(L/3,0);  left(60);
    draw(L/3,0);
  }
  if (level == 2) {
    draw(L/3,1);  left(60);
    draw(L/3,1);  right(120);
    draw(L/3,1);  left(60);
    draw(L/3,1);
  }
  if (level == 3) ...
}
```

Building a recursive function to draw a Koch curve

Base case:
Ensures that the recursion is not infinite



```
void draw(double L, int level) {
  if (level == 0) { forward(L); return; }
  draw(L/3, level-1); left(60);
  draw(L/3, level-1); right(120);
  draw(L/3, level-1); left(60);
  draw(L/3, level-1);
}
```

Demo in class

Tower of Hanoi

CS 101, 2025

Tower of Hanoi

- Famous puzzle consisting of three rods and disks of varying diameters. Disks are stacked on one rod, ordered bottom-to-top from largest to smallest (in diameter). The goal is to move all the disks from one (**source**) rod to one of the other two rods (i.e., a **target** rod, and the remaining rod is referred to as a **spare** rod), while satisfying the following rules:
  1. Only one disk is moved at a time.
  2. A valid move is taking the topmost disk from a stack and placing it on another stack or rod.
  3. A disk cannot be placed on top of a smaller disk.

Video derived from https://en.wikipedia.org/wiki/File:Tower_of_Hanoi_4.gif

Tower of Hanoi: Building the recursive solution

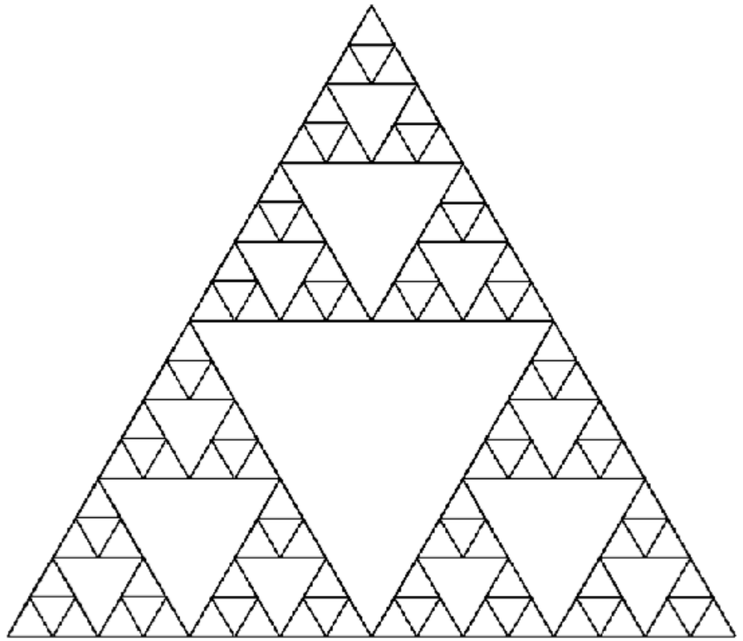
- Say we know how to solve Tower of Hanoi for `n-1` disks using `towerofHanoi(n-1, source, target, spare)`. Then, to solve it for `n` disks:
  1. Move `n-1` disks from the source rod to the spare rod.
     That is, call `towerofHanoi(n-1, source, spare, target)` //recursive call
  2. Disk `n` is the only one remaining on the source rod. Move it from source to target rod.
     This is a base case.
  3. Move `n-1` disks back from the spare rod to the target rod.
     That is, call `towerofHanoi(n-1, spare, target, source)` //recursive call

Demo in class

Video derived from https://en.wikipedia.org/wiki/File:Tower_of_Hanoi_4.gif

Homework Exercises

Write a recursive program to draw the following fractal with repeating equilateral triangles



Write a recursive function to print out all permutations of a string. Assume the string is a single word. If there are duplicates, you can print them all out.

If the input is "`out`", your code prints:

```
out
otu
uot
uto
tou
tuo
```

Fractal: https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle

Next class: Arrays

CS 101, 2025