

YEAH!

Meta Academy

Sahil Chopra - 1.26.2016

Adapted from SLs Rishi Bedi & Audrey Ho



META

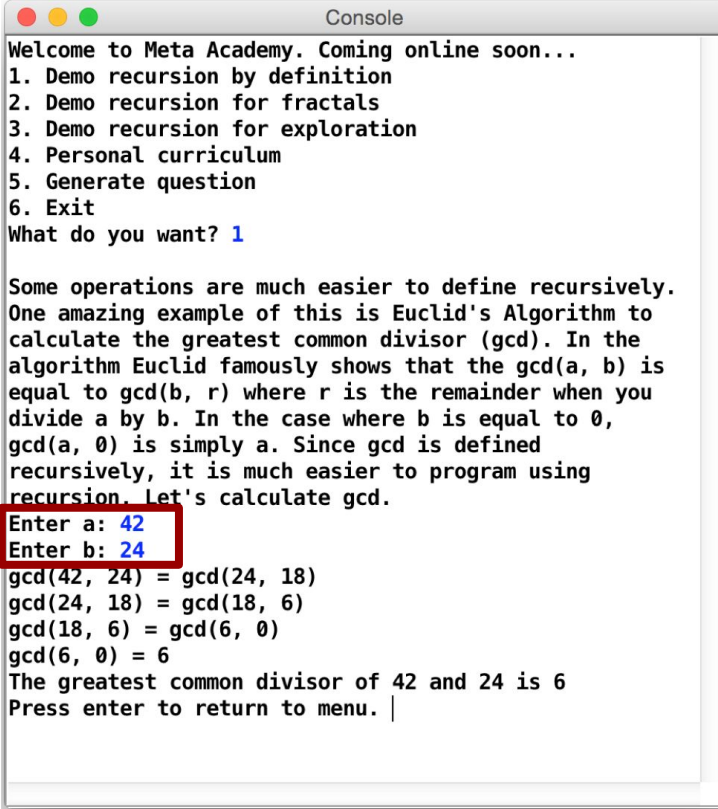
ACADEMY

Create a Teaching Tool For Recursion

1. Demo Recursion By Definition
2. Demo Recursion By Fractals
3. Demo Recursion For Exploration
4. Personal Curriculum
5. Generate Question

Milestone 1: GCD (Recursive By Definition)

```
int gcd (int a, int b):  
  
    // RECURSIVE ALGORITHM  
  
    if x b = 0 gcd(a, b) = a  
  
    else      gcd(a, b) = gcd(b, a%b)
```



The screenshot shows a terminal window titled "Console" with the following content:

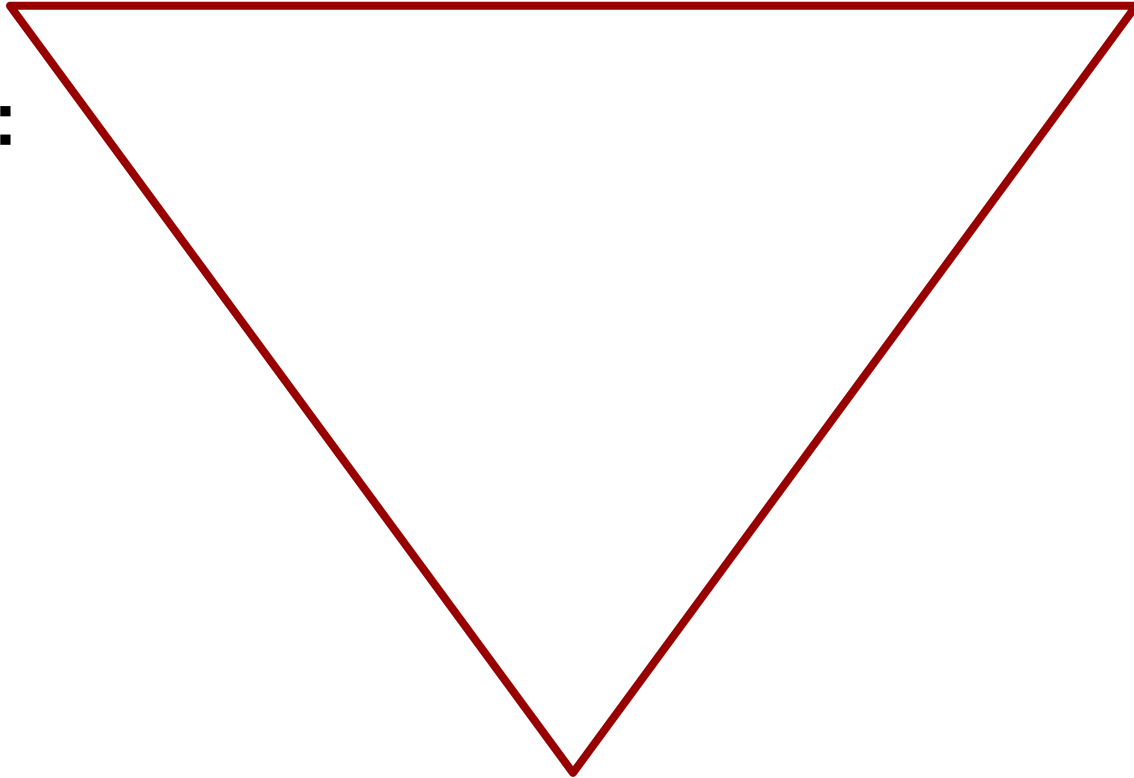
```
Welcome to Meta Academy. Coming online soon...  
1. Demo recursion by definition  
2. Demo recursion for fractals  
3. Demo recursion for exploration  
4. Personal curriculum  
5. Generate question  
6. Exit  
What do you want? 1  
  
Some operations are much easier to define recursively.  
One amazing example of this is Euclid's Algorithm to  
calculate the greatest common divisor (gcd). In the  
algorithm Euclid famously shows that the gcd(a, b) is  
equal to gcd(b, r) where r is the remainder when you  
divide a by b. In the case where b is equal to 0,  
gcd(a, 0) is simply a. Since gcd is defined  
recursively, it is much easier to program using  
recursion. Let's calculate gcd.  
Enter a: 42  
Enter b: 24  
gcd(42, 24) = gcd(24, 18)  
gcd(24, 18) = gcd(18, 6)  
gcd(18, 6) = gcd(6, 0)  
gcd(6, 0) = 6  
The greatest common divisor of 42 and 24 is 6  
Press enter to return to menu. |
```

Milestone 1: GCD (Recursive By Definition)

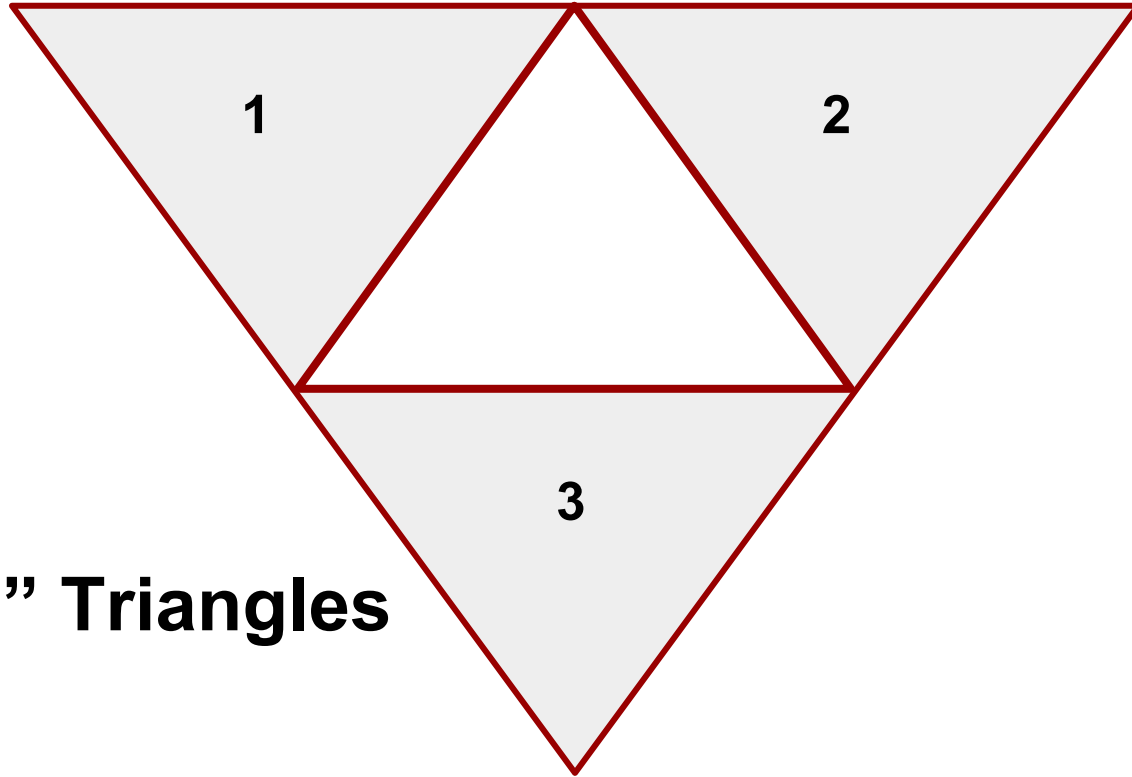
- 1) `gcd(42, 24)` // Apply $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$ since $b \neq 0$
- 2) `gcd(24, 18)` // Apply $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$ since $b \neq 0$
- 3) `gcd(18, 6)` // Apply $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$ since $b \neq 0$
- 4) `gcd(6, 0)` // Apply $\text{gcd}(a, b) = a$ (Base Case)

Milestone 2: Sierpinski (Recursion by Fractals)

Order 1:

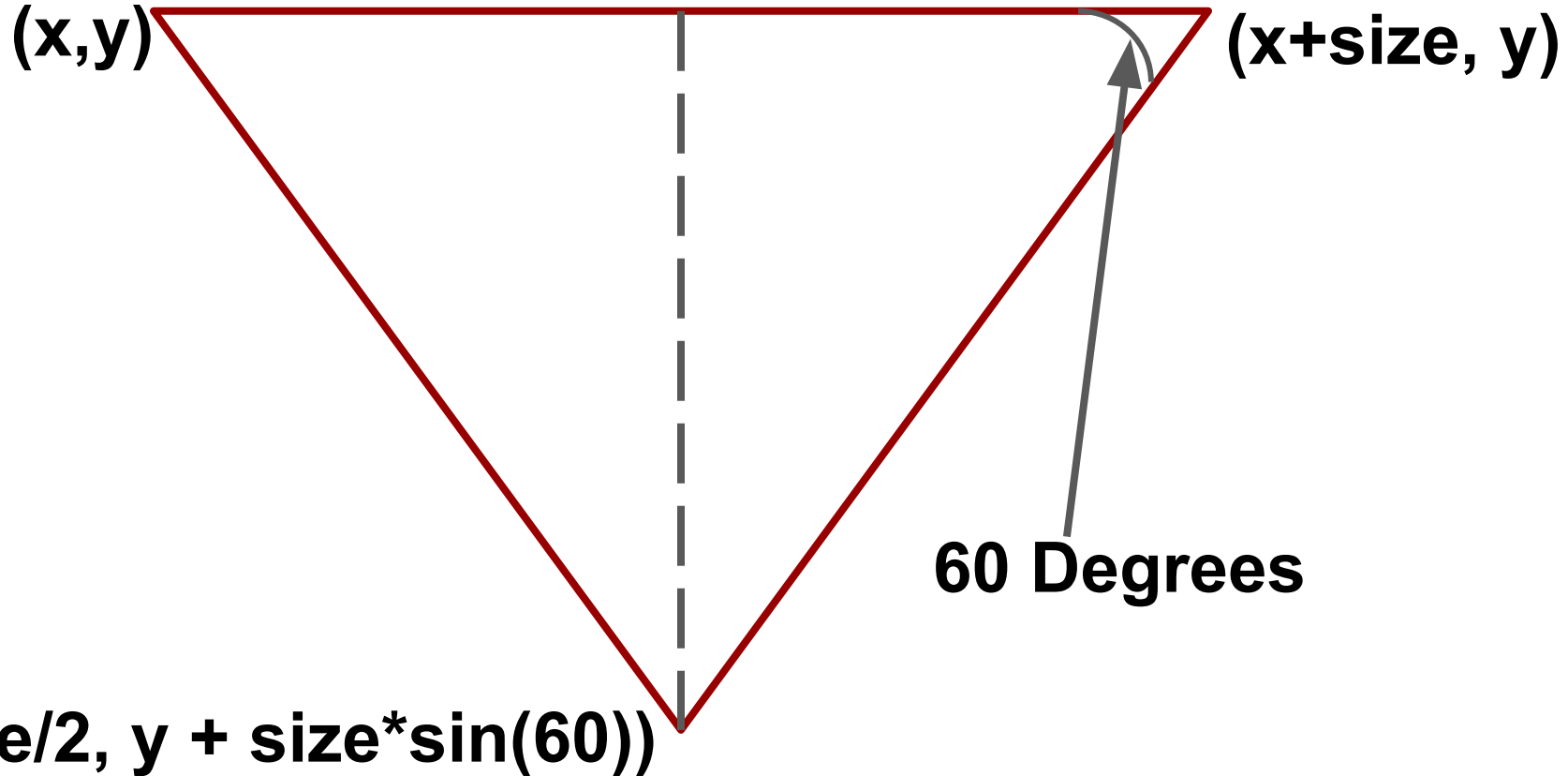


Milestone 2: Serpinski (Recursion by Fractals)



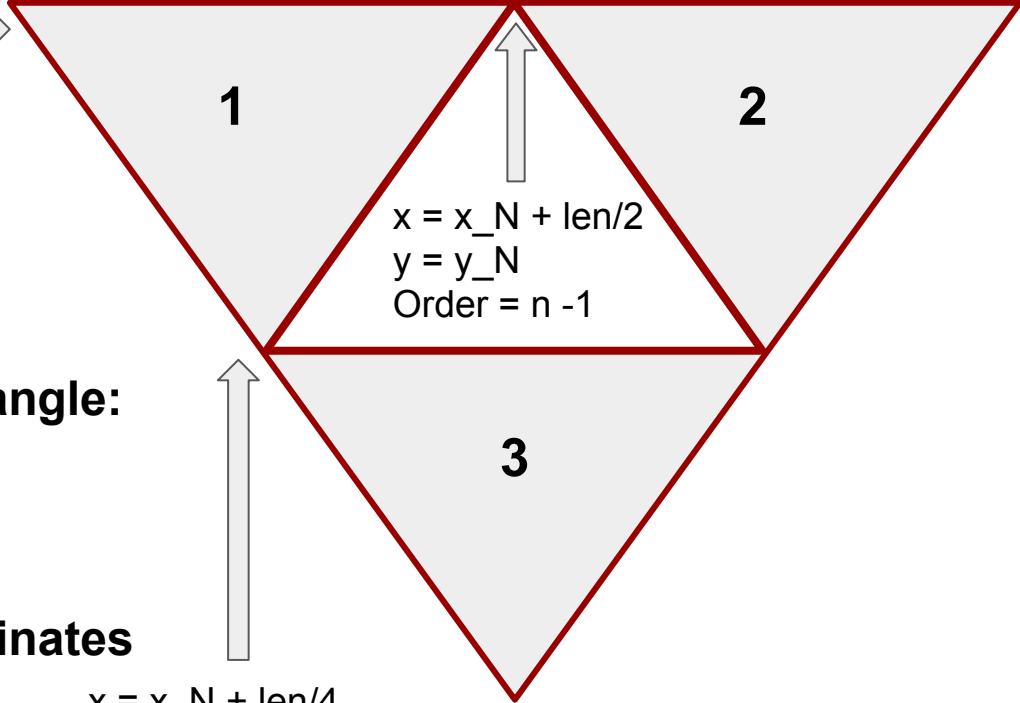
**Order 2 =
3 “Order-1” Triangles**

Milestone 2: Sierpinski (Recursion by Fractals)



Milestone 2: Serpinski (Recursion by Fractals)

$x = x_N$
 $y = y_N$
Order = $n - 1$



$x = x_N + \text{len}/2$
 $y = y_N$
Order = $n - 1$

Recursive Step: Drawing Order-N Triangle:

- 3 Triangles of Order N-1
- Each Tri. Has $\frac{1}{2}$ a Side Length
- (x_N, y_N) are (x,y) anchor coordinates

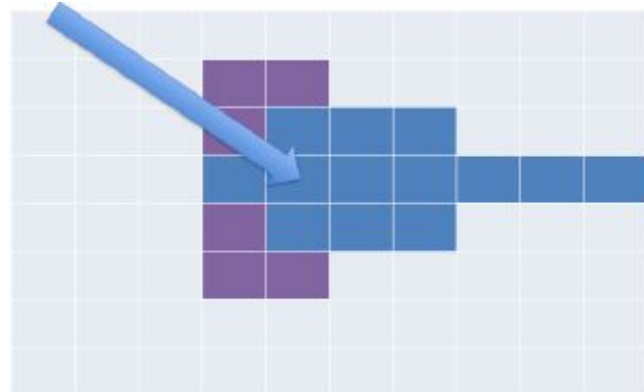
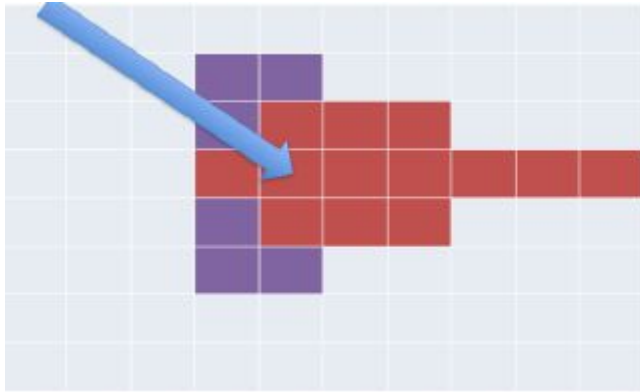
for Order-N Tri.

$x = x_N + \text{len}/4$
 $y = y_N + \text{len} * \sin(60)/2$
Order = $n - 1$

Milestone 3: Flood Fill (Recursion by Exploration)

```
int floodFill(GBufferedImage& image, int x, int y, int color)
```

```
// e.g. floodFill(image, 4, 3, blue)
```



Milestone 3: Flood Fill (Recursion by Exploration)

- Only fill boxes of old color that we clicked on ...
- How do we keep track of the old color?
 - Helper function lets us keep track of more variables

```
int floodFillHelper(image, x, y, newColor, oldColor)
```

- Recursion: What options can we explore for each pixel?

Milestone 4: Personalized Curriculum

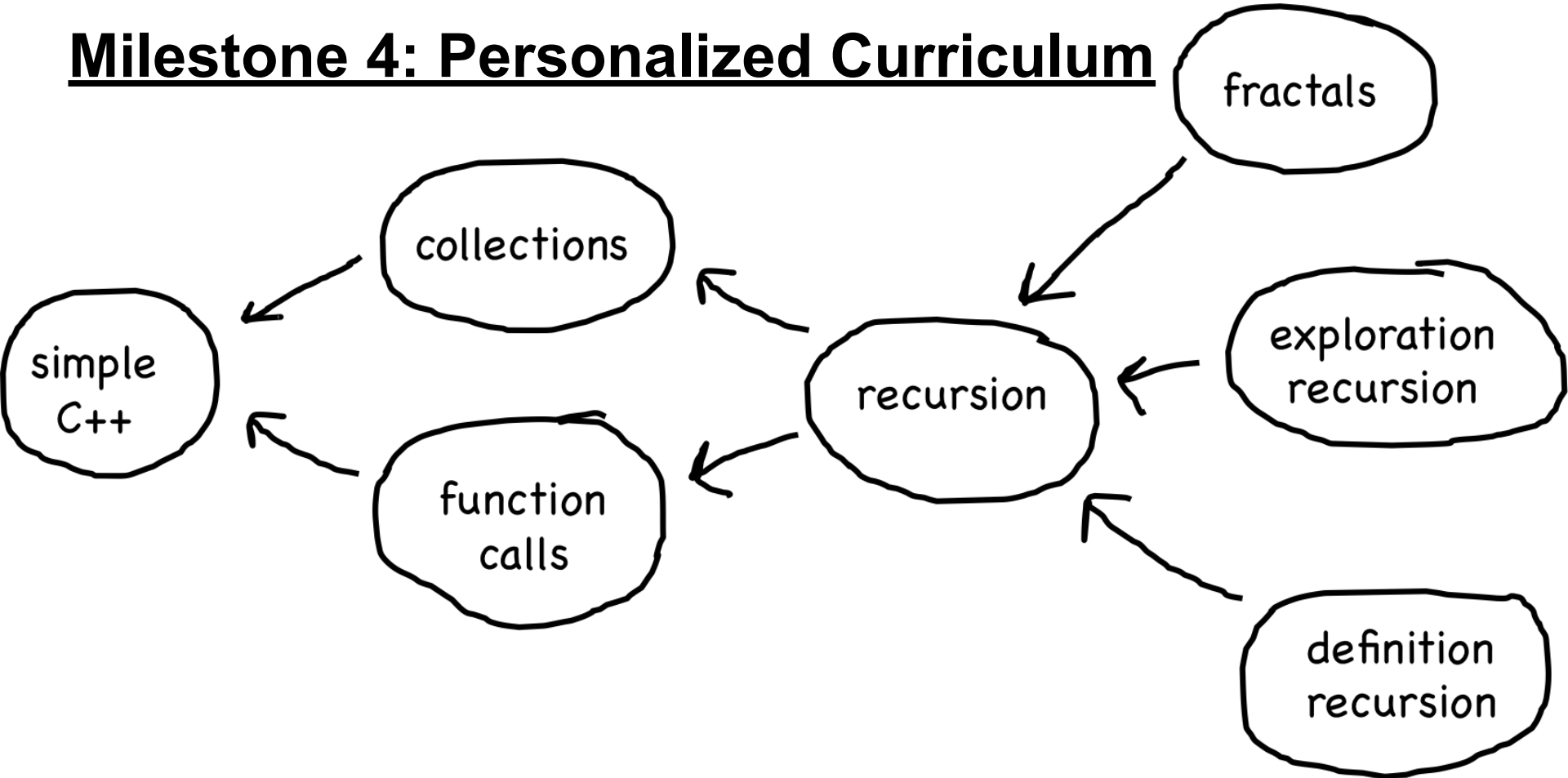
```
Console
Welcome to Meta Academy. Coming online soon...
1. Demo recursion by definition
2. Demo recursion for fractals
3. Demo recursion for exploration
4. Personal curriculum
5. Generate question
6. Exit
What do you want? 4
What course? recursion
Enter the concept the student would like to learn (or ?
to list concepts): ?
collections
definitionRecursion
explorationRecursion
fractals
functionCalls
recursion
Enter the concept the student would like to learn (or ?
to list concepts): explorationRecursion
The order you should learn concepts:
simpleC++
functionCalls
collections
recursion
explorationRecursion

Press enter to return to menu. |
```

```
Console
Welcome to Meta Academy. Coming online soon...
1. Demo recursion by definition
2. Demo recursion for fractals
3. Demo recursion for exploration
4. Personal curriculum
5. Generate question
6. Exit
What do you want? 4
What course? cs106b
Enter the concept the student would like to learn (or ?
to list concepts): dijkstra
The order you should learn concepts:
simpleC++
abstraction
controlFlow
functionCalls
passByReference
maps
sets
pointers
graphs
pQueues
bigO
queues
BFS
dijkstra

Press enter to return to menu.
```

Milestone 4: Personalized Curriculum



Milestone 4: Personalized Curriculum

```
allPrereqsOfConcept(prereqMap, concept) {  
    it's direct prerequisites and  
    for (childConcept : direct prerequisites) {  
        allPrereqsOfConcept(prereqMap, childConcept)  
    }  
}
```

```
"fractals"           → ["recursion"]  
"explorationRecursion" → ["recursion"]  
"definitionRecursion" → ["recursion"]  
"recursion"         → ["collections", "functionCalls"]
```

Q: Make sure to avoid repeating the same prerequisite multiple times in your list. How we can store prerequisites that we have already listed?

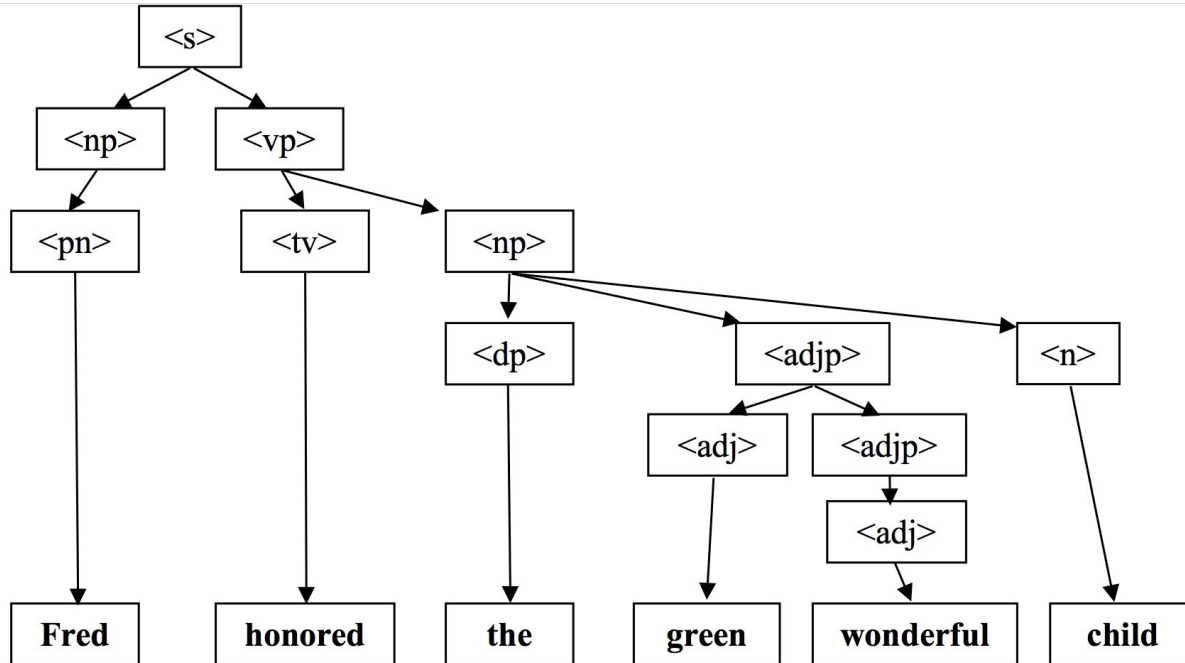
Q: Where should cout << statements go, in order to print the curriculum in the correct order?

Milestone 5: Generate Question

<s>:<np> <vp> // Non Terminal
<np>:<dp> <adjp> <n>|<pn> // Non Terminal
<dp>: the | a // Non Terminal
<adjp>:<adj>|<adj> <adjp> // Non Terminal
<adj>:big | fat | green | wonderful | faulty | subliminal | pretentious // Terminal
<n>:dog | cat | man | university | father | mother | child | television // Terminal
<pn>:John | Jane | Sally | Spot | Fred | Elmo // Terminal
<vp>:<tv> <np>|<iv> // Non Terminal
<tv>:hit | honored | kissed | helped // Terminal
<iv>:died | collapsed | laughed | wept // Terminal

Milestone 5: Generate Question

- Can Recursively Expand Non-Terminals Until Terminal Reached



Random expansion from sentence.txt grammar for symbol "<s>"

Milestone 5: Generate Question

- **Base Case: Terminal Reached**
- **Recursive Step:**
 - **Get the rules from the map for your current symbol**
 - **Get a random rule from those rules**
 - **For each symbol in that rule, recurse adding the result to an output string separated by a space**

Milestone 5: Generate Question

- Question grammars will have a non-terminal <QUESTION>
- To loop over a string expansion you will need to process the string one "token" at a time where a token could be a non-terminal or a terminal.

```
TokenScanner scanner(production);  
while (scanner.hasMoreTokens()) {  
    string token = scanner.nextToken();  
    // do something with token  
}
```