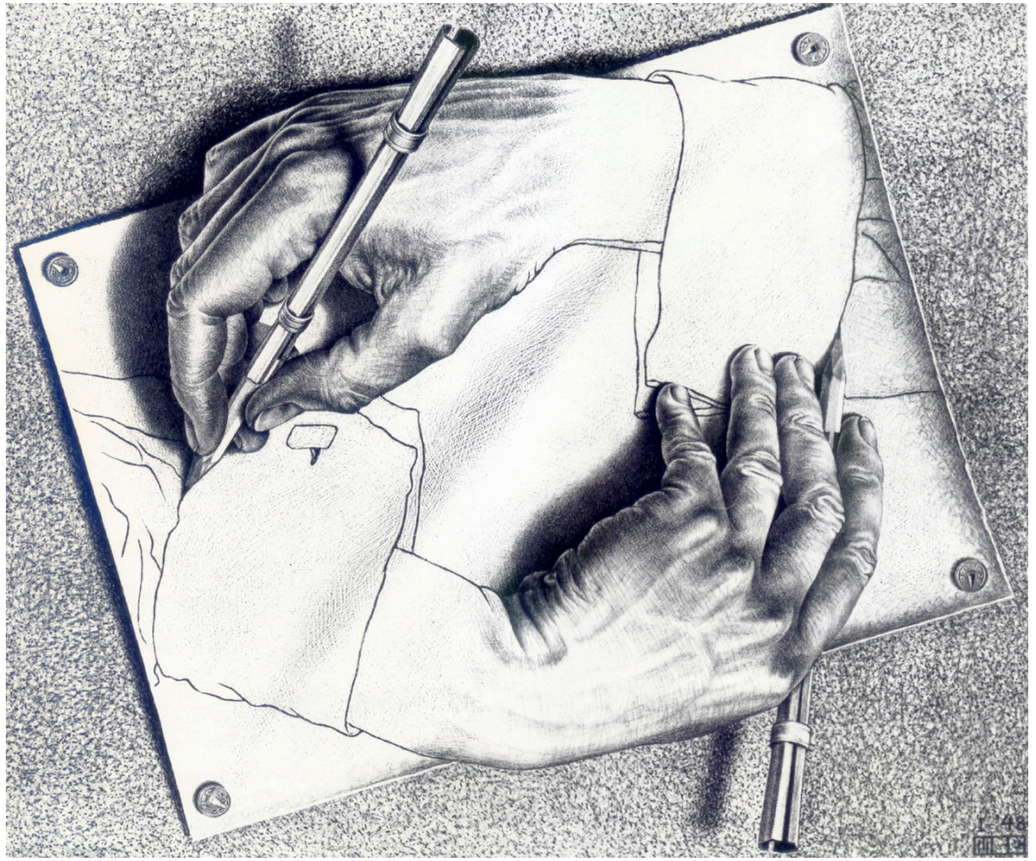
Essential Computing 1

Recursion



M.C. Escher

A method that calls itself

Death to the machine.

```
void repeat(){
  repeat();
}
```

A method that calls itself

Use a test for when to stop.

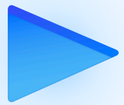
```
void repeat( int count ){
  if( count > 1 ) repeat( count - 1 );
}
```

```java
public static void main( String[] args ){
 repeat( 3 );
}
```

```java
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```


```
public static void main( String[] args ){
  repeat( 3 );
}
```

```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

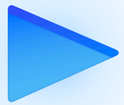


Stack

main | args


```
public static void main( String[] args ){
  repeat( 3 );
}
```

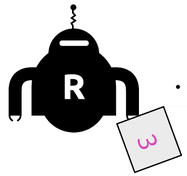
```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```



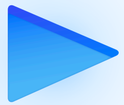
Stack

main	args
repeat	count

```
public static void main( String[] args ){
 repeat( 3 );
}
```




```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```



Stack

main	args
repeat	count 3

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main | args

repeat | count 3

Console

```
3
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3

Console

```
3
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2

Console

```
3
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2

Console

```
3
2
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2

Console

```
3
2
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2
repeat	count 1

Console

```
3
2
```


```java
public static void main( String[] args ){
 repeat( 3 );
}
```

```java
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2
repeat	count 1

Console

```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2
repeat	count 1

Console

```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```

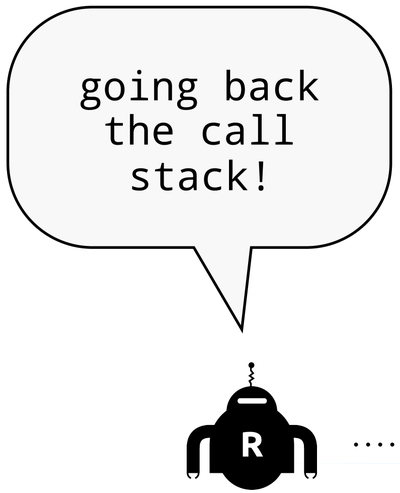

```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2
repeat	count 1

Console

```
3
2
1
```




```
public static void main( String[] args ){
 repeat( 3 );
}
```

```
static void repeat( int count ){
   System.out.println( count );
   if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2

Console

```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
   System.out.println( count );
   if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3
repeat	count 2

Console

```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
   System.out.println( count );
   if( count > 1 ) repeat( count - 1 );
}
```

Stack

main	args
repeat	count 3

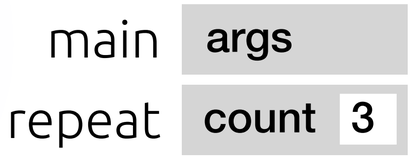
Console

```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
   System.out.println( count );
   if( count > 1 ) repeat( count - 1 );
}
```

Stack



Console

```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```


```
static void repeat( int count ){
   System.out.println( count );
   if( count > 1 ) repeat( count - 1 );
}
```

Stack

main | args


```
public static void main( String[] args ){
 repeat( 3 );
}
```

Console

```
3
2
1
```

```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

Stack

main | args

Console

```
3
2
1
```

end of program

```
public static void main( String[] args ){
  repeat( 3 );
}
```

```
static void repeat( int count ){
   System.out.println( count );
   if( count > 1 ) repeat( count - 1 );
}
```

Console

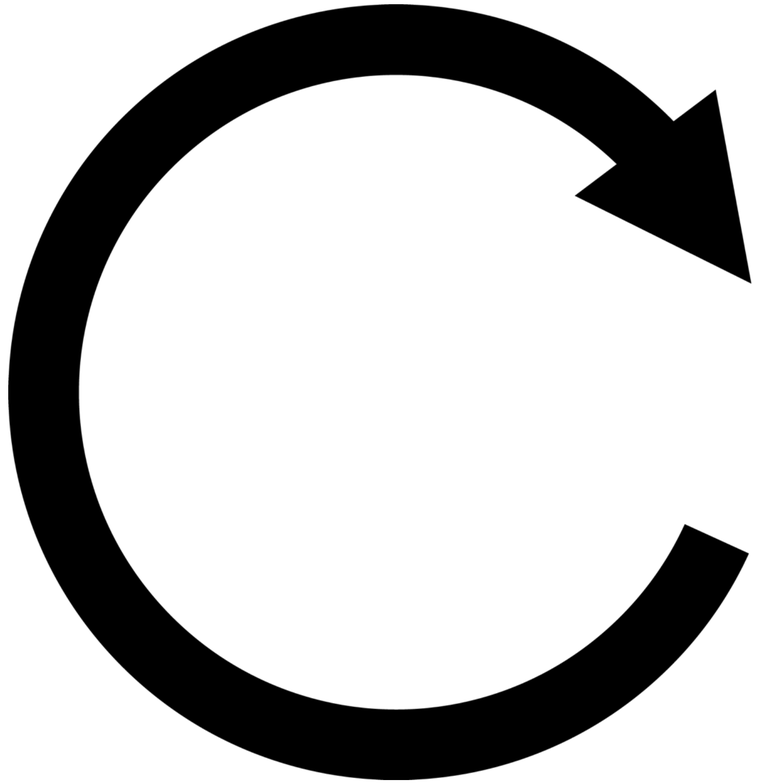
```
3
2
1
```

```
public static void main( String[] args ){
 repeat( 3 );
}
```

```
static void repeat( int count ){
  System.out.println( count );
  if( count > 1 ) repeat( count - 1 );
}
```

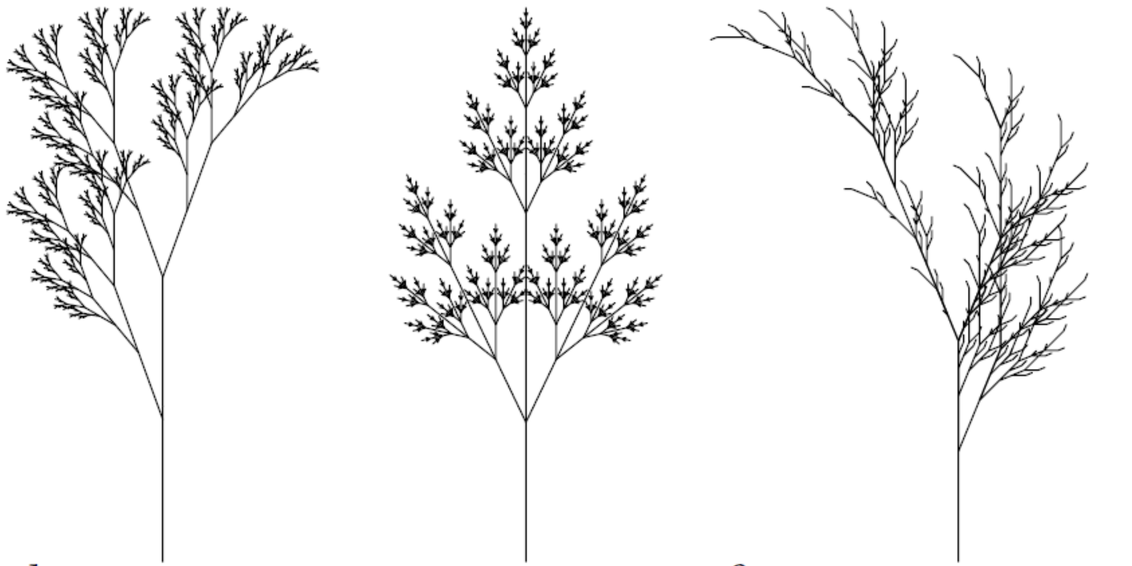
Examples of use

Not efficient for simple looping because we are accumulating a call stack (filling memory).



Examples of use

L-systems.
An algorithm for generating trees.



d
n=7, δ=20°
X
X → F[+X]F[-X]+X
F → FF

e
n=7, δ=25.7°
X
X → F[+X][-X]FX
F → FF

f
n=5, δ=22.5°
X
X → F-[[X]+X]+F[+FX]-X
F → FF

Figure 1.24: Examples of plant-like structures generated by bracketed OL-systems. L-systems (a), (b) and (c) are edge-rewriting, while (d), (e) and (f) are node-rewriting.

Examples of use

Traversing any branching tree structure