Recursion in TurtleWorld

Spirals, Trees, and Snowflakes

Friday, October 12, 2007

CS111 Computer Programming

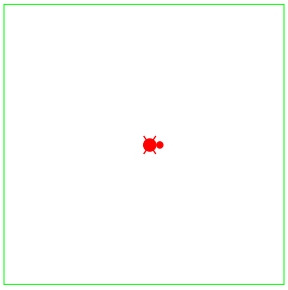
Department of Computer Science
Wellesley College

Honey, I shrunk the Buggle

- Turtles are like Buggles, only smaller* and more carefree (no grid).
- To create a new Turtle

```
new Turtle();
```

- At birth, Turtles are centered, pointing EAST, pen** down, and red.



* So small, in fact, that you cannot see them.

** Turtles have pens rather than brushes.

Turtle Commands

Go Forward:	public void fd (double dist);
Go Backward:	public void bd (double dist);
Turn Left:	public void lt (double angle);
Turn Right:	public void rt (double angle);
Pen Up:	public void pu();
Pen Down:	public void pd();
Get Heading:	public double getHeading();
Set Heading:	public void setHeading (double newHeading);
Get Color:	public Color getColor();
Set Color:	public void setColor (Color newColor);
Get Position:	public double getX(); public double getY();
Set Position:	public void double newX); public void setY(double newY);
	public void setPosition(double newX, double newY);
Go Home (to point (0,0), facing east, pen down):	public void home ();

Note: When a double parameter is specified, Java will automatically convert an int argument to a double. E.g. fd(36); is treated like fd(36.0);

What's a double?

It's a double-precision floating point number. E.g.:

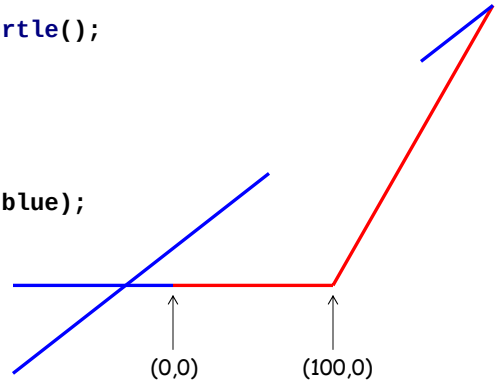
```
double x = 3.72;
x + 1.234  // Value = 4.954
x + 2      // Value = 5.72 (2 automatically converted to 2.0 first)
x * 2      // Value = 7.44 (2 automatically converted to 2.0 first)
x / 2      // Value = 1.86 (2 automatically converted to 2.0 first)

(int) x     // Value = 3 (int cast:  converts to integer by truncation)
Math.round(x)         // Value = 4 (rounds to nearest integer)
Math.floor(x)         // Value = 3 (rounds toward negative infinity)
Math.floor(-5.12)     // Value = -6 (rounds toward negative infinity)
Math.ceiling(x)       // Value = 4 (rounds toward positive infinity)
Math.ceiling(-5.12)   // Value = -5 (rounds toward positive infinity)

(double) 17    // Value = 17.0 (double cast:  converts to a double)
```

Be Careful with Division

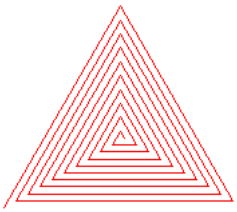
```
7/2                       // Value = 3 (integer division)

7.0/2                     // Value = 3.5 (floating point division)
7/2.0                     // Value = 3.5 (floating point division)
7.0/2.0                   // Value = 3.5 (floating point division)
((double) 7)/2            // Value = 3.5 (floating point division)
7/((double) 2)            // Value = 3.5 (floating point division)
((double) 7)/((double) 2) // Value = 3.5 (floating point division)
(double) 7/2              // Value = ((double) 7)/2 = 3.5
                          //   (floating point division)

(double) (7/2)            // Value = 3.0 (result of integer division
                          //              converted to double)
```


Myrtle the Turtle Goes on a Hike

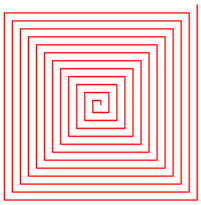
```
public void run() {

     Turtle myrtle = new Turtle();
     myrtle.fd(100);
     myrtle.lt(60);
     myrtle.fd(200);
     myrtle.rt(22.5);
     myrtle.setColor(Color.blue);
     myrtle.bd(56.78);
     myrtle.pu();
     myrtle.bd(120);
     myrtle.pd();
     myrtle.bd(200);
     myrtle.home();
     myrtle.bd(100);
}
```



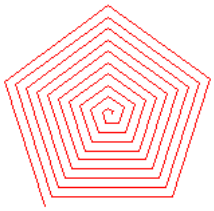
Spiro the Spiraling Turtle



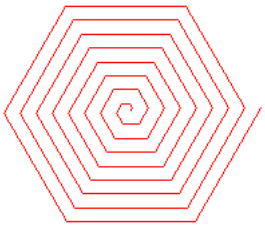
spiro.spiral(30,120,0,6);



spiro.spiral(50,90,0,3);



spiro.spiral(50,72,0,2);



spiro.spiral(40,60,0,6);

spiral(steps, angle, length, increment);

E.g., spiral(100, 90, 0, 3);

length

100 steps remain

angle

```
fd(0);
lt(90);
```

increment

99 steps remain

```
fd(0+3);
lt(90);
```

98 steps remain

```
fd(3+3);
lt(90);
```

97 steps remain

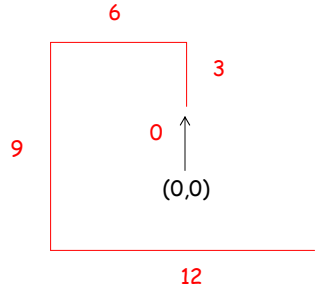
```
fd(6+3);
lt(90);
```

96 steps remain

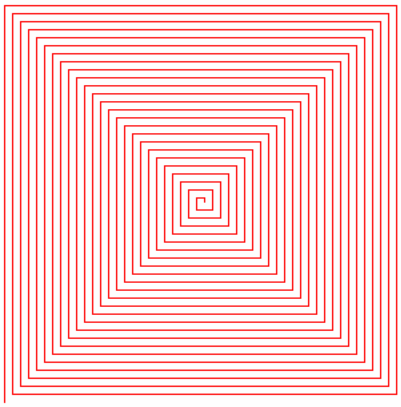
```
fd(9+3);
lt(90);
```

95 steps remain

. . .

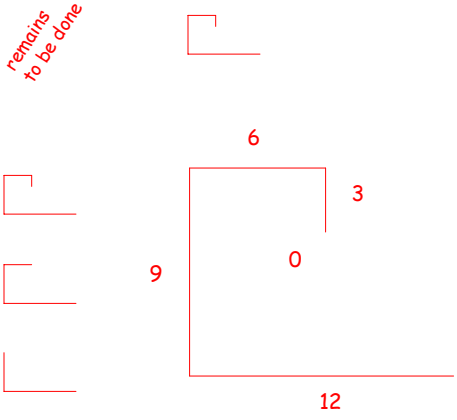


After 100 Steps:



Designing a Java spiral() Method

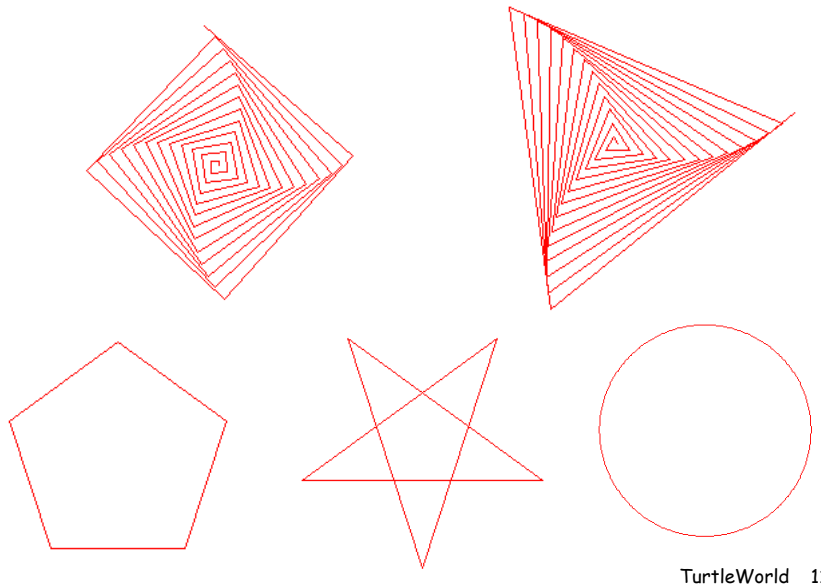
```
                steps  angle  length  increment   remains to be done
spiral(5, 90, 0, 3);
  fd(0);
  lt(90);
  spiral(4,90,3,3);
    fd(3);
    lt(90);
    spiral(3,90,6,3);
      fd(6);
      lt(90);
      spiral(2,90,9,3);
        fd(9);
        lt(90);
        spiral(1,90,12,3);
          fd(12);
          lt(90);
          spiral(0,90,12,3);
```



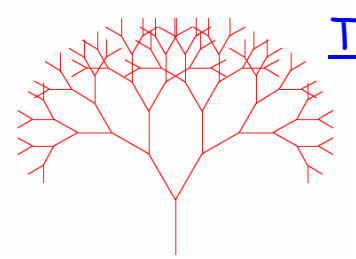
Let's Write the spiral() Method in Java

```
public void spiral(int steps, double angle,
                   double length, double increment) {




}
```

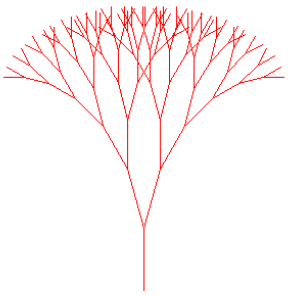

How Do You Draw These with spiral()?



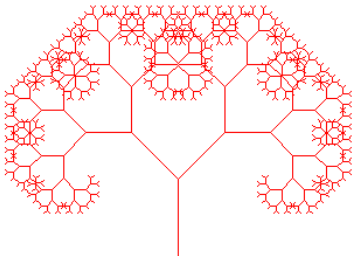
TreeWorld



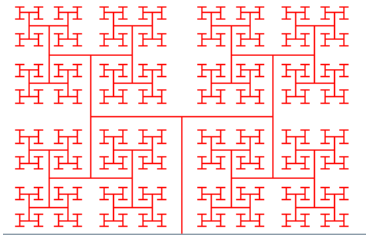
trina.tree(7,75,30,0.8)



trina.tree(7,75,15,0.8)

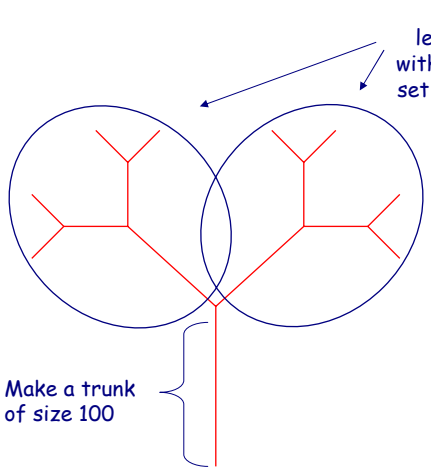


trina.tree(10,80,45,0.7)



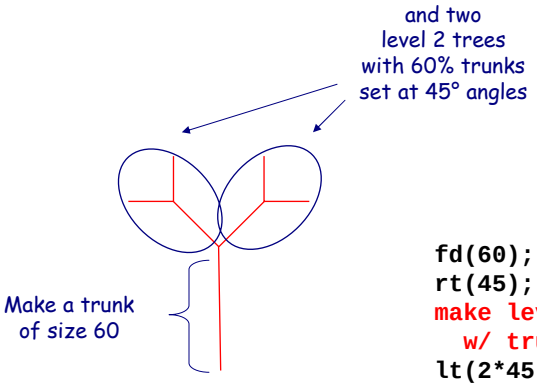
trina.tree(10,100,90,0.68)

How to make a level 4 tree



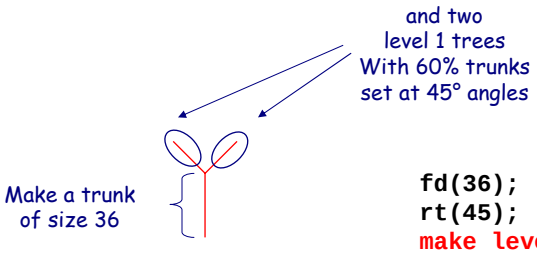

```
fd(100);
rt(45);
make level 3 tree
  w/ trunk 0.6*100
lt(2*45);
make level 3 tree
  w/ trunk 0.6*100
rt(45);
bd(100);
```

How to make a level 3 tree



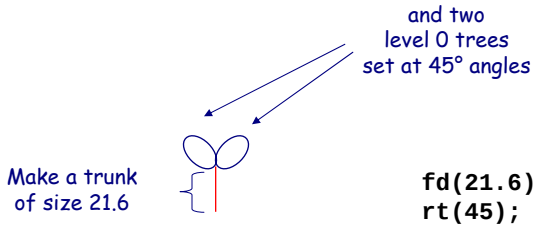

```
fd(60);
rt(45);
make level 2 tree
  w/ trunk 0.6*60
lt(2*45);
make level 2 tree
  w/ trunk 0.6*60
rt(45);
bd(60);
```


How to make a level 2 tree




```
fd(36);
rt(45);
make level 1 tree
  w/ trunk 0.6*36
lt(2*45);
make level 1 tree
  w/ trunk 0.6*36
rt(45);
bd(36);
```

How to make a level 1 tree




```
fd(21.6);
rt(45);
make level 0 tree
  w/ trunk 0.6*21.6
lt(2*45);
make level 0 tree
  w/ trunk 0.6*21.6
rt(45);
bd(21.6);
```


How to make a level 0 tree

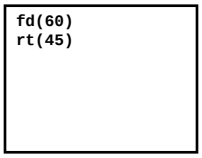
Do nothing!

Let's Write tree() in Java

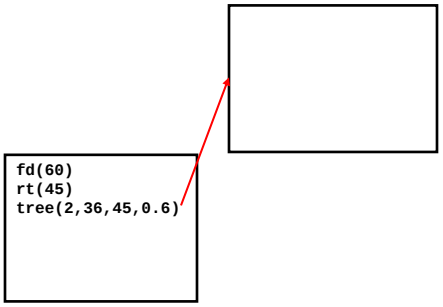
```
public void tree(int levels, double length,
                 double angle, double shrink) {




}
```


Tracing the invocation of `tree(3, 60, 45, 0.6)`

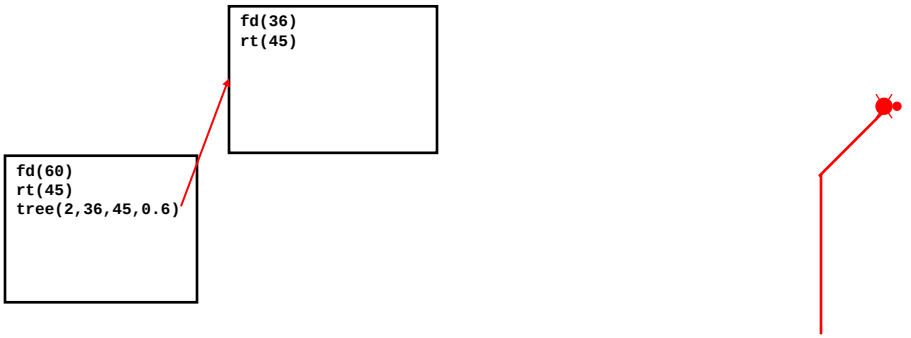
Draw trunk and turn to draw level 2 tree



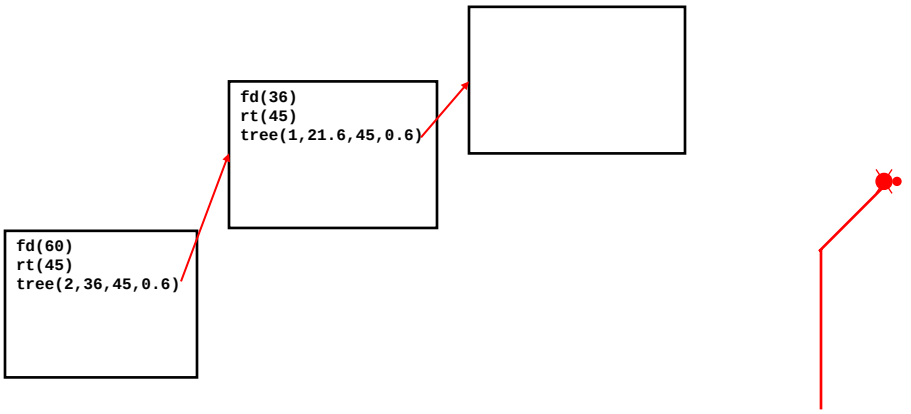
Begin recursive invocation to draw level 2 tree



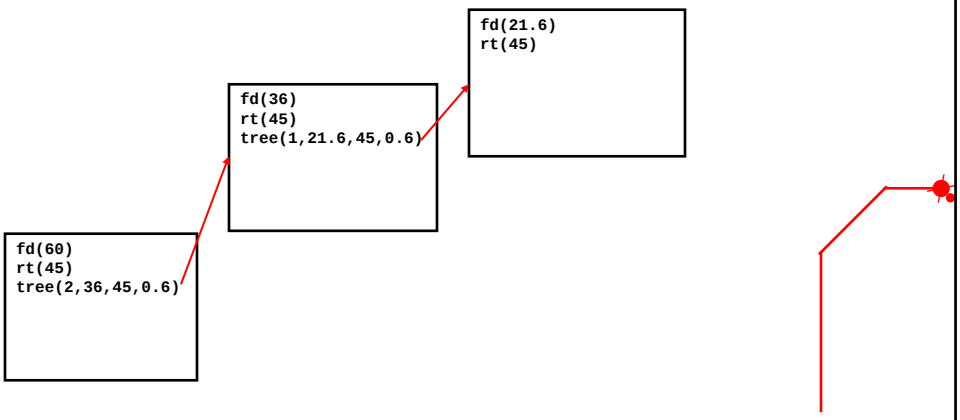
Draw trunk and turn to draw level 1 tree



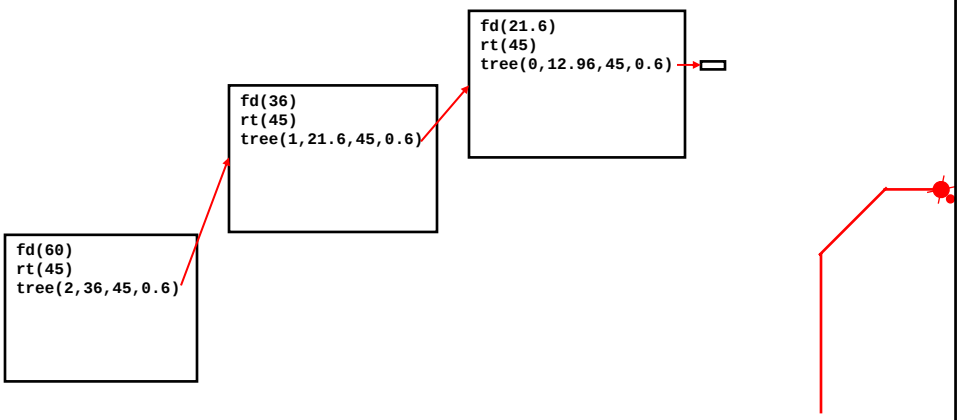
Begin recursive invocation to draw level 1 tree



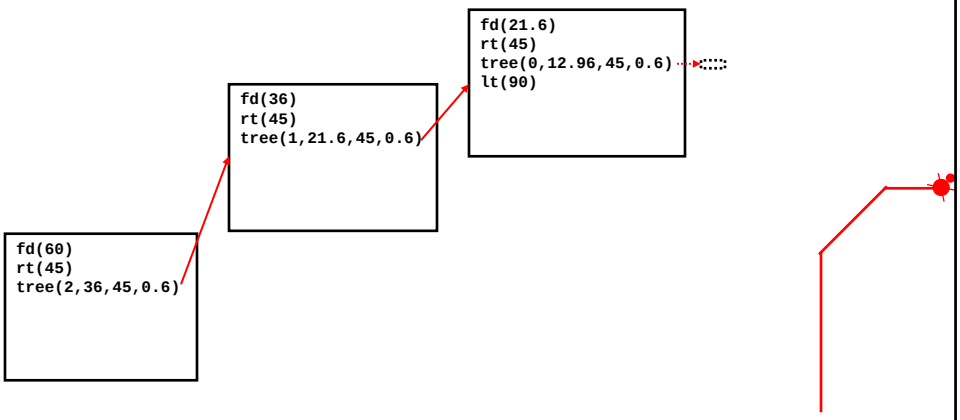
Draw trunk and turn to draw level 0 tree



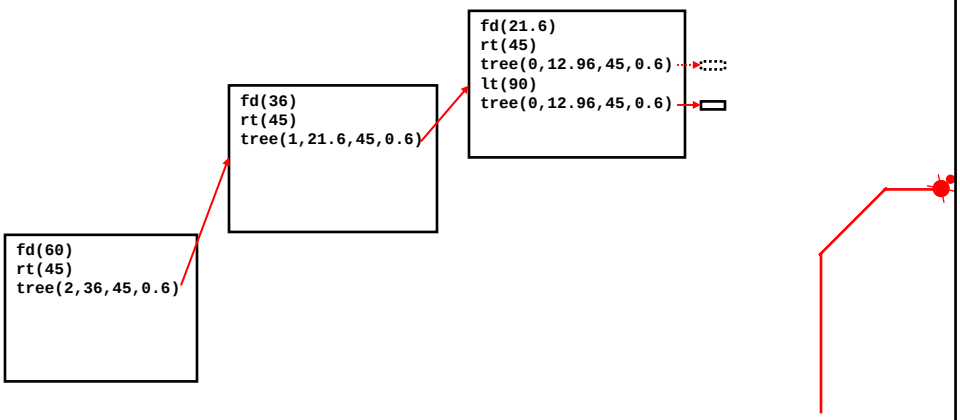
Begin recursive invocation to draw level 0 tree



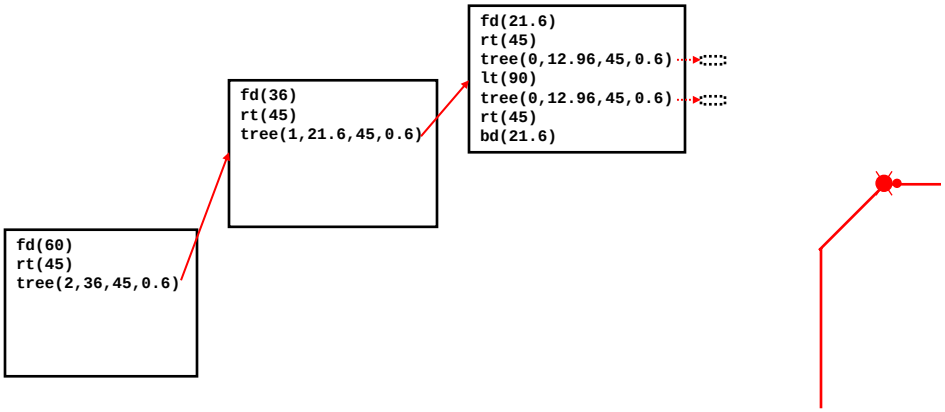
Complete level 0 tree and turn to draw another level 0 tree



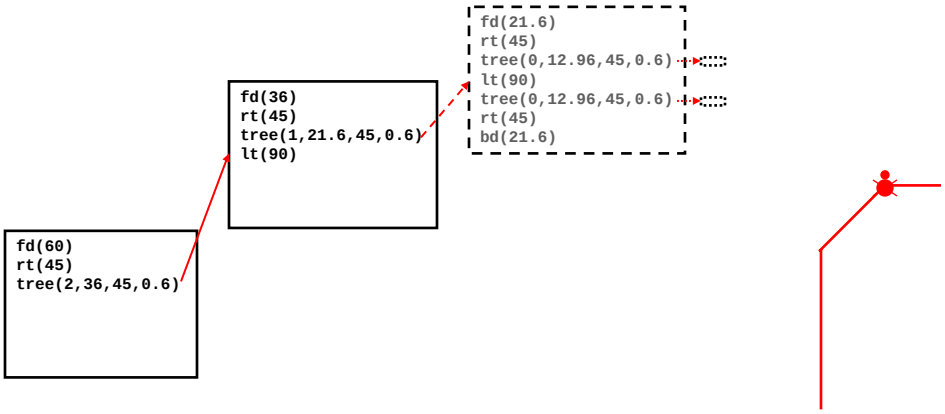
Begin recursive invocation to draw level 0 tree



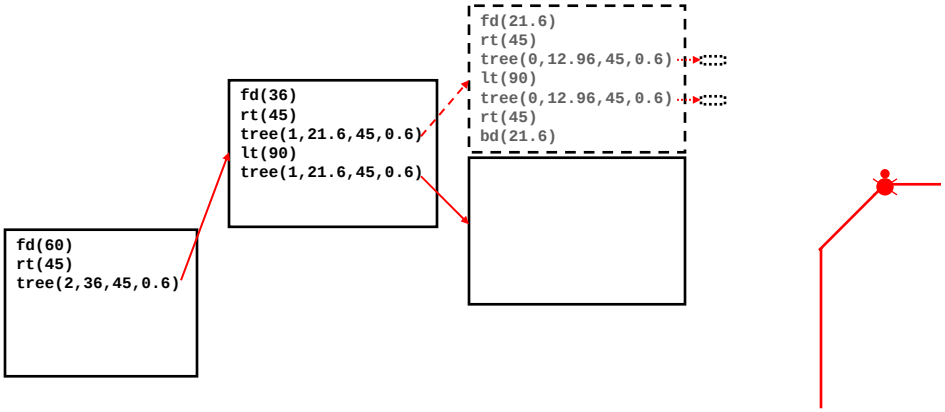
Complete level 0 tree and return to starting position of level 1 tree



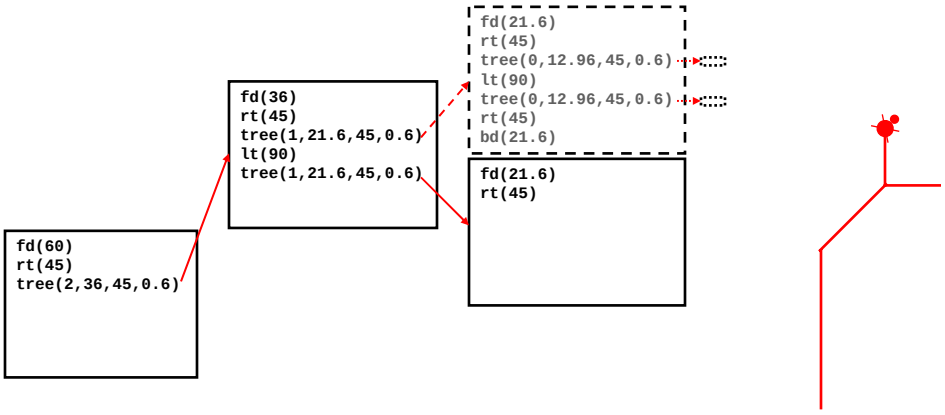
Complete level 1 tree and turn to draw another level 1 tree



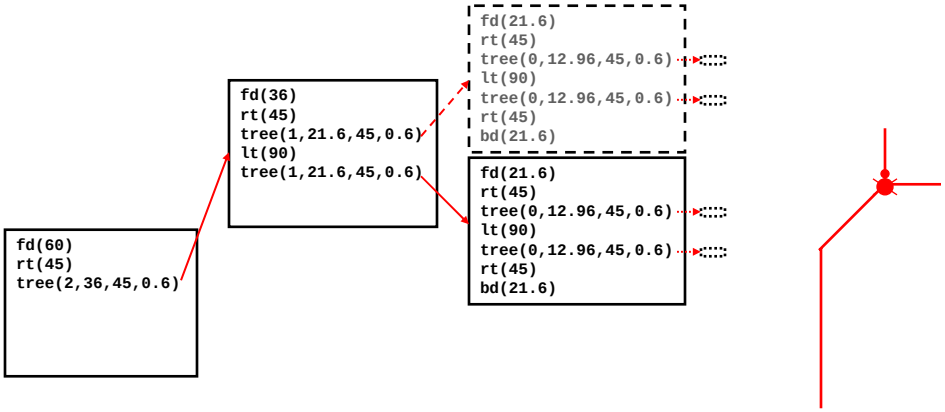
Begin recursive invocation to draw level 1 tree



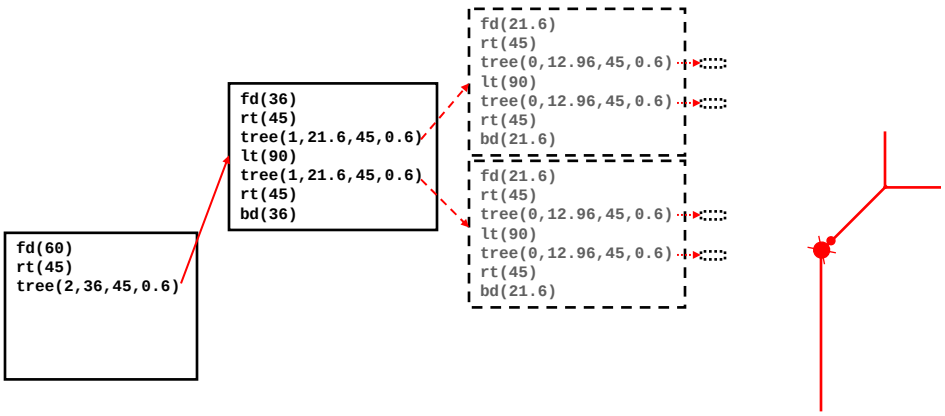
Draw trunk and turn to draw level 0 tree



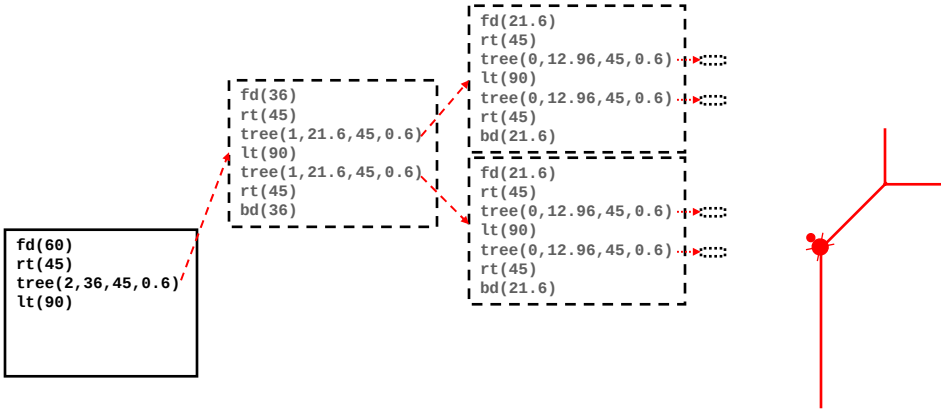
Complete two level 0 trees and return to starting position of level 1 tree



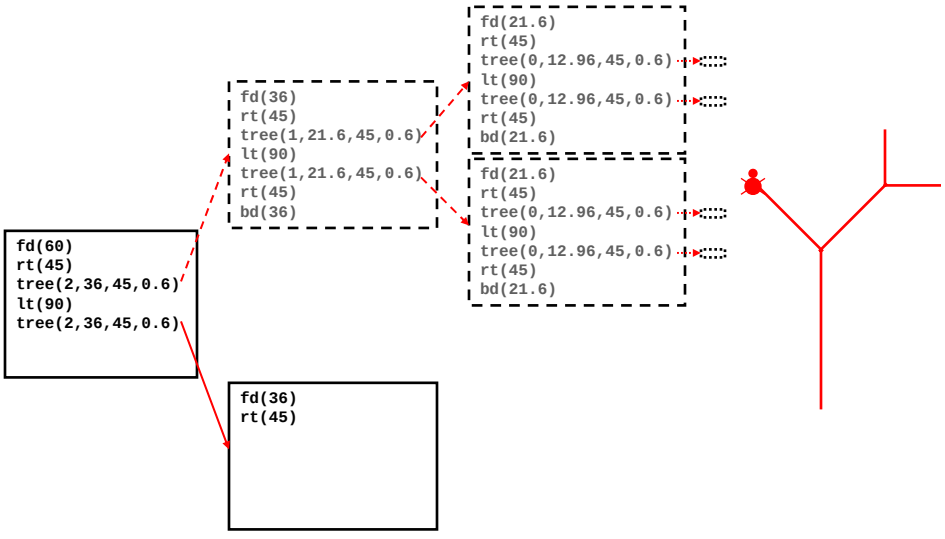
Complete level 1 tree and return to starting position of level 2 tree



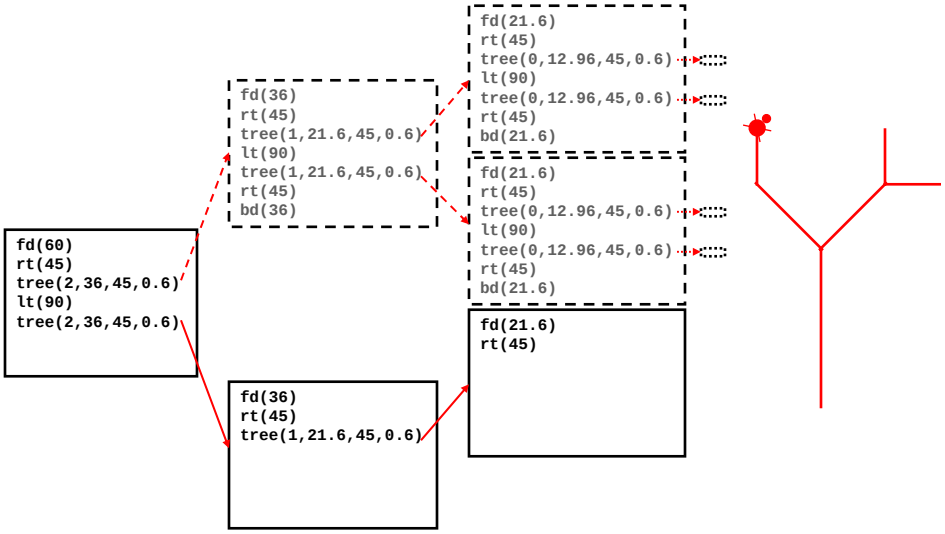
Complete level 2 tree and turn to draw another level 2 tree



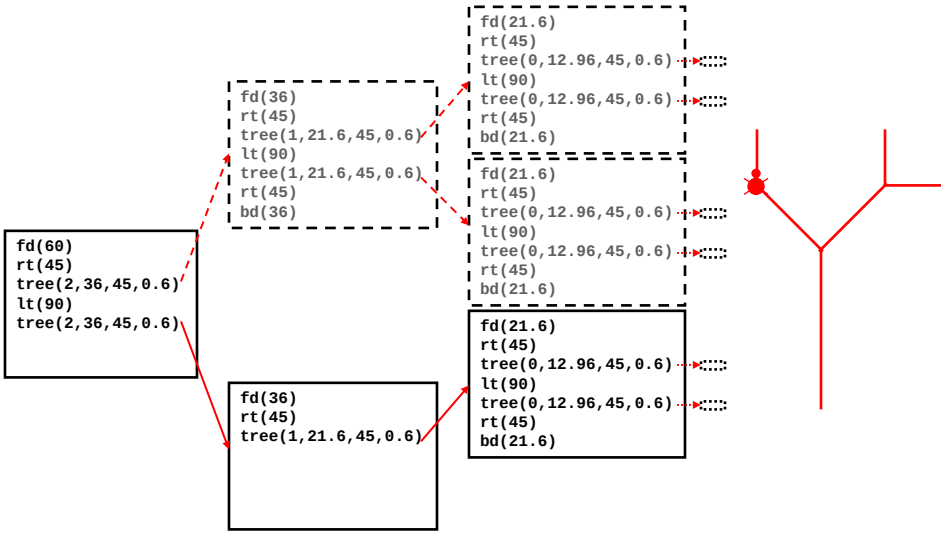
Draw trunk and turn to draw level 1 tree



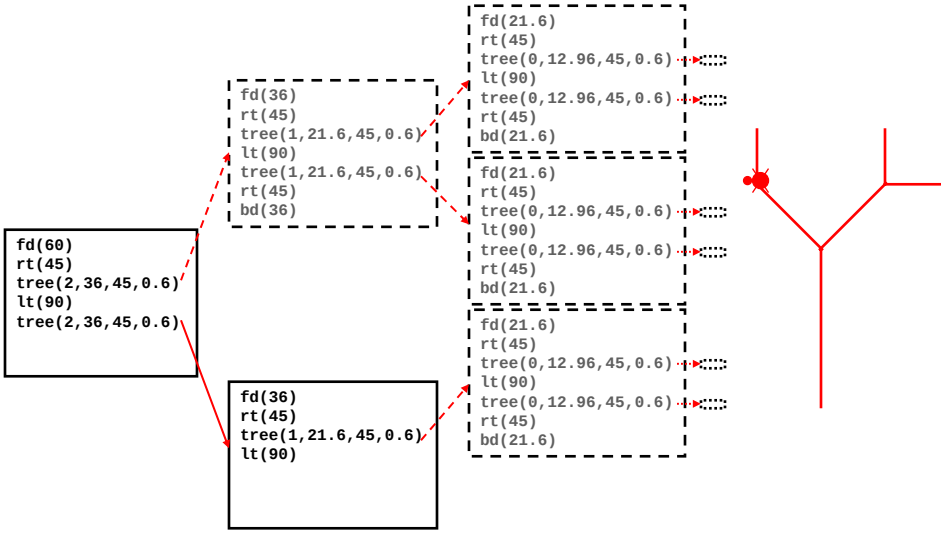
Draw trunk and turn to draw level 0 tree



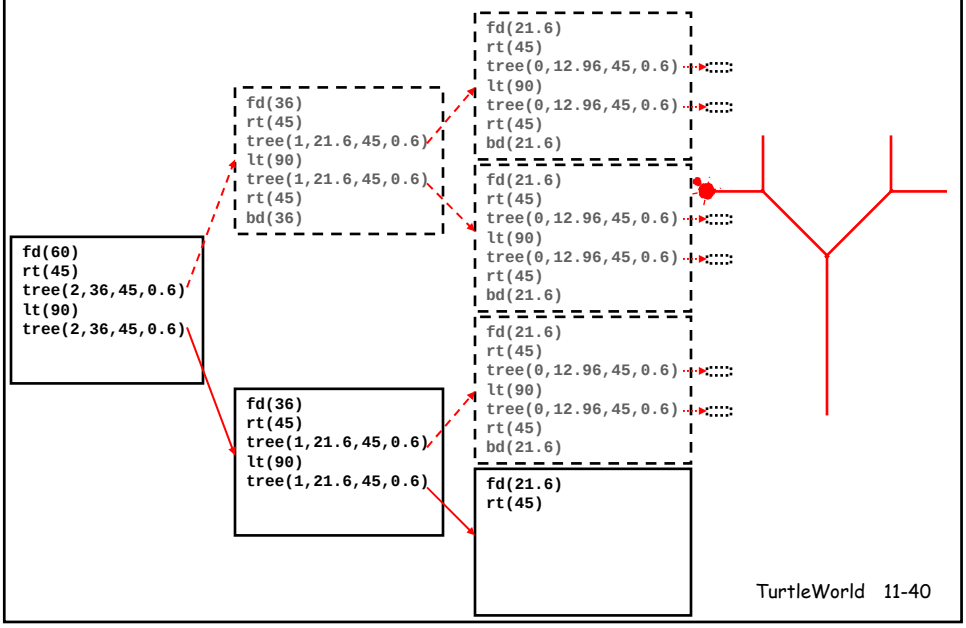
Complete two level 0 trees and return to starting position of level 1 tree



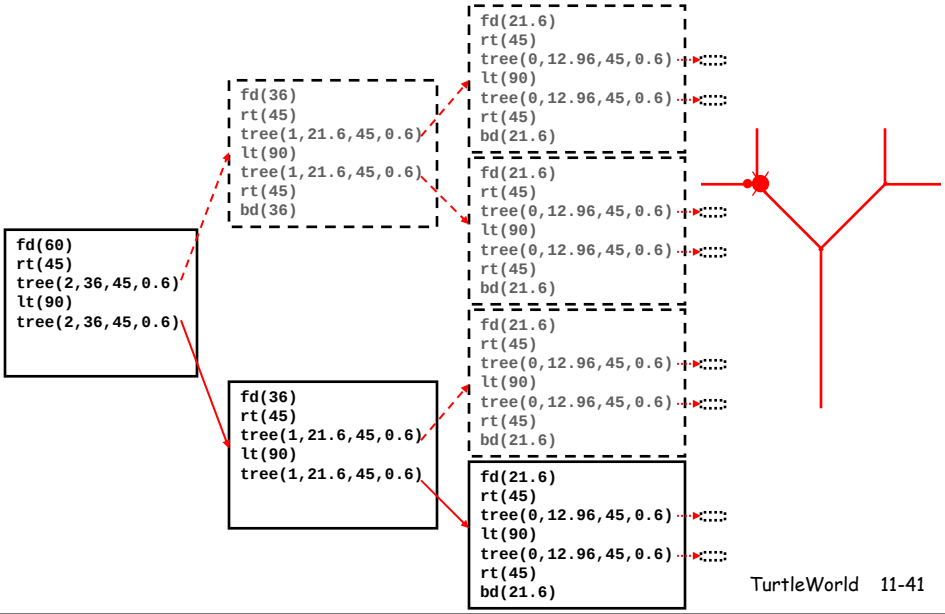
Complete level 1 tree and turn to draw another level 1 tree



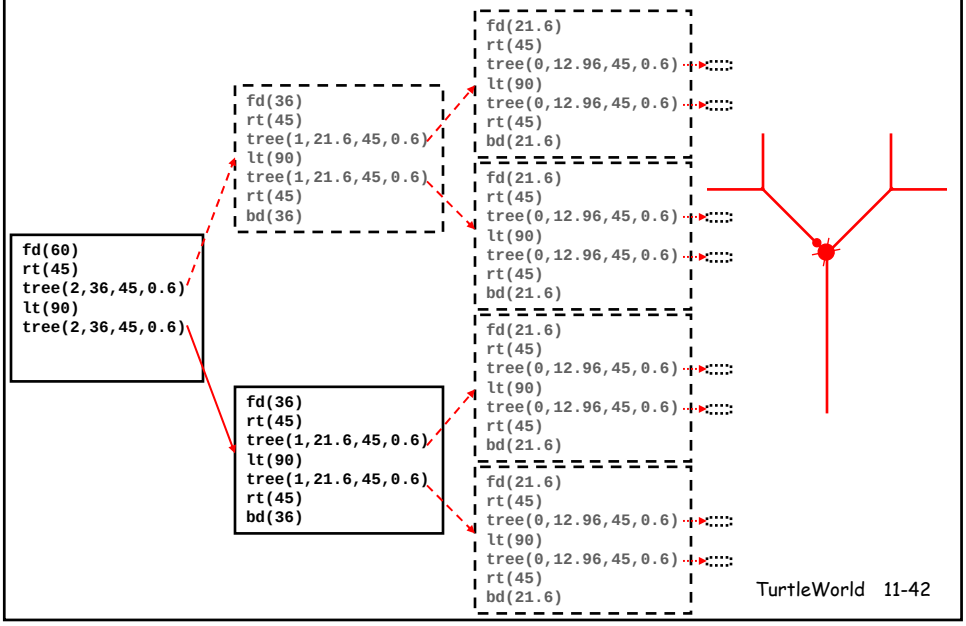
Draw trunk and turn to draw level 0 tree



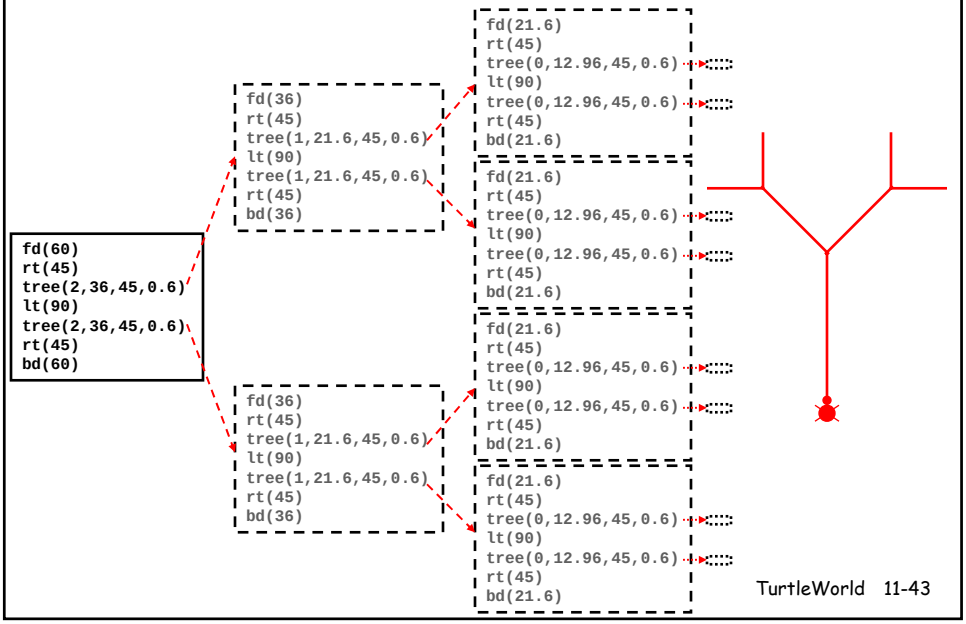
Complete two level 0 trees and return to starting position of level 1 tree



Complete level 1 tree and return to starting position of level 2 tree

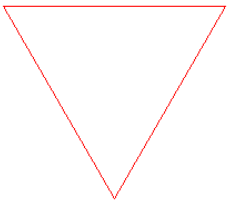


Complete level 2 tree and return to starting position of level 3 tree

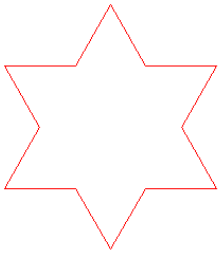


TurtleWorld 11-43

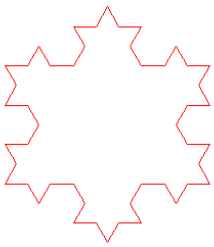
Snowflakes (the Koch curve)



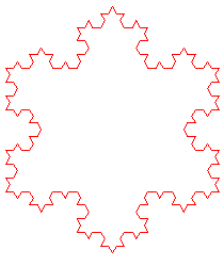
snowy.snowflake(0, 200);



snowy.snowflake(1, 200);



snowy.snowflake(2, 200);



snowy.snowflake(3, 200);

TurtleWorld 11-44

A Snowflake has 3 Sides

```
public void snowflake (int levels, double length) {

    snowside(levels, length);
    rt(120);
    snowside(levels, length);
    rt(120);
    snowside(levels, length);
    rt(120);
}
```


snowy.snowside(0,200);

snowy.snowside(1,200);

snowy.snowside(2,200);

snowy.snowside(3,200);

Let's Write snowside() in Java

```
public void snowside (int levels, double length) {




}
```