CS 101:
Computer Programming and Utilization

Jul - Nov 2016

Bernard Menezes
(cs101@cse.iitb.ac.in)

Lecture 13: Recursive Functions

About These Slides

- Based on Chapter 10 of the book
  An Introduction to Programming Through C++
  by Abhiram Ranade (Tata McGraw Hill, 2014)

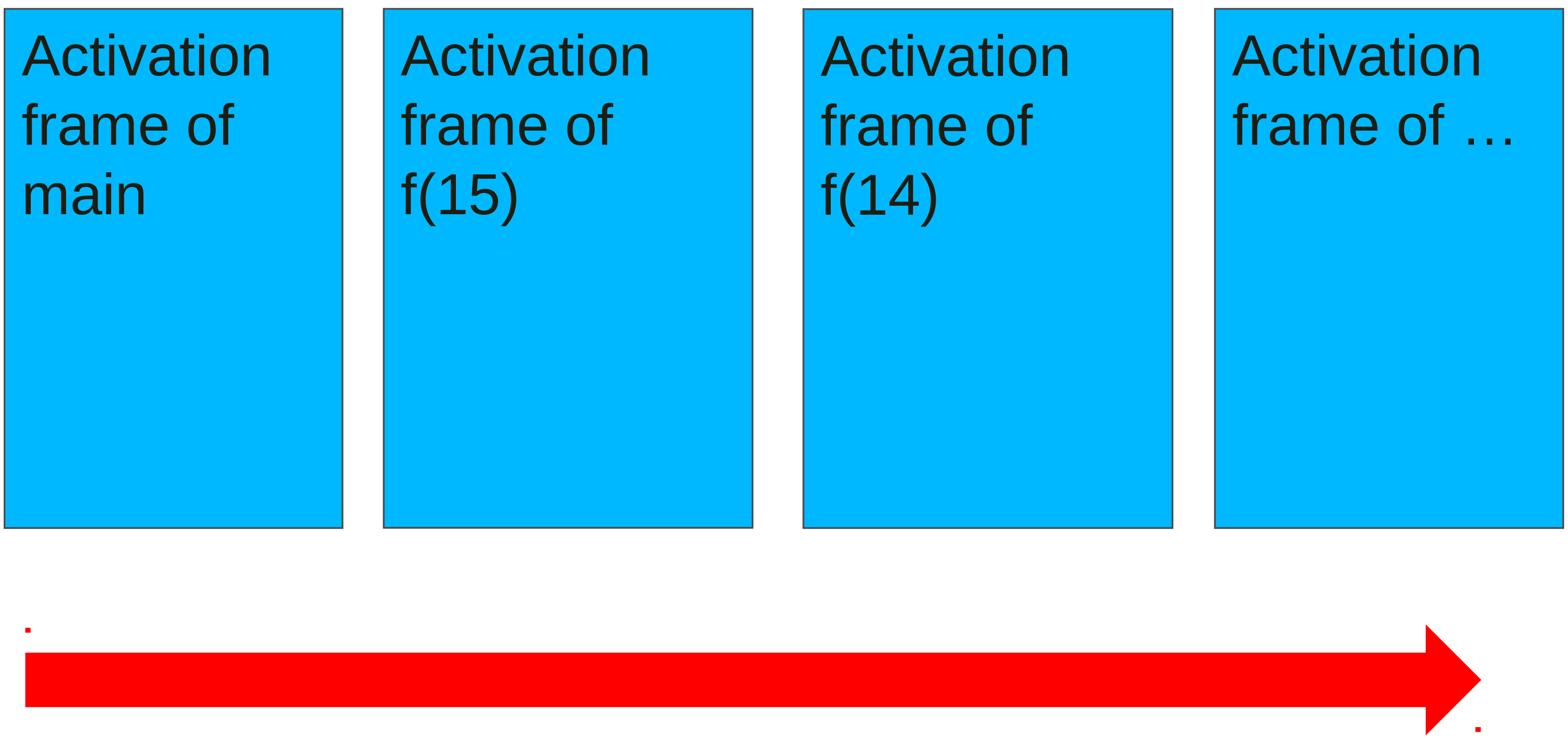
- Original slides by Abhiram Ranade
  – First update by Varsha Apte
  – Second update by Uday Khedker

Can a Function Call Itself?

```
int f(int n){
   …
   int z = f(n-1);
   …
}
main_program{
  int z = f(15);
}
```

- Allowed by execution mechanism
- main_program executes, calls f(15)
- Activation Frame (AF) created for f(15)
- f executes, calls f(14)
- AF created for f(14)
- Continues in this manner, with AFs created for f(13), f(12) and so on, endlessly

Activation Frames Keep Getting Created in Stack Memory



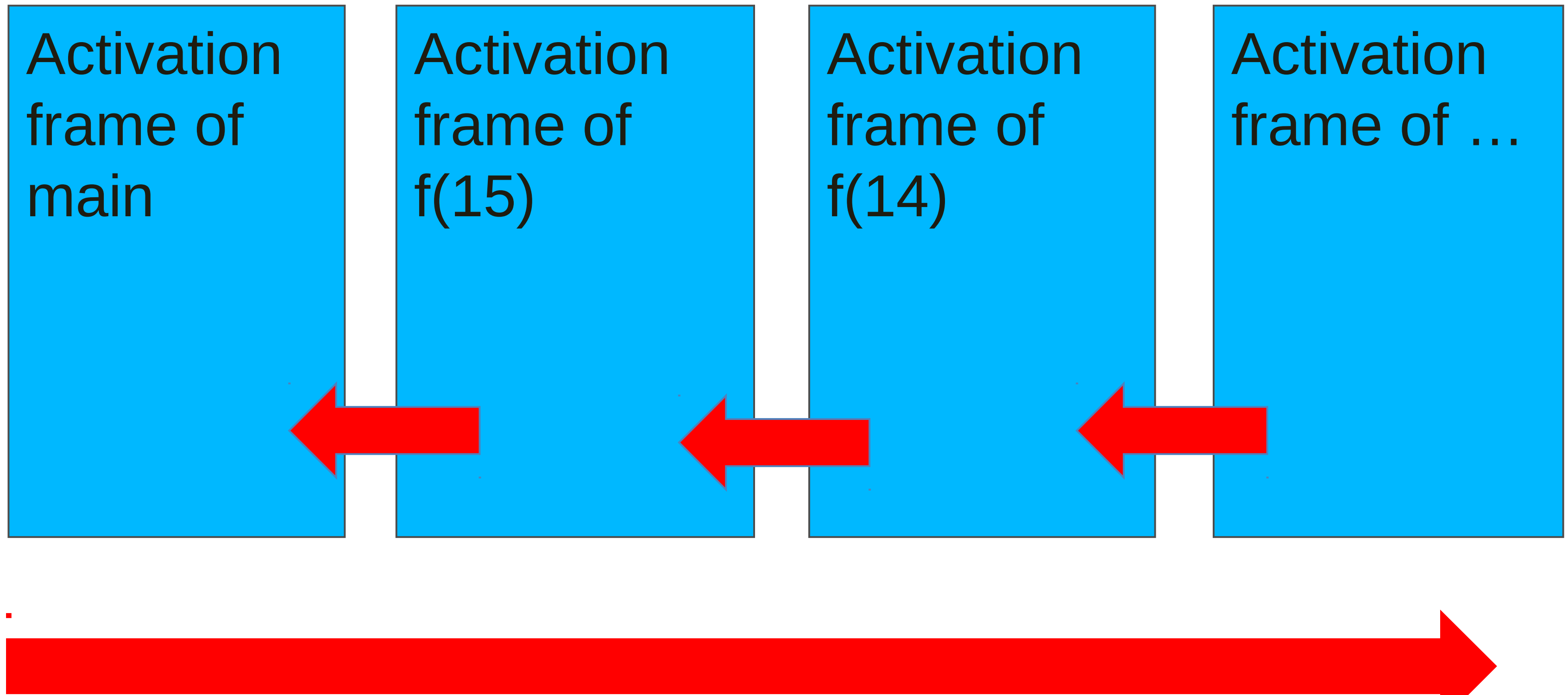
Another Function that Calls Itself

```
int f(int n){
   ...
   if(n > 13)
      z = f(n-1);
   ...
}
main_program{
  int w = f(15);
}
```

- main_program executes, calls f(15)
- AF created for f(15)
- f(15) executes, calls f(14)
- AF created for f(14)
- f(14) executes, calls f(13)
- AF created for f(13)
- f(13) executes, check n>13 fails. some result returned
- Result received in f(14)
- f(14) continues and in turn returns result to f(15)
- f(15) continues, returns result to main_program
- main_program continues and finished

Activation Frames Keep Getting Created in Stack Memory

and destroyed as the functions exit



Recursion

Function called from its own body

OK if we eventually get to a call which does not call itself

Then that call will return

Previous call will return…

But could it be useful?

Outline

- GCD Algorithm using recursion
- A tree drawing algorithm using recursion

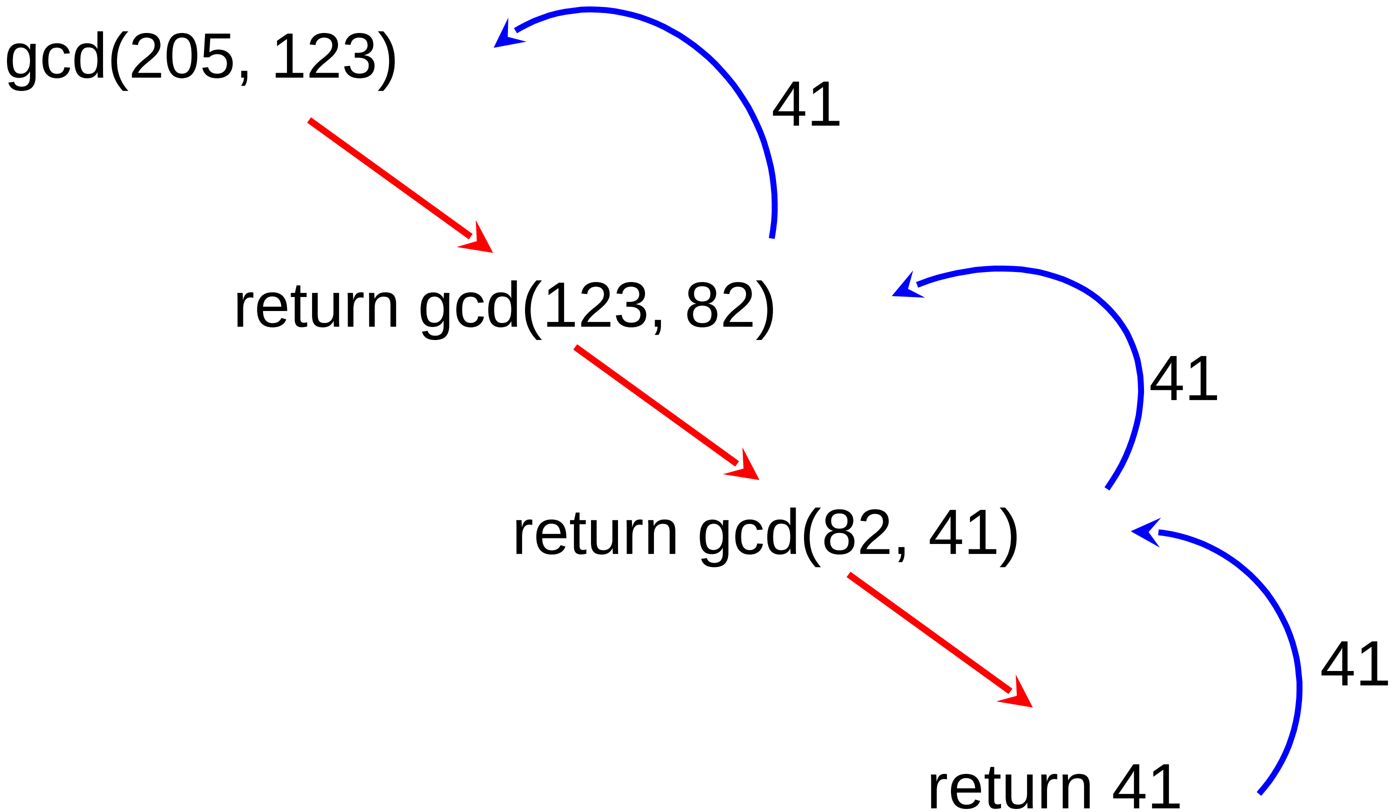
Euclid’s Theorem on GCD

```
//If m % n == 0, then
//            GCD(m, n) = n,
// else GCD(m,n) = GCD(n, m % n)

int gcd(int m, int n){
   if (m % n == 0) return n;
   else return gcd(n, m % n);
}
main_program{
      cout << gcd(205,123) << endl;
}
```

Will this work?

Execution of Recursive gcd



Euclid’s Theorem on GCD

```
int gcd(int m, int n){
   if (m % n == 0) return n;
   else return gcd(n, m % n);
}

main_program{
cout
<< gcd(205,123)
<< endl;
}
```

Execute this

Execute this

return 41

return 41

return 41

Activation frame of main created

Activation frame of gcd (205, 123) created

Activation frame of gcd (123, 82) created

Activation frame of gcd (82, 41) created

Recursion Vs. Iteration

- Recursion allows multiple distinct data spaces for different executions of a function body
  - Data spaces are live simultaneously
  - Creation and destruction follows LIFO policy
- Iteration uses a single data space for different executions of a loop body
  - Either the same data space is shared or one data space is destroyed before the next one is created
- Iteration is guaranteed to be simulated by recursion but not vice-versa

Correctness of Recursive gcd

We prove the correctness by induction on j

For a given value of j, gcd(i,j) correctly computes gcd(i,j) for all values of i

We prove this for all values of j by induction

- Base case: j=1. gcd(i,1) returns 1 for all i

  Obviously correct

- Inductive hypothesis: Assume the correctness of gcd(i,j) for some j
- Inductive step: Show that gcd(i,j+1) computes the correct value

Correctness of Recursive gcd

Inductive Step: Show that gcd(i,j+1) computes the correct value, assuming that gcd(i,j) is correct

- If j+1 divides i, then the result is j+1

  Hence correct

- If j+1 does not divide i, then gcd(i,j+1) returns the result of calling gcd(j, i%(j+1)

  - i%(j+1) can at most be equal to j
  - By the inductive hypothesis, gcd(j, i%(j+1) computes the correct value
  - Hence gcd(i, j+1) computes the correct value

Remarks

- The proof of recursive gcd is really the same as that of iterative gcd, but it appears more compact
- This is because in iterative gcd, we had to speak about “initial value of m,n”, “value at the beginning of the iteration” and so on
- In general recursive algorithms are shorter than corresponding iterative algorithms (if any), and the proof is also more compact, though same in spirit

Factorial Function

- Iterative factorial function

```
int fact(int n) {
int res=1;
for (int i=1; i<=n; i++)
res = res*i;
    return res;
}
```

- Recursive factorial function

```
int fact(int n) {
if (n<=0) return 1;
else return n*fact(n-1);
}
```

Fibonacci Function

- Iterative fibonacci function:

```
int fib(int n){
    if (n <= 0) return 0;
    if (n == 1) return 1;
    int n_2 = 0, n_1 = 1, result = 0;
    for (int i = 2; i <= n; i++) {
        result = n_1 + n_2;
        n_2 = n_1;
        n_1 = result;
    }
    return result;
}
```

Fibonacci Function

- Definition:

  fib(0) = 0

  fib(1) = 1

  fib(n) = fib(n-1) + fib(n-2),   n > 1

- Recursive fibonacci function:

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

An Important Application of Recursion: Processing Trees

Botanical trees…

Organization Tree

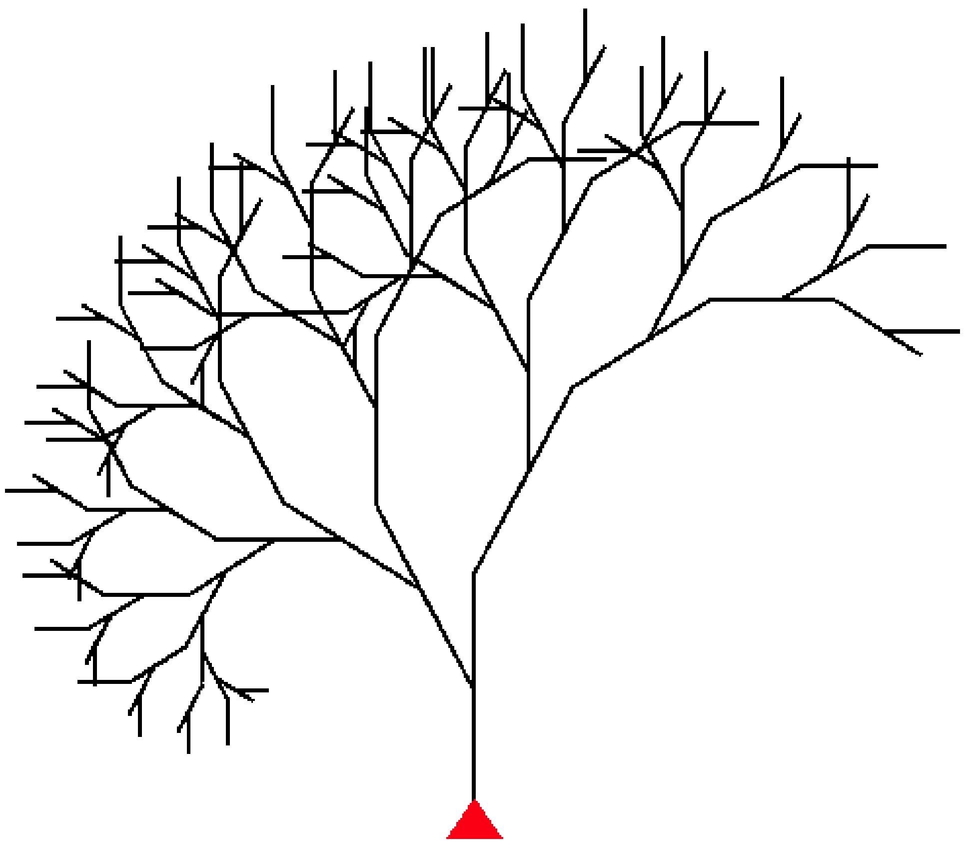
Expression Tree

Search Tree: later

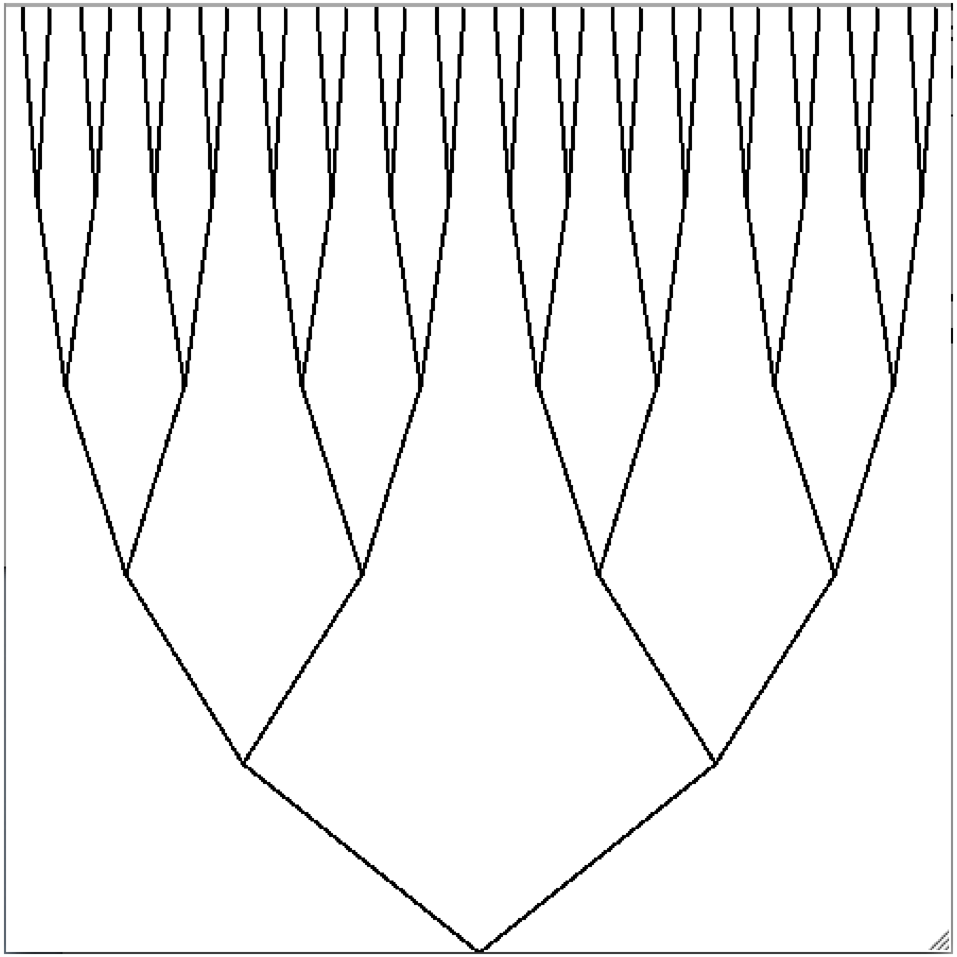
In this chapter we only consider how to draw trees

- Must understand the structure of trees
- Structure will be relevant to more complex algorithms

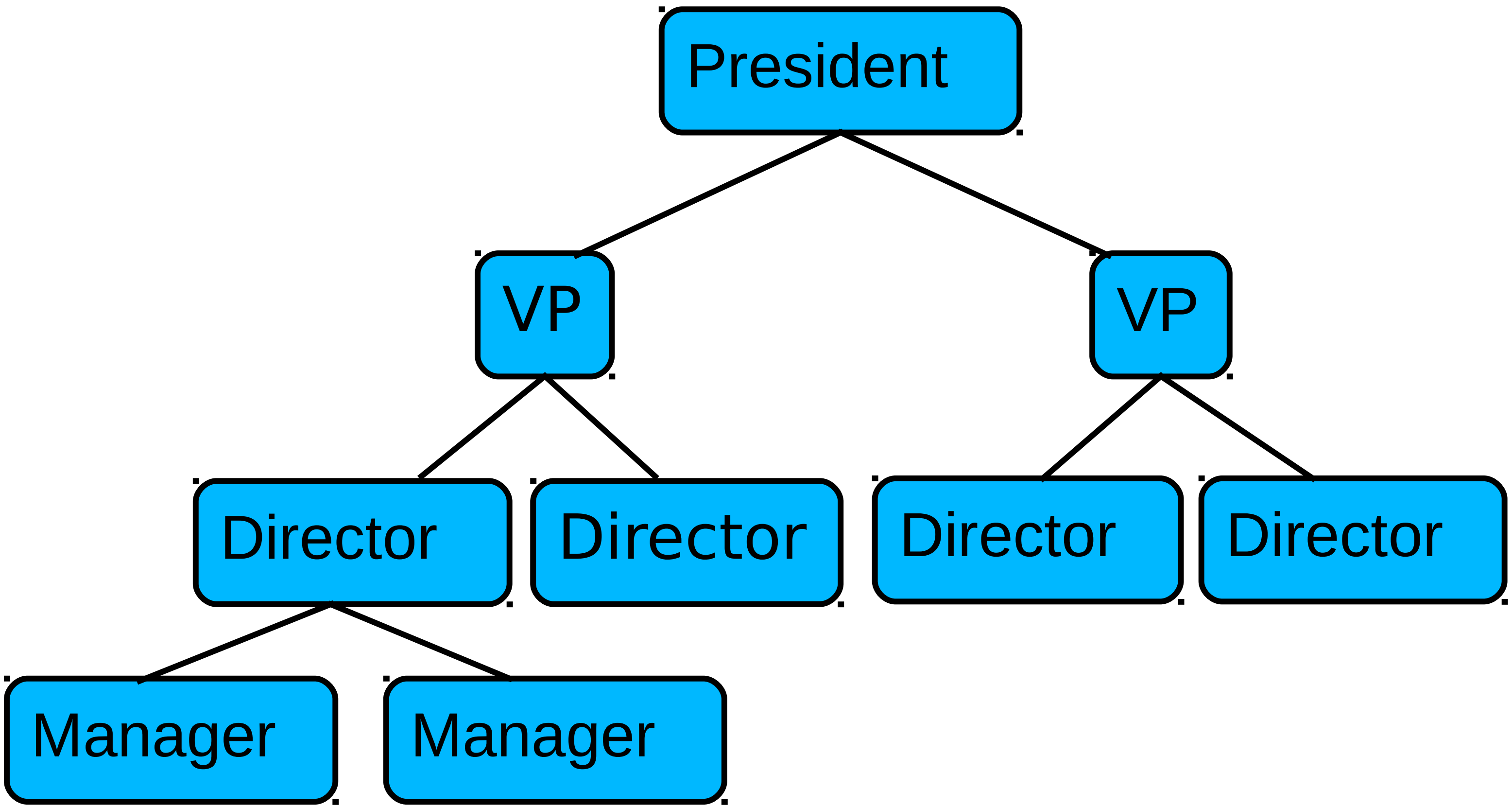
A Botanical Tree Drawn Using the Turtle in Simplecpp



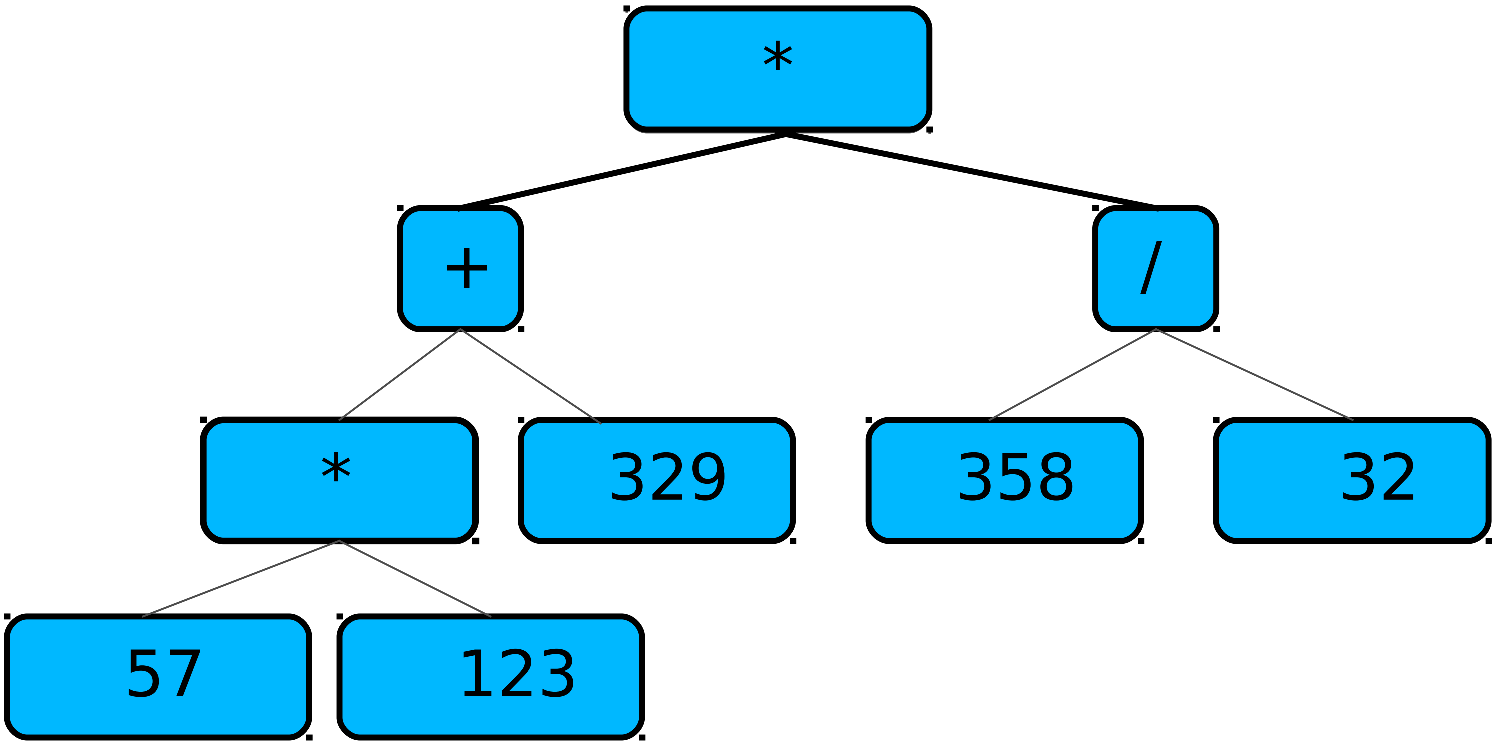
A More Stylized Tree Drawn Using simplecpp



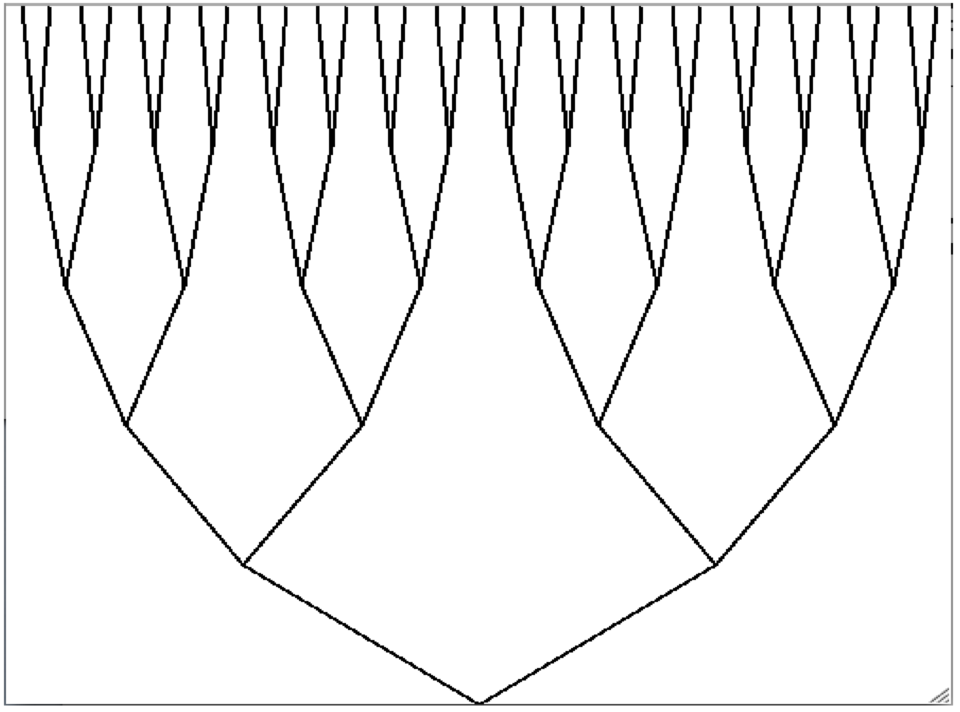
Organization Tree
(Typically “grows” Downwards)



Tree Representing ((57*123)+329)*(358/32)



1 Stylized Tree =
2 Small Stylized Trees + V



When a part of an object is of the same type as the whole, the object is said to have a recursive structure.

Drawing The Stylized Tree

Parts:

Root

Left branch,   Left subtree

Right branch, Right subtree

Number of levels: number of times the tree has branched going from the root to any leaf.

Number of levels in tree shown = 5

Number of levels in subtrees of tree: 4

Drawing The Stylized Tree

General idea:

To draw an L level tree:

if L > 0{

Draw the left branch, and a Level L-1 on top of it.

Draw the right branch, and a Level L-1 tree on top of it.

}

We must give the coordinates where the lines are to be drawn

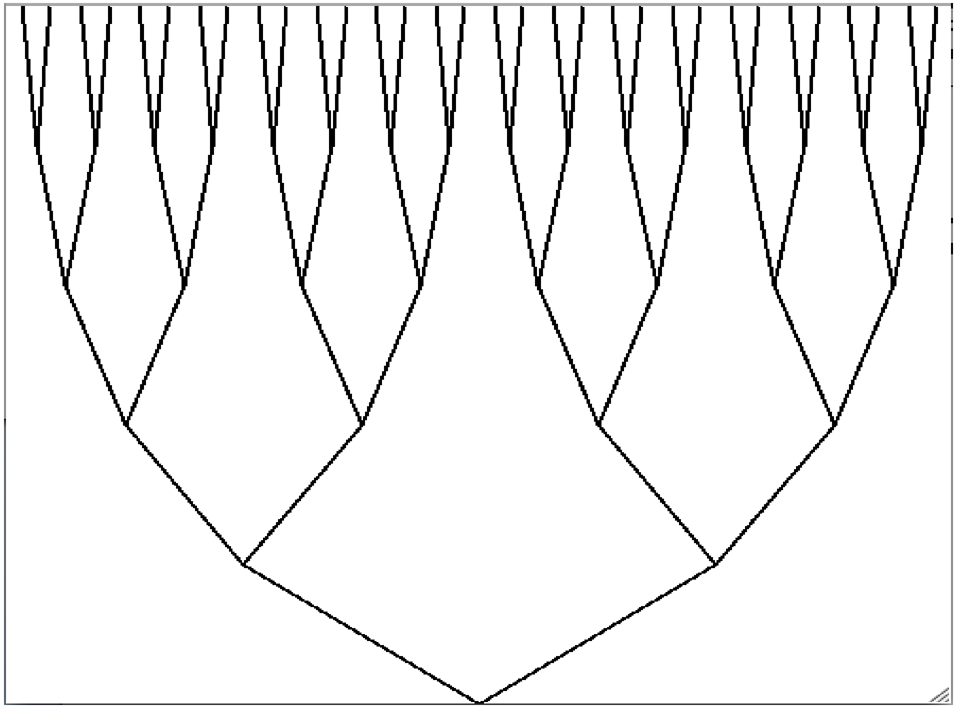
Say root is to be drawn at (rx,ry)

Total height of drawing is h.

Total width of drawing is w.

We should then figure out where the roots of the subtrees will be.

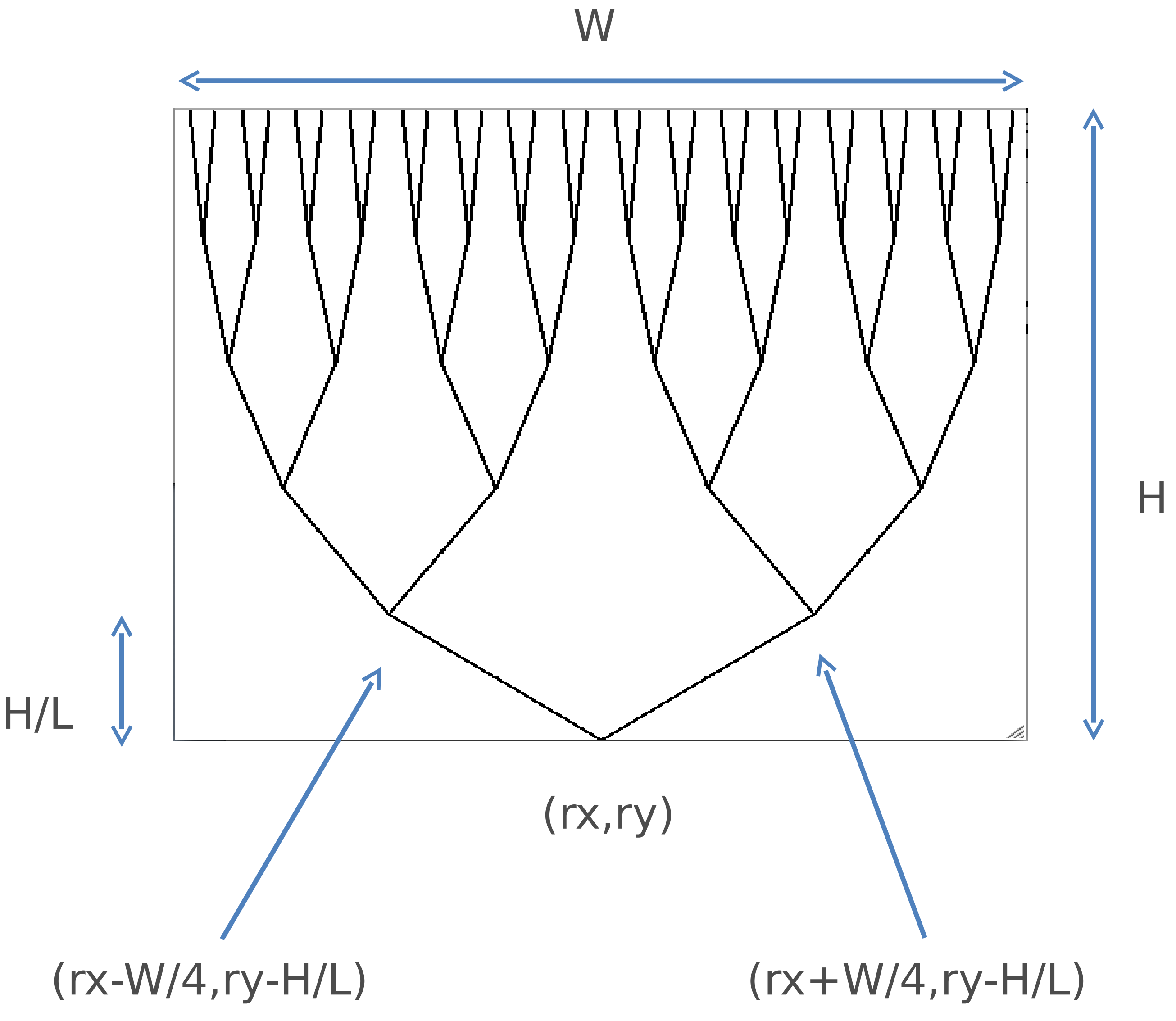
Drawing The Stylized Tree



Basic Primitive:

Drawing a line from (x1,y1) to (x2,y2)

Drawing The Stylized Tree



Drawing The Stylized Tree

Basic Primitive Required: Drawing a line

- Create a named shape with type Line

```
Line line_name(x1,y1,x2,y2);
```

- Draw the shape

```
line_name.imprint();
```

Drawing The Stylized Tree

```
void tree(int L, double rx, double ry,
                double H, double W) {
  if(L>0){
    Line left(rx, ry, rx-W/4, ry-H/L);     // line called left
    Line right(rx, ry, rx+W/4, ry-H/L); // line called right
    right.imprint();                // Draw the line called right
    left.imprint();                 // Draw the line called left
    tree(L-1, rx-W/4, ry-H/L, H-H/L, W/2); // left subtree
    tree(L-1, rx+W/4, ry-H/L, H-H/L, W/2);// right subtree
  }
}
```

Concluding Remarks

- Recursion allows many programs to be expressed very compactly
- The idea that the solution of a large problem can be obtained from the solution of a similar problem of the same type, is very powerful
- Euclid probably used this idea to discover his GCD algorithm
- More examples in the book