Functions 2

Abhiram Ranade

Functions: Summary from last time

- Executes on demand, like an independent program
- Calling program sends arguments and suspends.
  - ith argument placed in ith parameter
  - called function executes independently, in its own Activation frame.
  - General resources, such as cin, cout, available to function.
  - Canvas is also available, if it was created.
  - value returned, if any, = value of call.
- When function finishes, calling program resumes.

Summary (contd.)

- Function must be declared/defined above its use in the file.
- Functions can be placed in a file different from the file in which they are used.
- If a function in file F.cpp is needed by main programs in files main1.cpp and also main2.cpp, use:

s++ main1.cpp F.cpp to compile first program

s++ main2.cpp F.cpp to compile second program

Summary (contd.)

- main_program is a simplecpp abbreviation for int main()
- C++ requires the main program to be written as a function called main
- main cannot be called.
- main may return an int, but it is a special function and so need not.

Outline for Today

- More examples
- Recursive Functions

Function to draw a polygon

```
void polygon(int sides, double length){
  repeat(sides){
    forward(length); left(360.0/sides);
  }
  return;
}
int main(){
  turtleSim();
  for(int i=3, i<10; i++) polygon(i, 100);
}
```

Specification of polygon

Input:

sides: number of sides

length: of side

Output:

None. (Output: value returned)

Side-effect: Polygon is drawn on the screen.

At position, orientation…

Final position of pen, final orientation

Function with output value and side-effect

```
int printlcm(int m, int n){
  int res = lcm(m, n);
  cout << res << endl;
  return res;
}
```

Nothing new being said. Function-body can contain “statements”, including cout << …

Aside:
Creating shapes inside functions

```
void drawTriangle (double x1, double y1,
                   double x2, double y2, double x3, double y3){
   Line L1(x1, y1, x2, y2);  L1.imprint();
   Line L2(x2, y2, x3, y3);  L2.imprint();
   Line L3(x3, y3, x1, y1);  L3.imprint();
   return;
}
```

- L1, L2, L3 : variables, and also lines.
- Variables and lines both destroyed on return.
- Imprinting stays on canvas: side effect.

Shape arguments to functions

Not for now. But soon.

Button based turtle control

```
main_program{
  initCanvas();    Turtle t;    const float bFx=150,bFy=100, …
  Rectangle buttonF(…), buttonL(…);  Text tF(…,"Forward"), tL(…,"Left Turn");
  repeat(100){
    int clickPos = getClick();  int cx = clickPos/65536; int cy = clickPos % 65536;
    if(bFx-bWidth/2<= cx && cx<= bFx+bWidth/2 &&
       bFy-bHeight/2 <= cy && cy <= bFy+bHeight/2) t.forward(100);
    if(bLx-bWidth/2<= cx && cx<= bLx+bWidth/2 &&
       bLy-bHeight/2 <= cy && cy <= bLy+bHeight/2) t.left(10);
  }
}  // will functions help in this?
```

Function to determine if click is inside

```
bool inside(int cx, int cy, int bx, int by,
                    int w, int h){
  if(bx - w/2 <=  cx && cx <= bx + w/2 &&
      by – h/2 <= cy && cy <= by + h/2)
        return true;
  else return false;
}
```

Function to determine if click is inside

```
bool inside(int cx, int cy, int bx, int by,
                    int w, int h){
  return (bx - w/2 <=  cx && cx <= bx + w/2 &&
              by – h/2 <= cy && cy <= by + h/2);
}

// will improve this further later.
```

Practice problems

Write functions to

- Decide if a given number is perfect, i.e. equal to the sum of its divisors.
- Find the GCD of 3 numbers. LCM of 3 numbers.
- Draw a house. Use it to draw a colony of houses.
- Find cube roots. Use Newton's method.

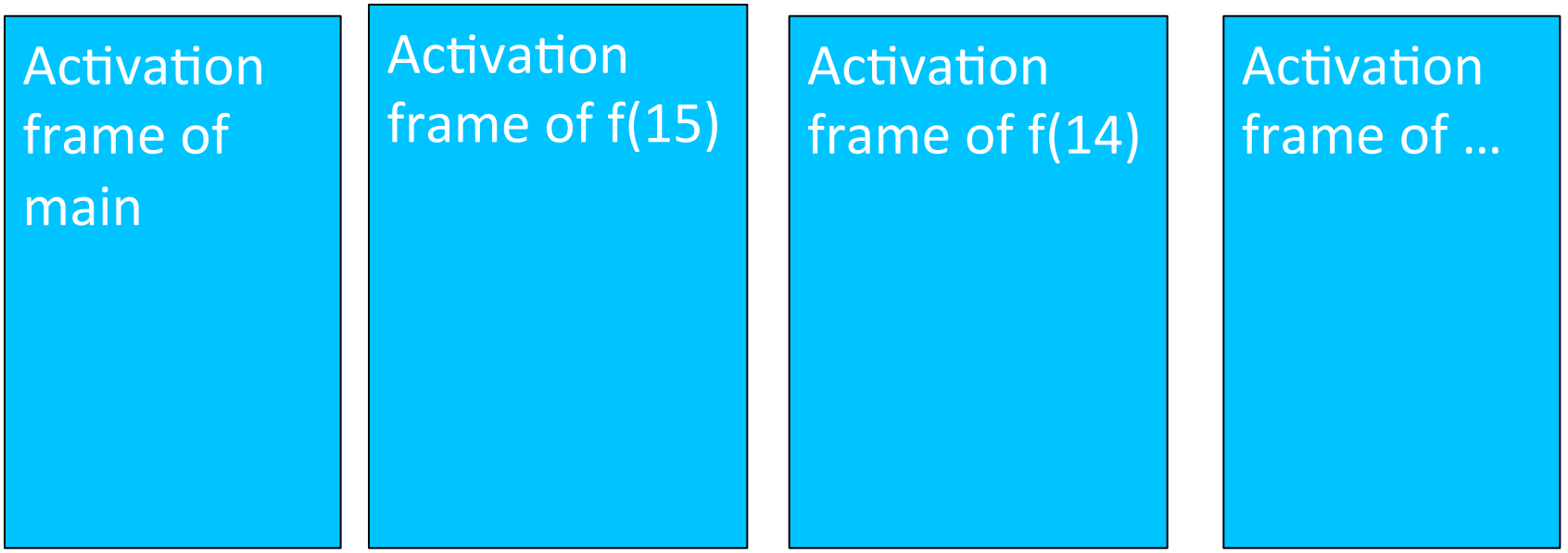
Everything we have done can be packaged as a function.

Can a function call itself?

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

int main(){
  int z = f(15);
}
```

Consistent with execution mechanism



Can a function call itself?

```
int f(int n){
   …    if(n > 13) int z = f(n-1) …
}

int main(){
  int z = f(15);
}
```

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?

Euclid's Observation

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);
}
```

Example

```
int main(){ cout << gcd(943,205)<<endl; }
AF of main
    suspend on: cout << gcd(943,205)<<endl;
AF of gcd(943,205):
    suspend on: return gcd(205,123);
AF of gcd(205,123):
    suspend on: return gcd(123,82);
AF of gcd(123,82):
    suspend on: return gcd(82,41);
AF of gcd(82,41):
    return 41
```

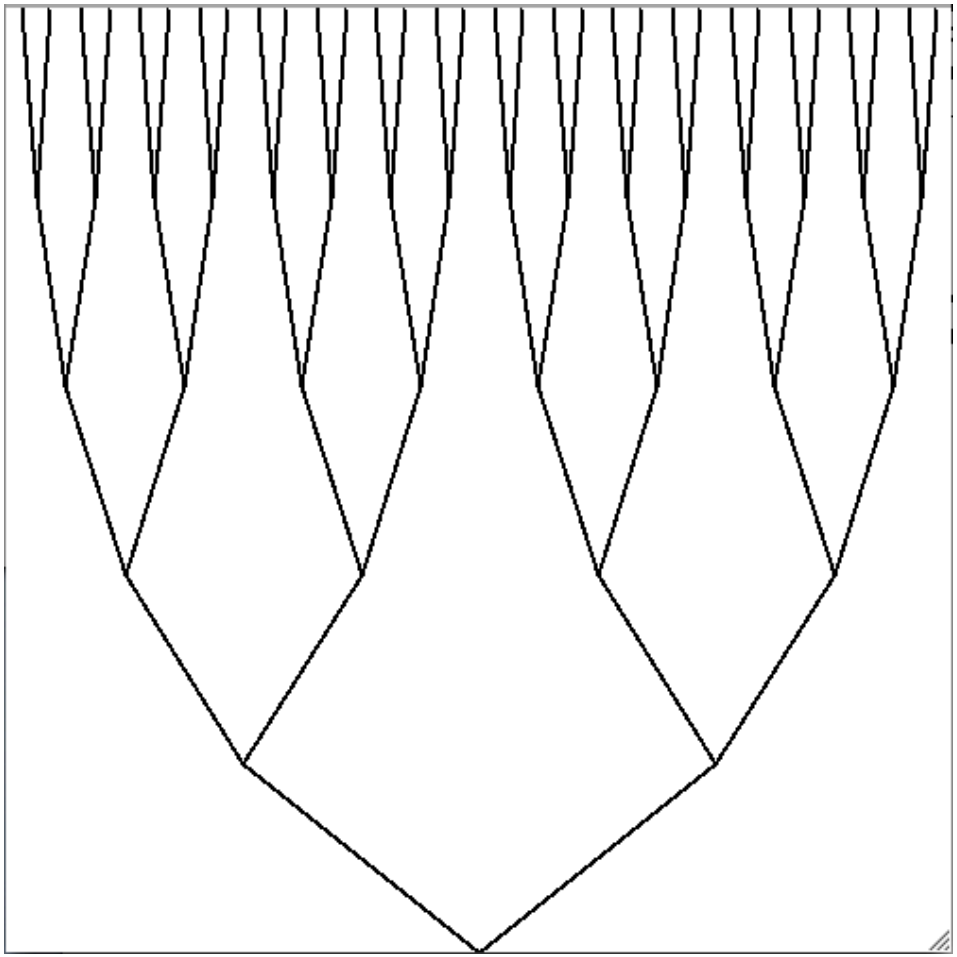
Correctness of Recursive Euclid

Very similar to iterative Euclid

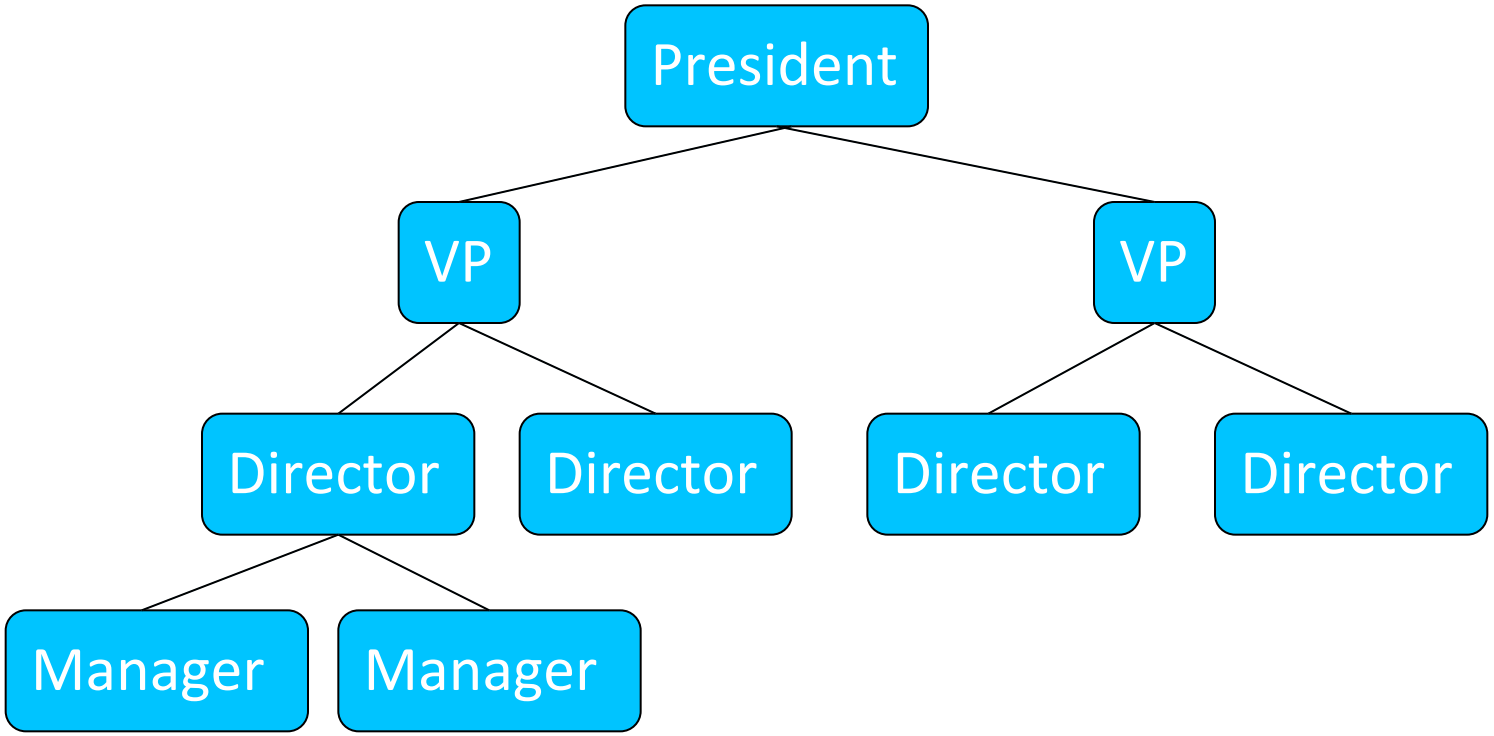
- Second argument always decreases,
- But cannot decrease below. Hence recursion is finite.
- By Euclid's theorem, GCD of values in new call is same as those in original call.

Recursive algorithms are often short and easier to prove.

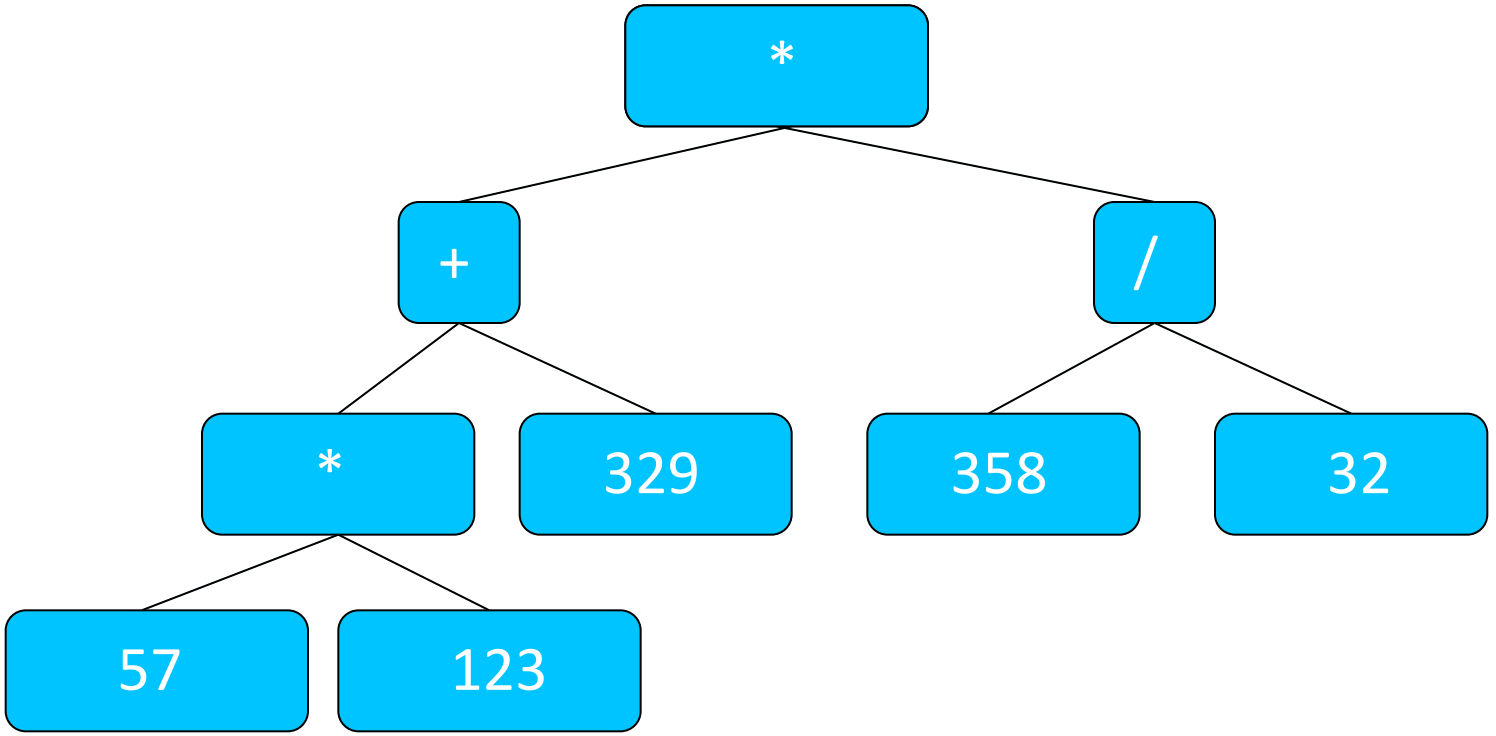
Trees 1



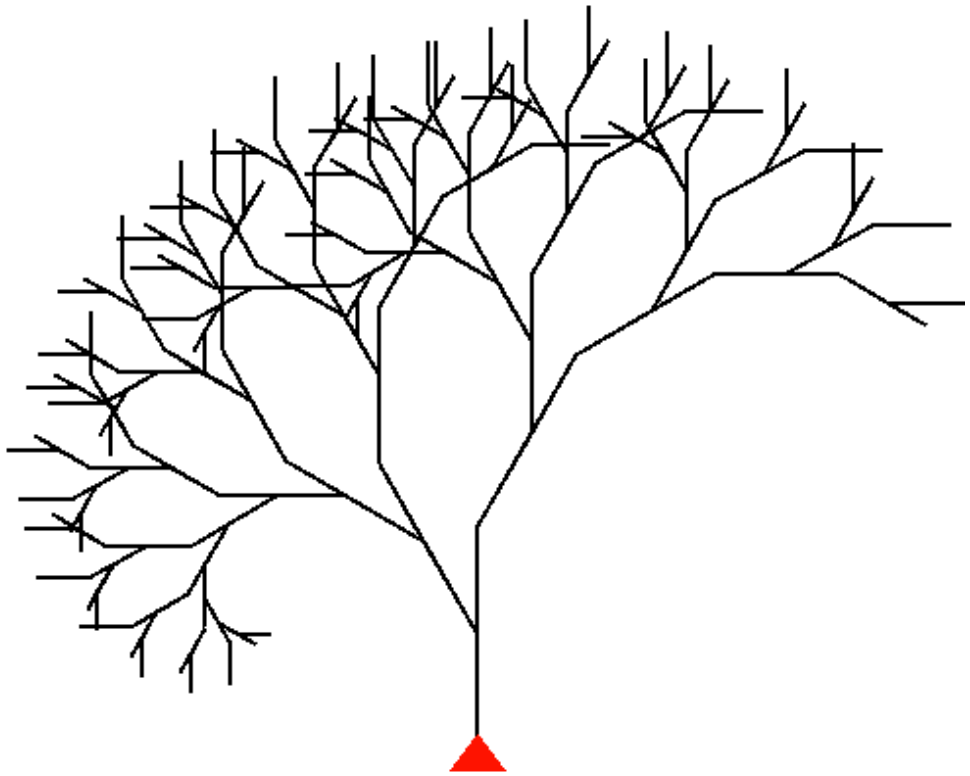
Trees 2


President
VP
VP
Director
Director
Director
Director
Manager
Manager

Trees 3



Trees 4



Trees are everywhere!

Organization Tree

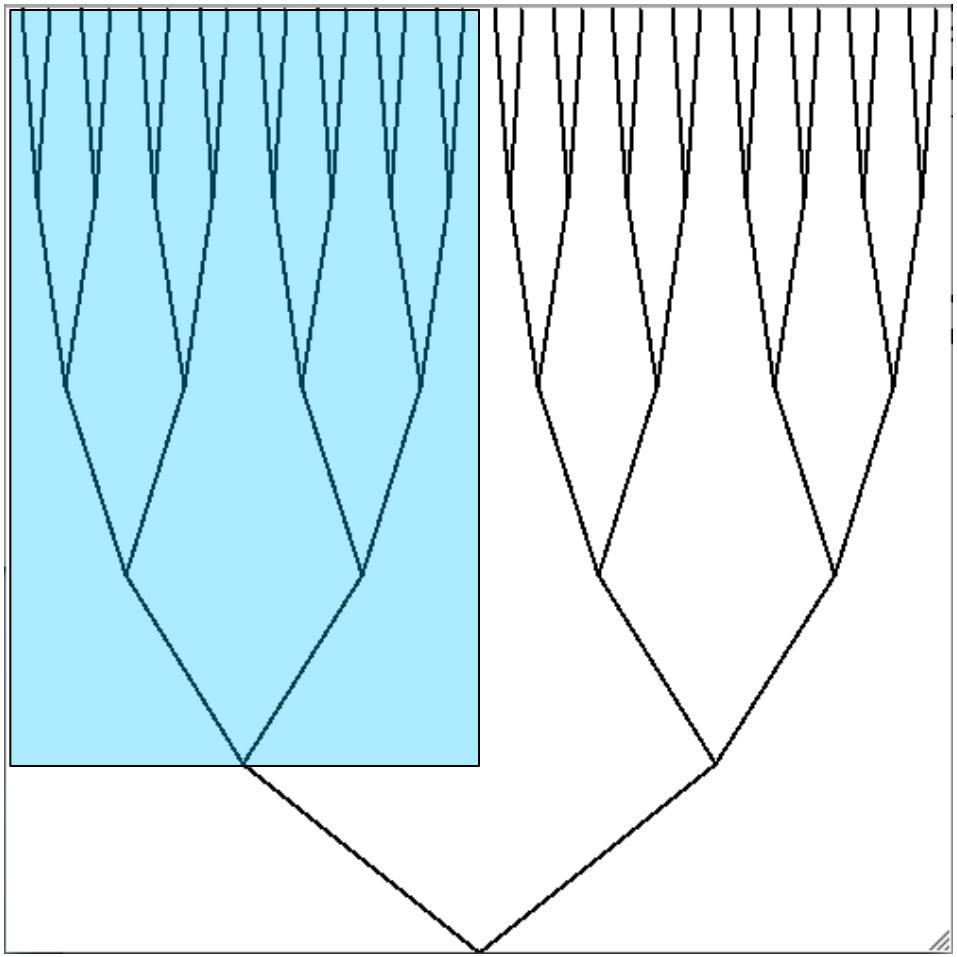
Expression Tree

Search Tree: later

Botanical trees…

Understand the structure of trees to design algorithms.

Trees = 2 small trees + V



Parts of a tree

Root

Left branch, Left subtree

Right branch, Right subtree

Subtrees have fewer levels, smaller width, smaller height

Levels = = 0? Then tree = only root.

```
void tree(int levels, double rx, double ry,
              double height, double width){
  if(levels>0){
    Line left(rx, ry, rx-width/4, ry-height/levels);
    Line right(rx, ry, rx+width/4, ry-height/levels);
    right.imprint();
    left.imprint();
    tree(levels-1, rx-width/4, ry-height/levels,
          height-height/levels, width/2);
    tree(levels-1, rx+width/4, ry-height/levels,
          height-height/levels, width/2);
  }
}
```