Eric Roberts
CS 106B

Handout #18
January 23, 2013

Recursive Strategies

Recursive Strategies

Eric Roberts
CS 106B
January 23, 2013

Recursion

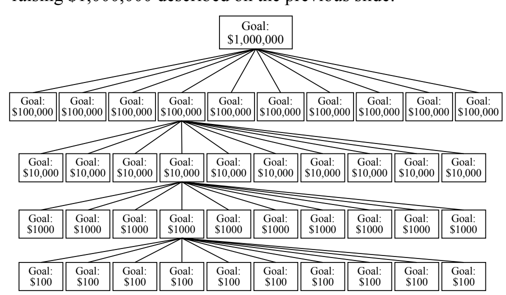
- One of the most important "Great Ideas" in CS 106B is the concept of ***recursion***, which is the process of solving a problem by dividing it into smaller subproblems *of the same form*. The italicized phrase is the essential characteristic of recursion; without it, all you have is a description of stepwise refinement of the sort we teach in CS 106A.
- The fact that recursive decomposition generates subproblems that have the same form as the original problem means that recursive programs will use the same function or method to solve subproblems at different levels of the solution. In terms of the structure of the code, the defining characteristic of recursion is having functions that call themselves, directly or indirectly, as the decomposition process proceeds.

A Simple Illustration of Recursion

- Suppose that you are the national fundraising director for a charitable organization and need to raise $1,000,000.
- One possible approach is to find a wealthy donor and ask for a single $1,000,000 contribution. The problem with that strategy is that individuals with the necessary combination of means and generosity are difficult to find. Donors are much more likely to make contributions in the $100 range.
- Another strategy would be to ask 10,000 friends for $100 each. Unfortunately, most of us don't have 10,000 friends.
- There are, however, more promising strategies. You could, for example, find ten regional coordinators and charge each one with raising $100,000. Those regional coordinators could in turn delegate the task to local coordinators, each with a goal of $10,000, continuing the process reached a manageable contribution level.

A Simple Illustration of Recursion

The following diagram illustrates the recursive strategy for raising $1,000,000 described on the previous slide:



A Pseudocode Fundraising Strategy

If you were to implement the fundraising strategy in the form of a C++ function, it would look something like this:

```
void collectContributions(int n) {
   if (n <= 100) {
      Collect the money from a single donor.
   } else {
      Find 10 volunteers.
      Get each volunteer to collect n/10 dollars.
      Combine the money raised by the volunteers.
   }
}
```

What makes this strategy recursive is that the line

Get each volunteer to collect n/10 dollars.

will be implemented using the following recursive call:

```
collectContributions(n / 10);
```

The Towers of Hanoi

In the great temple at Benares beneath the dome which marks the center of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priests transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

—Henri de Parville, *La Nature,* 1883

The Recursive "Leap of Faith"

- The purpose of going through the complete decomposition of the Towers of Hanoi problem is to convince you that the process works and that recursive calls are in fact no different from other method calls, at least in their internal operation.
- The danger with going through these details is that it might encourage you to do the same when you write your own recursive programs. As it happens, tracing through the details of a recursive program almost always makes such programs harder to write. Writing recursive programs becomes natural only after you have enough confidence in the process that you don't need to trace them fully.
- As you write a recursive program, it is important to believe that any recursive call will return the correct answer as long as the arguments define a simpler subproblem. Believing that to be true—even before you have completed the code—is called the ***recursive leap of faith***.

The Recursive Paradigm

- Most recursive functions you encounter in an introductory course have bodies that fit the following general pattern:

```
if (test for a simple case) {
    Compute and return the simple solution without using recursion.
} else {
    Divide the problem into one or more subproblems that have the same form.
    Solve each of the problems by calling this method recursively.
    Return the solution from the results of the various subproblems.
}
```

- Finding a recursive solution is mostly a matter of figuring out how to break it down so that it fits the paradigm. When you do so, you must do two things:
  1. Identify ***simple cases*** that can be solved without recursion.
  2. Find a ***recursive decomposition*** that breaks each instance of the problem into simpler subproblems of the same type, which you can then solve by applying the method recursively.

Generating Mondrian-Style Paintings

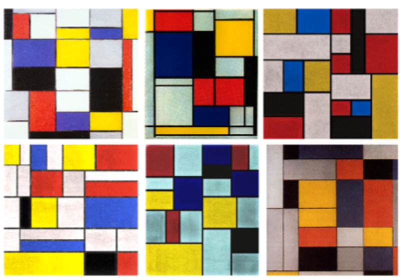
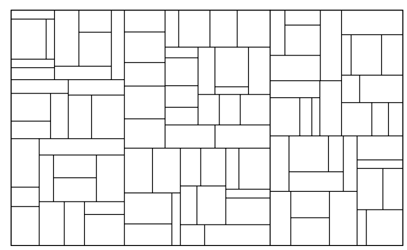


Fig. 11: Three real Mondrian paintings, and three samples from our targeting function. Can you tell which is which?

Source: Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun, "Metropolis Procedural Modeling," *ACM Transactions on Graphics*, April 2011.

Mondrian Decomposition



Methods in the Graphics Library

`GWindow gw(`*width***`,`** *height***`)`** Creates a graphics window with the specified dimensions.
`gw.drawLine(`x_0**`,`** y_0**`,`** x_1**`,`** y_1**`)`** Draws a line connecting the points (x_0, y_0) and (x_1, y_1).
`gw.drawPolarLine(`x_0**`,`** y_0**`,`** *r***`,`** *theta***`)`** Draws a line *r* pixels long in direction *theta* from (x_0, y_0). To make chaining line segments easier, this function returns the ending coordinates as a **`GPoint`**.
`gw.getWidth()` Returns the width of the graphics window.
`gw.getHeight()` Returns the height of the graphics window.

Many more functions exist in the **`gwindow.h`** and **`gobjects.h`** interfaces. The full documentation is available on the web site.

Code for the Mondrian Program

```
#include <iostream>
#include "gwindow.h"
#include "random.h"
using namespace std;

/* Constants */

const double MIN_AREA = 10000;   /* Smallest square that will be split */
const double MIN_EDGE = 20;      /* Smallest edge length allowed       */

/* Function prototypes */

void subdivideCanvas(GWindow & gw, double x, double y,
                                   double width, double height);

/* Main program */

int main() {
   GWindow gw;
   subdivideCanvas(gw, 0, 0, gw.getWidth(), gw.getHeight());
   return 0;
}
```

Code for the Mondrian Program

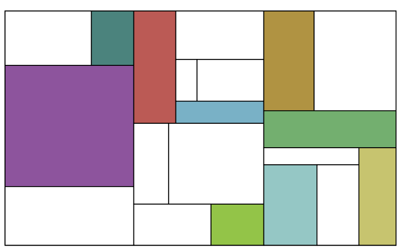
```
void subdivideCanvas(GWindow & gw, double x, double y,
                                  double width, double height) {
   if (width * height >= MIN_AREA) {
      if (width > height) {
         double mid = randomReal(MIN_EDGE, width - MIN_EDGE);
         subdivideCanvas(gw, x, y, mid, height);
         subdivideCanvas(gw, x + mid, y, width - mid, height);
         gw.drawLine(x + mid, y, x + mid, y + height);
      } else {
         double mid = randomReal(MIN_EDGE, height - MIN_EDGE);
         subdivideCanvas(gw, x, y, width, mid);
         subdivideCanvas(gw, x, y + mid, width, height - mid);
         gw.drawLine(x, y + mid, x + width, y + mid);
      }
   }
}
```

Exercise: A Better Mondrian Program

- Can you do a better job of emulating Mondrian's style?
- Suppose that you have the following additional functions:

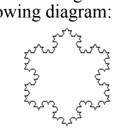
`gw.drawRect(`*x***`,`** *y***`,`** *width***`,`** *height***`)`** Draws the outline of a rectangle with the specified bounds.
`gw.fillRect(`*x***`,`** *y***`,`** *width***`,`** *height***`)`** Fills the outline of the specified rectangle using the current color.
`gw.setColor(`*color***`)`** Sets the pen color to the specified color string (such as `"BLACK"` or `"RED"`)
`gw.setColor("#`*rrggbb***`")`** Sets the red/green/blue components to the specified hexadecimal values.

Revised Mondrian Decomposition



Recursion and Fractals

- Recursion comes up in other graphical applications, most notably in the creation of ***fractals***, which are mathematical structures consisting of similar figures at various different scales. Fractals were popularized in a 1982 book by the late Benoit Mandelbrot entitled *The Fractal Geometry of Nature*.
- One of the simplest fractal patterns to draw is the ***Koch fractal***, named after its inventor, the Swedish mathematician Helge von Koch (1870-1924). The Koch fractal is sometimes called a ***snowflake fractal*** because of the beautiful, six-sided symmetries it displays as the figure becomes more detailed. as illustrated in the following diagram:



How Long is the Coast of England?

- The first widely circulated paper about fractals was a 1967 article in *Science* by Mandelbrot that asked the seemingly innocuous question, "How long is the coast of England?"
- The point that Mandelbrot made in the article is that the answer depends on the measurement scale, as these images from Wikipedia show.
- This thought-experiment serves to illustrate the fact that coastlines are ***fractal*** in that they exhibit the same structure at every level of detail.

Exercise: Fractal Coastline

- Exercise 15 on page 386 asks you to draw a fractal coastline between two points, A and B, on the graphics window.
  - The order-0 coastline is just a straight line.
  - The order-1 coastline replaces that line with one containing a triangular wedge pointing randomly up or down.
  - The order-2 coastline does the same for each line in the order-1.
  - Repeating this process eventually yields an order-5 coastline.