

## Introduction to Recursion

---

Today we'll start working with one of CS106X's neatest ideas: recursion. Recursion is the trick whenever the problem to be solved can be broken down into virtually identical (though smaller) sub-problems. The classic introductory example employing recursion is an implementation of the `factorial` function:

```
int factorial(int n)
{
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

### Readings from the Text:

Today's Lecture: Start Chapters 4 and 5.

Friday's Lecture: Continue with Chapters 4 and 5



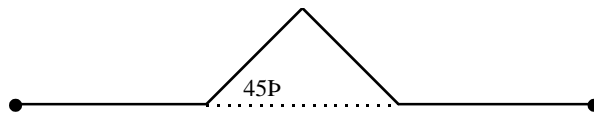
Every recursive function lists a sequence of base cases, and then one or more recursive cases. Occasionally, the problem to be solved is so simple that we can return or terminate execution without any further computation. The first of the two lines in `factorial` is an example of such a base case— $0!$  is always  $1$  and is easily understood. However, whenever the specified integer  $n$  is larger than  $0$ , it helps to calculate  $(n-1)!$  and multiply the result of that computation by  $n$  itself. That's precisely what the recursive call is doing.

We'll be spending a majority of next week learning recursion by example. Recursion is difficult for almost everyone who is learning it, and some of the examples work to help you understand it and others might not. My approach is to supply you will tons and tons of explanations to show how recursion works; hopefully there'll be a point sometime this week where everything start to make sense and fall into place.

## Graphical Recursion

Although fractals have been a mathematical curiosity for more than a century, modern interest in fractals as a practical tool can be traced largely to Benoit Mandelbrot, a researcher at IBM who made an extensive study of the field. As a way of getting people to understand the concept, Mandelbrot posed the following question: How long is the coastline of England? You can look up an answer in an encyclopedia, but that answer turns out to be meaningless unless you know the level of granularity at which it is measured. As you move through successively finer scales, the coast grows progressively longer as you count the length of each little peninsula or inlet.

The coastline problem in the text provides the background for one of today's example. Because the fractals come out a little cleaner if you do so, I've changed the dimensions of the triangle from the one in the text so that the angle of the bump in the fractal line is 45 rather than 60 degrees, like this:



The code for the finished implementation of `DrawCoastline` looks like this:

```
int main()
{
    InitGraphics();
    MovePen(0, GetWindowHeight()/2);
    DrawCoastline(GetWindowWidth(), 0, 7);
    return 0;
}

void DrawCoastline(double length, double theta, int order)
{
    if (order == 0) {
        DrawPolarLine(length, theta);
    } else {
        int sign = (RandomChance(0.5)) ? 1 : -1;
        DrawCoastline(length / 3, theta, order - 1);
        DrawCoastline(length / (3 * sqrt(2)), theta + sign * 45,
            order - 1);
        DrawCoastline(length / (3 * sqrt(2)), theta - sign * 45,
            order - 1);
        DrawCoastline(length / 3, theta, order - 1);
    }
}

void DrawPolarLine(double length, double theta)
{
    double dx = length * cos(3.1416 * theta / 180);
    double dy = length * sin(3.1416 * theta / 180);
    DrawLine(dx, dy);
}
```

## Listing Permutations And Subsets Of Strings

```

/**
 * Snapshot: Permutations
 * -----
 * Exhaustive recursive function to print the n! permutations.
 * At each level of recursion, try each of the choices
 * available (in this case, try all letters in string rest)
 * update the state (append to soFar, remove from rest), and
 * recur from there to try further choices. You hit the
 * base case when you run out of choices (rest is empty)
 */

void Permute(string rest, string soFar)
{
    if (rest.empty()) {
        cout << soFar << endl;
        return;
    }

    for (int i = 0; i < rest.length(); i++) {
        // try rest[i] as next character in permutation being assembled
        string remain = rest.substr(0, i) + rest.substr(i+1);
        Permute(remain, soFar + rest[i]);
    }
}

void ListPermutations(string str)
{
    Permute(str, "");
}

/**
 * Snapshot: Subsets
 * -----
 * Exhaustive recursive function to print the 2^n subsets.
 * At each level of recursion, take the first char of rest
 * and try both in the subset and out of the subset,
 * recur from there to try further choices. You hit the
 * base case when you run out of choices (rest is empty)
 */

void Subset(string rest, string soFar)
{
    if (rest.empty()) {
        cout << soFar << endl;
        return;
    }

    Subset(rest.substr(1), soFar + rest[0]); // recur with rest[0] included
    Subset(rest.substr(1), soFar);         // recur with rest[0] excluded
}

void ListSubsets(string str)
{
    Subset(str, "");
}

```