Recursion, pt. 1

The Foundations
What is Recursion?

• *Recursion* is the idea of solving a problem in terms of itself.
  – For some problems, it may not be possible to find a direct solution.
  – Instead, the problem is typically broken down, progressively, into simpler and simpler versions of itself for evaluation.
What is Recursion?

• One famous problem which is solved in a *recursive* manner: the factorial.
  – $n! = 1$ for $n = 0, n = 1$...
  – $n! = n \times (n-1)!$, $n > 1$.

• Note that aside from the $n=0, n=1$ cases, the factorial’s solution is stated in terms of a *reduced* form of itself.
What is Recursion?

• As long as n is a non-negative integer, n! will eventually reach a reduced form for which there is an exact solution.

  \[ 5! = 5 \times 4! = 5 \times 4 \times 3! = \ldots \]
  \[ = 5 \times 4 \times 3 \times 2 \times 1 \]
What is Recursion?

- From this point, the solution for the reduced problem will be used to determine the exact solution.

\[ 5 \times 4 \times 3 \times 2 \times 1 = 5 \times 4 \times 3 \times 2 \]
\[ = 5 \times 4 \times 6 \]
\[ = 5 \times 24 \]
\[ = 120. \]
Recursion

• Thus, the main idea of recursion is to reduce a complex problem to a combination of operations upon its simplest form.
  – This “simplest form” has a well-established, exact solution.
What is Recursion?

- As a result of how recursion works, it ends up being subject to a number of jokes:
  - “In order to understand recursion, you must understand recursion.”
  - Or, “recursion (n): See recursion.”
• Recursion is actually quite similar to a certain fairly well-known mathematical proof technique: *induction*.
A Mathematical Look

• Proof by induction involves three main parts:
  – A base case with a known solution.
    • Typically, for the most basic version of the problem. Say, for $i = 0$ in a series.
  – A proposed, closed-form solution for any value $k$, which the base case matches.
A Mathematical Look

• Proof by induction involves three main parts:
  – A proof that shows that if the proposed solution works for time step $k$, it works for time step $k + 1$.
  • Typically, it works by showing that the closed form solution for time step $k + 1$ is equal to that given by a known, correct alternative.
A Mathematical Look

- The main idea behind how induction works is the same as that for recursion.
  - The process is merely inverted: the way that induction proves something is how recursion will actually produce its solution.
The Basic Process

- There are two main elements to a recursive solution:
  - The base case: the form (or forms) of the problem for which an exact solution is provided.
  - The recursive step: the reduction of one version the problem to a simpler form.
The Basic Process

• Note that if we progressively reduce the problem, one step at a time, we’ll eventually hit the base case.
  – From there, we take that solution and modify it as necessary on the way back up to yield the true solution.
The Basic Process

• There are thus these two main elements to a recursive solution of the factorial method:
  – The base case: 0! and 1!
  – The recursive step: \( n! = n \times (n-1)! \)
  – Note that the "\( n \times \)" will be applied after the base case is reached.
  – \((n-1)!\) is the reduced form of the problem.
Coding Recursion

- As we’ve already seen, programming languages incorporate the idea of function calls.
  - This allows us to reuse code in multiple locations within a program.
  - Is there any reason that a function shouldn’t be able to reuse itself?
int factorial(int n)
{
    if(n == 0 || n == 1)
        return 1;
    else return n * factorial(n-1);
}

Coding Recursion

• Potential problem: how can the program keep track of its state?
  – There will multiple versions of “n” over the different calls of the factorial function.
  – The answer: stacks!
  – The stack is a data structure we haven’t yet seen, but may examine in brief later in the course.
Coding Recursion

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

• Each individual method call within a recursive process can be called a frame.
Coding Recursion

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

• We’ll use this to denote each frame of this method’s execution.
  – Let’s try n = 5.
Coding Recursion

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

Result: 1
Coding Recursion

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

Result: 2

```
n: 2
n: 3
n: 4
n: 5
```
Coding Recursion

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

Result: 6
Coding Recursion

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

Result: 24
Coding Recursion

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

Result: 120
n: 5