Recursion

What is recursion?

- A way of thinking about problems.
- A method for solving problems.
- Related to mathematical induction.

In programming:

- A function calls itself
  - direct recursion

- A function calls its invoker
  - indirect recursion

int f () {
  ... f(); ...
}

int g () {
  ... g(); ...
  ... f(); ...
}

Outline of a Recursive Function

if (answer is known)
  provide the answer
else
  make a recursive call to solve a smaller version of the same problem

Recursive Factorial Method

- n! = n * (n-1) * (n-2) * ... * 3 * 2 * 1
- n! = n * (n-1)!
- 0! = 1

Algorithm recursiveFactorial(n)

if n==0 then
  return 1
else
  return n * recursiveFactorial(n-1)
Fibonacci sequence

1, 1, 2, 3, 5, 8, 13, 21, ….

\[
\text{fib}(n) = \begin{cases} 
1 & \text{for } n == 1 \\
1 & \text{for } n == 2 \\
\text{fib}(n-2) + \text{fib}(n-1) & \text{for } n > 2 
\end{cases}
\]

Algorithm \(\text{fib}(n)\)
if \(n\leq 2\) then
return 1
else
return \(\text{fib}(n-2) + \text{fib}(n-1)\)

Tracing \(\text{fib}(6)\)

computation repeats!

Euclid's Algorithm

Finds the greatest common divisor of two nonnegative integers that are not both 0

Recursive definition of gcd algorithm
- \(\text{gcd}(a, b) = a\) (if \(b = 0\))
- \(\text{gcd}(a, b) = \text{gcd}(b, a \mod b)\) (if \(b \neq 0\))

Implementation:

```c
int gcd (int a, int b) {
    if (b == 0)
        return a;
    else
        return gcd (b, a \% b);
}
```
Iterative vs. recursive gcd

int gcd (int a, int b) {
    int temp;
    while (b != 0) {
        temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

Multiple recursion

- Tail recursion: a linearly recursive method makes its recursive call as its last step.
  - e.g. recursive gcd
  - Can be easily converted to non-recursive methods
- Binary recursion: there are two recursive calls for each non-base case
  - e.g. fibonacci sequence
- Multiple recursion: makes potentially many recursive calls (not just one or two).

Multiple recursion – Example

List all 'abc' strings of length l

void listAllStrings(int length, char* start) {
    if (length < 1) { //base case: empty string
        *start = '\0';
        output();
    } else { //recursive case: reduce length by 1
        for (char c = 'a'; c <= 'c'; c++) {
            *start = c;
            listAllStrings(length-1, start+1);
        }
    }
}

Why using recursion?

- Recursion makes your code faster? No!
  - overhead for function call and return
  - values recomputed
- Recursion uses less memory? No!
  - overhead for a function call and return (stack memory)
- Recursion makes your code simple? Sometimes.
  - readable code that is easy to debug