Recursion

- Break the task into smaller tasks
  - remarkable concept in Computer Science
  - technically, a recursive function is one that calls itself

- General form:
  - base case
    - solution for a trivial case
    - it is also used to stop the recursion in algorithms, preventing "stack overflow"
  - every recursive algorithm needs at least one base case
  - recursive call(s)
    - divide problem into smaller instances!
    - solve smaller instances with the same code

```c
int foo(int n) {
    int a = 10;
    n += a;
    return n;
}

int bar(int n) {
    int a = 10;
    n = foo(n);
    return a * n;
}
```

Functions
Definition. The factorial function is defined by the product:

\[ n! = \prod_{k=1}^{n} k \]

or by the recurrence relation:

\[ n! = \begin{cases} 1 & \text{if } n = 0 \\ (n - 1)! \times n & \text{if } n > 0 \end{cases} \]

Example: factorial

```c
int factorial(int n) {
    // base case
    if (n < 2) {
        return 1;
    }
    // recursive call
    return n * factorial(n-1);
}
```

Example: power of a number

Definition. The \(n\)-th power of \(b\) is defined by:

\[ b^n = b \times b \times \cdots \times b \quad \text{\(n\) times} \]

```c
int power(int x, int n) {
    // base case
    if (n == 0) {
        return 1;
    }
    // recursive call
    return x * power(x, n-1);
}
```

Example: sum of array

```c
int sum_array(int *A, int n) {
    if (n == 1) {
        return A[0];
    }
    // recursive call
    return A[n-1] + sum_array(A, n-1);
}
```
Example: decimal to binary

```cpp
void print_binary(int n) {
    if (n > 0) {
        print_binary(n/2);
        std::cout << n % 2;
    }
}
```

Indirect Recursion

```cpp
void f2(int n);

void f1(int n) {
    if (n > 1) {
        std::cout << "1";
        f2(n - 1);
    }
}

void f2(int n) {
    std::cout << "0";
    f1(n - 1);
}
```

Review

- Pointers!
- How programs are executed?
- Memory model
  - stack and heap
- How parameters are passed to functions?
  - passing arrays