Functions

```c
1 int foo(int n) {
2     int a = 10;
3     n += a;
4     return n;
5 }
6
7 int bar(int n) {
8     int a = 10;
9     n = foo(n);
10    return a * n;
11 }
```

Administrativia

- Gradescope
  - writing ‘correct’ code is not easy (especially in C/C++)
  - requires practice, discipline, perseverance, resilience

- Piazza
  - test cases can be shared, however, in an organized way
  - can also be used to discuss lecture topics

Recursion - Wikipedia
https://en.wikipedia.org/wiki/Recursion
Recursion occurs when a thing is defined in terms of itself or of its type. Recursion is used in a variety of disciplines ranging from linguistics to logic. The most common application of recursion is in mathematics and computer science, where a function being defined is applied within its own definition. Recursion (computer science) · Recursion · Recursion (disambiguation)

Recursion (computer science) - Wikipedia
https://en.wikipedia.org/wiki/Recursion_(computer_science)
In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions. Most computer programming languages support recursion by allowing a function to call itself within the program text.
Types of recursion · Recursive programs · Recursion versus iteration
Recursion

- Solve a task by reducing it to smaller tasks (of the same form)
  - powerful tool 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 using the same code

Demo

- Count the number of students behind you
  - … and include yourself

General From

```javascript
function() {
    if (this is the base case) {
        calculate trivial solution
    } else {
        break task into subtasks
        solve each task recursively
        merge solutions if necessary
    }
}
```

Example: factorial

**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);
}
```

Day in a life of a recursive call

```c
int val = factorial(4);
```

Precondition

- Specify the **input conditions** to a function (contract)
  - what is the expected input for the factorial example?
- Use **assert**
  - terminates program execution
- Use **throw, try, catch**
  - throws and exception that can be caught later
- .. or just let the program misbehave or crash later
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];
    }
    return A[n-1] + sum_array(A, n-1);
}
```

Example: decimal to binary

```c
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);
}
```

- `f1(6)` ?
- `f1(5)` ?
- `f1(10)` ?