CS 2: Introduction to Programming Methods

Solving Recursive Problems

Write Down The Method Header

Just like normal, you should begin by writing down the method header. Be careful to leave extra space, because sometimes recursion takes you by surprise and you end up needing extra arguments:

Identify The Recursive Argument

When you write a method recursively, there will be one argument that you are "recursing" on. This argument is the one that you are "checking" or "using". First, identify which argument this is.

The recursive argument is .

Finding The Base Case

The **type** of the recursive argument is super important to the rest of the problem. You will want to find the "smallest" version of that type. Or the version of that type where "there's nothing left". Try to fill in the following:

The type of the recursive argument is .

When we have , which is a particular thing the same type as the recursive argument, I'd say "there's nothing left to consider".

For example, if the type is `int`, when we have 0, there's nothing left to consider.

Filing In The Base Case

Now that we know what the base case is, we need to actually write a solution to it.

Splitting Up The Problem

The next step is to figure out how to split up the problem. Most types in recursive problems have indexes (lists, strings, arrays, . . .). There are generally two ways to split problems:

- Solve the problem for index 0 and use recursion to solve it for the rest of the indexes
- Solve the problem, recursively, for the first half of the indexes *and* the second half of the indexes.

Always try the first strategy first.

```
if (                                    ){
```
recursive argument = base case

solve the problem directly for this case

```
}
else {
                                        ;
```
get the value at index 0 and store in a variable

solve the problem directly for index 0

```
                                        ;
```
get the "rest" of the indexes and store in a variable

```
                                        ;
```
call the method recursively on the "rest"

```
                                        ;
```
combine the results for index 0 and the "rest" and return

```
}
```