

Recursion

Jan 31

Definition

Recurs: to occur again.

Methods/functions/procedures that call themselves.

Recursion is a very powerful programming technique that can be used in place of loops. (but requires a different way of thinking about the problem)

E.g. Factorial

Standard Defn.

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n * (n - 1) * (n-2) \dots (2 * 1) & \text{if } n > 0 \end{cases}$$

Recursive Defn.

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0. \end{cases}$$

Code:

```
// recursive definition
static long factorial (int n) {
    if (n == 0) {
        Return 1;
    } else {
        return n * factorial (n - 1);
    }
}

// non recursive algorithm
static long factorial (int n) {
    long product = 1;
    for (int i = 2; i <= n; i++) {
        product *= i;
    }
    return product;
}
```

Recursive methods are a natural for recursive data structures.

Verifying Recursive Methods

In verifying recursive methods, its useful to ask three questions:

- The base case: Is there a non recursive way out of the method and does it work correctly?
- Smaller case: Does each recursive call of the method involve a smaller case of the original problem?
- The General Case: Assuming the recursive calls work correctly, does the whole method work correctly?

How to write recursive methods

- Get an exact definition of the problem you want to solve
- Determine the size of the problem. (what needs to be made smaller with each recursive call?)
- Solve the base case. (When the problem can be expressed non recursively)
- Solve the general case correctly in terms of the smaller case of the same problem.

E.g. Recursive list processing

```
class ListNode {
    int info;
    ListNode next;
}
```

E.g. Printing a linked list:

```
/* Iterative solution */
static void printLinkedList (ListNode list) {
    while (list != null) {
        System.out.println (list.info);
        list = list.next;
    }
}
```

```
/* recursive solution */
static void printLinkedList (ListNode list) {
    if (list != null) {
        System.out.println(list.info);
        printLinkedList(list.next)
    }
}
```

The above method says “print this node, then print all the rest of the nodes.”
How would we print the list backwards?

```
/* recursively printing a linked list backwards */
static void reversePrint(ListNode list) {
    if (list != null) {
        RervePrint(list.next);
        System.out.println(list.info);
    }
}
```

“Print all the rest of the nodes, then print this node.” Or another way of thinking
This method effectively jumps to the end and starts running down, as opposed to starting at the beginning and running up.