Recursion

Recursive function: function that calls itself; useful when problem can be reduced to smaller instances of same problem

Example: void f( ) { ... f( ); ...}

Indirect recursion: Example: void f( ) { ... g( ); ...} void g( ) { ... f( ); ...}

Tail recursion: only recursive call is at very end of the function; useful in algorithms to eliminate recursion (*why eliminate recursion????....*)

Stopping case: at least one case where entire computation completed w/o recursion

Activation record: info recorded so that function knows where to return when it's done; contains values for local variables, parameters, and return location in code (where you left off); done internally using a stack

Example:

```
// Precondition: n ≥ 1
// Postcondition: value returned is (1 * 2 * … * (n-1) * n)

int factorial (const int n) {
int fact = 1;
if (n > 1)
  fact = n * factorial(n-1);
return(fact);
}
```

Variant expression: numeric qty that's decreased by some fixed amount on each recursive call

Threshold: value (for the variant expression) that guarantees a stopping case

Ensuring No Infinite Recursion

Find a variant expression and a threshold with following properties:

(1) Between one call of function and any succeeding recursive call of function, **value of variant expression decreases by at least some fixed amount**.

(2) If function called and **value of the variant expression is ≤ threshold**, then function **terminates** w/o making any recursive calls.

Example: n decreases by 1 in factorial(n-1) call

if n **≤** 1 we just return(fact)

Showing Recursive Function is Correct (via Inductive Reasoning)

(1) Show that there isn't infinite recursion

(2) Show that whenever function makes no recursive calls, it meets its pre- and post-conditions (i.e., **base step**)

(3) Show that whenever function called and recursive calls it makes meet their pre- and post-conditions, then original call also meets its pre- and post-conditions (i.e., **inductive step**)

Example: (1) already showed it terminates
(2) only terminates w/o recursive call if n **=** 1; returns fact = 1 (which is 1!)
(3) n! = n * (n-1)!

Frequently Asked Questions

Q: Does using recursion usually make your code **faster**?

Q: Does using recursion usually use **less memory**?

Q: So **why** use recursion?