What is Recursion, Operation of Functions Recursive or not, how Recursion works.

Making Hard Things Easy

CSI 310: Lecture 15
This idea is too broad to be helpful.

Understand the hard things by thinking about easier things instead...

Making Hard Things Easy:
Some values are like 2, 3, 4 and other values are used as addresses which specify value stored in each variable. Every data structure consists of variables (storage facilities) and the direction.

Hard: Data structures and programs that use pointers, references, and

Concrete Example 1:
This accurate understanding includes the idea of an activation (different from a situation that involves recursion. But accurate understanding of these topics is necessary for applying them to independently of whether the functions are recursive or not.

Arguments and parameters, meaning of basic programming statements, calling is, how it works, details about local external variables and function Easy: Don't think about recursion at all. Instead, concentrate on what function Hard: Recursion, self-reference, something defined in terms of itself.

Concrete Example 2:
But, running \( \text{fact}(0) \) calculates \( 0! = 0 \) is true and returns 1. Since \( \text{fact}(1) \) computes \( 1! = 1 \) and returns 1, then \( \text{fact}(2) \) computes \( 2! = 2 \times 1 \) and returns 2. For example, running \( \text{fact}(7) \) computes \( 7! = 5040 \) and returns 5040, which returns.

\[
\begin{cases}
\text{if } n = 0 \text{ then return } 1; \\
\text{else return } \text{fact}(n-1) \times n;
\end{cases}
\]

int fact(int n) {...}

Example: When the function runs, calls the same function, either directly or indirectly.

Definition: A function is recursive means the body of the function sometimes, re-implement the function itself.

(3) Elegantly way to write programs, performance can be improved with runtime.

(2) Powerful problem solving technique.

(1) Understanding, not just programming, data structures and algorithms.

Recursion: Trees, Recursion, Expressions. Stacks are closely related.

University at Albany Computer Science Dept.
SAME TIME while the recursive function runs.

...there will be allocated MANY DIFFERENT variables named _at the activation of the function._

**IMPORTANT:** There is a separate local extent variable for each and every which means something else.

Most people say „local” for short, but they get confused about „local scope” synonym for „extent”. Automatic variables are often called _local extent variables_. „Lifetime” is a

\[
\text{int\ fact(\int n)} \begin{cases} \text{return fact(n-1)}; & \text{if } n=0 \text{ return } 1; \\ \text{return } \text{fact}(n-1); & \end{cases}
\]
\begin{verbatim}
{ int C416(int n) { steep(1000000000); return -1; }
{ int C415(int n) { if (n==1) return 1 else return C415(n-1)*n; }
{ int C414(int n) { if (n==1) return 1 else return C414(n-1)*n; }
{ int C413(int n) { if (n==1) return 1 else return C413(n-1)*n; }
{ int C412(int n) { if (n==1) return 1 else return C412(n-1)*n; }
{ int HATCAT(int n) { int ret = HATCAT(n-1); count += ret; ret >> 2; main() { int ret; ret = HATCAT(4); } }
\end{verbatim}

But let's begin without any recursion at all. With apologies to Dr. Suehs,
Only true "logically"...this data is actually stored in the called activation.

**ACTIVATION** is destroyed, and its Record gets recycled.

Really: When an **ACTIVATION** executes the **return** statement, that

spot within the function's body it this activation **called a function**

Differently: One whose CALL operation created this one. (3) The return

An activation of **Activation Record**: The data structure that holds (1)

An activation will control what THAT activation does.

Really: A new **function activation** is created, and the function's body

Wrong: Control "jumps" or "goes to" the function's body.

What HAPPENS when the computer executes a **FUNCTION CALL**?
3. A stack is a sequence for which insertion and deletion are only done at ONE end (called the top).

2. Activation records are stored in a STACK (like a pile of bills).

First, note \( n \) is a local variable.

\[
\begin{align*}
\text{int fact(int n)} \{ \\
\text{if (n==0) return 1; else return fact(n-1)*n;} \\
\}
\end{align*}
\]
We now demonstrate what happens when the computer executes:

\[ \text{if} \ (n = 0) \ \text{return} \ 1; \]
\[ \text{if} \ (n < 0) \ \text{return} \ 1; \]
\[ \text{int func(int n);} \]
\[ \{ \text{else return func(n-1);} \} \]
\[ \{ \text{else return func(n+1);} \} \]
\[ \{ \text{else return func(n-1);} \} \]
Different activations—Different automatic variables

The store used for automatic variables is in the activation record.

The event of "calling one function once" is an activation.

Each time a function is called, an activation record is created.

University at Albany Computer Science Dept.
3. $\text{fact}(5)$

2. $\text{fact}(2)$

1. $\text{fact}(0)$

\[
\begin{array}{l}
\{ \text{else return fact(n-1)} \}
\end{array}
\]

\[
\begin{array}{l}
\{ \text{if(n==0) return 1;} \}
\end{array}
\]

\[
\begin{array}{l}
\text{int fact(n)}
\end{array}
\]

Let's do it again with a recursive function.
4. Now, automatic variables are in the activation record of the activation.

RETURNED TO.

3. The ACTIVATION whose CALL originally created this activation is

RETURNED TO.

2. This ACTIVATION'S ACTIVATION RECORD "goes away"

1. The return value (if any) is saved for use by the caller.

When a function ACTIVATION executes the return statement: ...