Program Example: Factorial Function

```c
int factorial(int n) {
    if(n > 1) {
        return n * factorial(n - 1);
    }
    else {
        return 1;
    }
}
```
Stacks for Recursion

- Computers use a structure called a stack to keep track of recursion
  - A stack is a memory structure analogous to a stack of paper
    - To place information on the stack, write it on a piece of paper and place it on top of the stack
    - To place more information on the stack, use a clean sheet of paper, write the information, and place it on the top of the stack
    - To retrieve information, only the top sheet of paper can be read, and thrown away when it is no longer needed
Last-in / First-out

- A stack is a last-in/first-out memory structure
  - The last item placed is the first that can be removed
- Whenever a function is called, the computer uses a "clean sheet of paper"
  - The function definition is copied to the paper
  - The arguments are plugged in for the parameters
  - The computer starts to execute the function body
Stacks and The Recursive Call

- When execution of a function definition reaches a recursive call
  - Execution stops
  - Information is saved on a "clean sheet of paper" to enable resumption of execution later
  - This sheet of paper is placed on top of the stack
- A new sheet is used for the recursive call
  - A new function definition is written, and arguments are plugged into parameters
  - Execution of the recursive call begins
The Stack and Ending Recursive Calls

- When a recursive function call is able to complete its computation with no recursive calls
  - The computer retrieves the top "sheet of paper" from the stack and resumes computation based on the information on the sheet
  - When that computation ends, that sheet of paper is discarded and the next sheet of paper on the stack is retrieved so that processing can resume
  - The process continues until no sheets remain in the stack
Activation Frames

- The computer does not use paper
- Portions of memory are used
  - The contents of these portions of memory is called an activation frame
  - The activation frame does not actually contain a copy of the function definition, but references a single copy of the function
Stack Overflow

- Because each recursive call causes an activation frame to be placed on the stack
  - infinite recursion can force the stack to grow beyond its limits to accommodate all the activation frames required
  - The result is a stack overflow
  - A stack overflow causes abnormal termination of the program