Function Call Implementation

- An easy way to implement factorial? [recursion]

```c
/**
 * factorial.c
 */
#include <stdio.h>

int factorial (int n) {
    if (n < 2)
        return 1;
    else
        return n * factorial (n-1);
}

int main () {
    int n = 3;
    int i = factorial (n);
    printf("factorial(%d) = %d\n", n, i);
    return 0;
}
```

- A key component of a recursive function?
  - The **base case**. Otherwise, the function recurs forever.
  - In the above program, the base case is the if branch.

- How does recursive function work?
  - The factorial function is first called in the main function, and passes value 3 to it.
  - When executing the factorial function, factorial function is called again, and passed value 2 to it.
  - When calculating factorial 2, the function is called for the third time to calculate factorial 1.
  - When get the return value of factorial 1, factorial 2 is calculated.
  - After getting the value of factorial 2, factorial 3 is calculated.
  - Then the final value is returned to the main function.

- This process makes sense logically. But **how the computer handles it? How are those active records saved for recursive calls?**
  - Generally, the computer uses the system or program **stack** to hold active records.
  - **Active records, also called stack frames**, need to hold the following types of information.
    - Return address
    - Return value
    - Arguments
    - Parameters
    - Local variables
  - This stack is a up-side-down stack, and for simplicity, stack frame only holds arguments and return values.
factorial (3) -> factorial (2) -> factorial (1)

push(r1)
- n=3
- return = ?

push(r2)
- n=3
- return = ?
- n=2
- return = ?

push(r3)
- n=3
- return = ?
- n=2
- return = ?
- n=1
- return = 1

pop(r1)
- n=3
- return = 6

pop(r2)
- n=3
- return = ?
- n=2
- return = 2

pop(r3)
- n=3
- return = ?
- n=2
- return = ?
- n=1
- return = 1