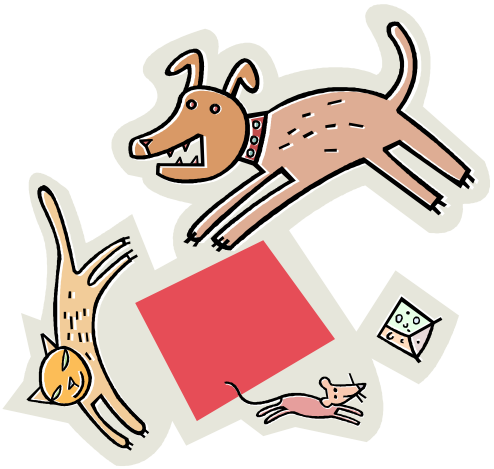
2장 순환

순환(recursion)이란?

- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법

- 정의자체가 순환적으로 되어 있는 경우에 적합한 방법



팩토리얼 프로그래밍 #1

- 팩토리얼의 정의

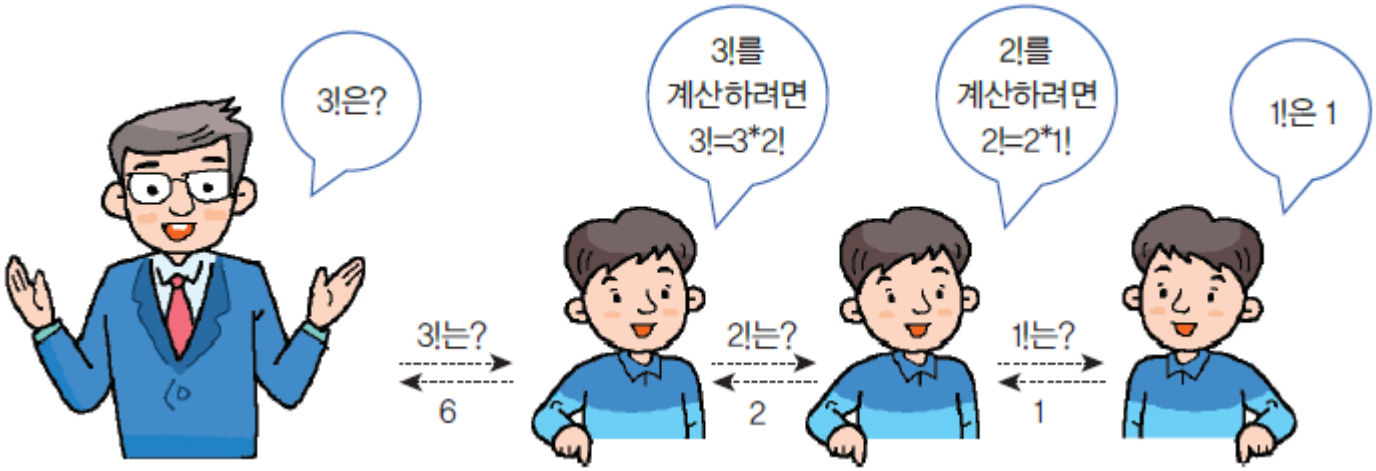
$$n! = \begin{cases} 1 & n = 0 \\ n*(n-1)! & n \geq 1 \end{cases}$$

팩토리얼 프로그래밍 #1

```
int factorial(int n)
{
   if( n<= 1 ) return(1);
   else return (n * factorial_n_1(n-1) );
}
```

팩토리얼 프로그래밍 #2

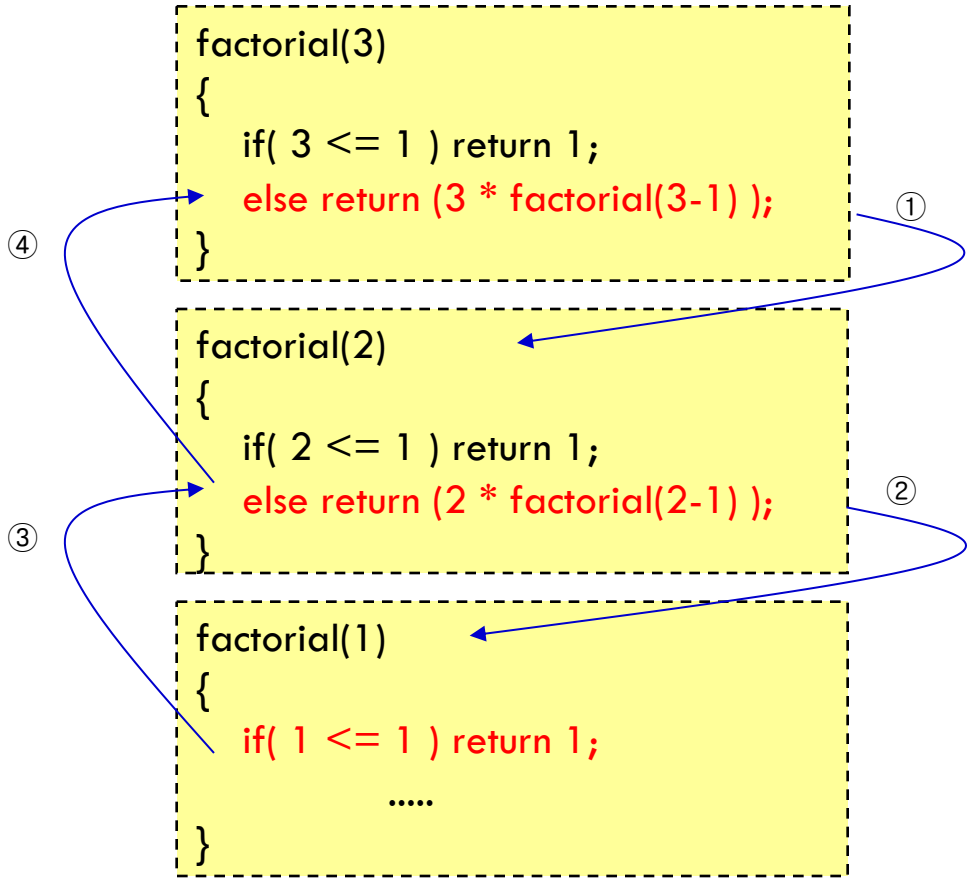
```
int factorial(int n)
{
    if( n <= 1 ) return(1);
    else return (n * factorial(n-1) );
}
```



순환호출순서

- 팩토리얼 함수의 호출 순서

factorial(5) = 5 * factorial(4)
= 5 * 4 * factorial(3)
= 5 * 4 * 3 * factorial(2)
= 5 * 4 * 3 * 2 * factorial(1)
= 5 * 4 * 3 * 2 * 1
= 120



순환 알고리즘의 구조

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

순환을 멈추는 부분

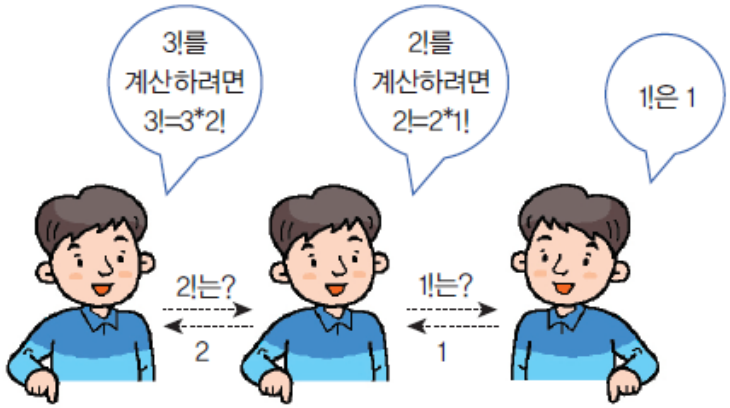
순환 호출을 하는 부분

- 만약 순환 호출을 멈추는 부분이 없다면?.
  - 시스템 오류가 발생할 때까지 무한정 호출하게 된다.

순환 <-> 반복

- 대부분의 순환은 반복으로 바꾸어 작성할 수 있다.

(a) 반복



(b) 순환

팩토리얼의 반복적 구현

$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)*(n-2)*\cdots*1 & n \geq 2 \end{cases}$$

```
int factorial_iter(int n)
{
    int k, v=1;
    for(k=n; k>0; k--)
        v = v*k;
    return(v);
}
```

거듭제곱 값 프로그래밍 #1

- 순환적인 방법이 더 효율적인 예제
- 숫자 x의 n제곱 값을 구하는 문제: x^n

반복적인 방법

```
double slow_power(double x, int n)
  {
        int i;
        double result = 1.0;
        for(i=0; i<n; i++)
              result = result * x;
        return(result);
  }
```

순환적인 방법

- 순환적인 알고리즘

```
power(x, n)

if n==0
        then return 1;
else if n이 짝수
        then return power(x^2, n/2);
else if n이 홀수
        then return x*power(x^2, (n-1)/2);
```

순환적인 방법

```
double power(double x, int n)
{
        if( n==0 ) return 1;
        else if ( (n%2)==0 )
                return power(x*x, n/2);
        else return x*power(x*x, (n-1)/2);
}
```

거듭제곱 값 프로그래밍 분석

- 순환적인 방법의 시간 복잡도
  - 만약 n이 2의 제곱이라고 가정하면 다음과 같이 문제의 크기가 줄어든다.

$$2^n \rightarrow 2^{n-1} \rightarrow \cdots 2^2 \rightarrow 2^1 \rightarrow 2^0$$

- 반복적인 방법과 순환적인 방법의 비교

	반복적인 함수 slow_power	순환적인 함수 power
시간복잡도	O(n)	O(logn)
실제수행속도	7.17초	0.47초

피보나치 수열의 계산 #1

- 순환 호출을 사용하면 비효율적인 예
- 피보나치 수열

  0,1,1,2,3,5,8,13,21,…

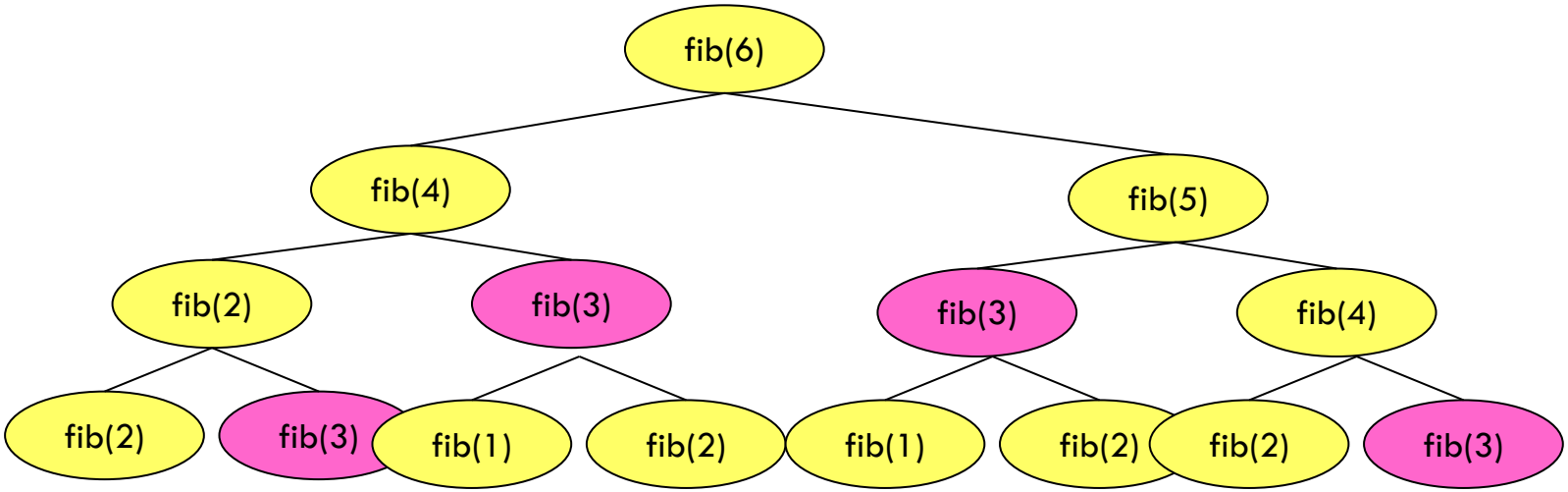
$$fib(n)\begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2)+fib(n-1) & otherwise \end{cases}$$

피보나치 수열의 계산 #1

```
int fib(int n)
{
    if( n==0 ) return 0;
    if( n==1 ) return 1;
    return (fib(n-1) + fib(n-2));
}
```

피보나치 수열의 계산 #1

- 순환 호출을 사용했을 경우의 비효율성
  - 같은 항이 중복해서 계산됨
  - 예를 들어 fib(6)을 호출하게 되면 fib(3)이 4번이나 중복되어서 계산됨
  - 이러한 현상은 n이 커지면 더 심해짐

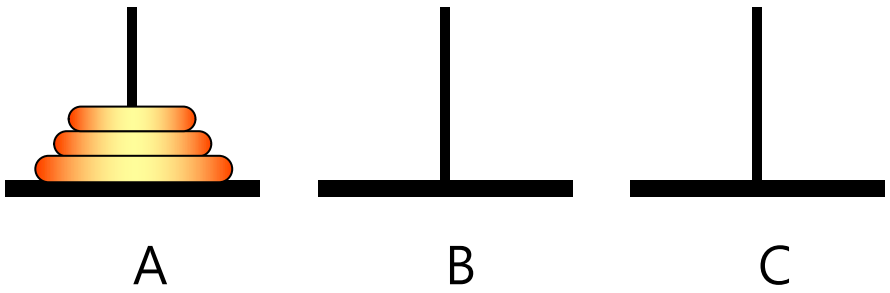


피보나치 수열의 반복구현

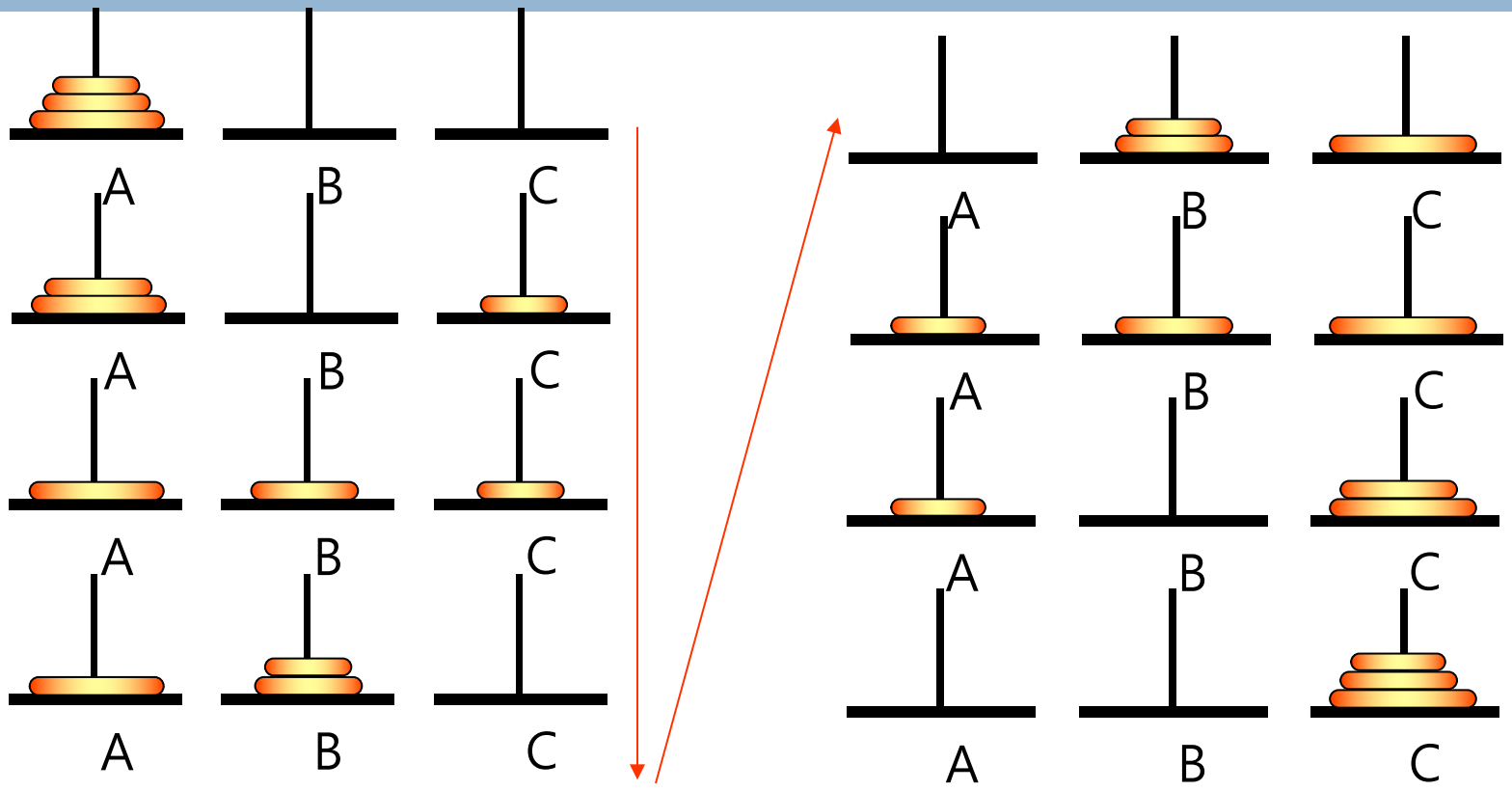
```
int fib_iter(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;

    int pp = 0;
    int p = 1;
    int result = 0;

    for (int i = 2; i <= n; i++) {
        result = p + pp;
        pp = p;
        p = result;
    }
    return result;
}
```

하노이 탑 문제

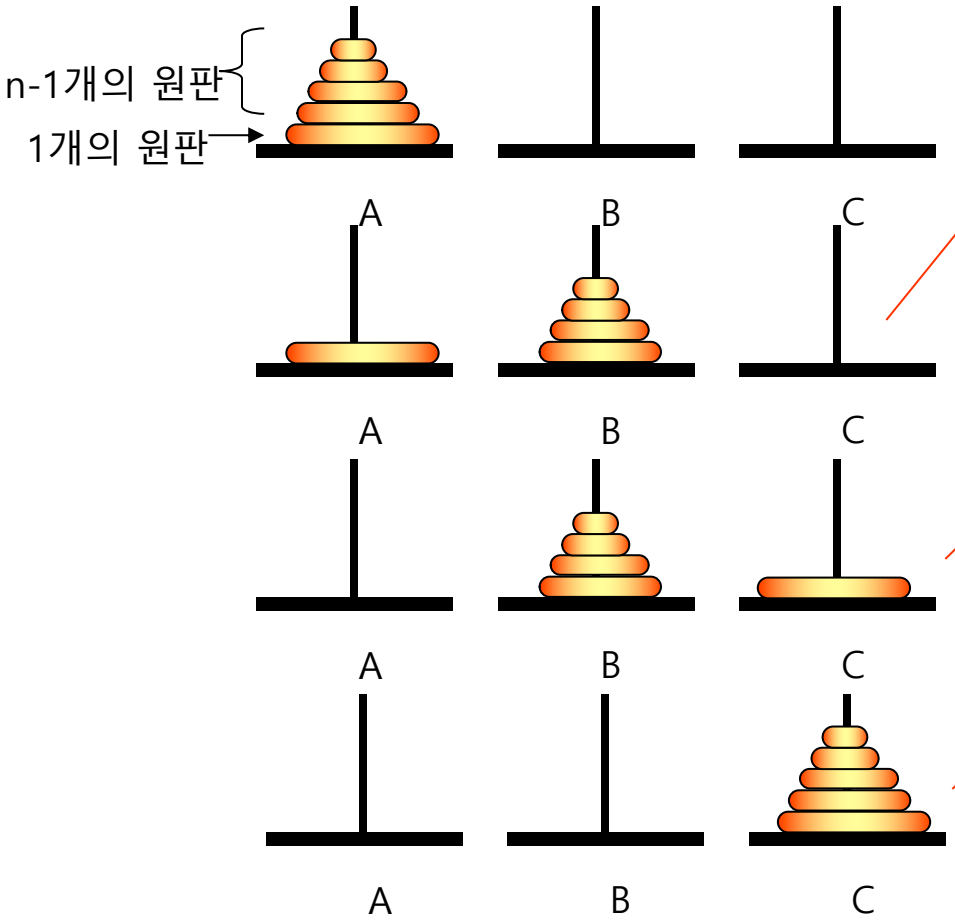
- 문제는 막대 A에 쌓여있는 원판 n개를 막대 C로 옮기는 것이다.
  - 한 번에 하나의 원판만 이동할 수 있다
  - 맨 위에 있는 원판만 이동할 수 있다
  - 크기가 작은 원판 위에 큰 원판이 쌓일 수 없다.
  - 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.



n=3인 경우의 해답



일반적인 경우에는?



C를 임시버퍼로 사용하여 A에 쌓여있는 n-1개의 원판을 B로 옮긴다.

A의 가장 큰 원판을 C로 옮긴다.

A를 임시버퍼로 사용하여 B에 쌓여있는 n-1개의 원판을 C로 옮긴다.

남아있는 문제는?

- 자,그러면 어떻게 n-1개의 원판을 A에서 B로, 또 B에서 C로 이동하는가?
- (힌트) 우리의 원래 문제가 **n**개의 원판을 **A**에서 **C**로 옮기는 것임을 기억하라.
- -> 따라서 지금 작성하고 있는 함수의 매개변수를 n-1로 바꾸어 순환 호출하면 된다.

남아있는 문제는?

```
// 막대 from에 쌓여있는 n개의 원판을 막대 tmp를 사용하여 막대 to로 // 옮
    긴다.
void hanoi_tower(int n, char from, char tmp, char to)
{
  if (n==1){
     from에서 to로 원판을 옮긴다.
  }
  else{
     hanoi_tower(n-1, from, to, tmp);
     from에 있는 한 개의 원판을 to로 옮긴다.
     hanoi_tower(n-1, tmp, from, to);
  }
}
```

하노이탑 최종 프로그램

```
#include <stdio.h>
void hanoi_tower(int n, char from, char tmp, char to)
{
  if( n==1 ) printf("원판 1을 %c 에서 %c으로 옮긴다.\n",from,to);
    else {
            hanoi_tower(n-1, from, to, tmp);
            printf("원판 %d을 %c에서 %c으로 옮긴다.\n",n, from, to);
            hanoi_tower(n-1, tmp, from, to);
  }
}
int main(void)
{
    hanoi_tower(4, 'A', 'B', 'C');
    retrun 0;
}
```

실행결과

```
원판 1을 A 에서 B으로 옮긴다.
원판 2을 A에서 C으로 옮긴다.
원판 1을 B 에서 C으로 옮긴다.
원판 3을 A에서 B으로 옮긴다.
원판 1을 C 에서 A으로 옮긴다.
원판 2을 C에서 B으로 옮긴다.
원판 1을 A 에서 B으로 옮긴다.
원판 4을 A에서 C으로 옮긴다.
원판 1을 B 에서 C으로 옮긴다.
원판 2을 B에서 A으로 옮긴다.
원판 1을 C 에서 A으로 옮긴다.
원판 3을 B에서 C으로 옮긴다.
원판 1을 A 에서 B으로 옮긴다.
원판 2을 A에서 C으로 옮긴다.
원판 1을 B 에서 C으로 옮긴다.
```

Q & A