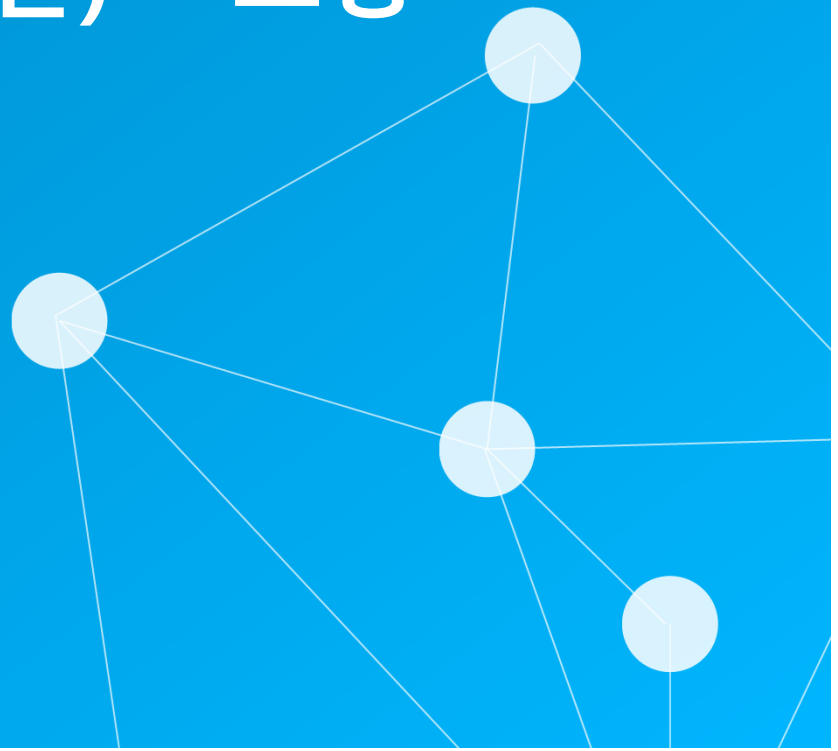
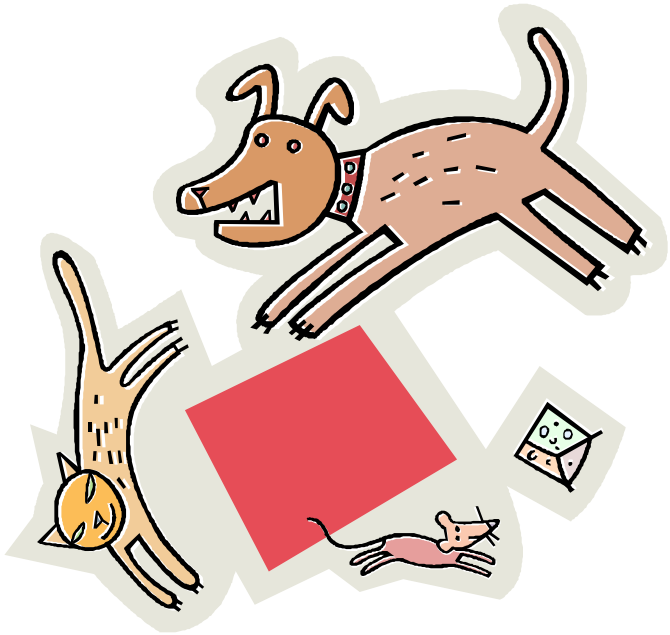
DATA
STRUCTURES
USING C++

재귀 (순환) – 보충



재귀(순환, recursion)이란?

- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법
- 정의자체가 순환적으로 되어 있는 경우에 적합한 방법



Some Definitions

- **Recursive algorithm** A solution that is expressed in terms of
  (a) a base case and
  (b) smaller instances of itself

  - **Base case:** the case for which the solution can be stated nonrecursively
  - **General (recursive) case:** the case for which the solution is expressed in terms of a smaller version of itself

- **Recursive call** : 함수 자신을 호출

General format for many recursive functions

```
if   (some condition for which answer is known)
                                          // base case
     solution statement

else                                      // general case
     recursive function call
```

- Each successive recursive call should bring you closer to a situation in which the answer is known.
- Each recursive algorithm must have at least one base case, as well as the general (recursive) case

재귀(recursion)의 이해

- 팩토리얼 값 구하기

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

- 피보나치 수열

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

- 이항계수

$${}_nC_k = \begin{cases} 1 & k = 0 \quad or \quad k = n \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & otherwise(0 < k < n) \end{cases}$$

- 하노이의 탑
- 이진탐색

팩토리얼 함수

n!을 구하는 함수

```
def factorial(n):
    if( n == 1 ):
        return 1
    else:
        return n * factorial(n-1)
```

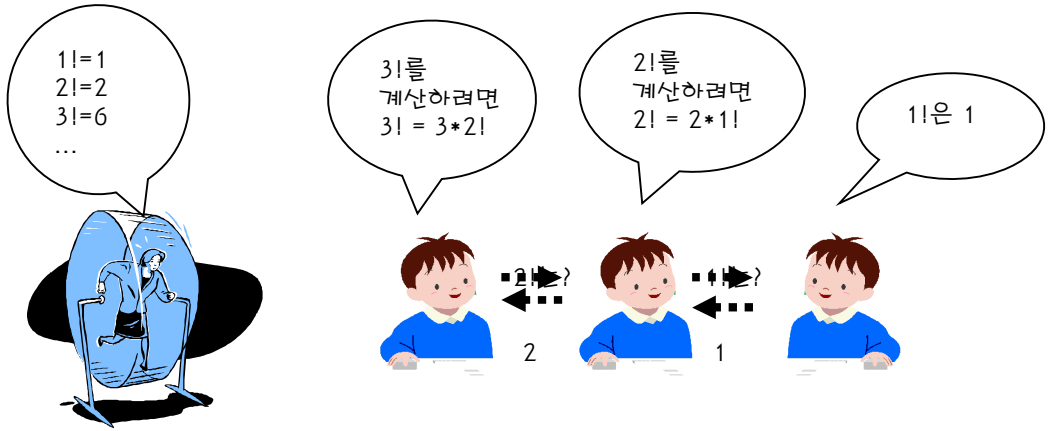
재귀 알고리즘의 구조

- 만약 재귀 호출을 멈추는 부분(base case)이 없다면?
  - 시스템 오류가 발생할 때까지 무한정 호출하게 된다

```
def factorial(n):
    return n * factorial(n-1)
```

재귀 <-> 반복

- 대부분의 재귀는 반복으로 바꾸어 작성할 수 있음
  - 재귀(recursion): 재귀 호출 이용
    - 재귀적인 방법인 경우에 자연스러운 방법
    - 함수 호출의 오버헤드 (실행시간에서 손해보지만 코드가 간단해서 가독성↑)
  - 반복(iteration): for나 while을 이용한 반복 (스택을 사용)
    - 수행속도가 빠르다.
    - 재귀를 반복을 이용할 경우 프로그램 작성이 아주 어려울 수도 있다.



Tail 재귀 <-> 반복

- Tail recursion인 경우 스택을 사용하지 않고 반복으로 바꾸어 작성할 수 있음
  - A recursive function is tail recursive when recursive call is the last thing executed by the function
    재귀 호출이 함수에서 마지막으로 수행되는 작업

```
def factorial(n):
    if( n == 1 ):
        return 1
    else:
        return n * factorial(n-1)
```

```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

base case
재귀를 멈추는 부분

recursive case
재귀호출을 하는 부분

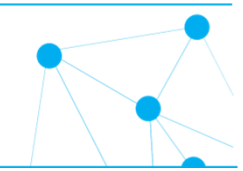
tail recursion

팩토리얼의 반복적 구현

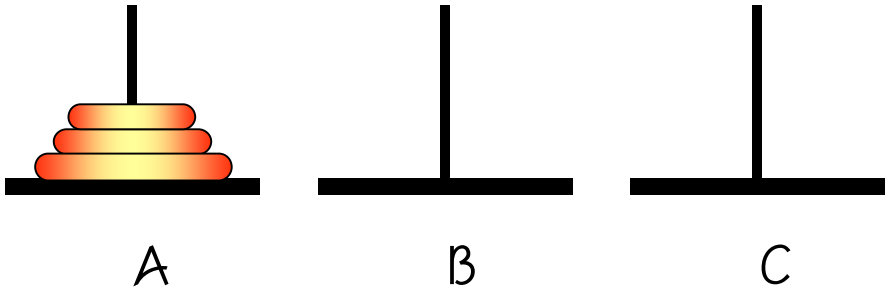
$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)*(n-2)*\cdots*1 \quad \ldots & n \geq 2 \end{cases}$$

```
def factorialIter(n):
    result=1
    for i in range(1,n+1):
        result = result * i
    return result
```

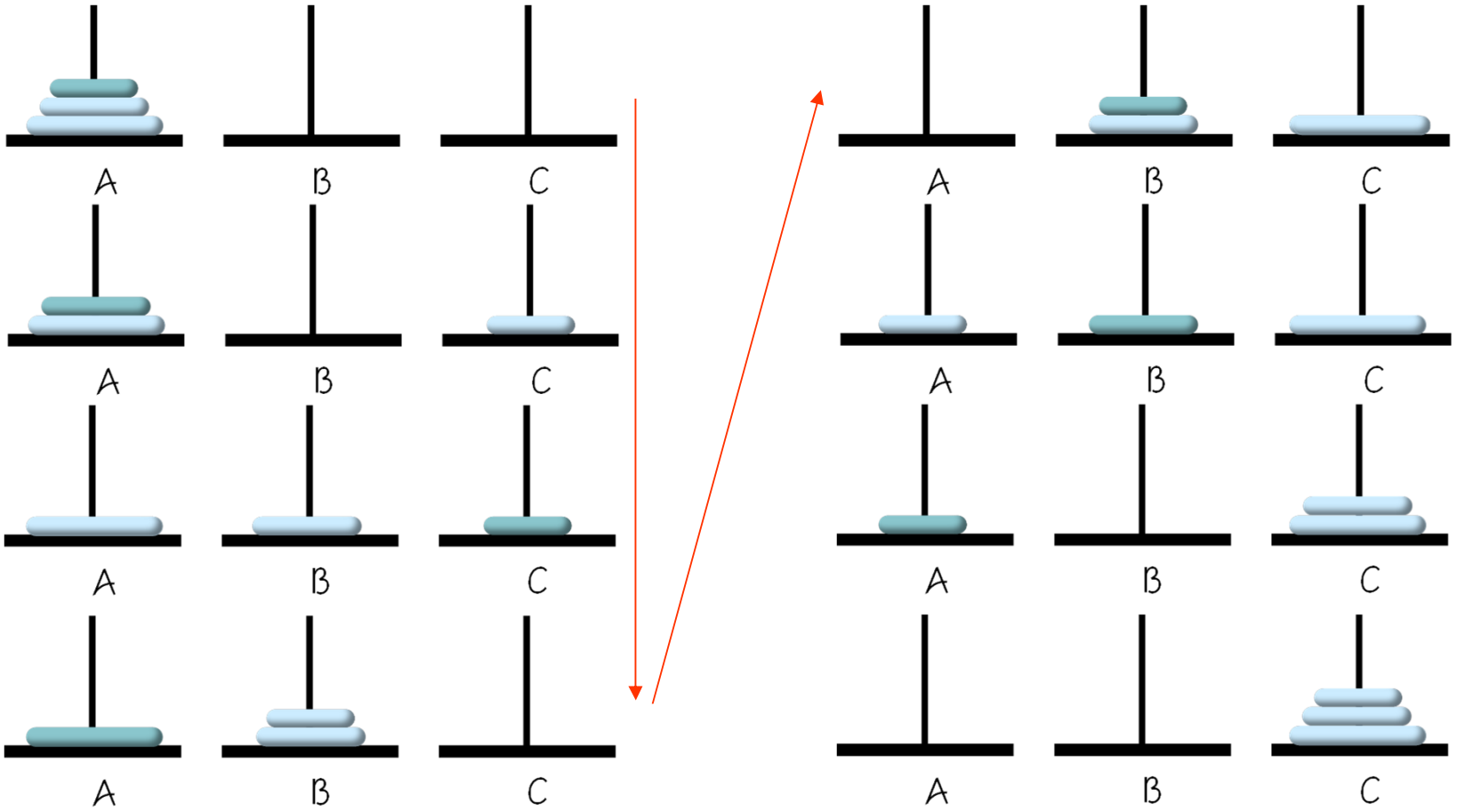
하노이 탑 문제



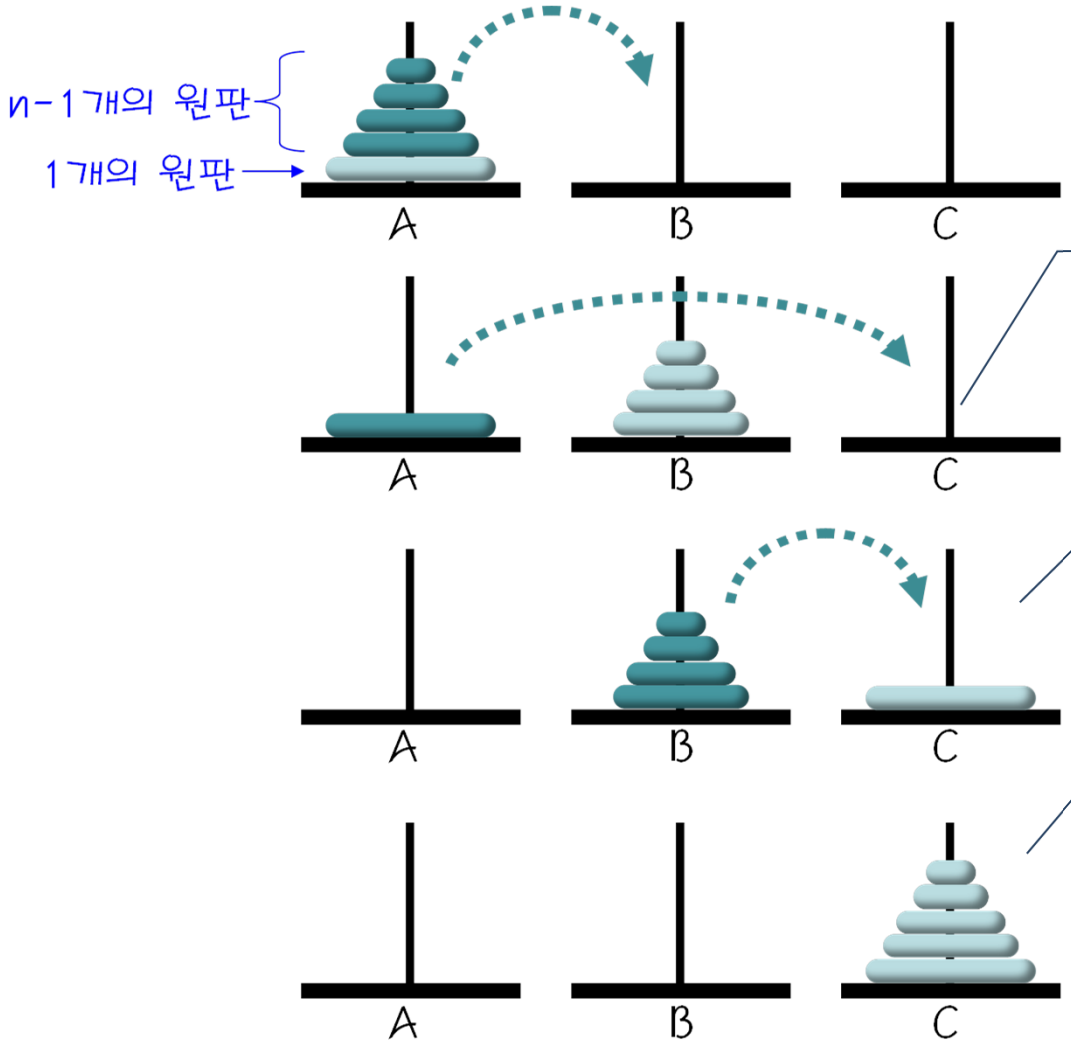
- 문제는 막대 A에 쌓여있는 원판 n개를 막대 C로 옮기는 것이다. 단 다음의 조건을 지켜야 한다.
  - 한 번에 하나의 원판만 이동할 수 있다
  - 맨 위에 있는 원판만 이동할 수 있다
  - 크기가 작은 원판위에 큰 원판이 쌓일 수 없다.
  - 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.



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.