

# 재귀

## 재귀란 ?

자기 자신을 호출하는 함수

## 재귀적 방법

자신의 복사본을 호출하여 더 작은 문제를 풀게함으로써 문제를 해결한다.

이를 재귀 단계라고 하는데, 재귀 단계는 더 많은 수의 재귀 단계를 만들 수 있다.

매 단계보다 함수는 원본 문제보다 조금 더 단순한 문제를 가지고 자기 자신을 호출한다.

작은 문제의 서열은 결국 기본 경우를 수렴해야 한다.

## 왜 재귀를 사용하는가?

재귀는 수학으로부터 빌려온 유용한 기법이다.

재귀 코드는 반복 코드보다 짧고 작성하기 쉽다.

일반적으로 루프는 컴파일되거나 인터프리터될 때 재귀 함수로 바뀐다.

재귀는 비슷한 하위 작업으로 정의될 수 있는 작업에 특히 유용하다.

예를 들어, 정렬, 검색, 그리고 탐색 문제들이 간단한 재귀 해법으로 해결된다.

## 재귀 함수의 형식

재귀 함수는 하위 작업을 수행하도록 자기 자신을 호출하여 작업을 수행한다.

어느 단계에 이르러서는, 자기 자신을 호출하지 않고도 수행할 수 있는 하위 작업을 수행한다.

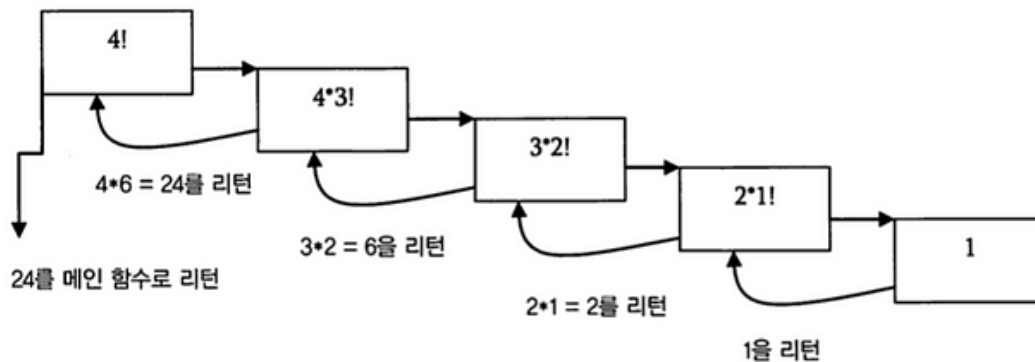
이렇게 함수가 재귀 호출하지 않는 것을 기본 경우라고 하고, 자기 자신을 호출해서 하위 작업을 수행하는 것을 재귀 경우라고 한다.

## 재귀와 메모리(시각화)

재귀 호출될 때마다 메서드의 복사본이 메모리에 만들어진다.

메서드가 종료할 때 리턴하는 메서드의 복사본은 메모리에서 삭제된다.

재귀 해법은 쉬워 보이지만, 시각화와 추적에는 시간이 좀 걸린다.



## 재귀와 반복 비교

### 재귀

- 기본 경우에 도달하면 종료한다.
- 각 재귀 호출은 스택 프레임(즉, 메모리)에 추가 공간을 필요로 한다.
- 무한 재귀에 들어가게 되면 메모리 용량을 초과해서 스택 오버플로우를 초래하게 된다.
- 어떤 문제들의 해답은 재귀적인 수식으로 만들기 쉽다.

### 반복

- 조건이 거짓일 때 종료한다.
- 각 반복이 추가 공간을 필요로 하지 않는다.
- 무한 루프는 추가 메모리가 필요하지 않으므로 무한히 반복된다.
- 반복적 해법은 재귀적 해법에 비해 간단하지 않을 때가 있다.

## 재귀에 대한 참고 사항

- 재귀적 알고리즘에는 기본 경우와 재귀적 경우 총 2가지 경우가 있다.
- 모든 재귀 함수는 기본 경우에 종료해야 한다.
- 일반적으로 반복 해법이 재귀 해법보다 효율적이다.(재귀 호출에 따른 추가적인 메모리 요구 때문)
- 재귀 알고리즘은 스택을 이용해서 재귀 호출 없이 구현될 수 있지만, 보통 문제를 더 일으키기 때문에 유용하지 못하다. 이 말은 **재귀적으로 풀 수 있는 문제는 반복적으로 풀**

수도 있다는 이야기이다.

- 어떤 문제들의 경우에 눈에 띄는 반복적 알고리즘이 없을 수도 있다.
- 어떤 문제는 재귀적 해법이 최적이고, 어떤 문제는 그렇지 않다.

## 재귀 알고리즘의 예

- 피보나치 수열, 팩토리얼 구하기
- 병합 정렬, 퀵 정렬
- 이진 검색
- 트리 탐색, 중위, 전위, 후위 등 여러 트리 문제들
- 그래프 탐색, 깊이 우선 탐색과 너비 우선 탐색
- 동적 계획법의 예
- 분할 정복 알고리즘
- 하노이의 탑
- 백트래킹 알고리즘