재귀 (Recursion)

위승빈

목차

재귀

- 재귀란?
- 재귀에 관해
- 팩토리얼 문제로 간단히 재귀 알아보기
- 재귀 함수의 내부적인 구현
- 재귀 알고리즘 구조
- 재귀와 반복
- 재귀 호출

재귀란?

어떤 알고리즘이나 함수가 자기 자신을 호출하여
문제를 해결하는 프로그래밍 기법

어떤 알고리즘이나 함수가 자기 자신을 호출하여
문제를 해결하는 프로그래밍 기법

어떤 알고리즘이나 함수가 자기 자신을 호출하여

재귀에 관해

함수가 본인 스스로를 호출해서 사용하는 것으로 **순환**, 되부름이라고 불리기도 한다.

재귀는 본질적으로 **재귀적으로 정의된 문제**나 그런 **자료 구조를 다루는 프로그램**에 적합하다.

예를 들어 거듭제곱, 피보나치, 하노이탑 문제등이 있다.

팩토리얼 문제로 간단히 재귀 알아보기

- 재귀적이지 않은 프로그래밍

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

- 재귀적인 프로그래밍

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

재귀 함수의 내부적인 구현

- 하나의 함수가 자기 자신을 다시 호출하는 것은 **다른 함수를 호출하는 것과 동일**하다.
- 복귀 주소가 시스템 스택에 저장되고 호출되는 함수를 위한 **매개 변수와 지역 변수를 스택으로부터 할당**받는다. (이런 함수를 위한 시스템 스택에서의 공간을 활성 레코드라 한다.)
- 호출된 함수가 끝나게 되면 시스템 스택에서 복귀 주소를 추출하여 호출한 함수로 되돌아가게 된다.
- **재귀 호출이 계속 중첩될수록 시스템 스택에 활성레코드들이 쌓이게 된다.**

재귀 알고리즘 구조

- 자기 자신을 **호출하는 부분**과 재귀 **호출을 멈추는 부분**으로 구성된다.
- 재귀 호출을 멈추는 부분이 없다면 시스템 스택을 다 사용할 때까지 순환적으로 호출되다가 결국 오류를 내면서 멈춘다.

재귀와 반복

- 반복은 for이나 while 등 반복 구조를 사용해 반복시키는 문장을 작성하는 것이며, 많은 경우에 간명하고 효율적으로 되풀이를 구현하는 방법이다.
- 반복을 사용하게 되면 지나치게 복잡해지는 문제들의 경우, 재귀가 좋은 해결책이 될 수 있다.
- 기본적으로 **재귀와 반복은 문제 해결 능력이 같고**, 대부분의 경우 서로를 바꾸어 쓸 수 있다.
    - 특히 꼬리 재귀의 경우 이를 반복 알고리즘으로 쉽게 바꾸어 쓸 수 있다.
    - 재귀는 어떤 문제에서는 **반복에 비해 알고리즘을 훨씬 명확하고 간결하게 나타낼 수 있다**는 장점이 있다.
- 일반적으로 **재귀는 반복에 비해 수행속도 면에서는 떨어진다.**
    - 이로인해 알고리즘을 설명할 때는 순환으로 하고 실제 프로그램에서는 반복 버전으로 코딩하는 경우도 있다.

재귀 호출

```
return n * factorial(n - 1);
```

꼬리 재귀 (tail recursion)

재귀 호출이 재귀 함수의 맨 끝에서 이루어지는 형태의 재귀이다.

쉽게 반복 형태로 변환이 가능하다.

```
return factorial(n - 1) * n;
```

머리 재귀 (head recursion)

재귀 호출이 앞에서 이루어 지는 머리 재귀의 경우나 여러 군데에서 자기 자신을 호출하는 경우(multi recursion)는 쉽게 반복적인 코드로 바꿀 수 없다.

동일한 알고리즘을 꼬리 재귀와 머리 재귀 양쪽 모두 표현할 수 있다면 당연히 꼬리 순환으로 작성해야 한다.

재귀적으로 문제의
크기를 줄여보자!