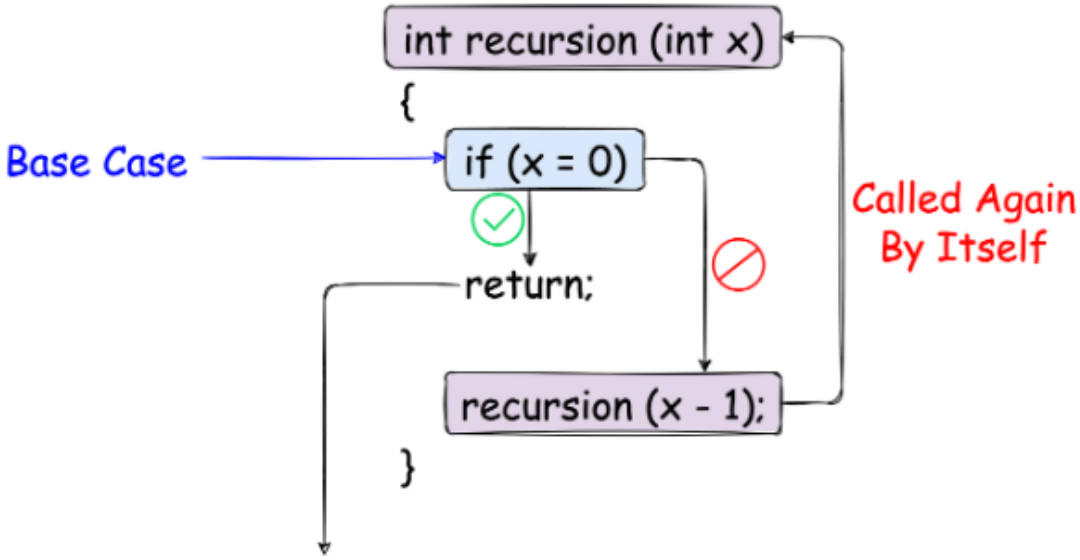
1주차 : 재귀

#재귀함수란?

재귀 (Recursion) 함수란 특정 함수 내에서 **자기 자신을 다시 호출하여 문제를 해결해나가는 함수**입니다. 문제를 해결하기 위해 원래 범위의 문제에서 더 작은 범위의 하위 문제를 먼저 해결함으로써 원래 문제를 해결해 나가는 방식입니다.



- 예제1

```
// 피보나치 수열
public static int fibonacci(int n) {
    if (n<2) {
        return n;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}

// 팩토리얼
public static int factorial(int n) {
    if (n==0) {
        return 1;
    } else {
```

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

(재귀 함수로 구현할 수 있는 대표적인 알고리즘으로는 피보나치수열, 팩토리얼, 거듭제곱, 최대 공약수 등이 있습니다.)

#재귀함수의 장점

- **첫 번째'**로 **가독성을 높일 수 있다는 것**입니다. 위에서 예로 든 피보나치 수열이나 팩토리얼 같은 경우 알고리즘을 기술한 그대로를 가지고 코드로 표현할 수 있기 때문에 for 문보다 더 직관적으로 코드를 이해할 수 있습니다.
- **'두 번째'**는 **변수의 사용을 줄여준다는 것**입니다. 변수의 사용을 줄여준다는 것은 변수가 저장되는 메모리에 대한 이야기가 아니라 'mutable state(변경 가능한 상태)'를 제거하여 프로그램 오류가 발생할 수 있는 가능성을 줄여준다는 이야기이며, 이는 변수의 수를 줄이는 것뿐만 아니라 변수가 가질 수 있는 값의 종류 또는 범위도 제한하게 되어 함수를 단순하게 만들고, 불변적으로 유지될 수 있도록 합니다.

#재귀함수의 단점

재귀함수를 잘못 사용하면 무한루프에 빠져 스택오버플로우를 발생시킬 수도 있고, 무한루프에 빠지지 않더라도 함수가 계속 호출되면서 함수의 매개변수, 지역변수, 리턴 값 함수 종료 후 돌아가는 위치가 스택 메모리에 저장되면서 stack이 쌓여 결국 마찬가지로 스택오버플로우를 발생시킬 수도 있습니다. 또한 스택 프레임을 구성하고 해제하는 과정에서 반복문보다 오버헤드가 들어 성능도 느려지게 됩니다.

#단점을 해결할 수 있는 방법이 있을까?

꼬리 재귀 최적화(TCO, tail call optimization)

꼬리 재귀 최적화 방식으로 구현된 재귀 함수는 위에서 언급된 재귀 함수의 스택오버플로우 문제를 해결할 수 있고, 반복문과 성능 차이도 발생시키지 않습니다.

- 예제2

```
// 꼬리 재귀 최적화가 되지 않은 재귀 함수
int recursive(int n)
{
 if(n==1) return 1;
 return n + recursive(n-1);
}

// 꼬리 재귀 최적화된 재귀 함수
int tailRecursive(int n, int acc)
{
 if(n==1) return acc;
 return tailRecursive(n-1, n + acc );
}
```

꼬리 재귀 최적화가 되지 않은 재귀 함수의 경우 return에서 **'n + 함수(n-1)'**이라는 연산이 필요한데, 이러한 연산으로 인해 함수가 호출될 때마다 호출 스택 메모리를 잡아먹게 되는 것입니다.

반면 꼬리 최적화된 재귀 함수를 보면 매개변수로 필요한 연산을 전달하기 때문에 return에서 따로 **'연산이 필요하지 않습니다.'**

- 꼬리 재귀 최적화 조건 2가지

1. 프로그래머가 재귀 함수를 꼬리 재귀 방식으로 구현해야 한다.
2. 컴파일러가 꼬리 재귀 최적화를 지원해야 한다.