Recursion

algorithm

기초

- 이해의 관점
  - Heap 영역의 새로운 주소값을 갖는 **복사본** 함수를 생성한다!
  - 가장 마지막에 호출된 함수가 가장 먼저 종료 된다! (**L**ast **I**n **F**irst **O**ut, Stack)
- 주의점!
  - 반드시 **탈출 조건**이 있어야 한다.(= 탈출조건이 없으면 계속 복사본 함수를 만들어내어 값을 반환(Return)하지 못한다.)

예시

```
#include <studio.h>

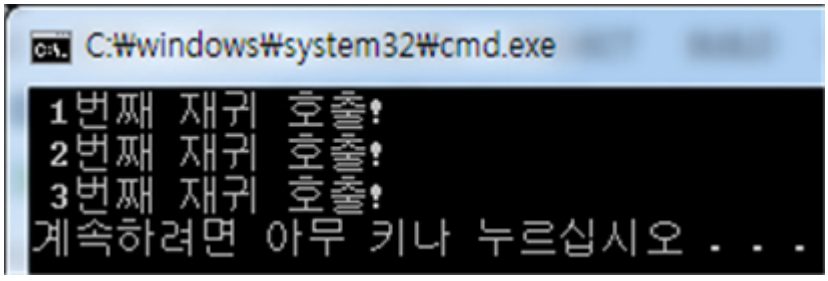
void Recursion(int num) {

  //탈출 조건
    if(num <= 0)
        return;

  //탈출 조건을 만들기 위한 인자(Argument)와 재귀 호출
    Recursion(num-1);

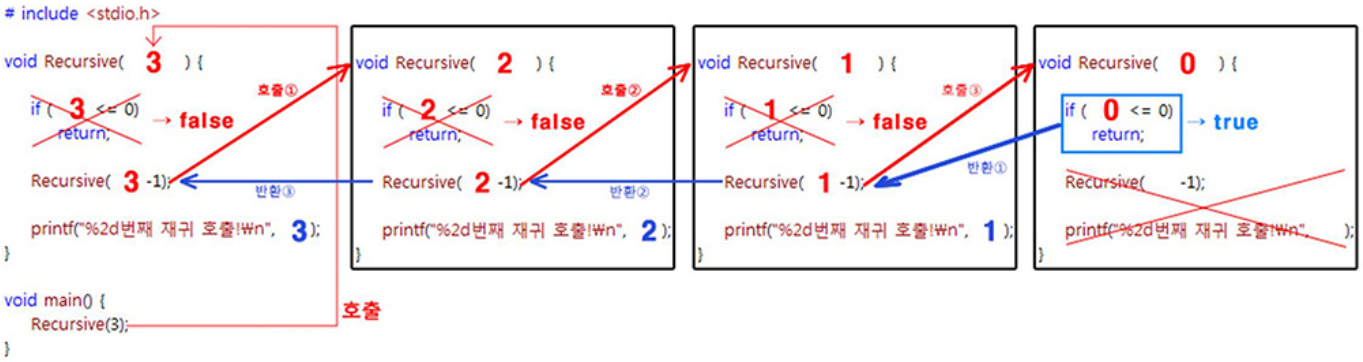
    printf("%2d번째 재귀 호출!\n", num);
}

void main() {
    Recursion(3);
}
```



해설

- → 연쇄적 호출
- → 연쇄적 반환



결론

- "연쇄적 호출 → 연쇄적 역반환"을 보이고 있다.
- 재귀함수 호출이 있으면 그 다음 코드는 그 시점에서 무시된다. (☞ printf()가 무시된 것에 주목!)
- "역반환"시에 출력되는 값이 자기 자신의 재귀함수에서 호출되는 매개변수(Parameter)값이다.
    - 반환시 복사본 함수를 호출하여 값을 읽어오기 때문

예제

합

```
public static int sum(int n) {
    if(n <= 0)
        return n;
    return n + sum(n - 1);
}
```

Factorial

```
private static int factorial(int n) {
    if(n == 0) //0! = 1
        return 1;
    else
        return n * factorial(n-1);
}
```

Fibonacci Array

```
public static int fibonacci(int n) {
    if(n < 1)
```

```
            throw new StackOverflowError("1보다 커야 합니다.");
        if(n == 1) {
            return 0;
        }
        else if (n == 2) {
            return 1;
        }
        else {
            return fibonacci(n - 1) + fibonacci(n - 2);
        }
    }
```

Hanoi Tower

```
    public static void hanoiTower(int n, String from, String to, String
temp) {
        if(n == 1) {
            System.out.println(String.format("원반 %d개 %s → %s", n, from,
temp));
        }
        else {
            hanoiTower(n - 1, from, temp, to);
            System.out.println(String.format("원반 %d개 %s → %s", n, from,
temp));
            hanoiTower(n - 1, temp, from, to);
        }
    }
```

From:
https://ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
https://ledyx.xyz/doku.php?id=algorithm:recursion

Last update: **2022/06/14 19:05**