

재귀함수 보충

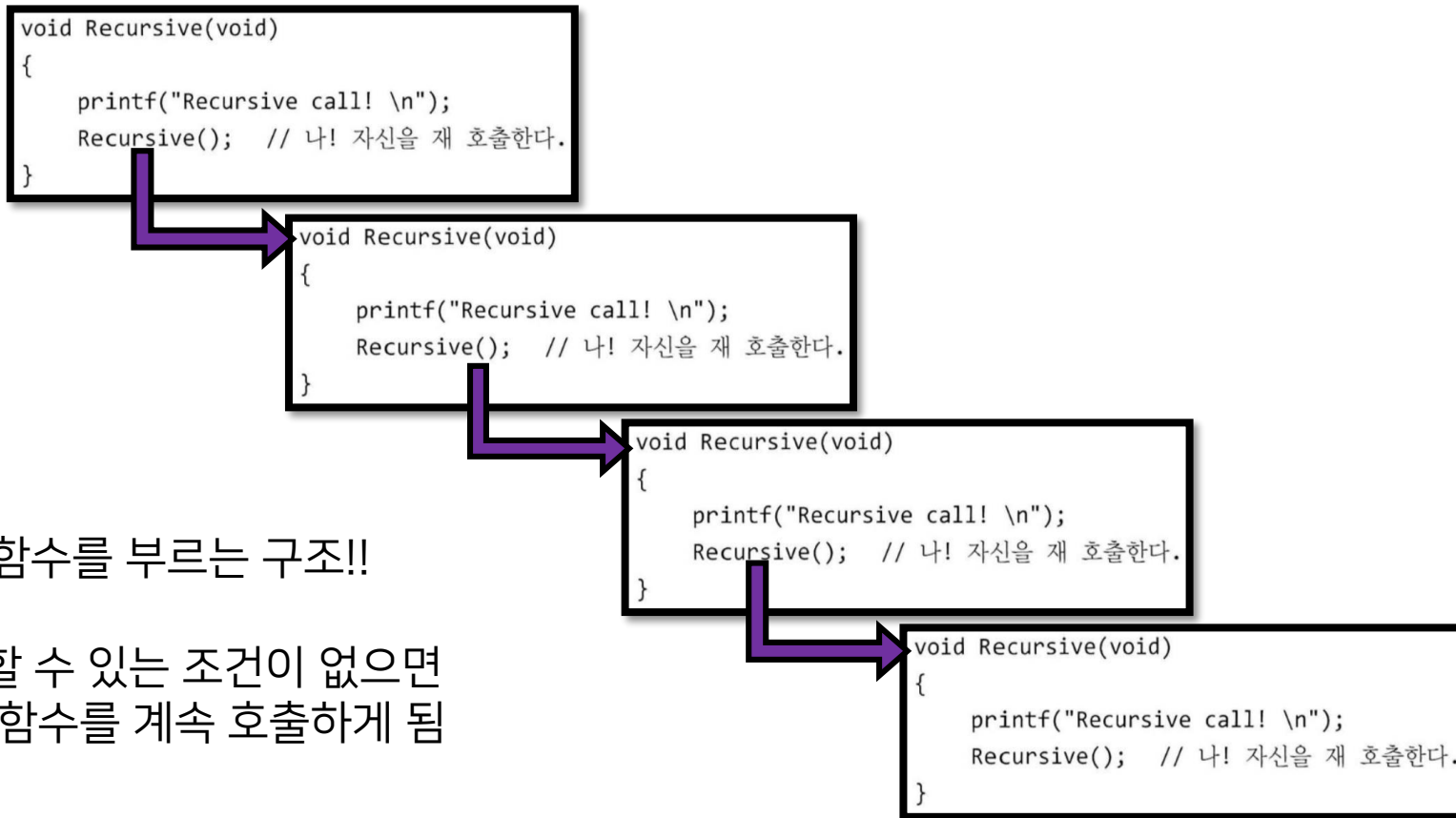
20200613

재귀함수의 기본적인 이해

```
void Recursive(void)
{
    printf("Recursive call! \n");
    Recursive(); // 나! 자신을 재 호출한다.
}
```

재귀함수란 다음과 같이 함수 내에서
자기 자신을 다시 호출하는 함수를 말함

재귀함수의 기본적인 이해



함수가 함수를 부르는 구조!!

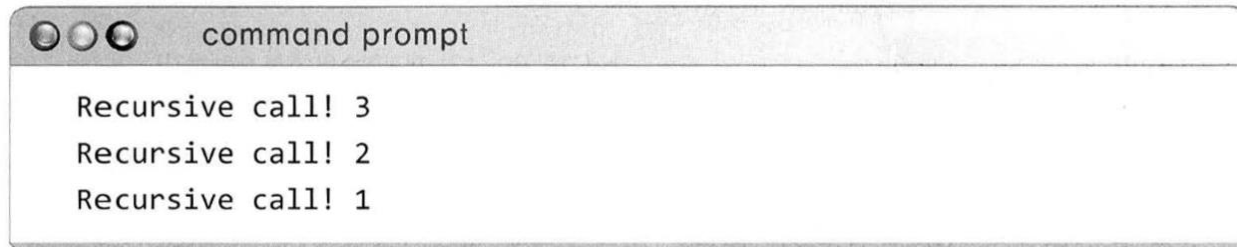
단, 멈추게 할 수 있는 조건이 없으면
무한정으로 함수를 계속 호출하게 됨

재귀함수와 return의 개념

❖ RecursiveFunc.c

```
1. #include <stdio.h>
2.
3. void Recursive(int num)
4. {
5.     if(num<=0)    // 재귀의 탈출조건
6.         return; //재귀의 탈출!
7.     printf("Recursive call! %d \n", num);
8.     Recursive(num-1);
9. }
10.
11. int main(void)
12. {
13.     Recursive(3);
14.     return 0;
15. }
```

❖ 실행결과: RecursiveFunc.c



```
command prompt

Recursive call! 3
Recursive call! 2
Recursive call! 1
```

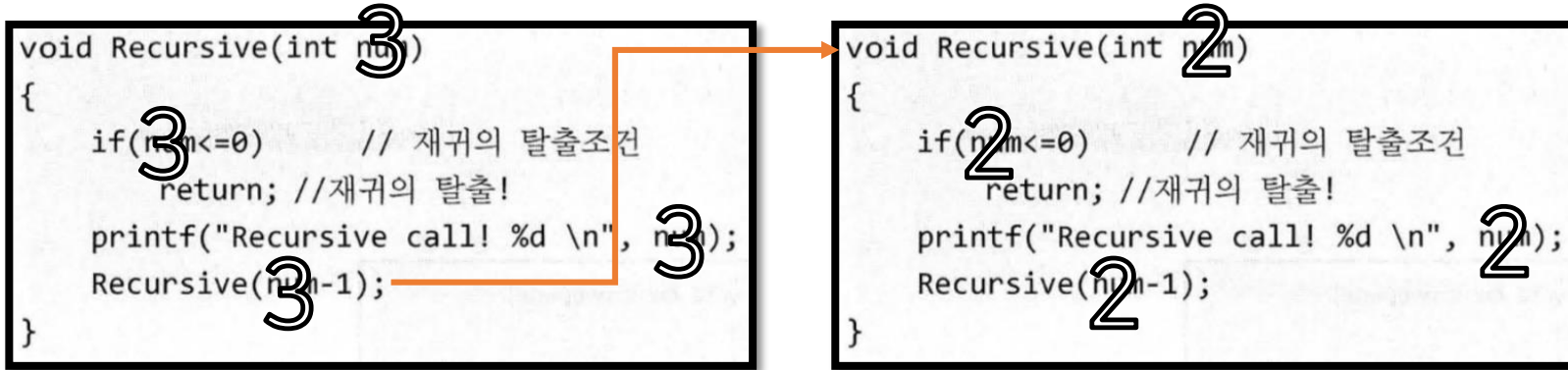
재귀함수와 return의 개념 - 함수 호출 단계

```
void Recursive(int n)
{
    if(n<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", n);
    Recursive(n-1);
}
```

실행창

Recursive call! 3

재귀함수와 return의 개념 - 함수 호출 단계

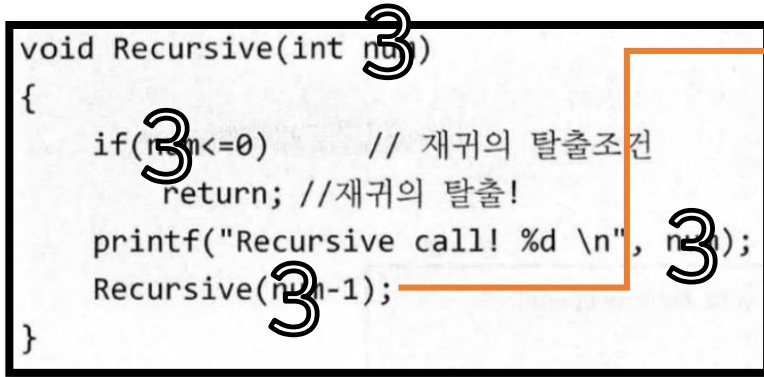


실행창

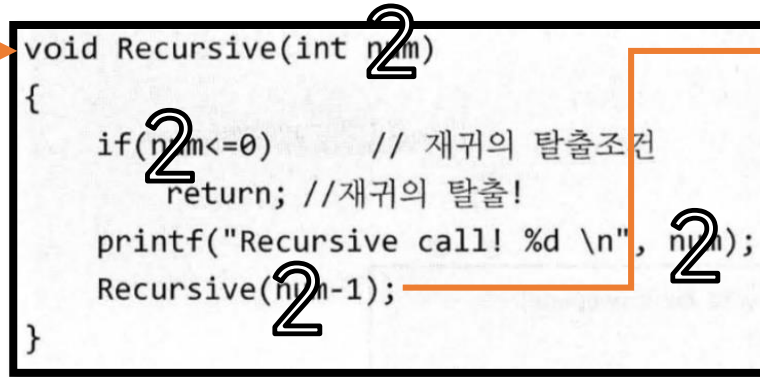
Recursive call! 3
Recursive call! 2

재귀함수와 return의 개념 - 함수 호출 단계

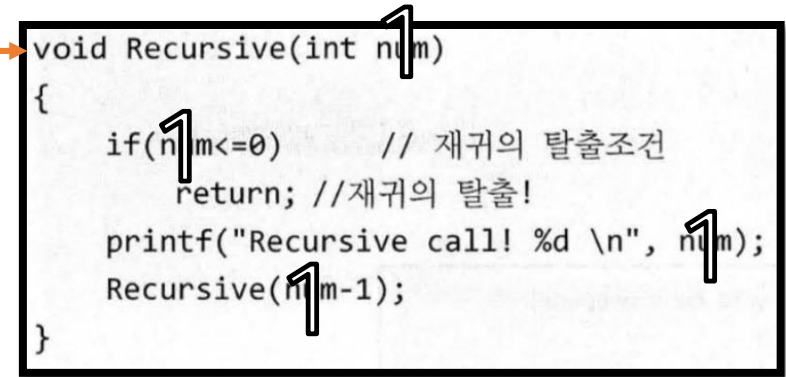
```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```



```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```



```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```



실행창

```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```

재귀함수와 return의 개념 - 함수 호출 단계

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

실행창

Recursive call! 3
Recursive call! 2
Recursive call! 1

★ 재귀함수에서 return은 중요하다 ★

```
int Add(int a, int b) {
```

```
    int y = a + b; Add 함수에서는 y를 return하고 함수 종료  
    return y;
```

```
}
```

2

함수가 끝났기 때문에,
Add 함수를 부른 함수(메인함수)로 다시 돌아감

```
int main(void) {
```

```
    int res; 메인함수가 Add 함수를 호출함
```

```
    res = Add(3,4);
```

```
    printf("Add 함수를 통한 덧셈의 결과: %d", res);
```

```
    return 0;
```

```
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

중요 ->

이것도 Recursive(1)가 Recursive(0)를 호출하고
return; 이라는 문장을 이용해서 함수를 종료했기 때문에
Recursive(0)을 부른 Recursive(1)로 다시 돌아감!!

재귀함수와 return의 개념 - 리턴 단계

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

반환 (하지만 반환 값이 없으므로
그냥 반환되면서 종료)

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

실행창

```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```

재귀함수와 return의 개념 - 리턴 단계

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

```
void Recursive(int num)
{
    if(num<=0) // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}
```

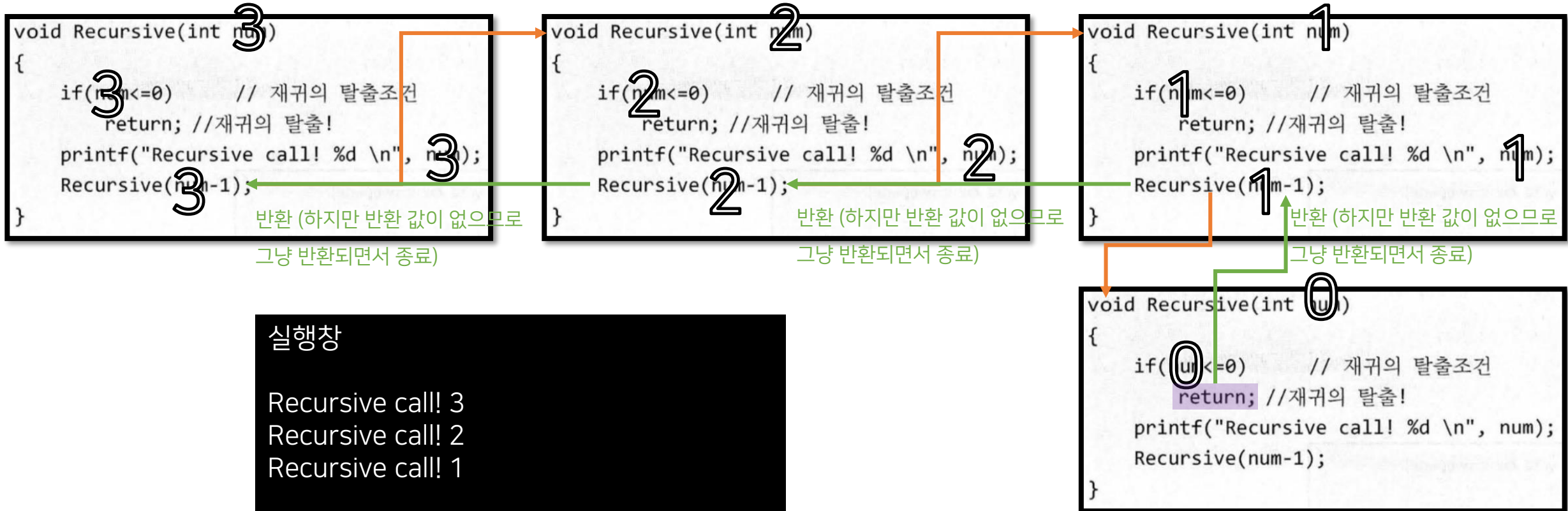
반환 (하지만 반환 값이 없으므로
그냥 반환되면서 종료)

반환 (하지만 반환 값이 없으므로
그냥 반환되면서 종료)

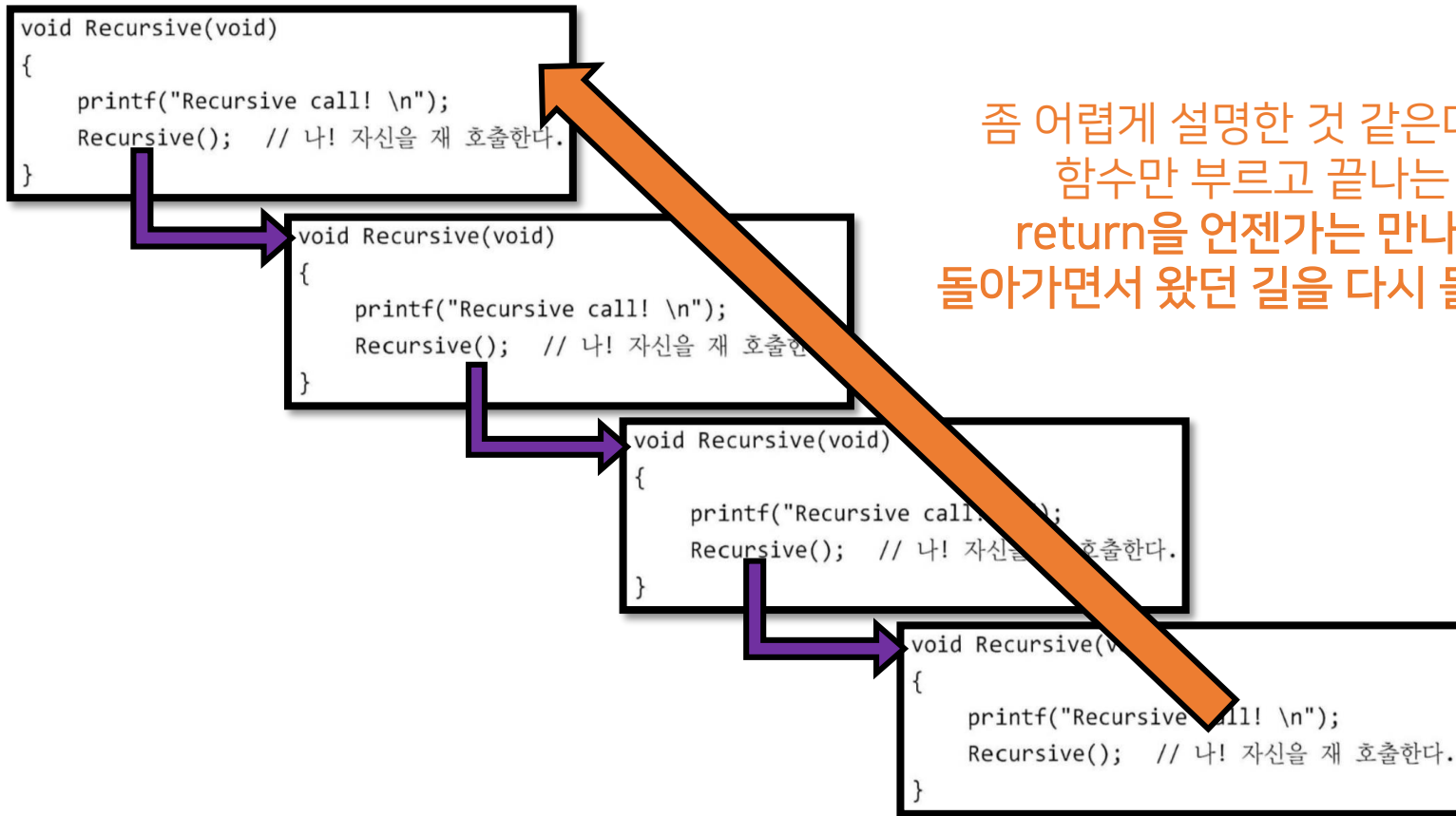
실행창

Recursive call! 3
Recursive call! 2
Recursive call! 1

재귀함수와 return의 개념 - 리턴 단계

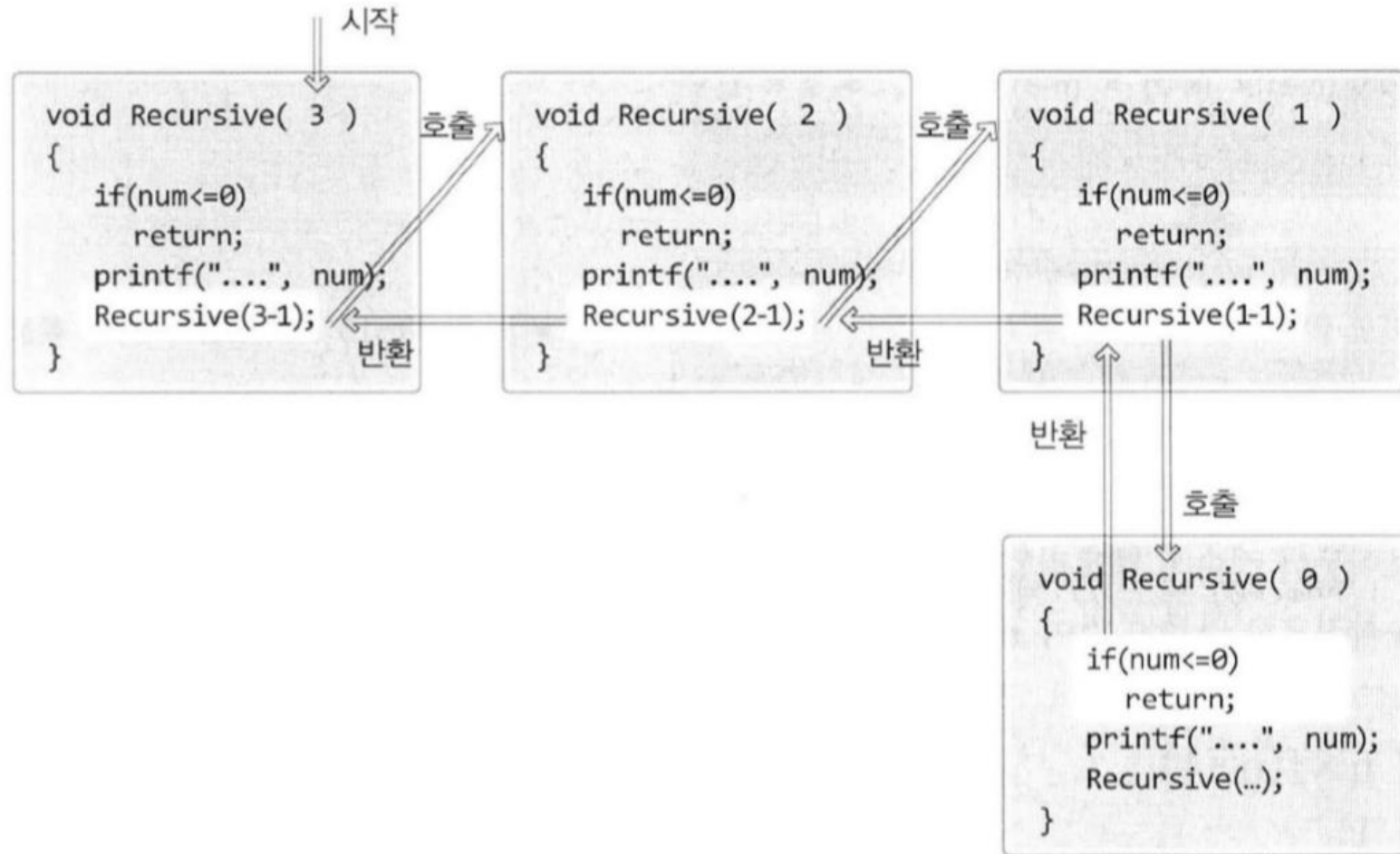


재귀함수에서 return은 중요하다



좀 어렵게 설명한 것 같은데, 핵심만 말하면
함수만 부르고 끝나는 것이 아니라,
return을 언젠가는 만나서 거꾸로 다시
돌아가면서 왔던 길을 다시 돌아가게 됨 (=반환)

재귀함수에서 return은 중요하다



▶ [그림 09-14: 탈출조건을 구성한 Recursive 함수의 호출과정]

재귀함수 중요 문제유형 아이디어

1. 팩토리얼 구하는 문제 (난이도 ★★)

❖ RecursiveFactorial.c

```
1. #include <stdio.h>
2.
3. int Factorial(int n)
4. {
5.     if(n==0)
6.         return 1;
7.     else
8.         return n * Factorial(n-1);
9. }
10.
11. int main(void)
12. {
13.     printf("1! = %d \n", Factorial(1));
14.     printf("2! = %d \n", Factorial(2));
15.     printf("3! = %d \n", Factorial(3));
16.     printf("4! = %d \n", Factorial(4));
17.     printf("9! = %d \n", Factorial(9));
18.     return 0;
19. }
```

<아이디어>

0까지 갔을 때 어떻게 처리하느냐가 중요
n이 0일 때의 처리를 신경 써서 풀면 쉬움

재귀함수 중요 문제유형 아이디어

2. 피보나치 수열 구하는 문제 (난이도 ★★★★★)

- 공학설계 파이썬 때 했던 거랑 비슷해

인자로 전달된 수만큼의 피보나치 수열을 출력하는 함수를 정의해보자. 예를 들어서 프로그램 사용자가 5를 입력하면 0에서부터 시작해서 총 5개의 피보나치 수열을 출력해야 한다. 참고로 피보나치 수열은 다음과 같다.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

이렇듯 피보나치 수열은 0과 1에서 시작한다. 그리고 세 번째 이후의 수열부터는 이전의 두 값의 합으로 구성된다. 즉, 세 번째 수는 0과 1의 합으로 이뤄져서 1이 되고, 네 번째 수는 1과 1의 합으로 이뤄져서 2가 된다.

<아이디어>

첫 번째 항과 두 번째 항에 대한 처리를 신경 써야함

왜냐면 시작할 때 0, 1은 주어져 있고

사실상 구하는 것은 첫 번째 항과 두 번째 항을 더한 세 번째 항 부터 시작임

재귀함수 중요 문제유형 아이디어

결론

재귀함수는 함수를 부르는 것보다

반환할 때 어떤 방식으로 반환되는지 아는게 더 중요함~!!

Summary

1. 재귀 함수는 자기 자신을 부르는 함수
2. 결국에는 더 이상 함수 호출을 멈출 수 있는 return 문이 필요함
3. 어떤 순서로 return이 되는지 항상 생각해보기

최대한 이해하기 쉽게 그림 넣느라고 자료가 길어졌는데

혹시 이해 안되면 반드시 질문하기!!