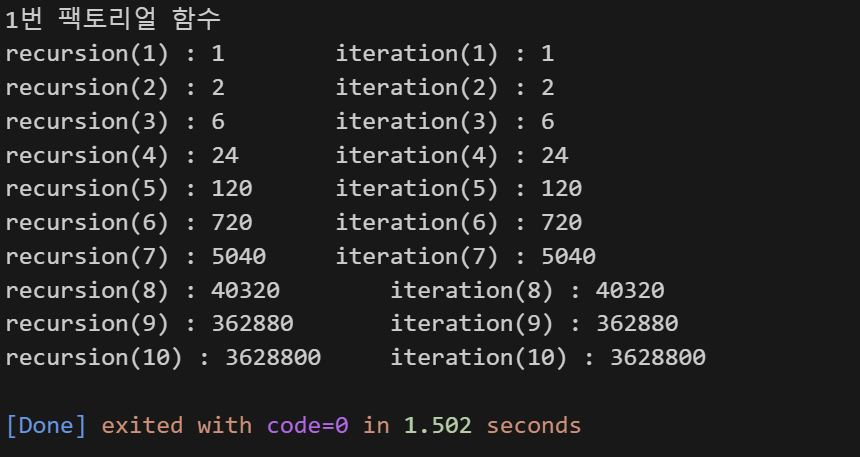
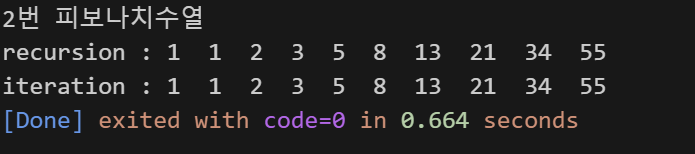
**<1 ~ 5번 실행 화면>**

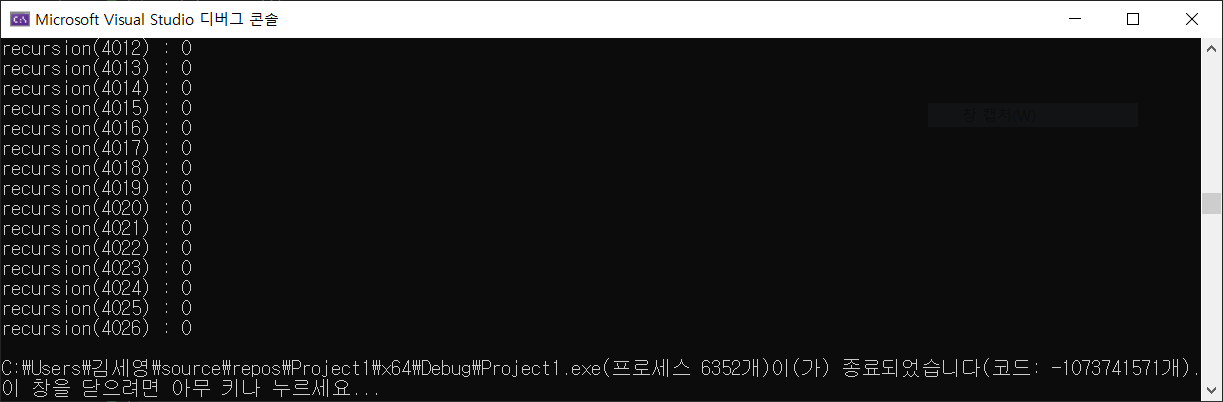
**1번**



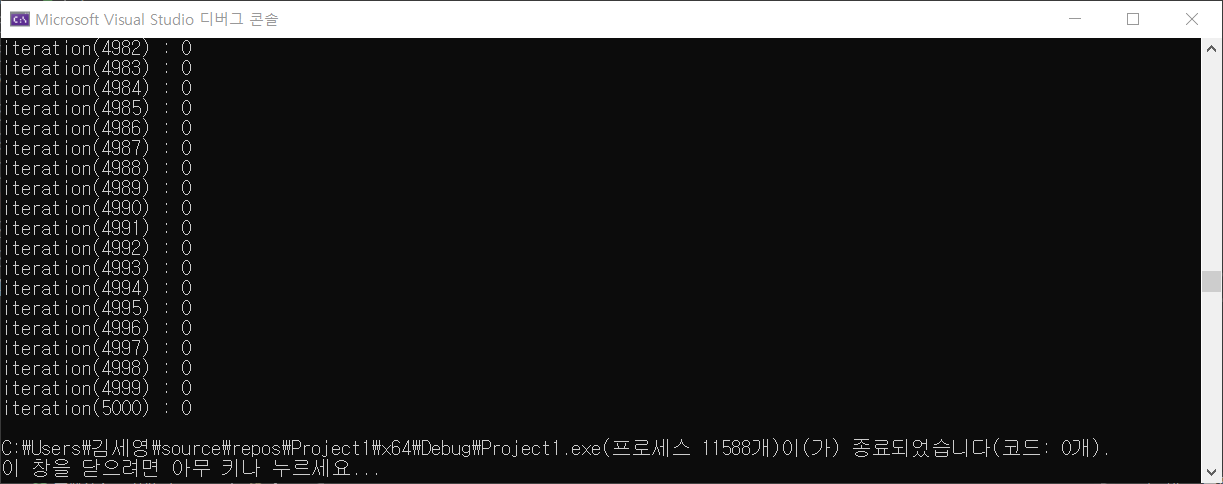
**2번**



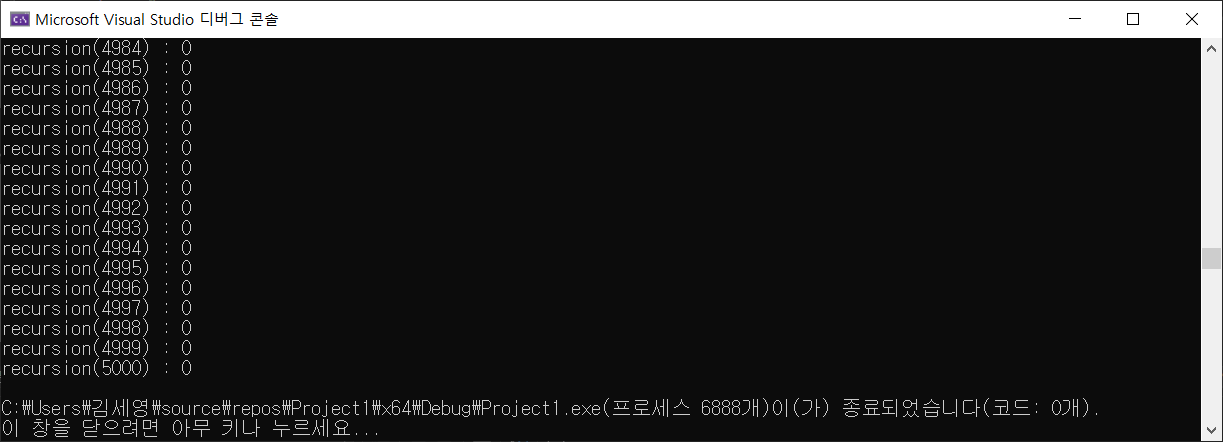
**3번**



**4번**



**5번**

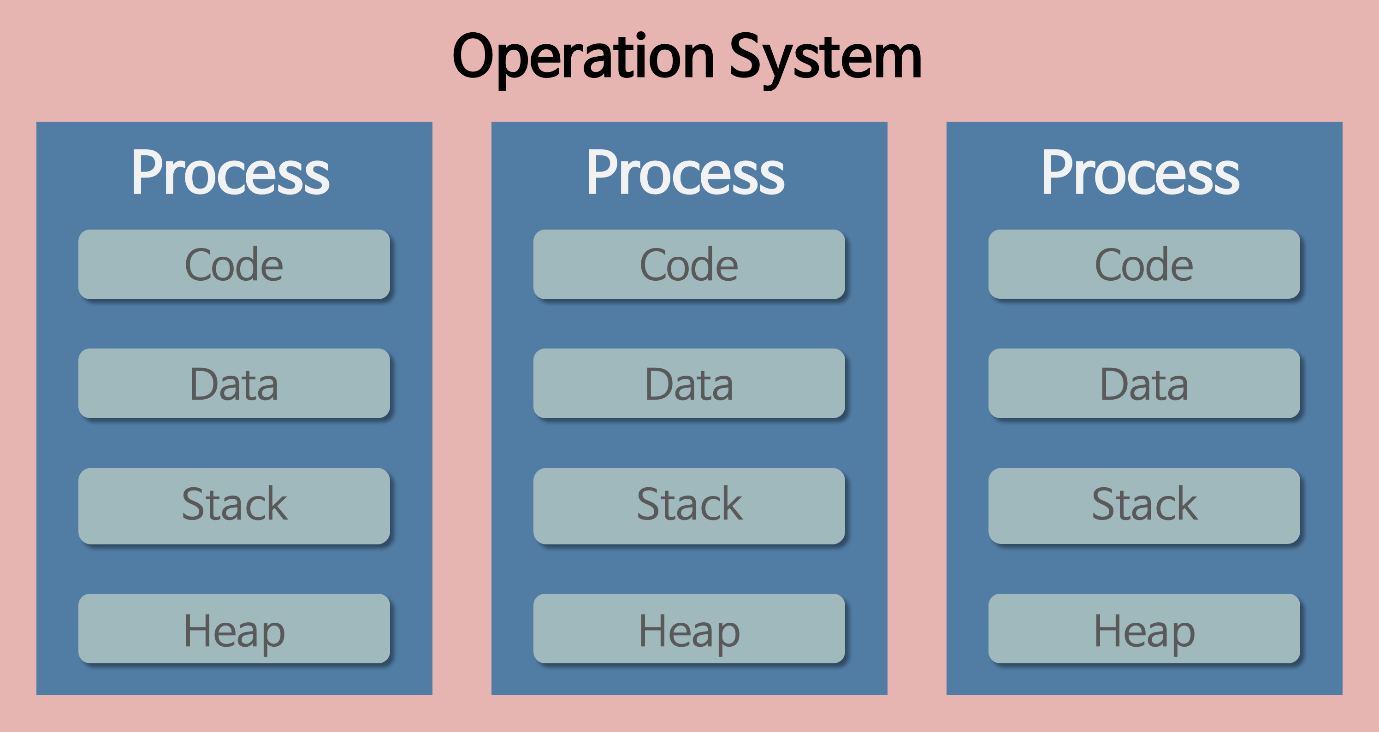


**<프로그램이 프로세스로서 실행될 때,**

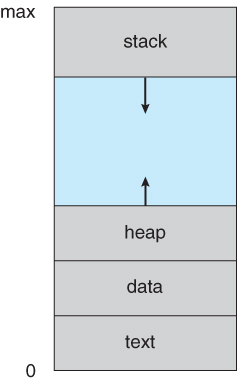
**구성되는 메모리 공간>**

# 프로세스

실행되고 있는 프로그램으로, **프로세스는 각 메모리 공간을 시스템으로부터 할당** 받습니다.



# 구성

  
메모리는 **code(text), data, heap, stack** 영역으로 구성되어 있습니다.

### **Code 영역**

**실행할 프로그램의 코드**가 저장됩니다. CPU는 이 영역에서 명령어를 하나씩 가져와 처리하게 됩니다.

### **Data 영역**

**전역변수와 정적변수(static)**가 저장됩니다. 이 변수들은 프로그램이 시작될 때 할당되어 프로그램 종료 시 소멸됩니다.

### **Stack 영역**

지역변수, 매개변수, 리턴값 등 **잠시 사용되었다가 사라지는 데이터**를 저장하는 영역입니다. 함수 호출 시 할당되고 함수 반환 시 소멸됩니다. 로드 시(컴파일 타임) 크기가 결정됩니다.

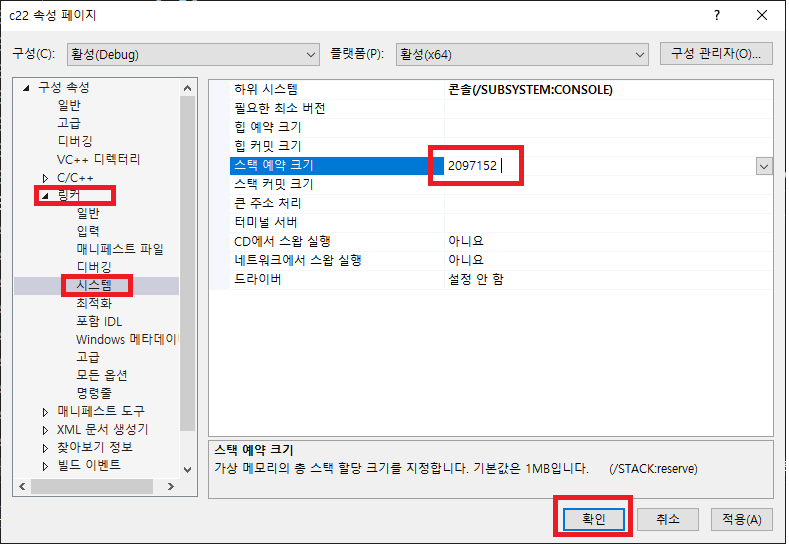
위 세 영역은 컴파일 할 때 data, stack 영역의 크기를 계산해 메모리 영역을 결정합니다!

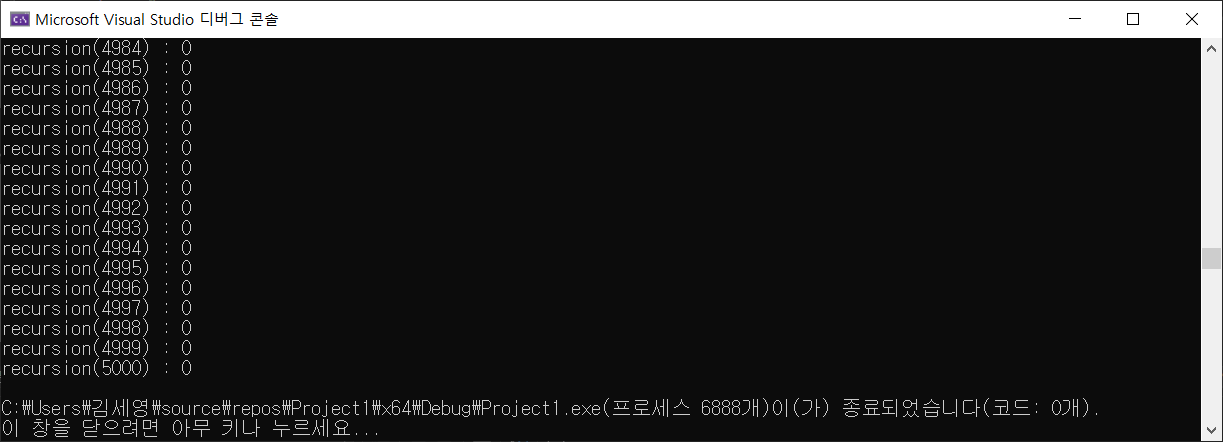
### **Heap 영역**

**동적 데이터 영역**입니다. 메모리 주소 값에 의해서만 참조되고 사용되는 영역입니다. 따라서, 프로그램 동작 시(런타임)에 크기가 결정됩니다.

예를 들어, stack에서 pointer 변수를 할당하면 pointer가 가리키는 heap 영역의 임의의 공간부터 원하는 크기만큼 할당해 사용하게 됩니다.

* **Visual Studio 🡪 Stack 영역 크기 조정하기**





* **Visual Studio의 컴파일 옵션인 ‘디버그모드’와 ‘릴리즈모드’의 차이점**

### Release, Debug 차이점1**:** 프로젝트 속성의 최적화로 들어가 보시면 Release는 위와 같이 최대 최적화로 되어 있는 반면 Debug는 사용 안 함으로 되어있는 것을 확인하실 수 있습니다. 여기서 **코드 최적화**란 무엇일까요? 코드 최적화는 고급 프로그래밍 언어(C언어나 C++)를 컴파일하는 과정에서 컴파일러가 판단하여 더 효율적인 코드로 바꿀 수 있는 부분을 판단하여 바꿔주는데 여기서 말하는 최적화는 바로 이것을 뜻합니다. ( Release에서는 코드 최적화를 하고 Debug에서는 하지 않는다.)

**디버그 vs 릴리즈**

### Debug모드에서는 컴파일 시 코드의 안정성을 위해 여러 가지 장치를 추가한다.

### 생성되는 파일의 크기가 다르다.

### 사용되는 런타임 라이브러리가 다르다.

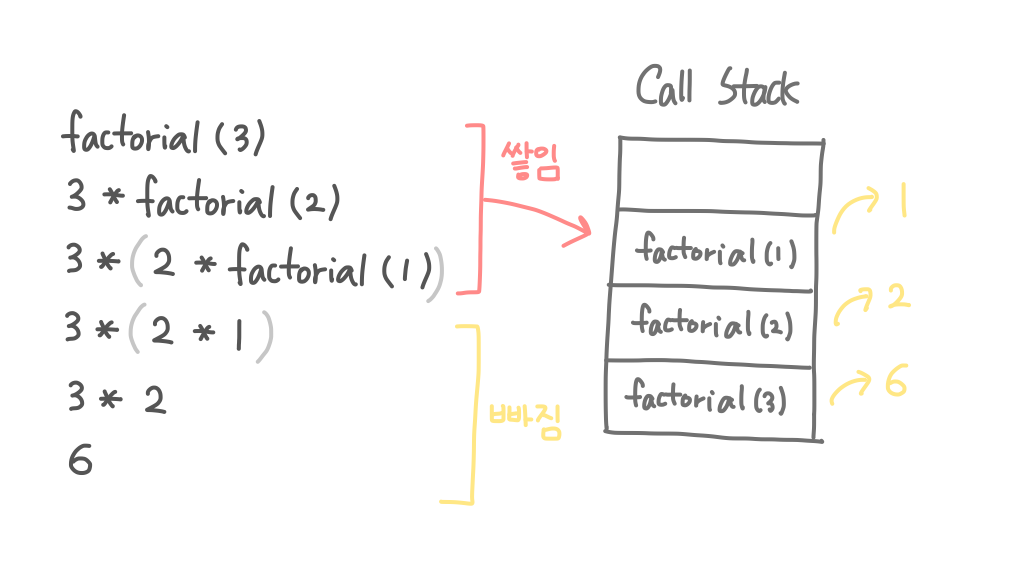
* 어셈블리어 상황에서 릴리즈에 비해 디버그는 코드라인이 길다는 점을 확인할 수 있다.

**<Tail recursion(꼬리 재귀)>**

* What is Tail recursion?

: 먼저, 재귀함수의 특징을 생각해보자.(= 왜 Tail recursion이 등장했는가?)

🡪 재귀 함수는 호출 할 때 마다 **Stack공간**에 **잠시 사용되었다가 사라지는 데이터(**지역변수, 매개변수, 리턴값,  **리턴 후 돌아갈 위치** **)**를 저장하기 때문에, 무리하게 호출하면 '**스택 오버플로우**'가 발생할 수 있다!



<재귀 사용의 문제, 단점 요약>

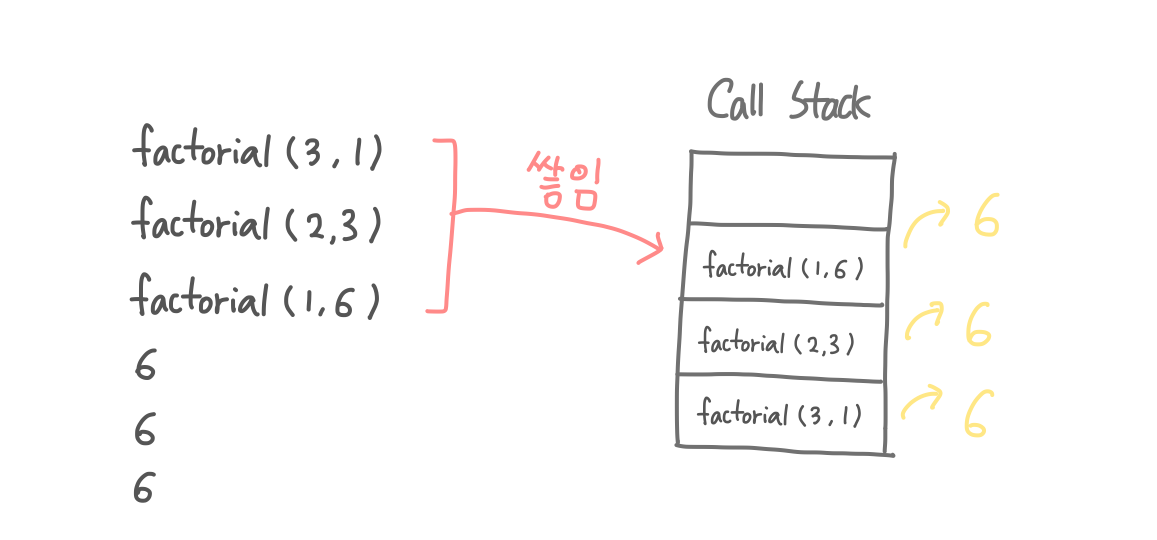
1. 재귀 호출로 구하려니 함수 호출 횟수가 너무 커서 문제다.
2. Stack 깊이가 너무 깊어져도 문제다.

**Tail recursion** = **원래 자리에서 해야할 일이 남아있지 않다면 돌아갈 원래 자리를 Stack에 추가로 저장할 필요가 없다.** 원래 자리에서 해야할 일을 남겨두지 않는 방법이다.

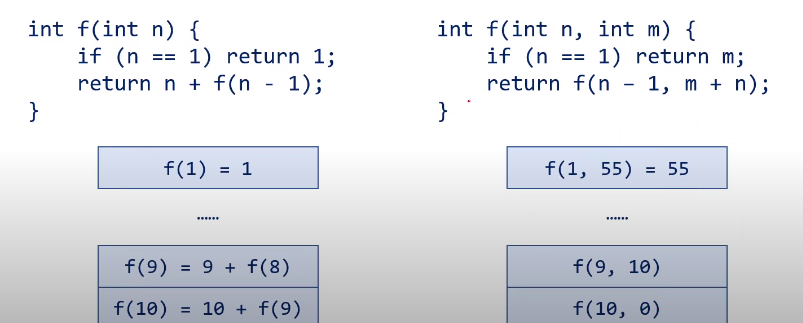
( **아무것도 하지 않고 return만 한다!** )

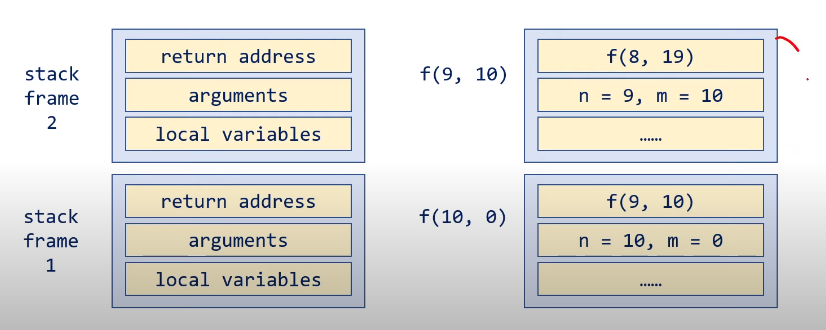
: 재귀 호출을 함수의 마지막 return문으로만 사용할 경우

: 호출 스택을 추가로 사용할 필요 없이 스택의 현재 영역을 재사용할 수 있다.



EX) sum함수 (1 ~ n까지 자연수의 합)





이러한 과정은 컴파일러가 최적화를 진행함 / 자바스크립트 같은 경우 인터프리터가 진행

컴파일할 때, 최적화 옵션을 주는 방법을 통해 실행한다.

<팩토리얼 소스 코드 : C++에서는 디폴트매개변수(int total = 1)로 사용>

#include <stdio.h>

int factorial(int n, int total)

{

if (n == 1)

return total;

else

return factorial(n - 1, n \* total);

}

int main()

{

// tail recursion

printf("tail recursion: %d\n", factorial(10, 1));

return 0;

}