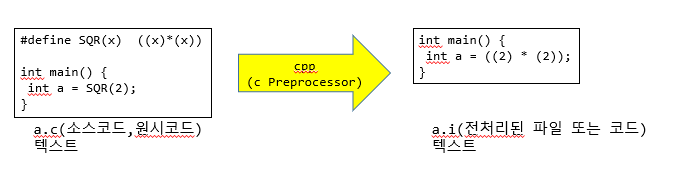
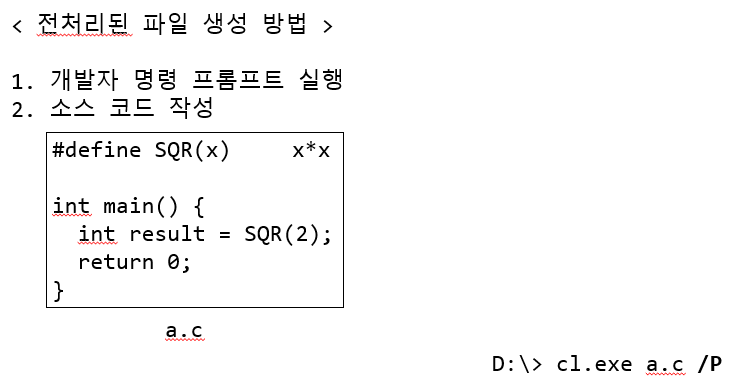
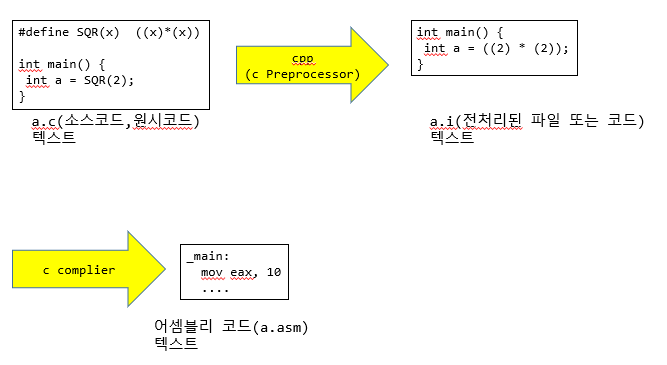
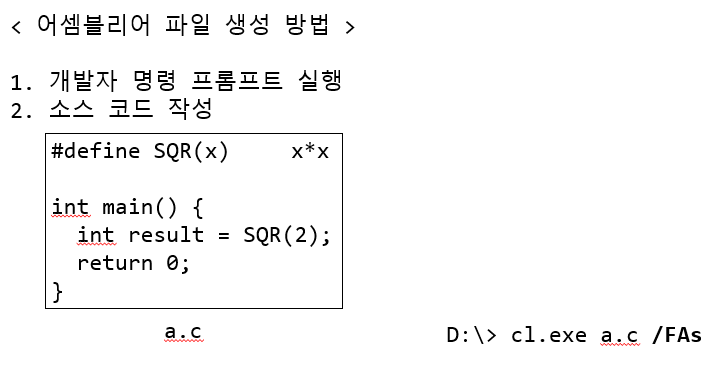
Day0519

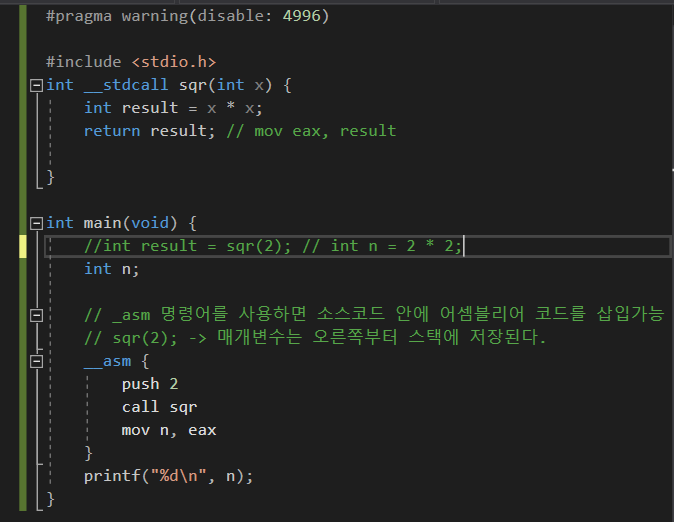
<전처리기>







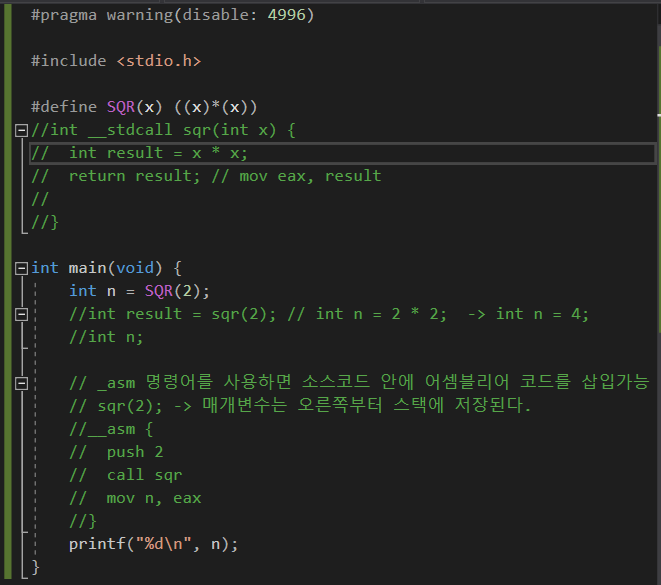


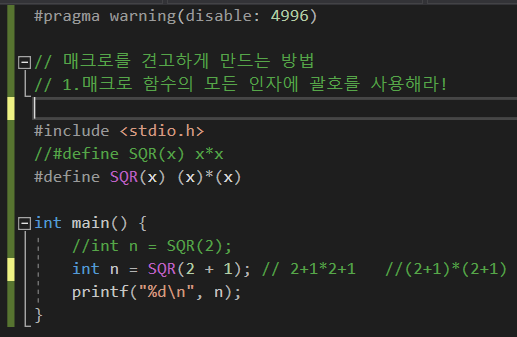


이렇게 간단한 것 조차 함수를 쓰게 되면.

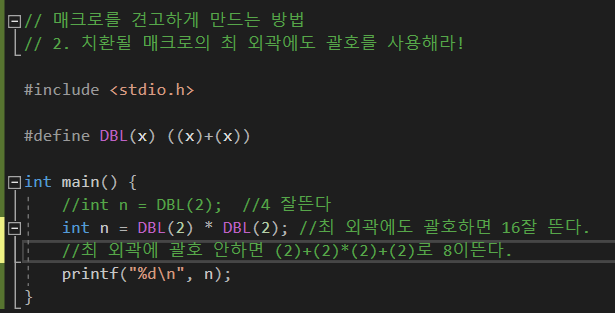
이 코드가 어셈블리로 바뀌면서 함수는 어셈블리어로 코드의 줄이 길어진다.. 비효율적

그래서 메크로를 쓰게 되면 더 효율적이다.

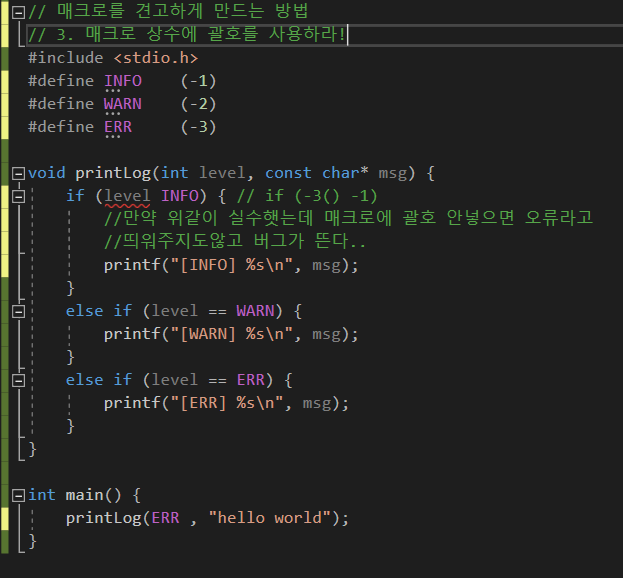




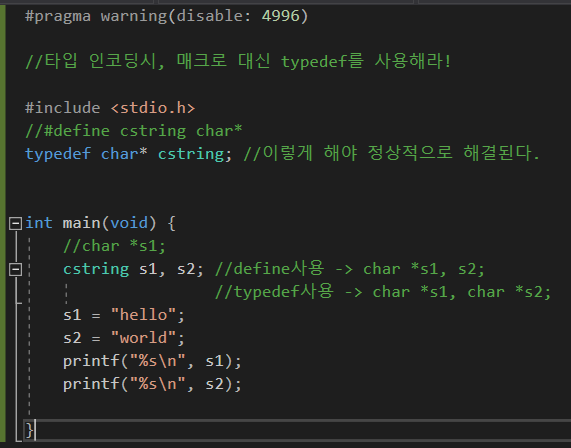
매크로 견고하게 만드는 방법1



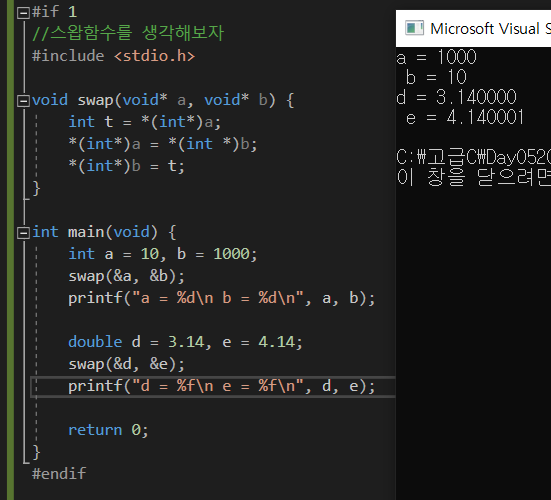
매크로 견고하게 만드는 방법2



매크로 견고하게 만드는 방법3

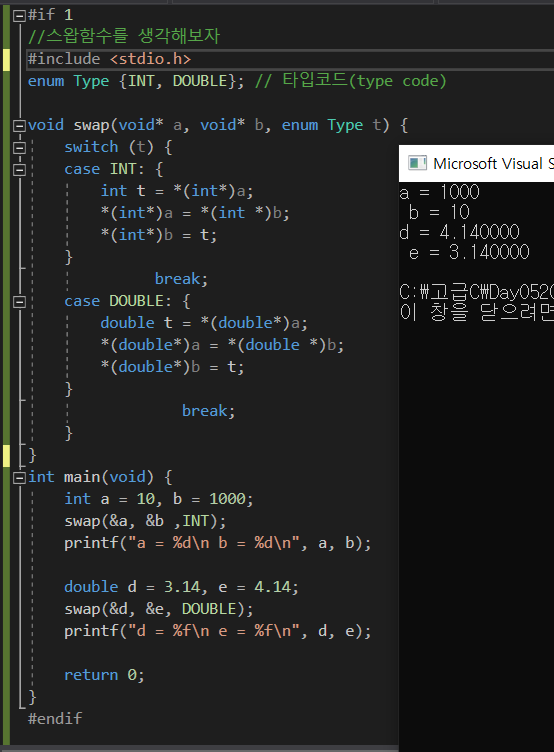


Day0520



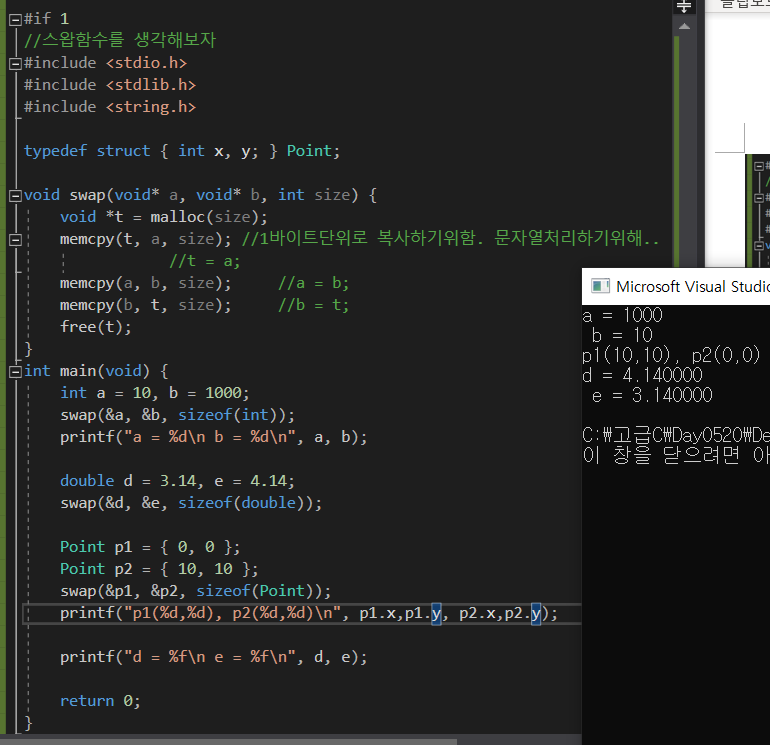
Step1. 위 코드는 정수타입만 스왑 가능하다는 문제가 있다.

모든 타입에 대해 스왑가능하도록 타입 정보를 사용자로부터 받는다.(아래코드)



Step2. 위 코드는 타입 코드가 사용되었으므로 새로운 타입이 추가될때마다 함수를 뜯어고쳐야하므로 유지보수가 어렵다. 타입 코드는 분기문을 수반하기 때문이다.

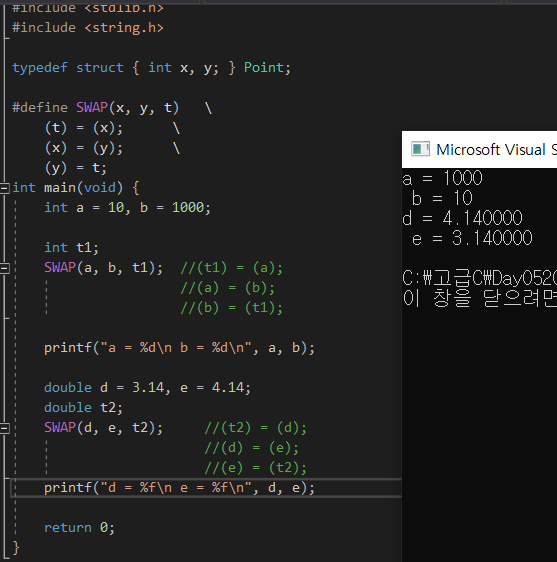
그래서 아래코드를 보면



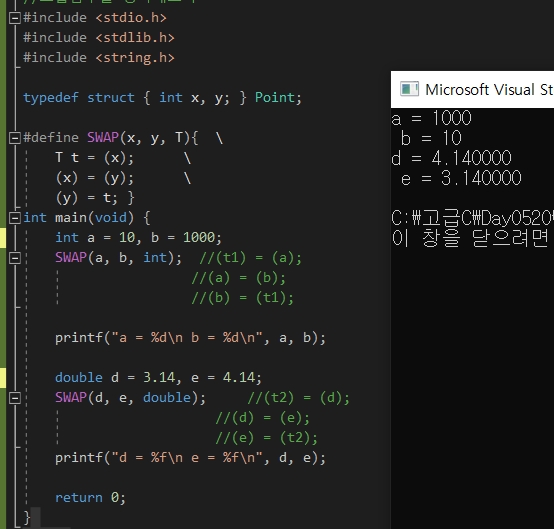
Step3. 위 코드는 내부적으로 동적할당을 하기 때문에 성능상의 이슈가 발생한다..

이를 해결하기 위해 매크로를 도입한다.

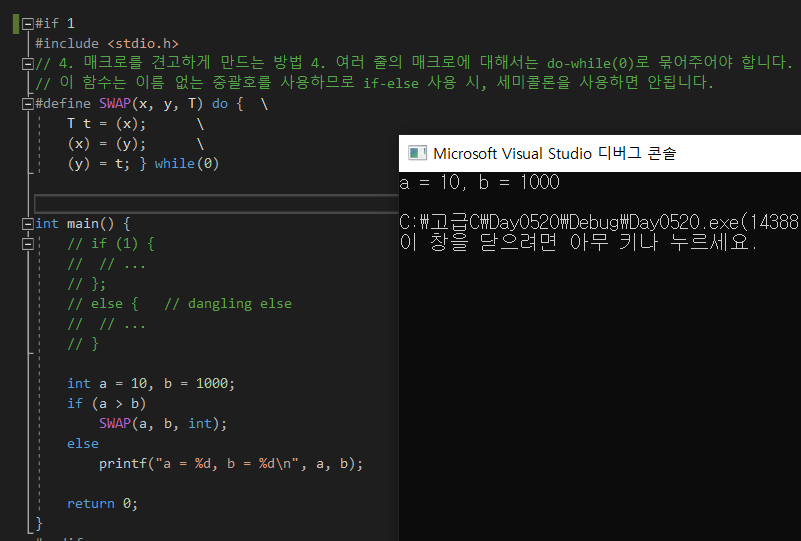
Cf. 구조체 복사 -> memcpy



Step4. 이전코드는 임시 변수를 사용한다는 단점이 있다. 그래서 이름없는 중괄호를 도입하여 단점을 극복하자.



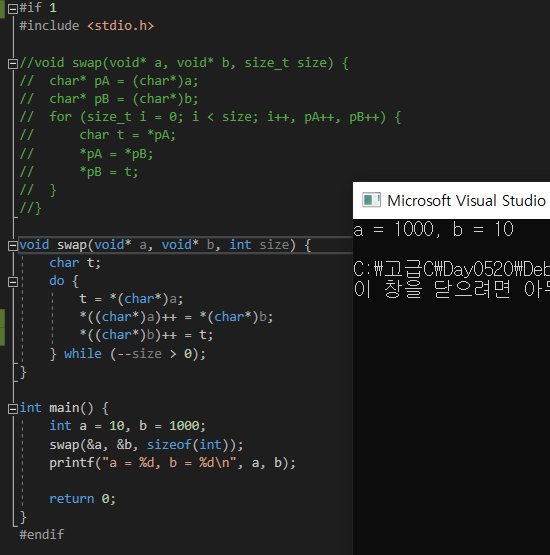
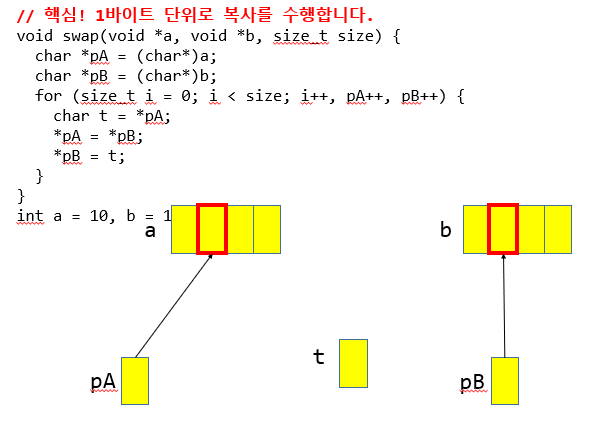
Step5. 함수호출이나 메모리 할당에 대한 오버헤드 없으면서 완벽히 스왑된다.



매크로를 견고하게 만드는방법 4 : 여러줄의 매크로에 대해서는 do-while(0)로 묶어주어야한다.

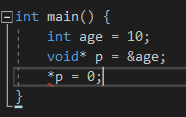
Do-while안쓰면 아래에 if else 에서 if(){}; else 가 되므로 else만 대롱대롱 남게된다..(dangling else) 오류!!

연습문제: 매크로를 사용하지 않고 일반함수를 사용하여 완벽하게 일반화된 스왑함수를 구현

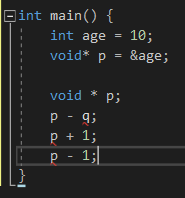


Cf. void \* : 순수하게 주소만 담을수 있는 포인터

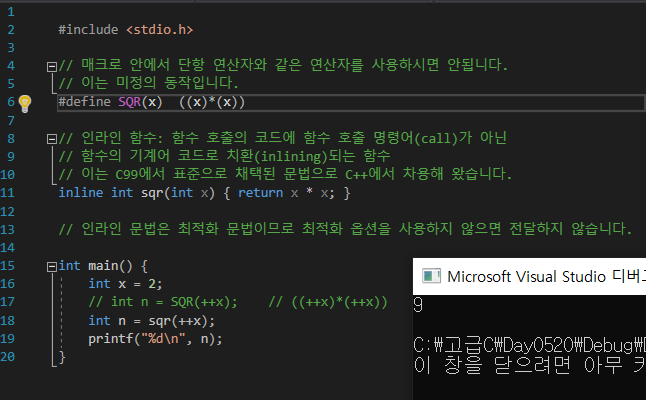
대상체의 정보가 없기 때문에 역참조, 제한적인 산술연산이 불가능



불가능!!



다 불가능!!



메크로함수에 단항연산자 같은 연산자를 사용하지마라! 어떤 프로세스에선 16 어떤 프로세스에선 12 가 뜰수있다.. (미정의 동작이다)

Vs 함수를 써서 간단히(대신 함수를 실행하면 비효율적..)

->해결책: inline함수

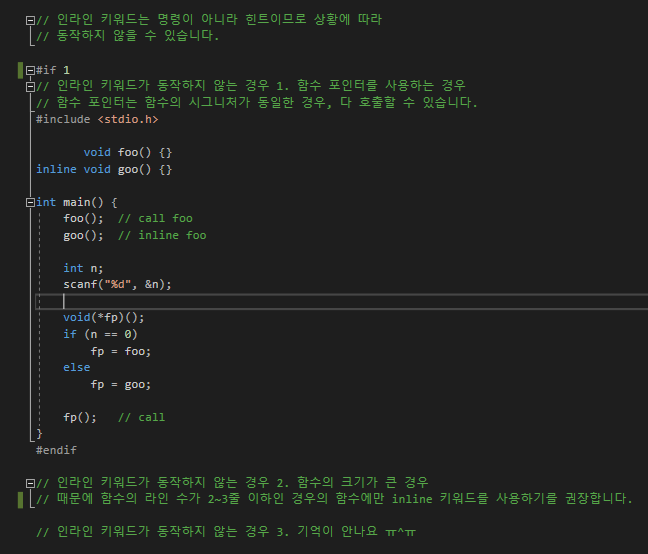
Inline 함수의 명령어코드를(기계어코드) 그대로 때려박는 함수..

인라인 함수: 함수 호출의 코드에 함수호출 명령어(call)이 아닌 함수의 기계어 코드로 치환(inlining) 되는 함수. 이는 c99에서 표준으로 채택된 문법으로 c++에서 차용해 왔다.

->인라인 문법은 최적화 문법이므로 최적화 옵션을 사용하지 않으면 전달하지 않는다.

11번째줄의 기계어 코드가 바로 18번째줄에 박힌다.

* 함수가 호출되어 함수정의부로 올라가는게 아니라 그 함수의 코드가 복제되어 호출부로 간다(즉, 함수로의 이동이 아니라 그 함수가 복제되어서 호출부로 온다)



인라인 키워드는 명령이 아니라 힌트이므로 상황에 따라 동작하지않을수 있다.

**인라인 키워드가 동작하지 않는 경우** 1. 함수포인터를 사용하는 경우

즉, 함수포인터는 함수의 시그니처가 동일한 경우(매개변수의 타입과 개수가 동일), 다 호출될수 있다.’’

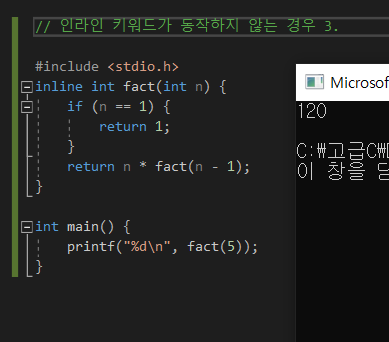
아래 예제의 if문은 런타임에 결정되고 inline은 컴파일타임에 실행되므로 inline은 동작되지x

**인라인 키워드가 동작하지 않는 경우** 2. 함수의 크기가 큰 경우

Cf.크기가 작은 함수에서 inline함수는 효율적이지만 큰 함수에서는 inline을 쓰게 되면 실행 안될수도있다.

함수의 라인수가 2~3줄 이하인 경우의 함수에만 inline 키워드를 사용하기를 권장한다.

**인라인 키워드가 동작하지 않는 경우 3.** 재귀호출



* 인라인 함수의 단점: 자칫 목적 파일의 크기가 커질수 있다.

(오히려 2~3줄 길이의 함수는 목적파일의 크기가 작아진다. )

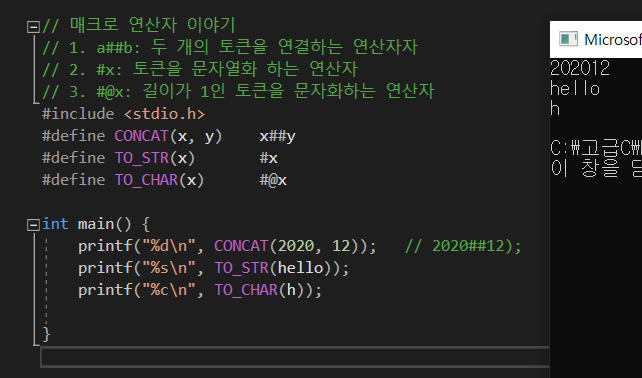
<매크로 연산자>

1. a##b : ab (두개가 붙어버림) -> 두개의 토큰을 연결하는 연산자

연산의 개념이 아니라 연산의 개념이므로 ( ) 넣지 말고 그대로 써야한다.

1. #x : 토큰을 문자열화하는 연산자
2. #@x : 길이가 1인 토큰을 문자화하는 연산자(이거는 잘 쓰이진 않음)

마이크로소프트 전용이라 이식성이 떨어짐..잘 안씀..



<미리 정의된 메크로>

🡺4.c 처음부분

\_\_FILE\_\_ : 매크로가 치환되는 순간의 소스파일의 경로와 파일의 이름

메크로이므로 컴파일 타임이 아니라 전처리동안에 처리됨.

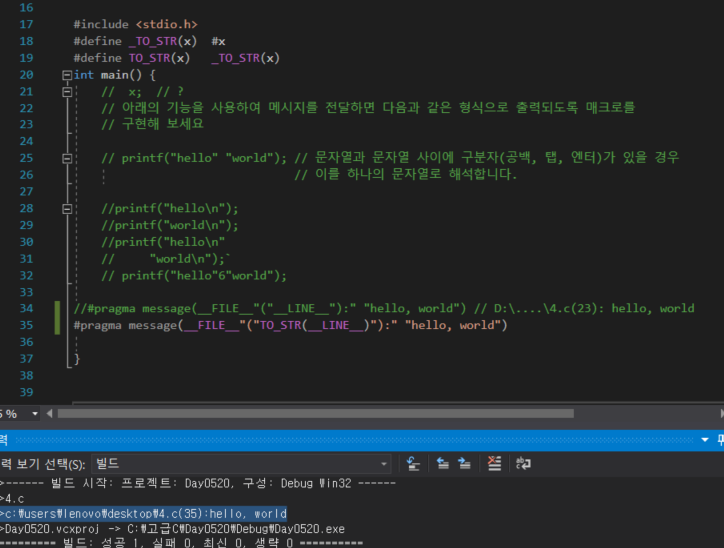
\_\_DATE\_\_ : 현재날짜

\_\_TIME\_\_ : 현재 시간

\_\_LINE\_\_ : 현재 해석되고있는 라인수 보고싶을 때(정수형)

\_\_func\_\_ : 함수의 이름 (위 4개는 전처리기가 제공하는것. 이것은 컴파일러가 제공하는 심볼)(표준)

\_\_FUNCTION\_\_ : 이식성에 떨어지므로 위에것 쓰자.(확장문법이므로 표준은 아님)(비표준)



19번째줄- > 바로 넣게 되면 \_\_LINE\_\_이 숫자로 안바뀌고 출력되므로 define을 두번거쳐서 가야지 숫자로 바뀌고 그게 18번째 라인으로 들어가게된다.

조건부 매크로

중복안되게(include guard) - ifndef ~ endif 사용

->조건부 매크로를 사용하여 헤더파일이 중복 포함되지 않도록 하는 기법(include guard)

