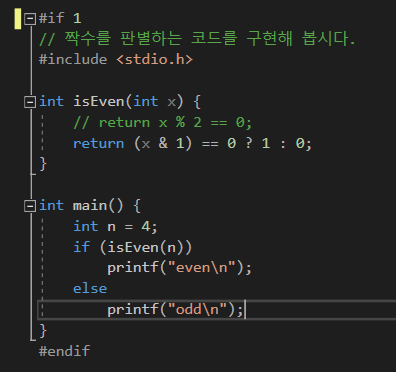
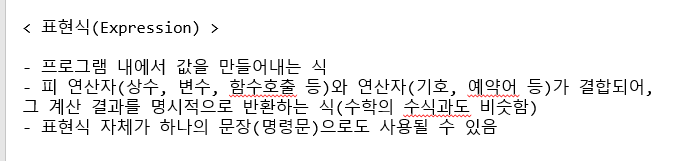
<표현식 exp >



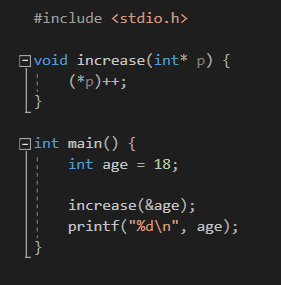
비트 연산시 끝이 1이면 홀수, 끝이 0 이면 짝수임을 이용한 코드

0001 -> 1 홀

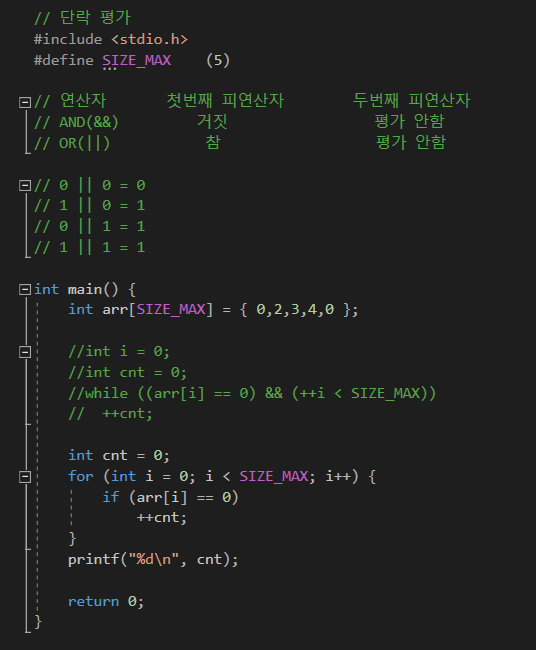
0010 -> 2

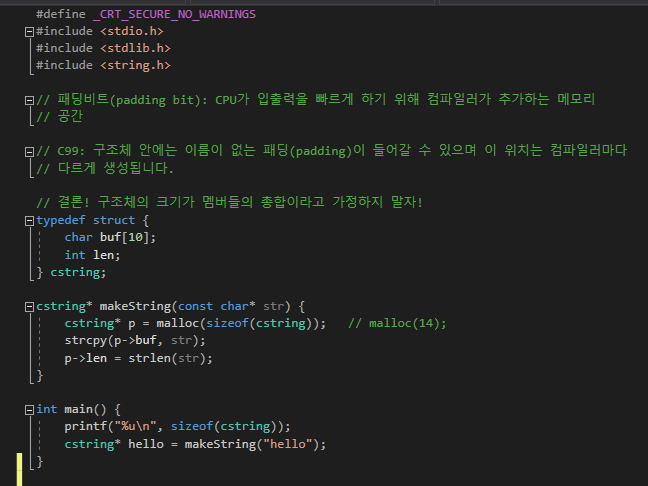
0011 -> 3 홀

0100 -> 4



단락 평가 && ||





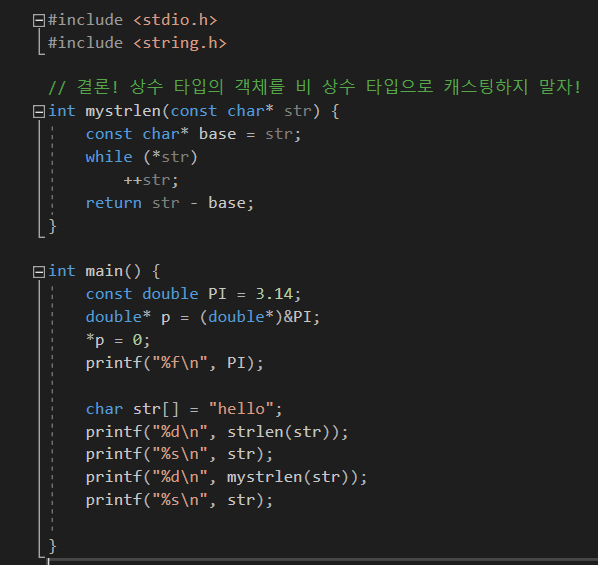
구조체 안에 패딩비트가 있으므로 char와 int의 크기만 더해서 malloc에 넣지 말자.

하드코딩 하지말고 sizeof를 사용하자!

패딩비트(padding bit) : CPU가 입출력을 빠르게 하기위해 컴파일러가 추가하는 메모리 공간.

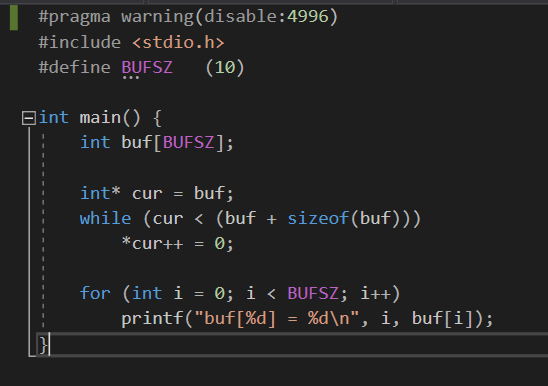
C99에서는 구조체안에 이름이 없는 패딩(padding)이 들어갈 수 있으며 이 위치는 컴파일러마다 다르게 생성된다.

🡪결론 : 구조체의 크기가 멤버들의 총합이라고 가정하지 말자!



상수타입의 객체를 비 상수 타입으로 캐스팅 하지말자

base 만들때 앞에도 const꼭 써주자.



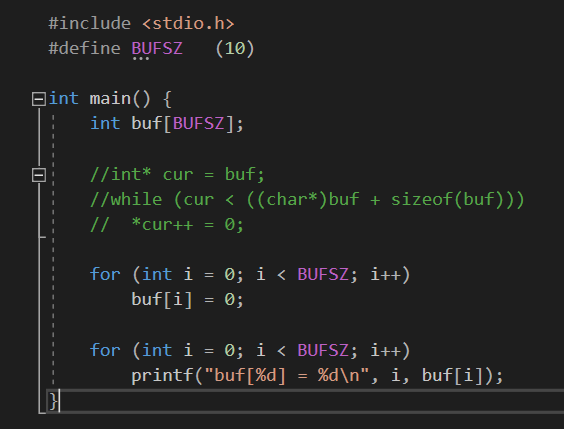
Buf + sizeof(buf) 를 하게 되면 sizeof에 \*4가 처리되어 buf의 크기를 넘어서게 되므로 buf를 char\* 로 형변환 하여 더해준다. 왜냐하면 char \* 만이 p + 1 = 1이 되므로.

포인터였다면 p+1 을 하게 되면 4를 더해준다..

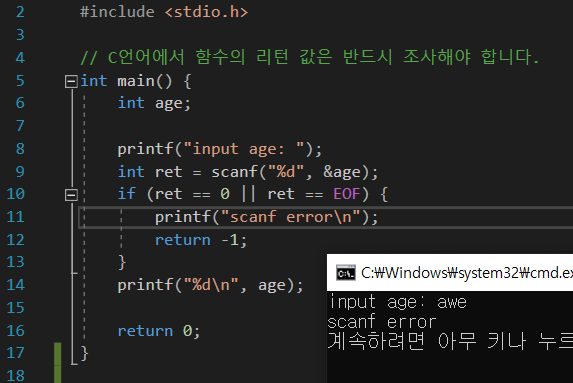
즉, 어떤 임의의 위치에서 원하는 만큼 이동하게 위해서는 char \* 를 써야한다. (1바이트단위로 입출력이 이루어지는 char!)

그러나 위 코드보다는 아래 코드가 더 좋다. 주소 계산보다 첨자연산이 더 좋다.

그러나 루프를 돌면 성능이 떨어지므로, memset을 쓰면 더 좋다.



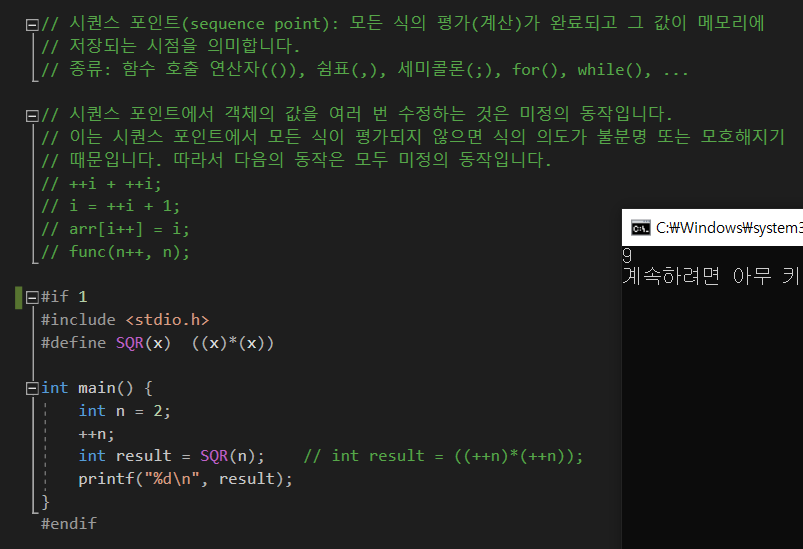
Day0522



printf 의 리턴값 : 출력된 개수(int)

Scanf 의 리턴값: 잘못된 입력이 들어오면 0 / 제대로 들어오면 입력받은 개수(int)

* c언어에서 함수의 리턴값은 반드시 조사해야한다.



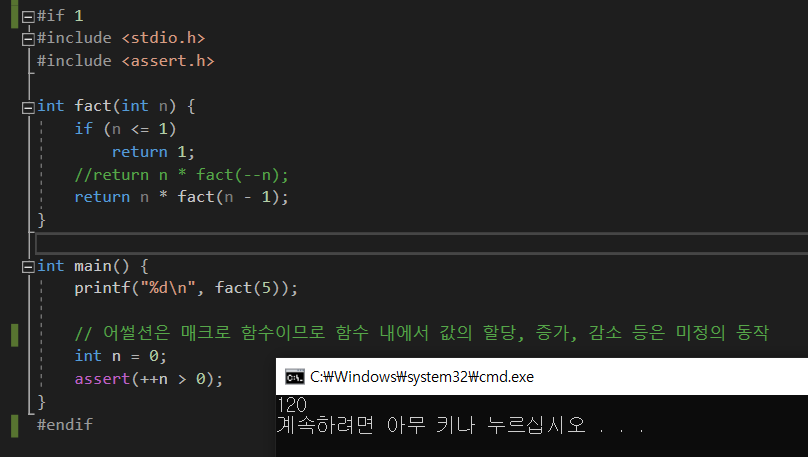
시퀀스 포인트(sequence point) :모든 식의 평가(계산)가 완료되고 그 값이 메모리에 저장되는 시점

시퀀스 포인트의 종류 : 함수호출 연사자 ( ) , 쉼표 , , 세미클론 ; , for(), while() , …..

시퀀스 포인트에서 객체의 값을 여러 번 수정하는 것은 미정의 동작이다.

이는 시퀀스 포인트에서 모든 식이 평가되지 않으면 식의 의도가 불분명 또는 모호해지기때문이다. 따라서 다음의 동작은 모두 미정의 동작이다.

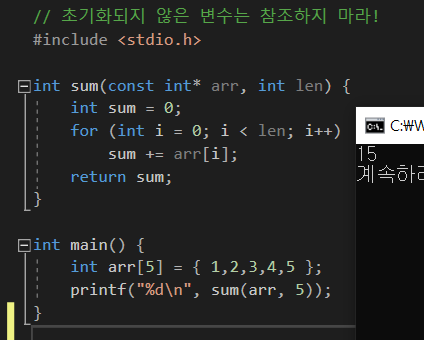
Ex> ++i + ++i; i = ++i + 1; arr[i++]=i; func(n++, n); 등..



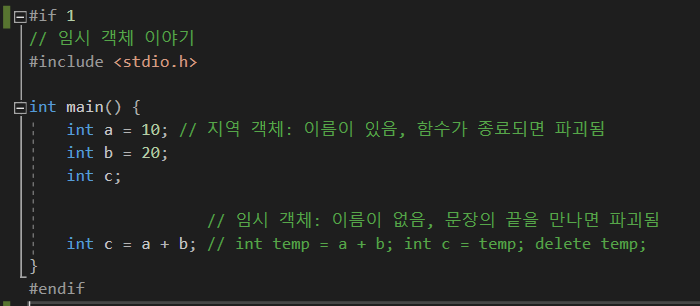
재귀에서의 return n \* fact(--n); 은 미정의동작이므로 24가 나온다.. 그러므로 -–n을 쓰지말고 n-1을 사용하자

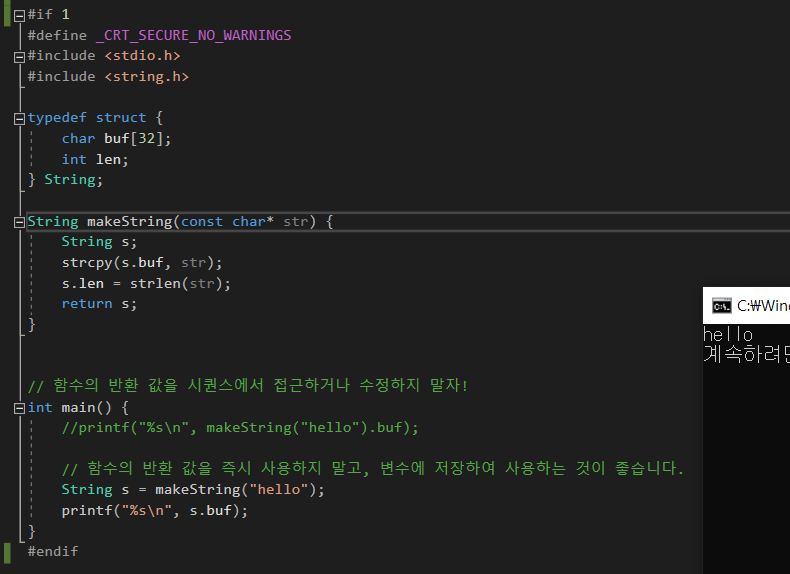
assert는 찾아보면 매크로 함수이므로 함수내에서 값의 할당, 증가, 감소 등은 미정의 동작이므로 사용하지말자

=>그냥 한줄에는 하나의 연산자만 있으면 된다!!



초기화되지 않은 변수는 참조하지 마라

임시객체에 대하여(지역객체와 임시객체)



함수의 반환 값을 시퀀스에서 접근하거나 수정하지말자!

임시객체를 바로 받아서 쓰게되면 임시객체가 남아있는지 아닌지 모르므로 지역객체를 써서 처리하자! 즉, 함수의 반환값을 즉시 사용하지말고, 변수에 저장하여 사용하는 것이 좋다.