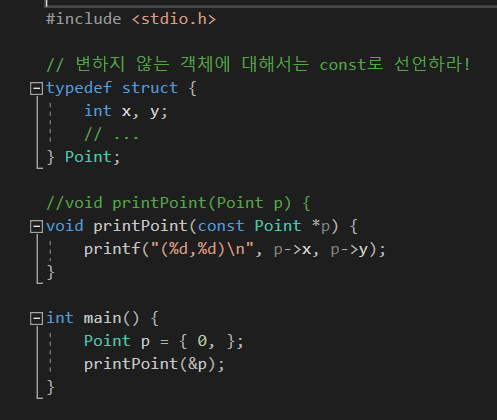
c언어코드 ----------🡪 .i ----------🡪.asm-------------🡪기계어(목적파일)--------🡪 런타임단계

전처리단계 컴파일단계 어셈블단계 링크단계



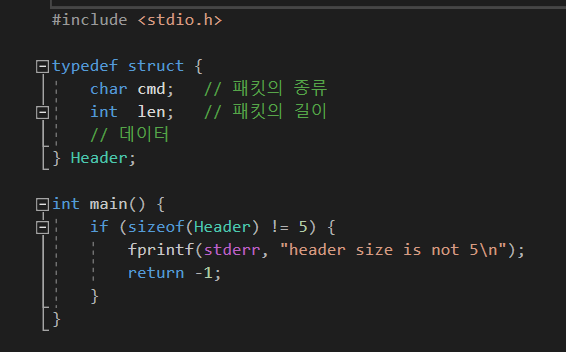
복사에 의한 오버헤드가 발생하는 단점이있다.. (스택에 복사를 하여 올리기 때문에..)

그래서 포인터를 사용하여 고침(Point \*p) 라는 식으로.

하지만 그래도 단점이있다. -> const 붙이면 오류(컴파일타임의 오류라는 것을 알수있다)

방어적 테크닉(const를 쓰고보자!) -> 값을 변경안할시에 이거 사용하고, 만약 오류나면 컴파일 타임에 오류가 뜬다라는 사실을 알수있다!

강사님의 요약: 변하지 않는 객체에 대해서는 const로 선언하라!



구조체의 크기(패딩비트는 프로세서마다 다르기 때문에) 만약 서로 다른 크기를 가진 것을 연결시키면 깨지게된다. 그래서 if문으로 크기가 다르다면 return 해주면 된다.

그러나 위에 코드는 복잡하므로 assert()를 써준다. -> 헤더의 크기를 5라고 단정해주었지만 이가 실패되었다고 출력된다. (방어적 프로그래밍의 기법중 하나)

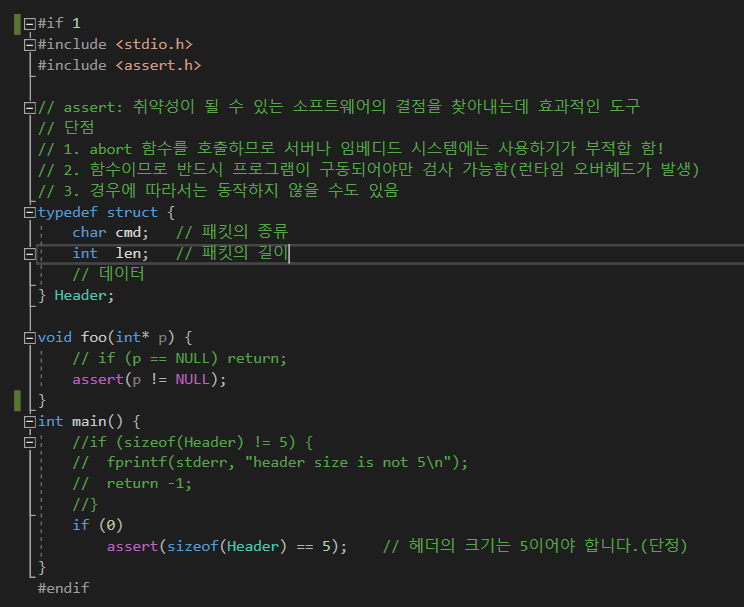
->assert는 취약성이 될수 있는 소프트웨어의 결점을 찾아내는데 효과적인 도구이다. 그러나 단점이 있는데

1. abort 함수를 호출하므로 ‘서버’나 ‘임베디드 시스템’에서는 사용하기 부적합하다.

2. 함수이므로 반드시 프로그램이 구동되어야만 검사 가능함(런타임 오버헤드가 발생)

3. 경우에 따라서는 동작하지 않을수도 있다.

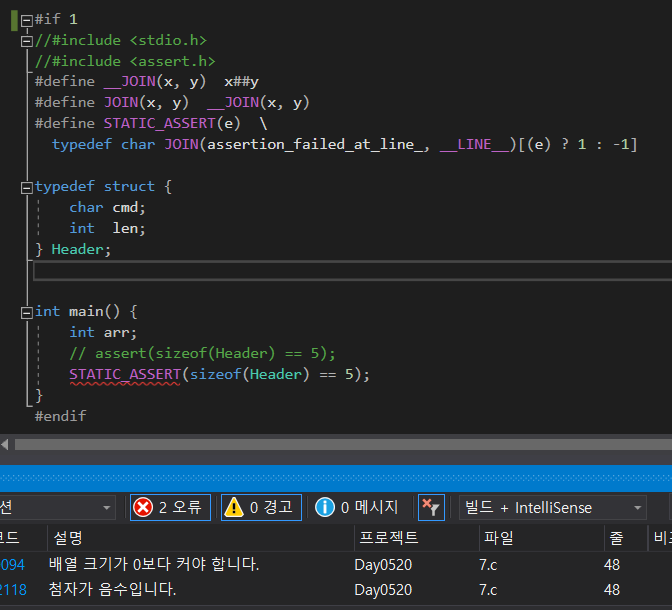
assert는 런타임에서만 작동하므로(c에서) 직접만들어서 써야한다. -> static정적 assert를 만들자



정적 assert 만들기(컴파일타임에 검사해줌)

그러나 arr의 용량이 차지되므로 단점이다. 그래서 typedef를 써서 정의가 아니라 선언을 하게 되면 메모리를 잡아먹지 않는다.(스택에) 그냥 선언후 갖다 버리기 때문에..

그러나 또 문제가 발생하는데 arr라는 심볼을 다른곳에서 사용할수 있다. 그러므로 아래코드처럼



매직 넘버를 회피하기 위한 방법 1. 상수(const) 로 바꾸기

단점 1. switch문에서 쓸수 없다. 2. 메모리를 차지한다.

Switch에 오는 값은 반드시 컴파일타임에 결정되는 literal값이 와야한다.

매직 넘버를 회피하기 위한 방법 2. 매크로 상수를 사용하기

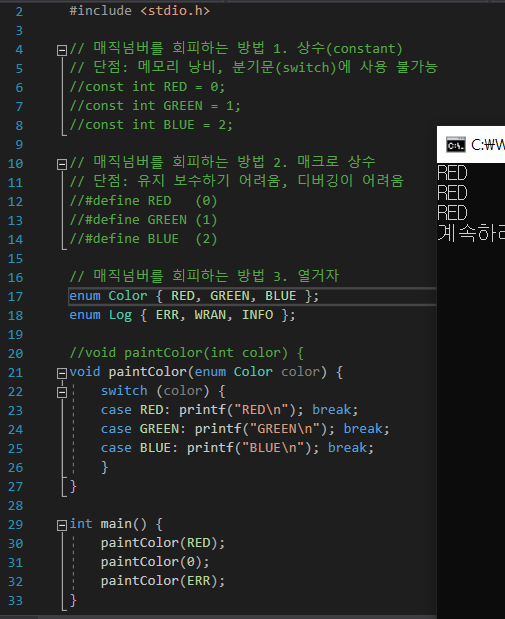
단점 1. 추가 제거시 숫자가 바뀌므로 유지보수가 쉽지 않다. 2. 디버깅이 어렵다. 색상이 넘어가는것이 아니라 숫자가 넘어가서 디버깅 어려움.

매직 넘버를 회피하기 위한 방법 3. 열거자를 사용하기(enum)

디버깅이 쉽다/ 메모리를 잡아먹지도 않는다 /

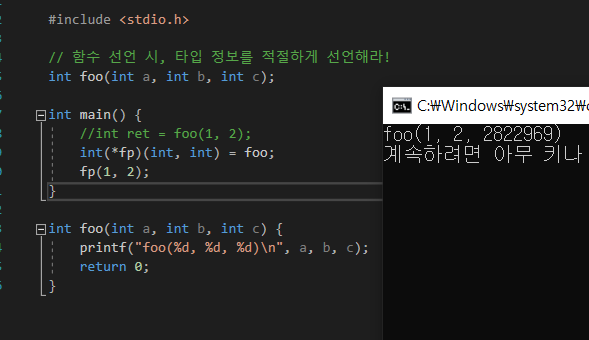
그러나 단점도 잇는데, enum은 완벽한 타입으로 설정되어있지 않아서 다른 enum에 대한게 들어가도 작동이된다.. 0이 들어가도 되고, 다른 enum으로 설정한 err가 들어가도 되고…

타입 안정성이 살짝 떨어지지만 그래도 이게 제일 낫다.



1.함수선언시엔 타입 정보를 적절하게 선언해야한다.

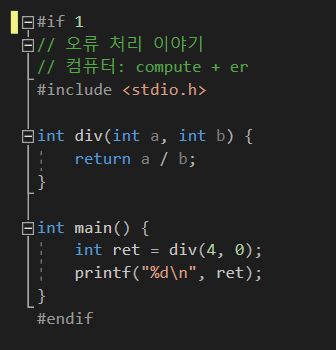
2. 함수포인터 사용시에도 적절하게 잘 선언해야한다.(경고도 내주지 않는다..) – 조심하자



Day0521

오류처리 이야기

제수에 0이 들어갈 경우가 있으므로 좋은 코드가 아니다.(즉, 0이 들어오면 미정의 동작이기 때문에)



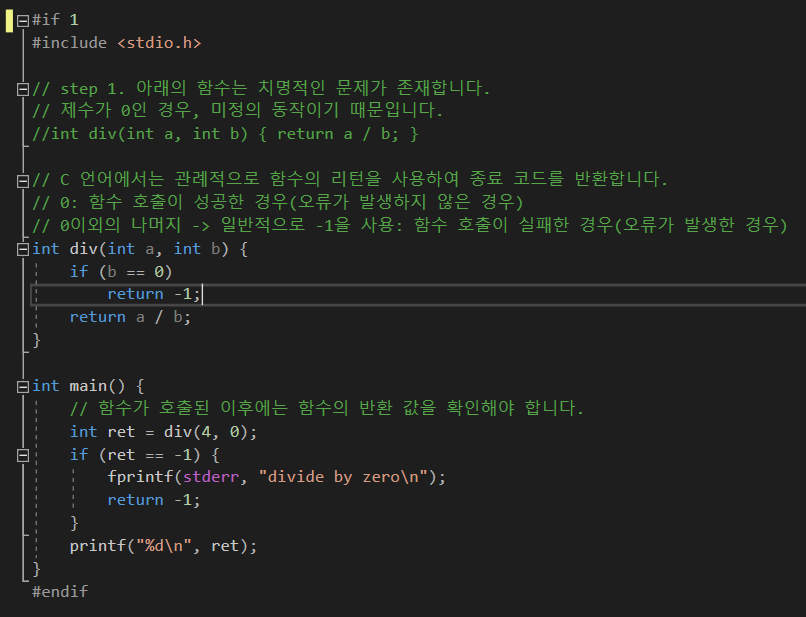
c언어에서는 관례적으로 함수의 리턴에 종료 코드를 반환한다.

0: 함수호출이 성공한 경우(오류가 발생하지 x)

0이외의 나머지(보통은 -1) : 함수 호출이 실패한 경우(오류가 발생한 경우)

함수가 호출된 이후에는 함수의 반환 값을 확인해야한다.

그러나 반환 타입 int에서 return이 정상적으로 -1이 나올때랑 오류가 발생하여 -1을 리턴할때가 혼재되어있다. (그래서 객체지향에서는 exception이라는 것이 생기게 되었다.)



위와 같은 문제들을 해결하기 위해 결과와 오류코드를 분리해보자.

해결방법

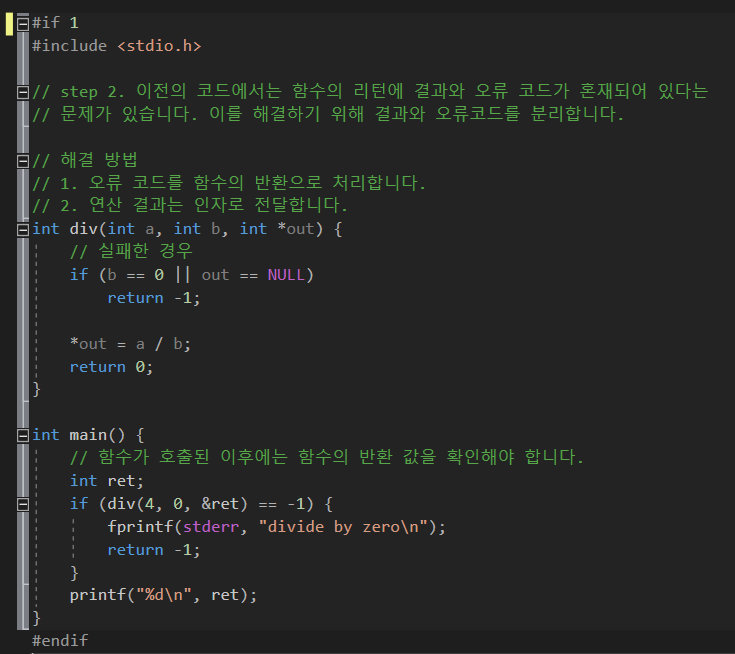
1. 오류코드를 함수의 반환으로 처리한다.
2. 연산결과는 인자로 전달한다.

(Int \*ret은 값을 꺼내기위한 변수이므로 아웃파라미터 라고한다.

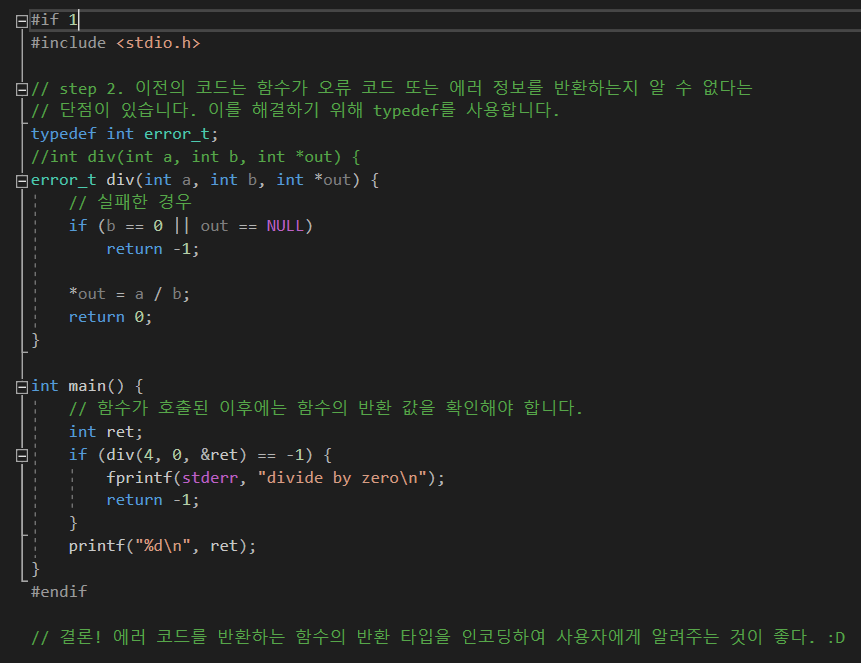
만약 값을 전달하기 위해 사용한 변수라면 인파라미터 라고한다.??(인파라미터는 const를 앞에 붙인다.))

인자(argument) – 함수호출시 사용되는 괄호안 변수들

매개변수(parameter) – 함수정의시 사용되는 괄호안 변수들



이전 코드는 함수가 오류 코드 또는 에러 정보를 반환하는지 알 수 없다는 단점이 있다. 이를 해결하기 위해 typedef를 사용한다.( 코드 자체를 주석으로 만드는 좋은 방법)

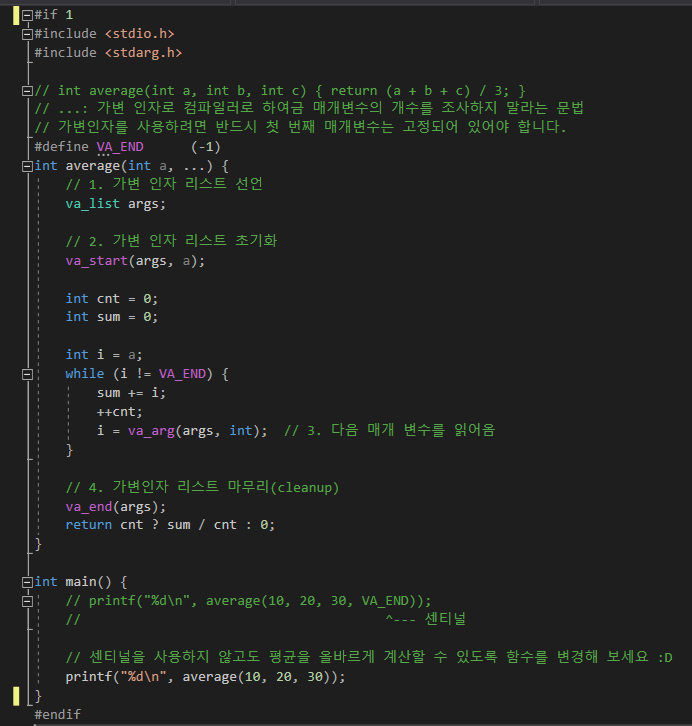


* 결론! : 에러 코드를 반환하는 함수의 반환 타입을 인코딩하여 사용자에게 알려주는 것이 좋다.

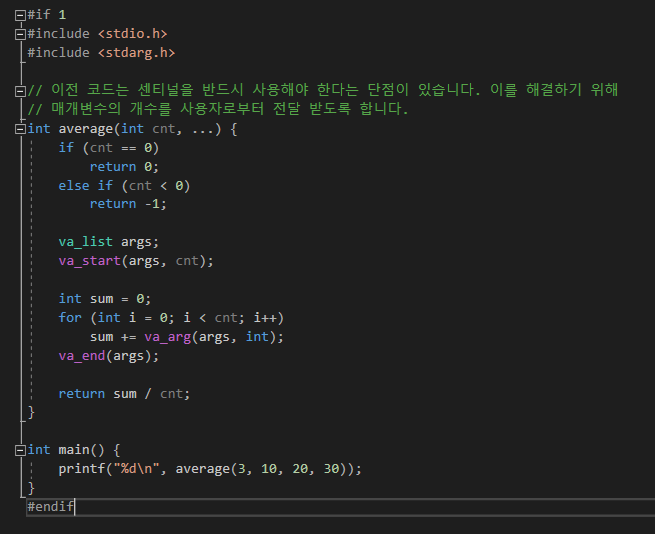
… : 가변인자로 컴파일러로 하여금 매개변수의 개수를 조사하지 말라는 문법임

가변인자를 사용하려면 반드시 첫번째 매개변수는 고정되어있어야한다.

함수 호출부에서의 마지막 부분엔 ‘센티널’값을 주어 끝을 지정해준다.



위 코드에선 센티널을 사용하였지만 안쓰고도 맨 처음 매개변수를 전체 매개변수의 개수로 받아주면 굳이 센티널을 사용하지 않아도 된다. 즉, 위 코드는 센티널을 반드시 사용해야한다는 단점이 있기 때문에 매개변수의 개수를 사용자로부터 전달받도록 하는 코드.

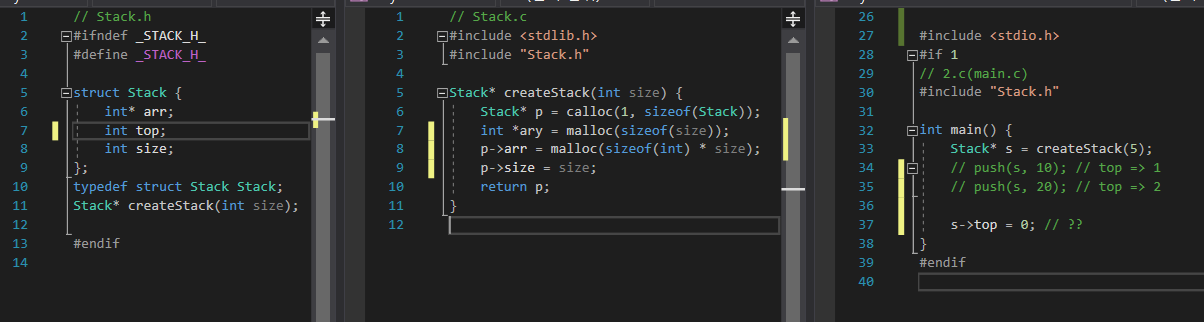


정보은닉 / 정수를 저장하는 스택 자료구조에 대해서.

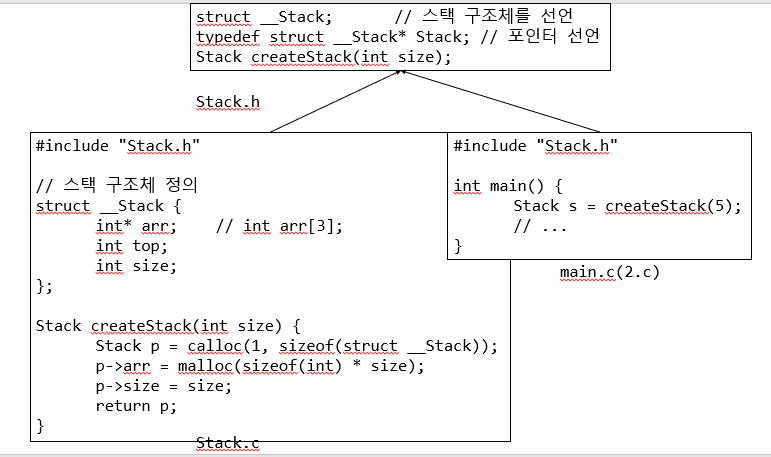
🡺2.c 첫부분 / circle.h / circle.c

위코드를 모듈화 시키기

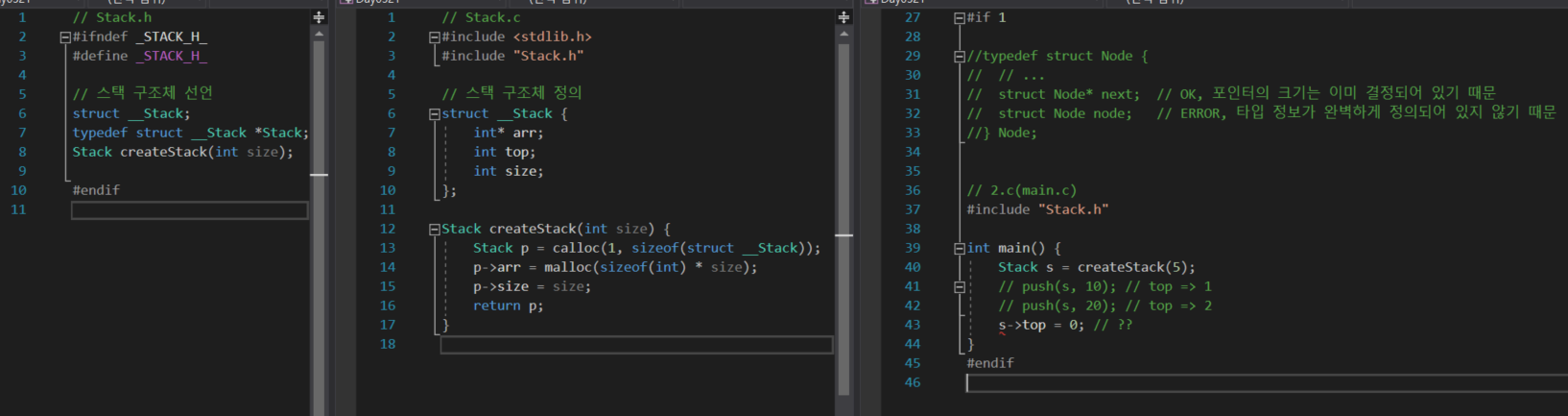
🡺2.c 27줄부터(main.c) //Stack.h / Stack.c



위 코드에서 top을 중간에 바꿔버리면 그 스택은 버려지게된다. 그래서 top의 값을 못바꾸게. 즉 c언어는 정보은닉이 없으므로 이것을 만들어보자



🡺2.c 27줄부터(main.c) //Stack.h / Stack.c



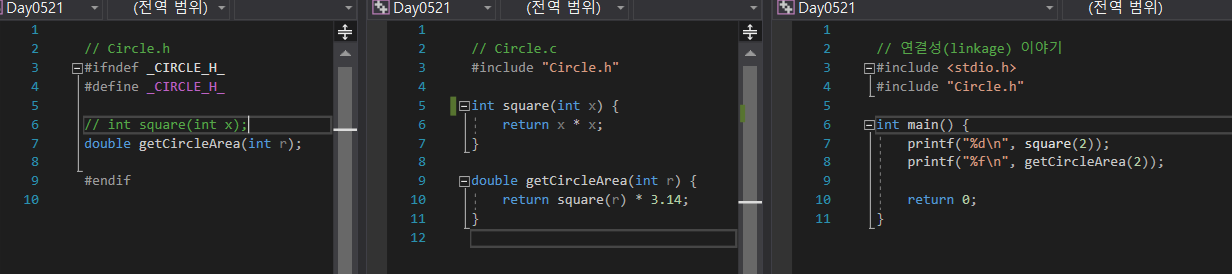
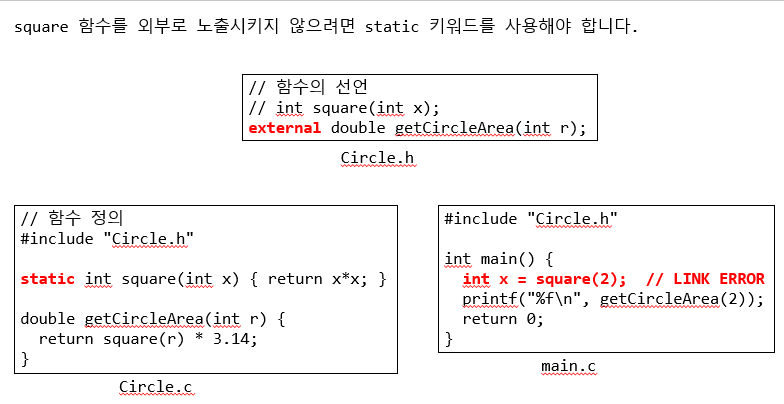
연결성(linkage) 이야기

(Cf. Circle.c안에 circle.h를 선언해주는 이유.(굳이 안해도되나?)

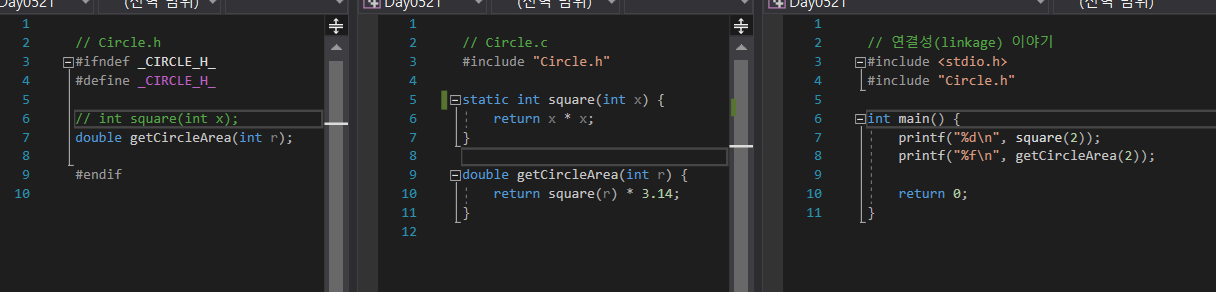
교차호출했을 때 선언이 미리 되어있어야하는데 include안해주면 교차 호출시 해당 함수를찾지 못한다.)

Static 키워드를 안쓰게 되면 int square(int x);를 선언안해도 컴파일러가 자동으로 해준다..

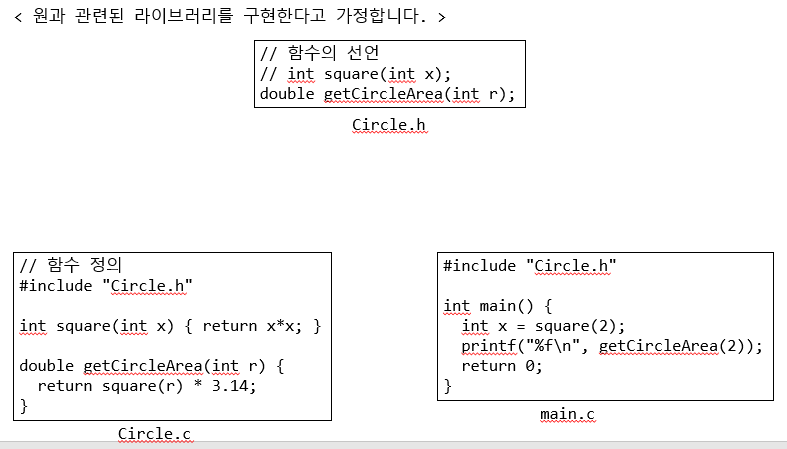
이런 점을 바로잡기위해 static키워드를 써주자!(아래코드)



Static 삽입후..

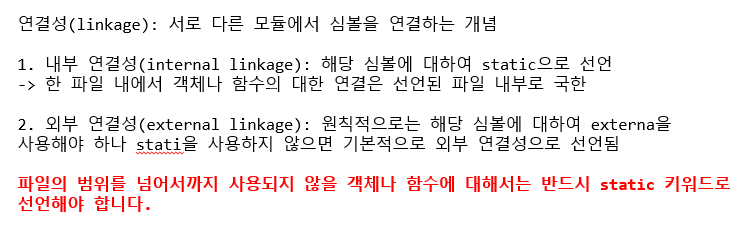


Square 함수를 외부로 노출시키지 않기 위해서 static 키워드를 사용한다.(실행 안되고 오류 잘 뜬다) – static(정적)함수는 헤더에 x



연결성(linkage) : 서로 다른 모듈에서 심볼을 연결하는 개념

1. 내부 연결성(internal linkage)
2. 외부 연결성(external linkage)

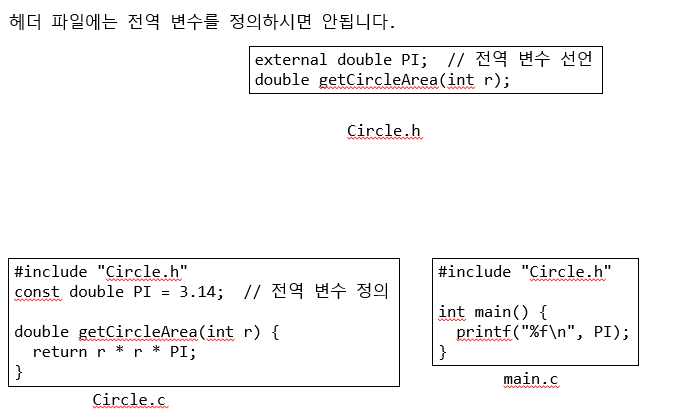


헤더파일에는 전역변수를 정의하면 안된다.

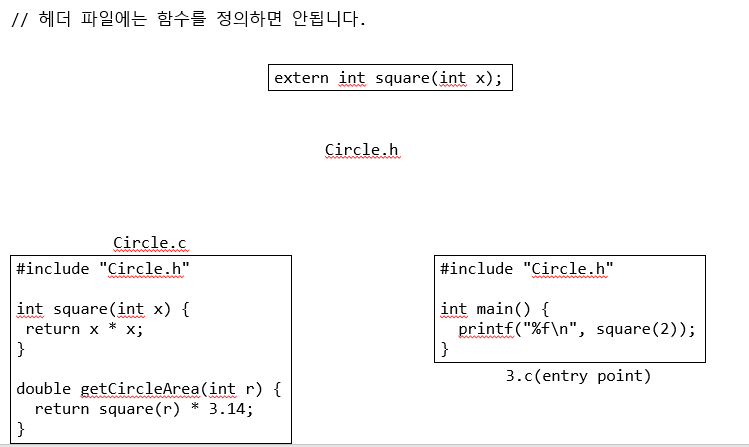
만약 헤더에 넣게되면 전처리기 단계까지는 문제가 없다. 그러나 링커는 circle.c 에도. main.c에도 둘다 똑 같은 PI가 존재하므로 링크에러가 발생한다.(링커 단계에서)

그래서 해결책으로 .h에서 전역변수를 정의하는게 아니고 전역변수는 circle.c에서 정의를 해주고 .h에서는 선언만 해주면 된다.(전역으로 external 키워드 붙여서) 그리고 나서 링커 단게에서 메인에서 사용된 PI가 어디있나 뒤지게 되는데 그게 circle.c에서 3.14라는걸 발견하고 잘 실행된다. 즉, 헤더파일에는 정의를 둬서는 안된다. 아까 구조체는 .h에 올라갓는데 왜냐하면 typedef로 선언하여 메모리에 잡히지 않았기 때문에 가능한것이었다.

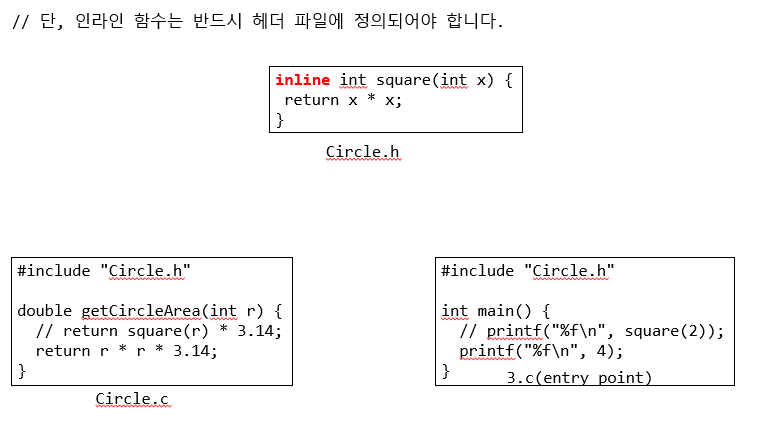
매크로/인라인 함수는 반드시 헤더파일에 있어야한다. (internal linkage의 속성을 가진다) -> 이것에 대한 설명은 아래코드에.



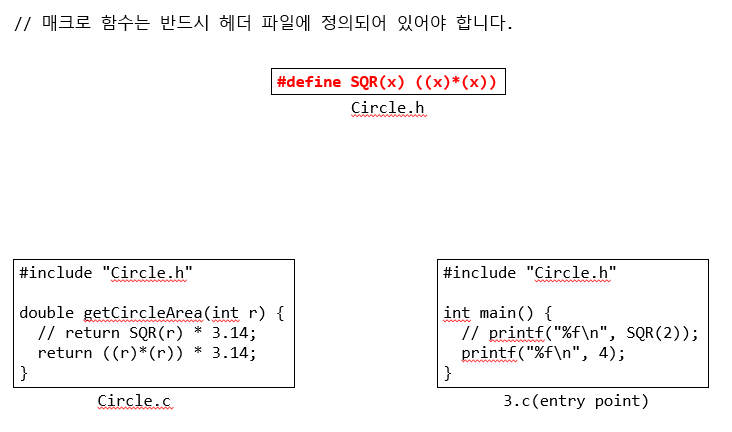
헤더파일에는 함수를 정의하면 안된다.



단, 인라인함수는 반드시 헤더파일에 ‘정의’되어야한다. 코드가 이동하는 것이 아니라 .c 나 main에 기계어 코드가 박혀야하므로 .h에서 인라인함수를 정의해줘야한다. 인라인함수를 모든 곳에 기계어코드로 바뀌게 되면 헤더에 있는 인라인코드는 사라진다.



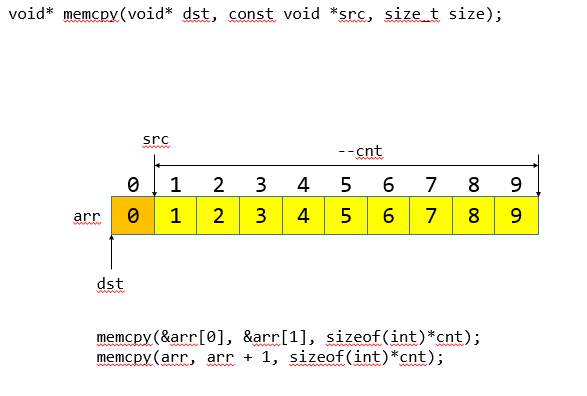
메크로 함수도 반드시 헤더파일에 ‘정의’되어있어야한다.

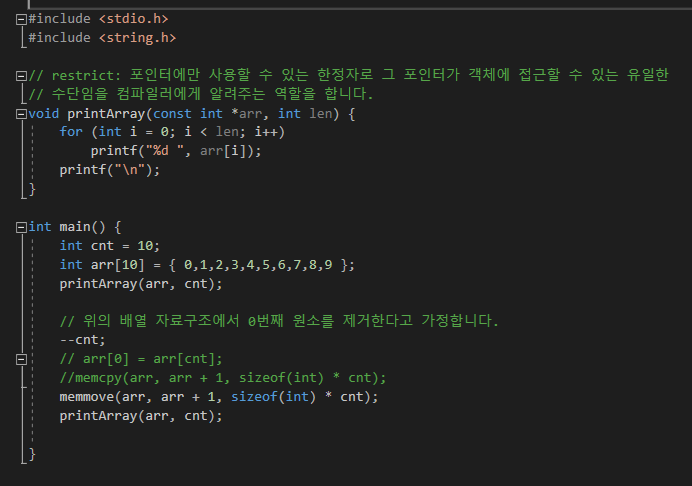


총 정리!! -> 헤더파일 설계시 주의할점

1. 인클루드 가드를 설정해야한다..
2. 객체(변수)나 함수는 정의를 하면 안된다.
3. 인라인 함수나 매크로는 헤더파일에 정의해야한다.

단, 타입에 대한 정보는 헤더 파일에 정의할 수 있다(ex 구조체).

🡺



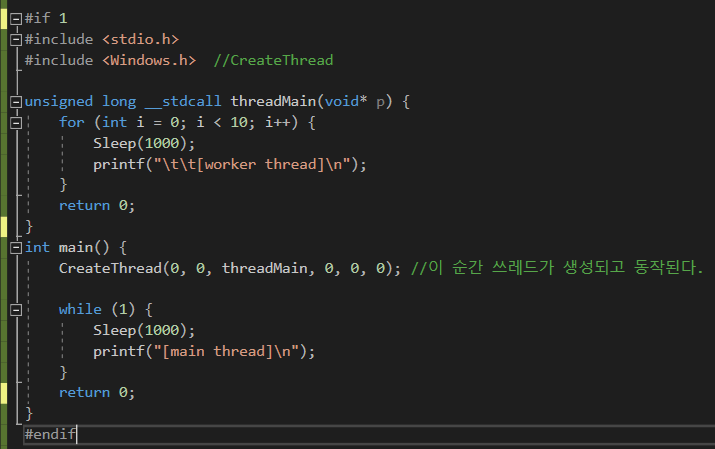
memcpy는 서로 같은 메모리의 복사를 하면 중첩되기 때문에 사용하면 안된다!  
그럴때엔 memmove를 사용해야한다.(버퍼를 한번더 복사하고 쓰기 때문에 속도는 살짝 더 느리지만 더 안전하다!- 절대 데이터가 유실될 일 없으며 안전히 복사가 가능하다)

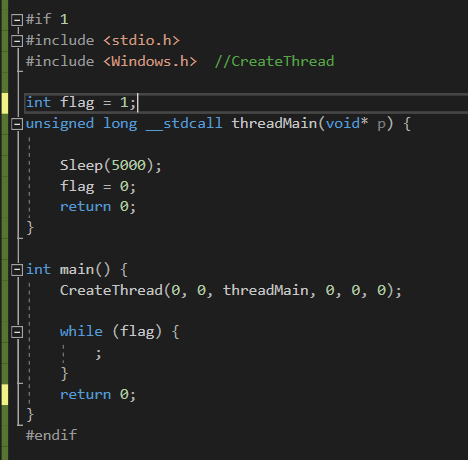
즉 같은 메모리에 대해서는 memcpy가 아니라 memmove를 사용하자!

총정리!

* restrict : 포인터에만 사용할수 있는 한정자로 그 포인터가 해당 객체에 접근할수 있는 유일한 수단임을 컴파일러에게 알려주는 역할을 한다. 포인터가 restrict 키워드로 한정되는 그 포인터가 가리키는 객체는 그 포인터만 접근 가능하므로 컴파일러가 좀더 적극적인 최적화를 수행 할 수있다. 만약 이 한정자를 사용한 포인터에 대하여 같은 객체를 가리키면(참조) 그 결과는 미정의 동작이다.

c언어에서의 쓰레드





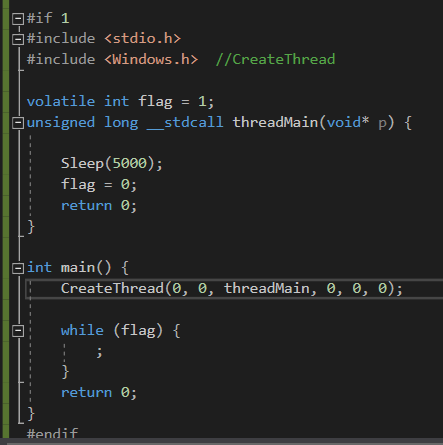
c언어에서의 쓰레드2

원래 라면 flag돌다가 thred가 5초가 지나면 flag가 0이되고 while문에 flag가 0이 적용되어 while문 빠져나와야 정상인데 release모드에서는 안끝난다.. debug모드에선 정상 처리된다.

왜냐하면 flag값을 cpu값에 캐싱하는데, 캐시해서 읽어오므로 물리메모리의 값이 바뀐다해서 캐시의 값이 바뀌지 않기 때문이다.

이를 해결하기 위한 해결책은 volatile이다.

위 코드에 volatile 추가한 코드



정상적으로 5초 후에 잘 종료된다.

🡺정리: volatile : 값이 프로그램이 아닌 외부적인 요인에 의하여 그 값이 변경될 수 있다고 컴파일러에게 알려 캐싱을 제한할 때 사용하는 키워드이다.

따라서 캐시되어서는 안되는 데이터에 대해서는 반드시 volatile을 사용해야한다.