항목 30 : 인라인 함수는 미주알고주알 따져서 이해해 두자

인라인 함수의 장점

* 함수 호출 비용(오버헤드)이 없습니다.
* 매크로보다 훨씬 안전하고 쓰기 좋습니다.
* 컴파일러가 함수 본문에 대해 문맥별 최적화를 걸기가 용이해집니다.

그러나 장점만 보고 인라인을 남발해서는 안됩니다. 왜 안 되는지 알아보도록 하겠습니다.

**우선 목적 코드의 크기가 커집니다.** 인라인 함수를 호출하는 함수의 본문에 인라인 함수 코드가 들어가기 때문입니다. 특히 템플릿을 inline함수로 만들다간 코드 부풀림 현상이 더욱 심해질 것입니다. 따라서 부풀려진 코드는 메모리를 많이 차지할 수 있고 명령어 캐시 적중률이 떨어질 수 있습니다.

inline함수를 만드는 방법은 명시적인 것과 암시적인 것 두 가지가 있습니다. 헤더 파일 안의 클래스 정의 부분에 함수를 만들면 컴파일러는 인라인 함수 후보로 찍습니다. **후보라는 표현을 쓴 것은 무조건 인라인 함수가 되는 것이 아니기 때문입니다.**

명시적인 방법은 함수 정의 앞에 inline 키워드를 붙이는 것입니다. 하지만 이 것 또한 컴파일러가 무조건 인라인 함수로 만들어 주는 것은 아닙니다.

**inline함수를 만들어 주는 것은 컴파일러만의 권한입니다.** 조건을 충족 시켜 주어야만 컴파일러는 인라인 함수로 만들어 줍니다.

인라인 함수가 될 수 있는 조건

* 루프 혹은 재귀 함수가 없어야 합니다. **즉, 복잡한 함수는 안 됩니다.**
* 간단한 함수라 할지라도 **가상 함수를 호출하는 함수는 인라인 될 수 없습니다.** ( 가상 함수는 실행 중에 어떤 함수가 호출 될지 결정되고 인라인 함수가 만들어지는 시점은 컴파일 도중이라 절대 인라인 될 수 없습니다. )

*다음 코드는 인라인 될 것같이 보이는데 안 되는 경우의 예제 입니다.*

*class Base {*

*private:*

*string bm1, bm2; // Base 클래스의 멤버*

*};*

*class Derived: public Base {*

*public:*

***Derived( ) { } // 비어 보이는 생성자***

*private:*

*string dm1, dm2, dm3; // Derived의 멤버*

*};*

Derived 생성자는 아무것도 없어 보이므로 인라인하기에 딱 좋아 보입니다. **하지만 컴파일러가 볼 때는 많은 코드가 숨어있다는 것을 알고 있습니다.** 우선 기본 클래스인 Base생성자를 호출 하는 코드가 숨어 있을 것이고 string dm1, dm2, dm3의 각각 생성자를 호출 해주는 코드가 있을 것 입니다.

**텅 빈 생성자 처럼 보이지만 실제로는 그렇지 않습니다.** 복잡하고 인라인으로 만들었다간 코드 부풀림이 심해질 것입니다.

라이브러리를 만들 때 인라인으로 만들고 어느 날 그 라이브러리 내부가 바뀌면 이 라이브러리를 사용하는 모든 사용자들은 다시 컴파일해야 합니다. **더 큰 문제는 디버깅이 힘들어 진다는 것입니다. 있지도 않은 함수에 중단점을 걸 수는 없잖아요?** 이렇다 보니 어떤 함수를 인라인으로 만들지 말지 고민이 됩니다.

**우선 인라인 할지 말지는 컴파일러에게 맡기세요. 꼭 인라인으로 만들고 싶다면 함수를 최대한 간단하게 만드세요.** (예를 들어 멤버변수 값만 리턴 하는 함수) 그리고 자주 호출되는 함수에 대해서만 만드세요.