**항목 30. 인 라인 함수는 미주알고주알 따져서 이해해 두자.**

인 라인 함수는 좋은 아이디어이다. 함수처럼 보이며 함수처럼 동작하는데다, 매크로보다 훨씬 안전하고 쓰기 좋으며 함수 호출 시 발생하는 오버헤드도 걱정할 필요가 없다.

또, 잘 모르는 이점이 하나 더 존재 하는데, 함수 호출 비용이 면제 되는 것은 눈에 보이는 것에 불과 하며, 대체적으로 컴파일러 최적화는 함수 호출이 없는 코드가 연속적으로 이어지는 구간에 적용되도록 설계 되었기 때문에, 인라인 함수를 사용하면 컴파일러가 함수 본문에 대해 문맥 별 최적화를 걸기가 용이 해진다.

그러나, 단점도 존재하는데 **인 라인 함수의 아이디어는 함수 호출 문을 그 함수의 본문으로 바꿔 치기 하자는 것이기 때문에, 목적 코드의 크기가 커지기 때문에 메모리가 제한된 환경에서 난발하게 되면, 프로그램의 크기가 그 환경에서는 사용할 수 없을 만큼 커진다.**

**또, 인 라인 함수의 남용은 페이징 횟수가 늘어나며, 명령어 캐시 적중률이 떨어질 가능성이 커집니다.**

하지만, 본문의 길이가 짧은 인 라인 함수를 사용하면, 함수 본문에 대해 만들어지는 코드의 크기가 함수 호출 문에 대해 만들어지는 코드보다 작아질 수 있다. 이 경우 목적 코드의 크기가 작아지며 명령어 캐시 적중률도 높아 진다.

인 라인 함수는 컴파일러에 ‘요청’하는 것이지 ‘명령’하는 것이 아니다.

인 라인 함수를 사용하는 데에는 명시적, 암시적 방법이 존재합니다.

1. **암시적 방법**

**클래스 정의 안에 함수를 바로 정의해 넣으면 컴파일러는 그 함수를 인 라인 함수 후보로 찍습니다.**

|  |
| --- |
| class Person {  public:  int age() const {return theAge;} // 암시적인 인 라인 요청: age는 클래스 정의 내부에서 정의되었다.  private:  int theAge;  }; |

이런 경우에는 암시적으로 인 라인 함수로 선언 됩니다.

1. **명시적인 방법**

인 라인 함수를 선언하는 명시적인 방법은 **정의 앞에 inline 키워드를 붙이는 것**

한 예로, 표준 라이브러리의 max 템플릿은 대개 다음과 같이 구현되어 있다.

|  |
| --- |
| template<typename T>  inline const T& std::max(const T& a, const T& b) // 명시적인 인라인 요청, std::max 앞에 inline이 존재  { return a < b ? b : a; } |

max가 템플릿 이라는 점 때문에 ‘**인라인 함수와 템플릿은 대개 헤더 파일안에 정의한다**.’라는 이야기가 생각난다. 이점을 오해 해서 함수 템플릿은 반드시 인라인 함수이어야 한다고 생각할 수 있다. 말하자면 이런 것이다.

인 라인 함수는 대체적으로 헤더 파일에 들어 있어야 하는 게 맞다. 왜냐하면 대부분의 경우 인 라인을 컴파일 도중에 수행하기 때문에 인 라인 함수 호출을 그 함수의 본문으로 바꿔 치기 하려면, 일단 컴파일러는 그 함수가 어떤 형태인지 알고 있어야 하기 때문이다.

템플릿 역시, 대체적으로 헤더 파일에 있어야 하는 것이 맞다. 템플릿이 사용되는 부분에서 해당 템플릿을 인스턴스로 만들려면 그것이 어떻게 생겼는지 컴파일러가 알아야 하기 때문이다.

그러나 템플릿 인스턴스화는 인라인과 완전 별개로, 관련이 없다. 어떤 템플릿에 그 템플릿에 inline을 붙여 선언하는 것이고, 그게 끝이다. 함수 템플릿이 궂이 인라인 될 이유가 없다면 템플릿을 인라인 함수로 선언하지 않아도 된다.

“**inline은 컴파일러 선에서 무시할 수 있는 요청이다**.” 대부분의 컴파일러의 경우, 아무리 인라인 함수로 선언되어 있어도 컴파일러가 보기에 복잡한 함수는 절대로 인라인 확장의 대상이 되지 않습니다. 또, 정말 **간단한 함수라고 가상 함수 호출 같은 것은 인라인화 해주지 않습니다**.

virtual의 의미가 “**어떤 함수를 호출할지 결정하는 작업을 런타임중에 한다.”** 인데, inline의 의미는 “함수 호출 위치에 호출된 함수를 끼워 넣는 작업을 프로그램 실행 전에 한다.”라는 뜻이므로 주어진 시점에 호출할 함수가 무엇이 될지 컴파일러가 알 수 없다면 어쩔 수 없는 것이다.

**인라인 함수가 실제 인라인이 되느냐 하는 것은 전적으로 컴파일러의 몫이다.**

완벽한 인라인 조건을 갖춘 코드라 하여도 컴파일러가 인라인 함수의 본문에 대해 코드를 만드는 경우가 있다. 어떤 인라인 함수의 주소를 취하는 코드가 있으면, 컴파일러는 이 코드를 위해 아웃라인 함수 본문을 만들 수밖에 없을 것이다. 있지도 않은 함수에 대해 어떻게 포인터를 가지고 오겠는가?

|  |
| --- |
| inline void f() {…} // 이 f의 호출은 컴파일러가 반드시 인라인해 준다고 가정  void (\*pf)() = f; // pf는 f를 가리키는 함수 포인터 이다.  f(); // 이 호출은 인라인될 것입니다. 평범한 함수 호출 이기 때문에..  pf(); // 이 호출은 인라인 되지 않을 것이다. 함수 포인터를 통해 호출되고 있기 때문에 |

인라인 되지 않는 인라인 함수는 여러분이 함수 포인터를 전혀 사용하지 않아도 여러분 주변을 떠돌며 괴롭힐 것이다.

또, 생성자와 소멸자는 인라인 하기에 그리 좋지 않은 함수이다. 아래의 Derived클래스에 들어 있는 생성자

|  |
| --- |
| class Base  {  public:  private:  std::string bm1, bm2; // Base 클래스의 멤버1, 멤버2  };  class Derived: public Base  {  public:  Derived() {} // Derived의 생성자가 비어 있다. 정말 비어 있을까?  private:  std::string dm1, dm2, dm3; // Derived의 멤버1, 멤버2, 멤버3  } |

Derived의 생성자는 비어있어 인라인 하기 딱 좋아 보이나 그렇지 않다.

C++는 객체가 생성되고 소멸될 때 일어나는 일들에 대해 여러 가지 작업을 한다.

예를 들면, new를 하면 동적으로 만들어지는 객체를 생성자가 자동으로 초기화하며, delete를 하면 이에 대응되는 소멸자가 호출되는 것, 객체가 생성되는 도중에 예외가 던져지더라도, 이미 생성이 완료된 부분의 말끔한 소멸도 보장된다.

**C++은 무엇을 해야 하는지는 정해 두었지만 그것을 어떻게 해야 하는지는 정하지 않았다는 점.**

즉, 눈에 보이지 않지만 이런 일을 가능하게 하는 어떤 코드가 이미 포함되어 있다는 것이다.

실제 비어있다고 생각되던 Derived 생성자는 사실 어떤 구현환경에서는 다음과 같은 코드로 만들어 진다.

|  |
| --- |
| Derived::Derived()  {  Base::Base();  try{dm1.std::string::string();}  catch (…){  Base::~Base();  throw;  }  try{dm2.std::string::string();}  catch (…){  Base::~Base();  throw;  }  try{dm3.std::string::string();}  catch (…){  Base::~Base();  throw;  }  } |

실제 코드가 이렇게 생성되지는 않지만 Derived의 텅빈 생성자가 동작하는 잘 보여 주는 코드이다.

inline으로 선언할 때 그 영향에 대해 많은 고민을 해야 합니다. 사용자의 눈에 빤히 보이는 인라인 함수에 대해서는 라이브러리 차원에서 바이너리 업그레이드를 제공할 수 없기 때문이다.

어떤 함수를 인라인으로 선언해야 하고 또 어떤 것을 선언하지 말아야 하는지는 기본 전략이 존재.

우선, 아무것도 인라인하지 않는다. 아니면 꼭 인라인해야하는 가장 간단한 함수에 한해서만 인라인을 선언한다. 인라인을 주의해서 사용하는 버릇을 들여서, 디버깅하고 싶은 부분에서 디버거를 제대로 쓸 수 있도록 만들고, 정말 필요한 위치에 인라인 함수를 놓도록 하자.

**요약**

- 함수 인라인은 작고, 자주 호출되는 함수에 대해서만 하는 것으로 묶어 둡시다. 이렇게 하면 디버깅 및 라이브러리의 바이너리 업그레이드가 용이해지고, 자칫 생길 수 있는 코드 부풀림 현상이 최소화되며, 프로그램의 속력이 더 빨라질 수 있는 여지가 최고로 많아집니다.

- 함수 템플릿이 대개 헤더 파일에 들어간다는 일반적인 부분만 생각해서 이들을 inline으로 선언하면 안됩니다.