**항목 43. 템플릿으로 만들어진 기본 클래스 안의 이름에 접근하는 방법을 알아 두자.**

몇 개 회사에 메시지를 전송하는 응용프로그램을 만들어야 할 경우를 생각해 보자.

|  |
| --- |
| class CompanyA {  public:  void sendClear(const std::string msg);  void sendEncrypted(const std::string msg);  };  class CompanyB {  public:  void sendClear(const std::string msg);  void sendEncrypted(const std::string msg);  };  class MsgInfo { ... };  template<typename C>  class MsgSender {  void sendClear(const MsgInfo& info) {  std::string msg = info.toString();  C c;  c.senderClear(msg);  }  void sendEncrypted(const MsgInfo& info) { ... };  };  template<typename C>  class LoggingMsgSender : public MsgSender<C> {  void sendClear(const MsgInfo& info) {  // 메세지 보내기전 Log 기록  sendClear(info); // Compile 시 오류 발생  // 메세지 보낸 후 Log 기록  }  }; |

위의 코드는 컴파일 되지 않는다. LoggingMsgSender에서 오류를 발생시킨다. Template의 경우 Compile-time에 그 형태가 정해지는데, LoggingMsgSender<C> 입장에서 MsgSender<C>클래스가 어떤 형태인지 알 방법이 없다. sendClear() 함수가 들어 있는지 없는지도 모르는 것이 당연하다.

|  |
| --- |
| class CompanyZ {  public:  void sendEncrypted(const std::string msg);  };  template<> // CompanyZ 에 대해서만 특수화된 템플릿  class MsgSender<CompanyZ> {  void sendEncrypted(const MsgInfo& info) { ... };  }; |

Template<>라는 낯선 표현이 보인다. 이건 템플릿도 아니고 클래스도 아니다. 특정 Type에 대해서만 사용 가능한 템플릿을 **완전 템플릿 특수화(total template specialization)** 이라고 한다.

위의 MsgSender<CompanyZ>를 LoggingMsgSender로 보내면 어떻게 될까? 저긴 sendClear() 함수 자체가 없다. 이럴 가능성이 있기 때문에 **C++ 컴파일러는 템플릿으로 만들어진 기본 클래스를 뒤져서 상속된 이름을 찾는 것을 거부 한다.**

이 난국을 돌파하기 위해서는 어떻게든 C++이 “난 템플릿화된 기본 클래스를 멋대로 안 뒤질 꺼야”라고 생각하지 않도록 해줘야 한다. 방법에는 세가지 있다.

**첫째, 기본 클래스 함수에 대한 호출문 앞에 “this->”를 붙인다.**

|  |
| --- |
| template<typename C>  class LoggingMsgSender : public MsgSender<C> {  void sendClear(const MsgInfo& info) {  // 메세지 보내기전 Log 기록  this->sendClear(info);  // 메세지 보낸 후 Log 기록  }  }; |

**둘째, using 선언을 사용한다.**

|  |
| --- |
| template<typename C>  class LoggingMsgSender : public MsgSender<C> {  using MsgSender<C>::sendClear;  void sendClear(const MsgInfo& info) {  // 메세지 보내기전 Log 기록  sendClear(info);  // 메세지 보낸 후 Log 기록  }  }; |

**셋째, 명시적으로 지정한다.**

하지만 이 방법은 추천하고 싶지 않다. 만약 가상함수인 경우 명시적으로 지정하면 가상 함수 바인딩이 무시된다.

|  |
| --- |
| template<typename C>  class LoggingMsgSender : public MsgSender<C> {  void sendClear(const MsgInfo& info) {  // 메세지 보내기전 Log 기록  MsgSender<C>::sendClear(info);  // 메세지 보낸 후 Log 기록  }  }; |

위 세가지 방법의 동작 원리는 모두 같다. 기본 클래스 템플릿이 이후에 어떻게 특수화되더라도 원래의 일반형 템플릿에서 제공하는 인터페이스를 그대로 제공할 것이라고 C++에게 약속하는 것이다. 그 약속이 비리정치인 선거공약 같은 것이었다는 것이 들통 나면 이후의 컴파일 과정에서 겨레의 응징이 들어갈 것이다.

**요약**

- 파생클래스 템플릿에서 기본 클래스 이름을 참조할 떄는 “this->”를 접투사로 붙이거나 기본 클래스 한정문을 명시적으로 써주는 것으로 해결합시다.