

Chapter 11. 분리컴파일과 네임스페이스

박 종 혁 교수

UCS Lab

(<http://www.parkjonghyuk.net>)

Tel: 970-6702

Email: jhpark1@snut.ac.kr

분리 컴파일

- 프로그램 부분
 - 분리된 파일에 저장
 - 분리되어 컴파일
 - 프로그램이 실행되기 전에 상호간에 링크됨
- 클래스 정의
 - 사용 프로그램에서 분리됨
 - 클래스 라이브러리 생성
 - 많은 다른 프로그램에 재사용 됨
 - 사전정의 라이브러리와 유사

클래스 분리

- 클래스 독립성
 - 클래스 정의 분리
 - 인터페이스
 - 클래스 구현 분리
 - 두 개의 파일에 존재
- 구현 부분의 변경 → 해당 파일만 변경
 - 클래스 정의부의 변경은 필요 없음
 - 사용(응용) 프로그램의 변경도 필요 없음

캡슐화 리뷰

- 캡슐화 원칙:
 - “(프로그래머에게) 어떻게 사용되는 가”와 “구현의 세부사항”을 분리
- 완전한 분리
 - 구현 부분을 수정해도 → 다른 프로그램에는 영향이 없음
- OOP 기본 원칙

캡슐화 규칙

- 분리 규칙:
 1. 모든 멤버 변수는 `private`
 2. 클래스의 기본 함수:
 - `public` 멤버 함수
 - 프렌드 또는 일반 함수
 - 오버로딩 된 함수클래스 정의와 함수원형의 그룹화
 - 클래스 인터페이스
 3. 클래스 사용자에게 클래스 구현을 감춤

클래스 분리

- 인터페이스 파일
 - 함수의 원형을 포함하는 클래스의 정의 부분
 - 사용자는 인터페이스 파일을 보고 프로그래밍
 - 분리된 컴파일 단위
- 구현 파일
 - 멤버 함수 정의 부분
 - 분리된 컴파일 단위

클래스 헤더 파일

- 클래스 인터페이스는 헤더 파일에 위치
 - *.h
- 클래스를 사용하는 프로그램은 헤더파일을 include
 - #include "myclass.h"
 - 사용자 정의 파일
 - 작업 디렉토리
 - 라이브러리 includes → <iostream>
 - <> → 사전 정의 라이브러리 헤더파일 지정
 - 라이브러리 디렉토리

클래스 구현 파일

- 클래스 구현 부분은 *.cpp 파일에 존재
 - 일반적으로 인터페이스 파일과 구현 파일은 같은 이름을 사용
 - myclass.h, myclass.cpp
 - 클래스의 모든 멤버 함수의 정의
 - 구현파일은 반듯이 클래스 헤더 파일을 include 해야 함
- *.cpp 파일에는 일반적으로 실행 코드가 포함 됨
 - 예) 함수 정의, main() 함수

클래스 파일

- 클래스 헤더 파일은 다음의 파일에 의해 #included 됨:
 - 구현 파일
 - 프로그램 파일
 - 응용 파일 또는 드라이버 파일이라고도 함
- 파일의 구성은 시스템 의존적
 - 일반적으로 IDE는 project나 workspace를 가짐
 - 구현 파일과 헤더 파일의 위치는 IDE에 따라 다름

헤더 파일의 중복 컴파일

- 헤더 파일
 - 일반적으로 여러 번 `include` 됨
 - 클래스 인터페이스는 클래스 구현 파일과 프로그램 파일에서 `include` 됨
 - 한번 만 컴파일 되게 해야 함!
 - 어떠한 곳에서 먼저 컴파일 될 지 장담 할 수 없음
- 선행처리기(preprocessor) 사용
 - 컴파일러에게 헤더 파일을 한번 만 `include` 하라고 알려줘야 함

#ifndef 사용

- 헤더 파일 구조:
 - #ifndef FNAME_H
#define FNAME_H
... //Contents of header file
...
#endif
- FNAME은 일관성과 가독성을 위해 일반적으로 파일이름을 사용
- 위의 구문은 헤더 파일의 중복 정의를 방지

기타 라이브러리 파일

- 라이브러리에 클래스 많이 있는 것은 아님
- 관련 함수
 - 원형 → 헤더 파일
 - 정의 → 구현 파일
- 기타 자료형 정의
 - 구조체, typedef → 헤더 파일
 - 상수 선언 → 헤더 파일

네임스페이스

- 네임스페이스 정의:
네임 정의의 집합
 - 클래스 정의
 - 변수 선언
- 프로그램에서 많은 클래스와 함수를 사용
 - 일반적으로 중복된 이름이 존재
 - 네임스페이스는 이러한 충돌을 방지
 - On/Off 가능
 - 충돌 → Off

using 지시자

- using namespace std;
 - std 네임스페이스에 있는 모든 정의의 사용 가능
- 사용자가 std 네임스페이스를 사용 하지 않은 다면?
 - cout, cin 등을 다른 의미로 사용 가능
 - cout, cin 의 재정의 필요
 - 나머지도 재정의 가능

Namespace std

- 많은 표준 라이브러리 파일에 정의된 모든 네임을 포함
- 예:
`#include <iostream>`
 - cin, cout, etc.
 - 프로그램은 네임을 모름
 - 프로그램에서 이러한 네임으로 접근하기 위해서는 `std` 네임스페이스를 명시해야 함

전역 네임스페이스

- 모든 코드는 특정한 네임스페이스에 속함
- 명시하지 않으면 → 전역 네임스페이스
 - using 지시자가 필요 없음
 - 전역 네임스페이스는 항상 사용가능
 - 묵시적으로 using 지시자 사용

중복된 네임

- 복수의 네임스페이스 사용
 - 전역과 std가 일반적으로 사용
- 네임 정의가 양쪽에 모두 존재할 경우
 - 에러
 - 사용시 어떠한 네임스페이스에 위치하는지 명시해야 함

네임스페이스 명시

- 두 개의 네임스페이스 NS1, NS2
 - 두 개 모두 `void myFunction()`를 가질 경우

```
{
    using namespace NS1;
    myFunction();
}
{
    using namespace NS2;
    myFunction();
}
```
 - `using` 지시자는 블록 영역을 가짐

네임스페이스 생성

- 네임스페이스 그룹화:
`namespace Name_Space_Name`
`{`
 `Some_Code`
`}`
- 블록 내에 정의된 모든 정의는 `Name_Space_Name` 소속
- 사용:
`using namespace Name_Space_Name;`

네임스페이스 생성 예제

- 함수 선언:

```
namespace Space1
{
    void greeting();
}
```

- 함수 정의:

```
namespace Space1
{
    void greeting()
    {
        cout << "Hello from namespace Space1.\n";
    }
}
```

using 선언

- 사용 가능한 네임스페이스에서 유일한 네임을 생성
- 다음의 예:
네임스페이스 NS1, NS2
각각은 함수 fun1(), fun2() 을 가짐
 - 선언 구문:
`using Name_Space::One_Name;`
 - 각각의 네임이 어떠한 네임스페이스 소속인지 명시:
`using NS1::fun1;`
`using NS2::fun2;`

using 선언과 지시자 사용

- 차이점:
 - using 선언
 - 네임스페이스 내에 하나의 네임만 사용
 - 같은 네임스페이스 내의 다른 네임을 사용 할 수 없음
 - using 지시자
 - 네임스페이스 내의 모든 네임의 사용 가능
 - 사용할 수 있는 잠재적인 네임들을 알림

이름 제한 (Qualifying Name)

- 네임이 어느 네임스페이스 소속인지 명시 가능
 - :: 사용
 - 단 한번 또는 적은 사용
- `NS1::fun1();`
 - 함수 `fun()`는 네임스페이스 `NS1` 소속임을 명시
- 매개 변수에 유용:
`int getInput(std::istream inputStream);`
 - 매개변수는 `std` 네임스페이스 `istream`
 - 코드에 `using` 지시자와 `using` 선언을 제거 할 수 있음

네임스페이스 명명

- 유일한 문자열 포함
 - 예) 자신의 성
- 다른 네임스페이스와 중복을 줄여 줌
- 종종 같은 프로그램을 위해 복수의 프로그래머가 네임스페이스를 정의
 - 네임스페이스명은 구분되어야 함
 - 고려하지 않으면 → 에러 발생

디스플레이 11.6 네임스페이스 안에서 클래스 정의하기 (헤더파일)

Display 11.6 Placing a Class in a Namespace (Header File)

```
1  //This is the header file dtime.h.  
2  #ifndef DTIME_H  
3  #define DTIME_H  
  
4  #include <iostream>  
5  using std::istream;  
6  using std::ostream;  
  
7  namespace DTimeSavitch  
8  {  
9  
10     class DigitalTime  
11     {  
12  
13         <The definition of the class DigitalTime is the same as in Display 11.1.>  
14     };  
15  
16 }// DTimeSavitch  
  
17 #endif //DTIME_H
```

A better version of this class definition will be given in Displays 11.8 and 11.9.


Note that the namespace DTimeSavitch spans two files. The other is shown in Display 11.7.

디스플레이 11.7 네임스페이스 안에서 클래스 정의하기 (구현파일)

Display 11.7 Placing a Class in a Namespace (Implementation File)

```
1 //This is the implementation file dtime.cpp.
2 #include <iostream>
3 #include <cctype>
4 #include <cstdlib>
5 using std::istream;
6 using std::ostream;
7 using std::cout;
8 using std::cin;
9 #include "dtime.h"
```

*You can use the single **using** directive **using namespace std;** in place of these four **using** declarations. However, the four **using** declarations are a preferable style.*



```
10 namespace DTimeSavitch
11 {
12     <All the function definitions from Display 11.2 go here.>
13
14
15 }// DTimeSavitch
```

명명되지 않은 네임스페이스

- 컴파일 단위 정의:
 - 파일 단위로
 - 예) 구현파일, 헤더파일
- 모든 컴파일 단위는 명명되지 않은 네임스페이스의 사용 가능
 - 같은 방식으로 사용, 하지만 네임이 없음
 - 컴파일 단위에서만 사용 가능
- 명명되지 않은 네임스페이스의 영역은 컴파일 단위
- helping function 등의 감춤에 사용

전역 네임스페이스 vs. 명명되지 않은 네임스페이스

- 같지 않음
- 전역 네임스페이스:
 - 네임스페이스 그룹화를 하지 않음
 - 전역 영역
- 명명되지 않은 네임스페이스:
 - 네임스페이스 단위를 네임 없이 그룹화
 - 지역 영역

내포된 네임스페이스

- 구문

```
namespace S1
{
    namespace S2
    {
        void sample()
        {
            ...
        }
    }
}
```

- 네임스페이스를 모두 명시:
 - S1::S2::sample();

도움 함수 감추기

- helping function:
 - 저 수준 도구
 - public 섹션에 위치 시키지 않음
- 두 가지 방법:
 - private 멤버 함수
 - 호출 객체 사용시
 - 명명되지 않은 네임스페이스에 위치!
 - 함수가 호출 객체를 사용하지 않을 경우
 - 코드를 명확하게 함 (디스플레이 11.2)

요약 1

- 클래스 정의와 구현부의 분리 가능 → 분리된 파일
 - 분리 컴파일 단위
- 네임스페이스는 클래스 정의 및 변수 선언들과 같은 이름 정의의 집합
- 네임스페이스에서 이름을 사용하는 방법:
 - Using 지시어
 - Using 선언
 - 이름 제한 (qualifying name)

요약 2

- 네임스페이스 정의는 네임스페이스 그룹 내에 존재
- 명명되지 않은 네임스페이스
 - 지역 네임 정의에 사용
 - 컴파일 단위의 영역을 가짐
- 전역 네임스페이스
 - 특정한 네임스페이스에 그룹화 되지 않았을 경우
 - 전역 영역

Q&A