C 프로그래밍 프로젝트

Chap 27. 파일의 분할과 헤더파일의 디자인

2013.09.11.

오 병 우

컴퓨터공학과

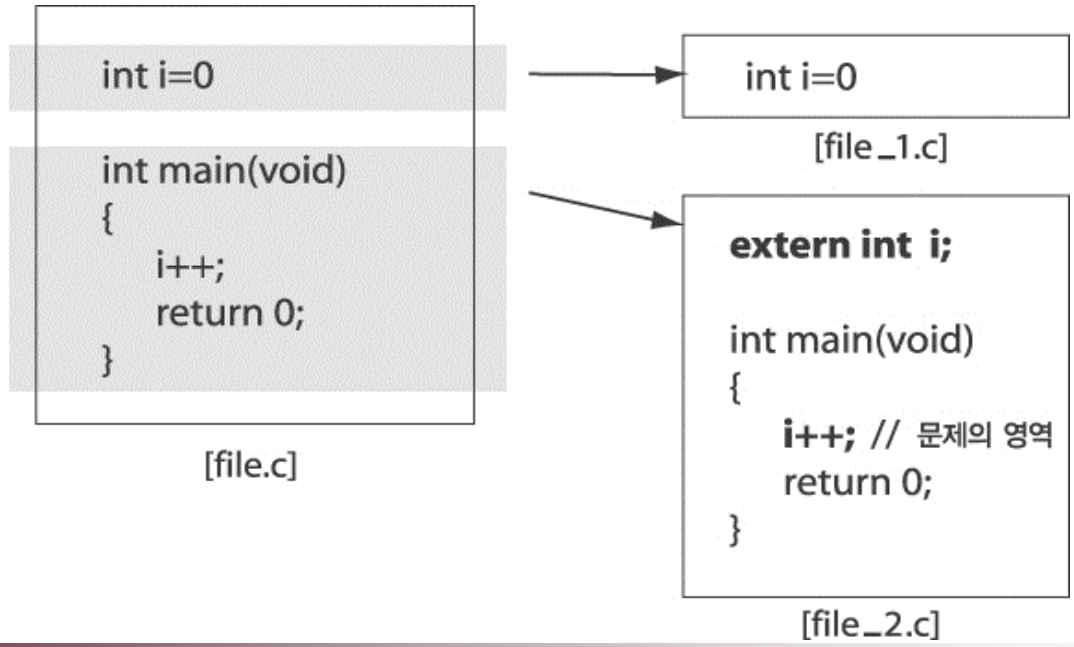
kit 국립금오공과대학교
Kumoh National Institute of Technology

27-1 프로그램의 모듈화

- 설계 (design) 중요
  - 변경, 확장 등의 유지 보수가 용이하도록 설계
  - C 언어에서는 module 구성 중요
  - C++, Java 등의 객체 지향 언어에서는 class, abstraction 중요
    - Design Patterns에 대해 2학년 여름방학이나 겨울방학에 공부해 보시기 바랍니다.
- 모듈(module)이란 무엇인가?
  - 프로그램을 구성하는 구성 요소의 일부
  - 관련된 데이터와 함수들이 묶여서 모듈을 형성한다.
  - 파일 단위로 나뉘는 것이 보통
- 모듈화 (modular) 프로그래밍
  - 기능별로 파일을 나눠가며 프로그래밍하는 것
  - 유지 보수성이 좋아진다.

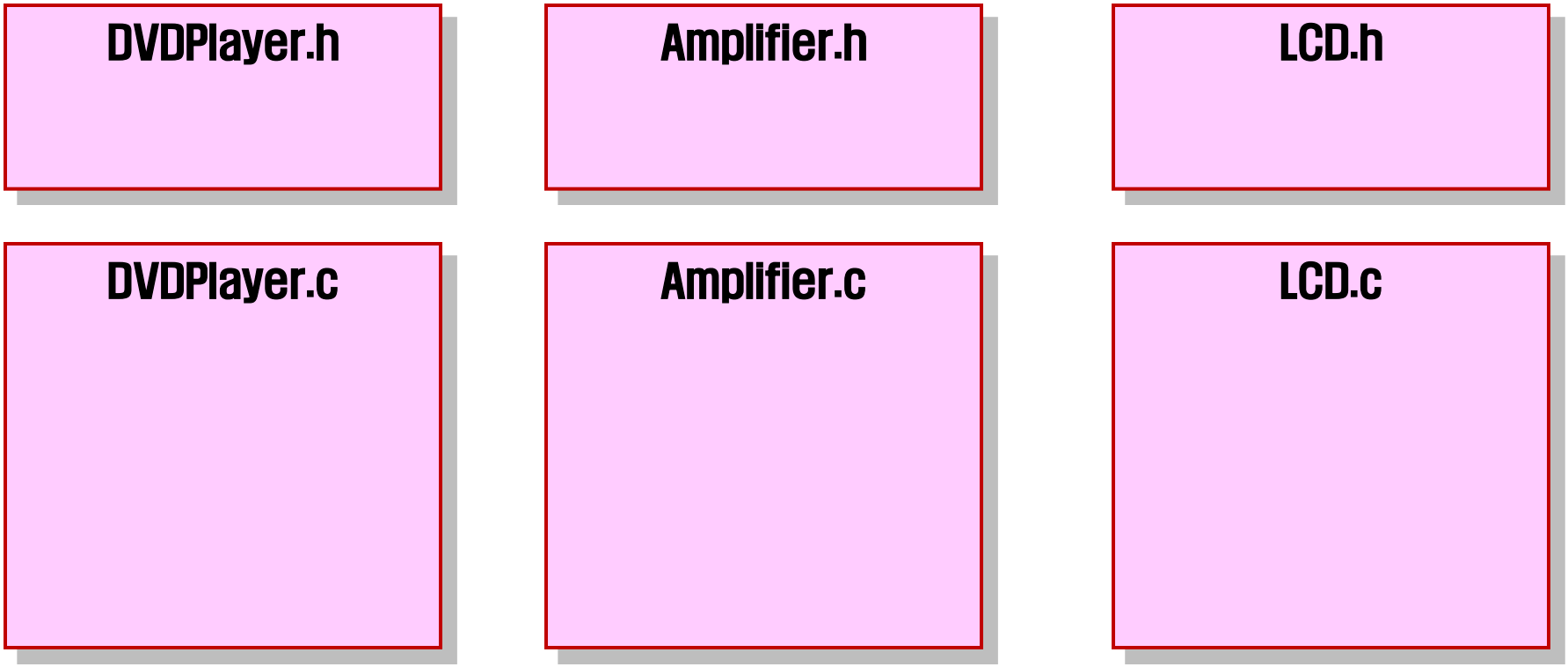
27-1 프로그램의 모듈화

- 파일의 분할 및 컴파일
  - 파일을 나눌지라도 완전히 독립되는 것은 아니다.
  - 파일이 나뉘어도 상호 참조가 발생할 수 있는데, 이는 전역 변수 및 전역 함수로 제한된다.
  - 외부 파일의 변수/함수 참조 시 extern 선언



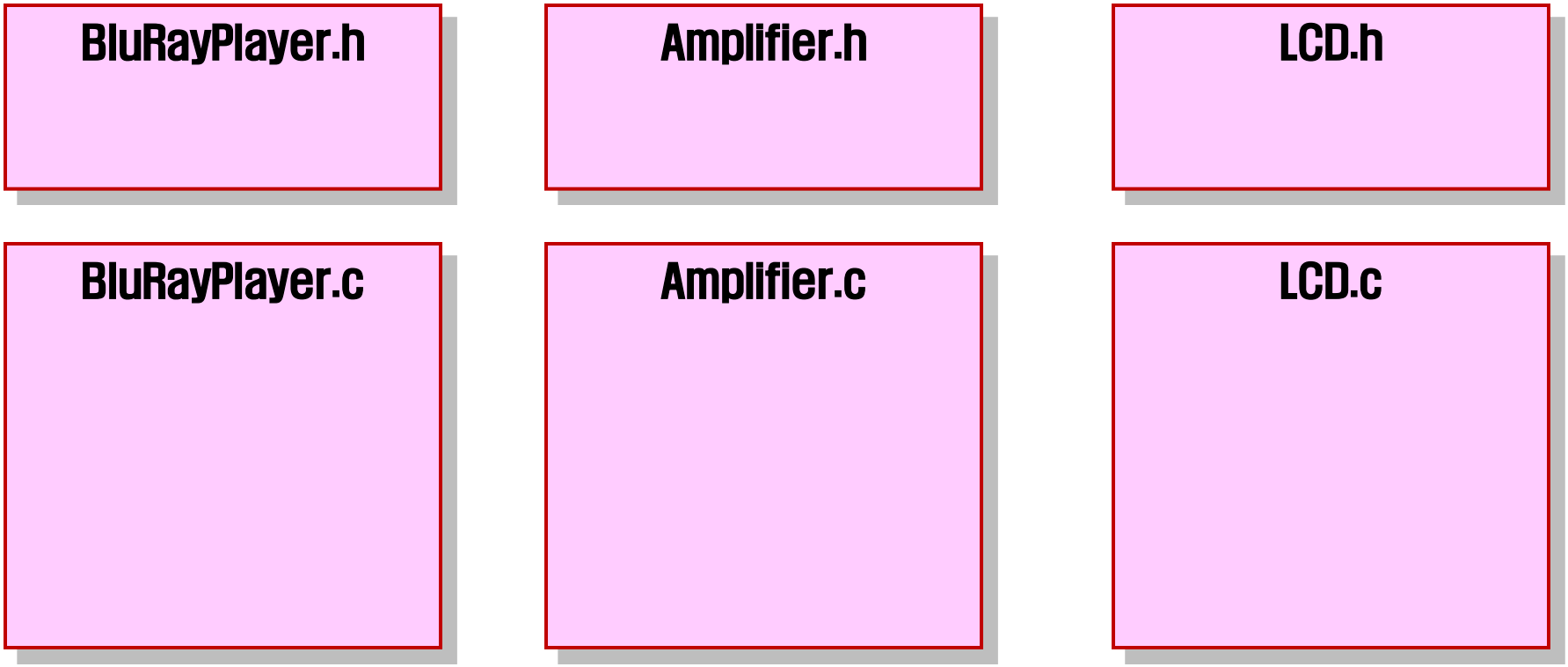
모듈 구성 예제

- Home theater



모듈 구성 예제

- Home theater
  - DVD Player를 Blu-ray Player로 업그레이드



모듈 구성 예제

- Home theater
  - 42” LCD 모니터를 100” Projector로 업그레이드

다른 부분과
독립적으로
변경할 수 있도록
설계할 것

BluRayPlayer.h	Amplifier.h	Projector.h
BluRayPlayer.c	Amplifier.c	Projector.c

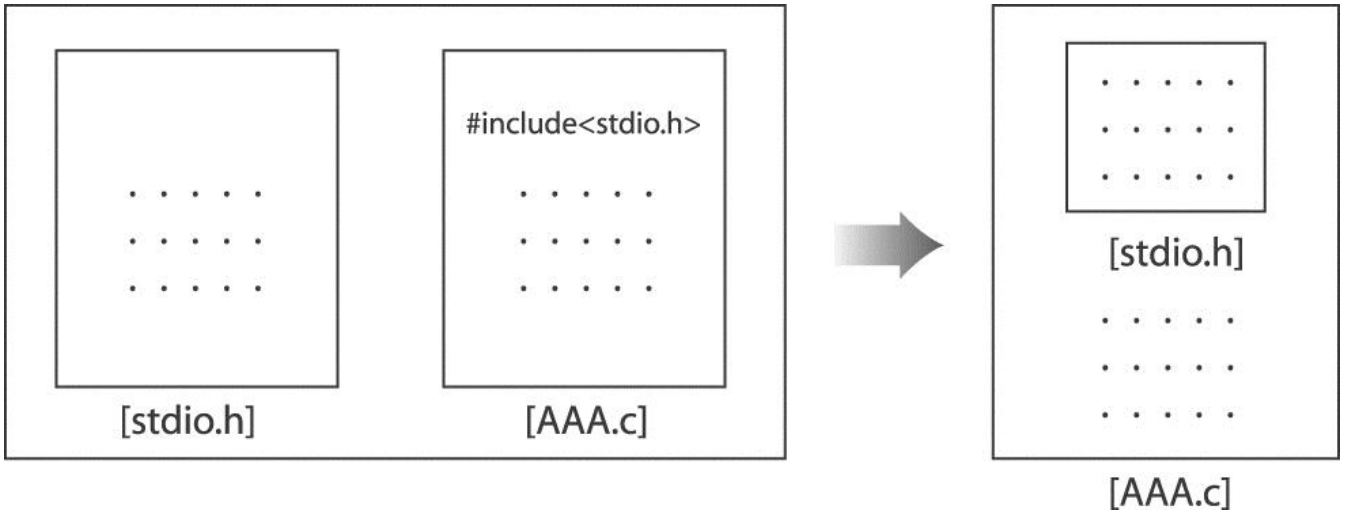
27-1 프로그램의 모듈화

- 파일의 분할 및 컴파일
  - 컴파일: 변경 사항이 있는 파일만 별도 컴파일
  - 링크: 실행 파일을 만들기 위해 모든 object 파일을 모아서 링크
    - extern 선언된 함수나 변수를 찾아서 연결

- 외부 접근 금지
  - static 키워드에 의한 접근의 제한

27-2 헤더 파일의 구현과 유용성

- 헤더 파일(header file)의 포함(include)이 지니는 의미
  - 선행처리기에 의해 하나의 파일을 다른 하나의 파일에 포함시키는 작업



27-2 헤더 파일의 구현과 유용성

- 헤더 파일 포함 방법

옵션에서 include 디렉토리 추가할 수 있음

  - <file> : 표준 디렉토리 파일 포함
    - Linux: /usr/include
    - Visual C: …\Microsoft Visual Studio\VC98\Include
  - "file" : 현재 디렉토리 또는 지정된 경로에 존재하는 파일 포함

/ (slash) 추천, ₩ (back slash), ₩₩ 모두 사용 가능

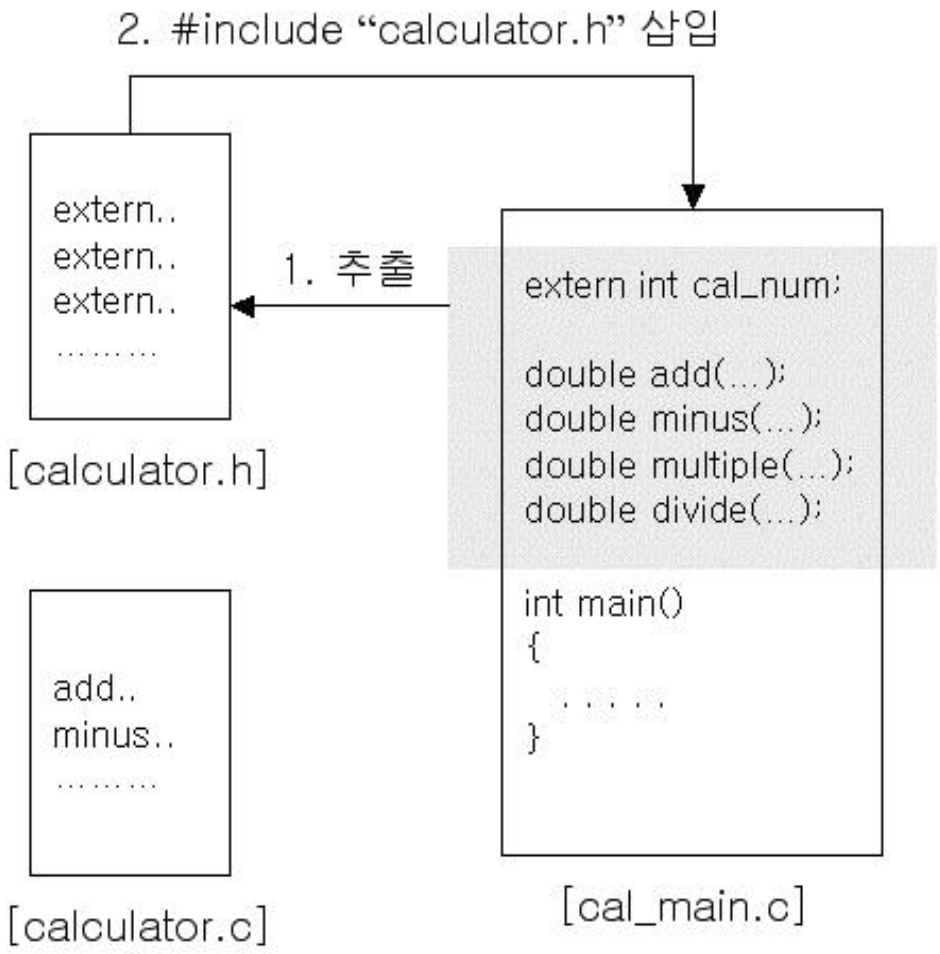
```
#include <abc.h>                 // 표준 디렉토리
#include "plus.h"                // 현재 작업 디렉토리에  있는 plus.h 포함
#include "c:/include/abc.h"      // c:\include에서 abc.h 포함
#include "..\plus.h"             // 현재의 상위 디렉토리에 있는 plus.h 포함
```

27-2 헤더 파일의 구현과 유용성

- 헤더 파일의 정의
  - 여러 소스 파일에서 공통으로 사용되는 선언의 집합
    - include
    - extern 변수 및 함수
    - 구조체 및 typedef (22, 23장)
- 왜 헤더 파일을 사용하는가?
  - 선언의 간략화
  - 파일 변경 최소화!
    - 함수를 사용하려는 .c 파일이 많을 때, 함수의 인자가 변경되면 모든 .c 파일을 수정해야 함
    - Header file를 사용하면 .h만 수정하면 됨 (.c에서는 .h를 include)
- 주의
  - 중복 선언이 되지 않도록 한다!
    - 조건부 컴파일로 해결 (#ifdef, #ifndef, #else, #endif)

26장 참조

27-2 헤더 파일의 구현과 유용성



27-2 헤더 파일의 구현과 유용성

calc.h

```
extern int cal_num;
double add (double a, double b);
double minus (double a, double b);
double multiple (double a, double b);
double divide (double a, double b);
```

calc.c

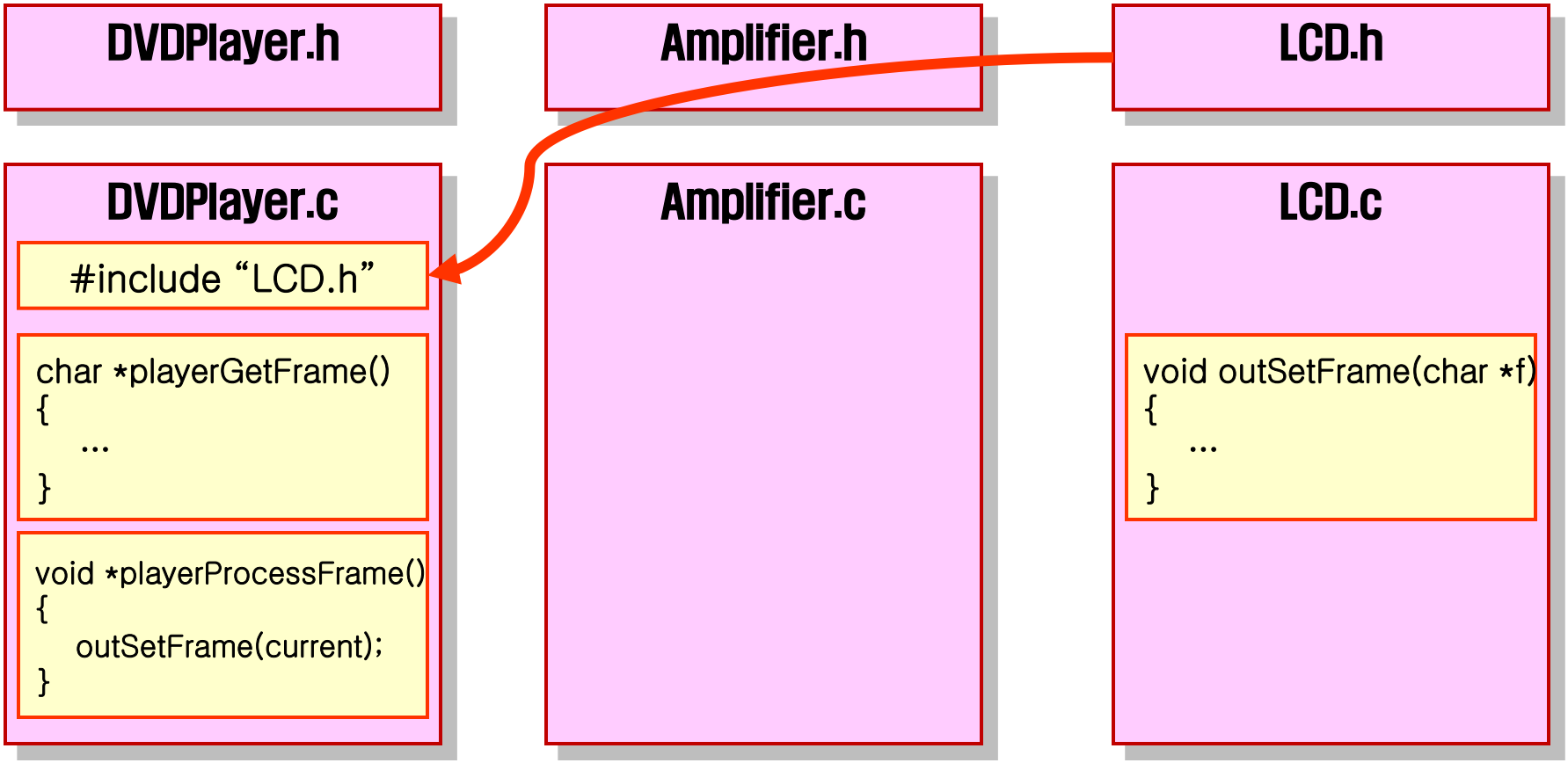
```
int cal_num = 0;

double add (double a, double b)
{
    cal_num++;
    return a+b;
}
double multiple (double a, double b)
…
```

cal_main.c

```
#include <stdio.h>
#include “calc.h”

int main (void)
{
    double a = 10.2;
    double b = 2.1;

    printf (“덧 셈: %f\n”, add(a, b));
    …
    printf (“총 연산 수: %d\n”, cal_num);
    return 0;
}
```

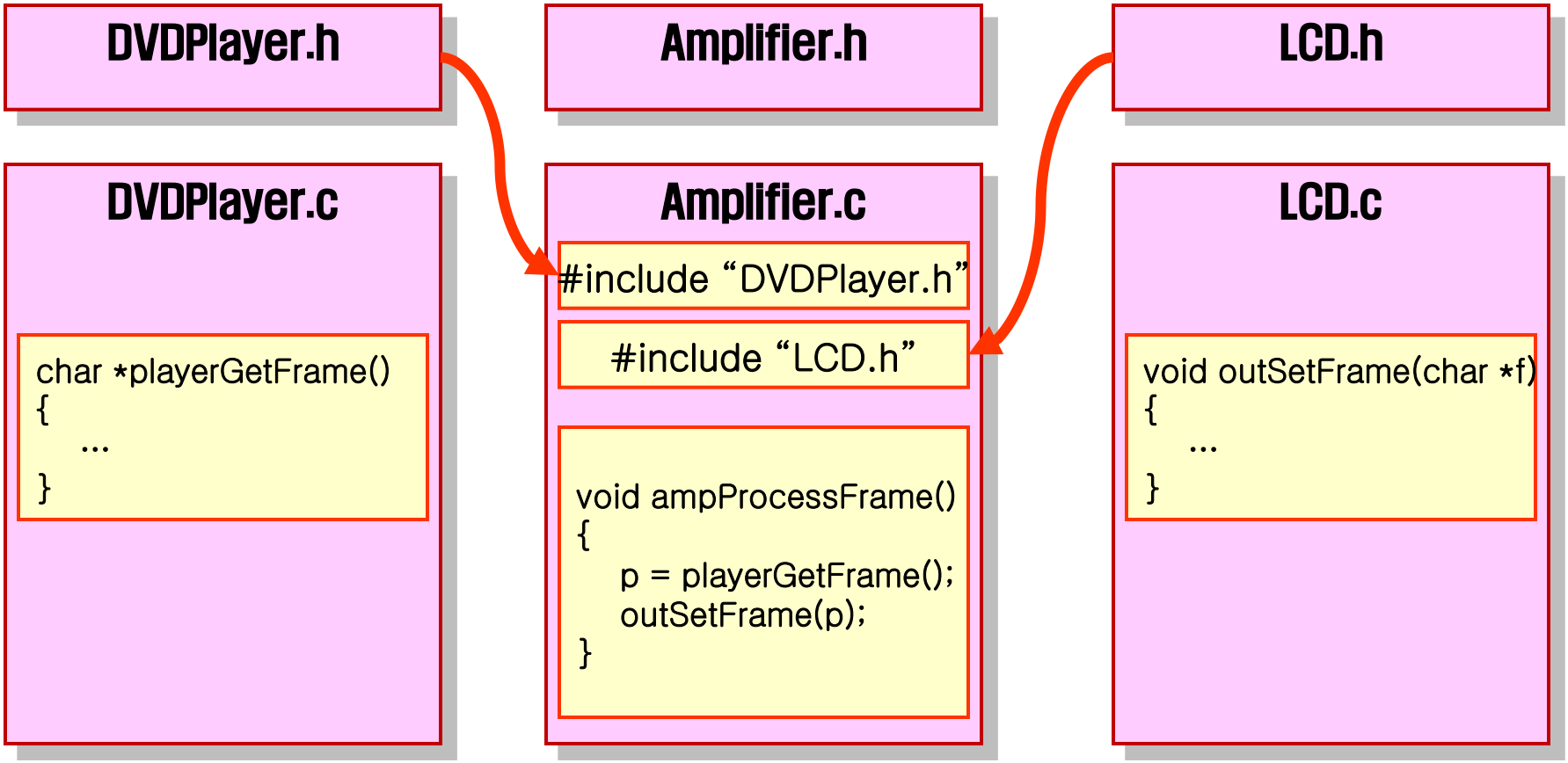
헤더 파일 사용 예제

- DVD에서 Amplifier를 거치지 않고 바로 LCD로 출력



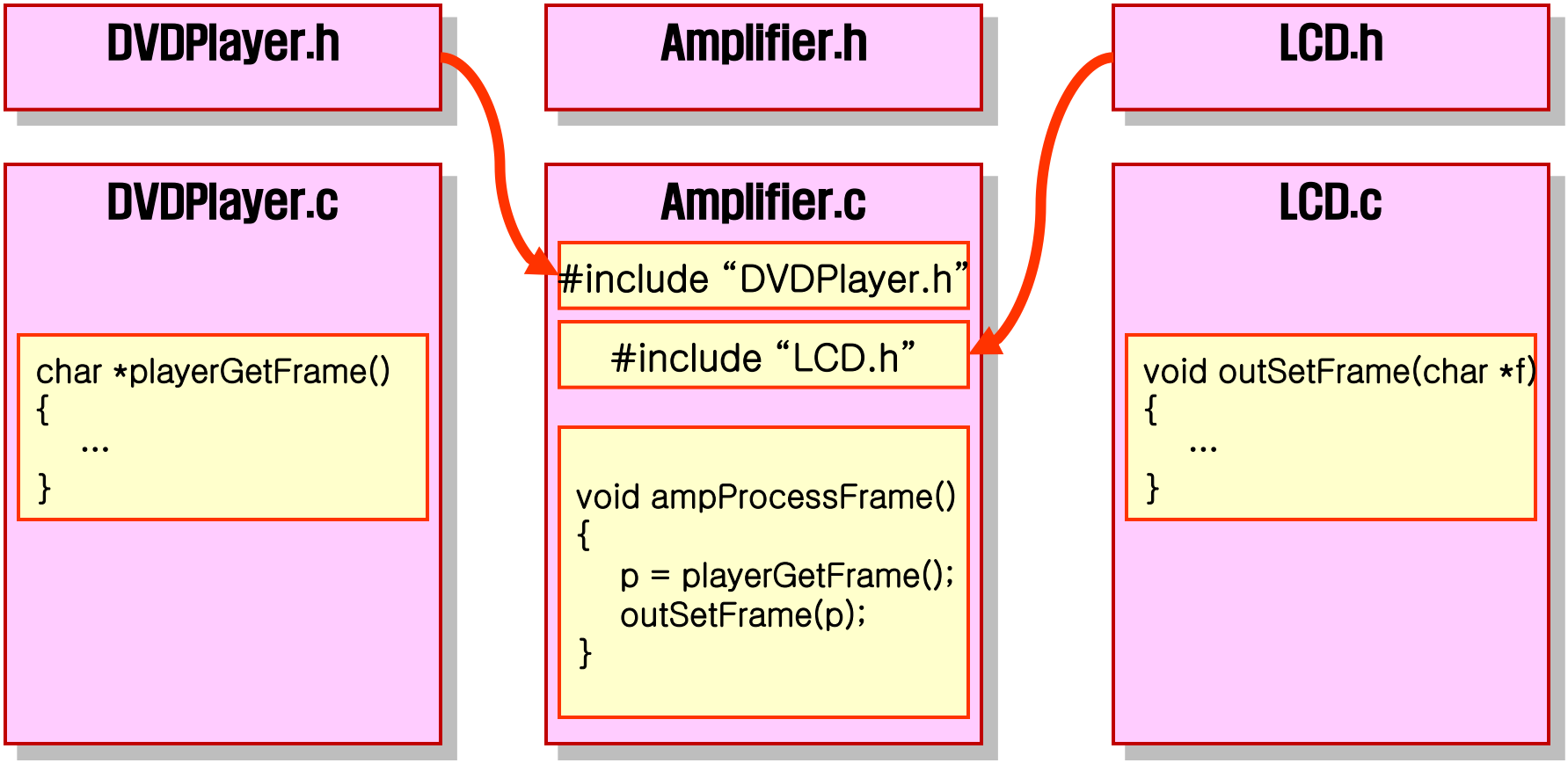
헤더 파일 사용 예제

- DVD-Amplifier-LCD 순으로 화면 출력



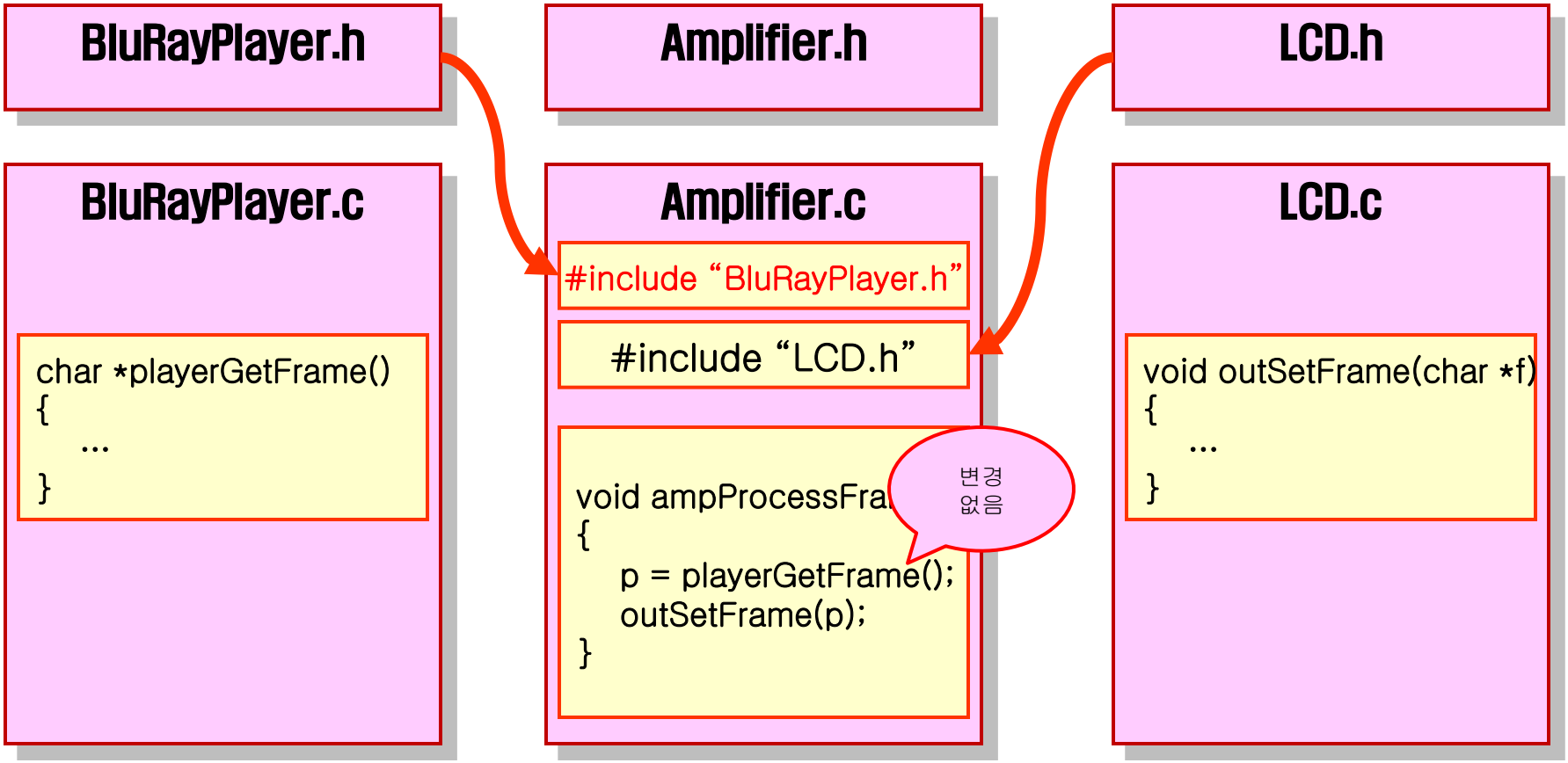
모듈별 헤더 파일 사용 예제

- DVD Player



모듈별 헤더 파일 사용 예제

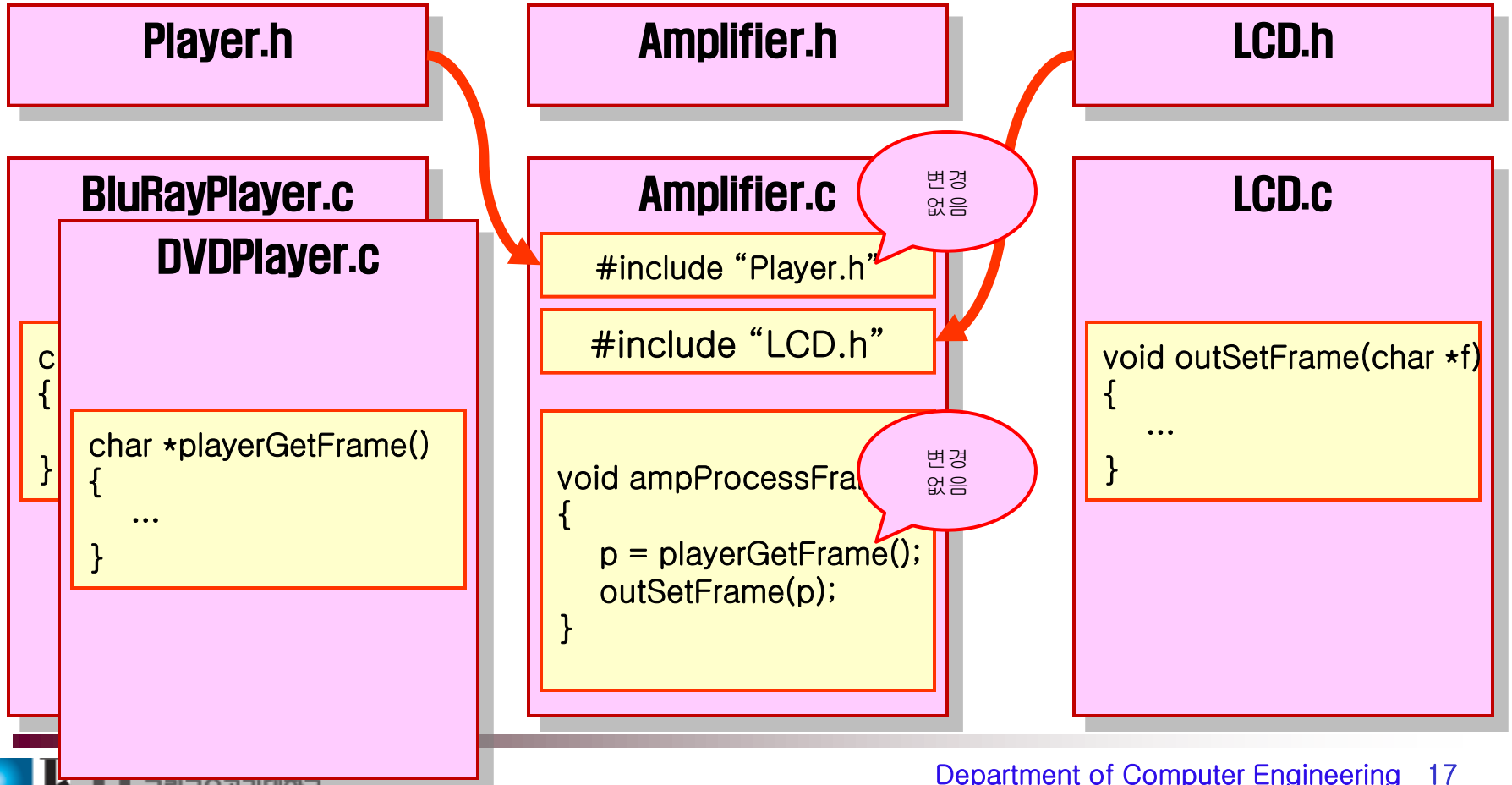
- Blu-ray Player로 업그레이드
  - Amplifier 모듈에서 include 변경



공통 헤더 파일 사용 예제

프로젝트에 어떤 .c 파일을 넣을지에 의해 결정

- 공통 인터페이스 (Interface)
  - DVD Player 모듈과 Blu-ray Player 모듈에서 공통적으로 쓰는 인터페이스를 Player.h에 저장하고 있다면 include를 변경하는 것도 필요 없음



27-3 조건부 컴파일

- #if, #elif, #else, #endif 기반 조건부 컴파일
  - 컴파일하기 전에 실제 컴파일할 소스 부분을 지정

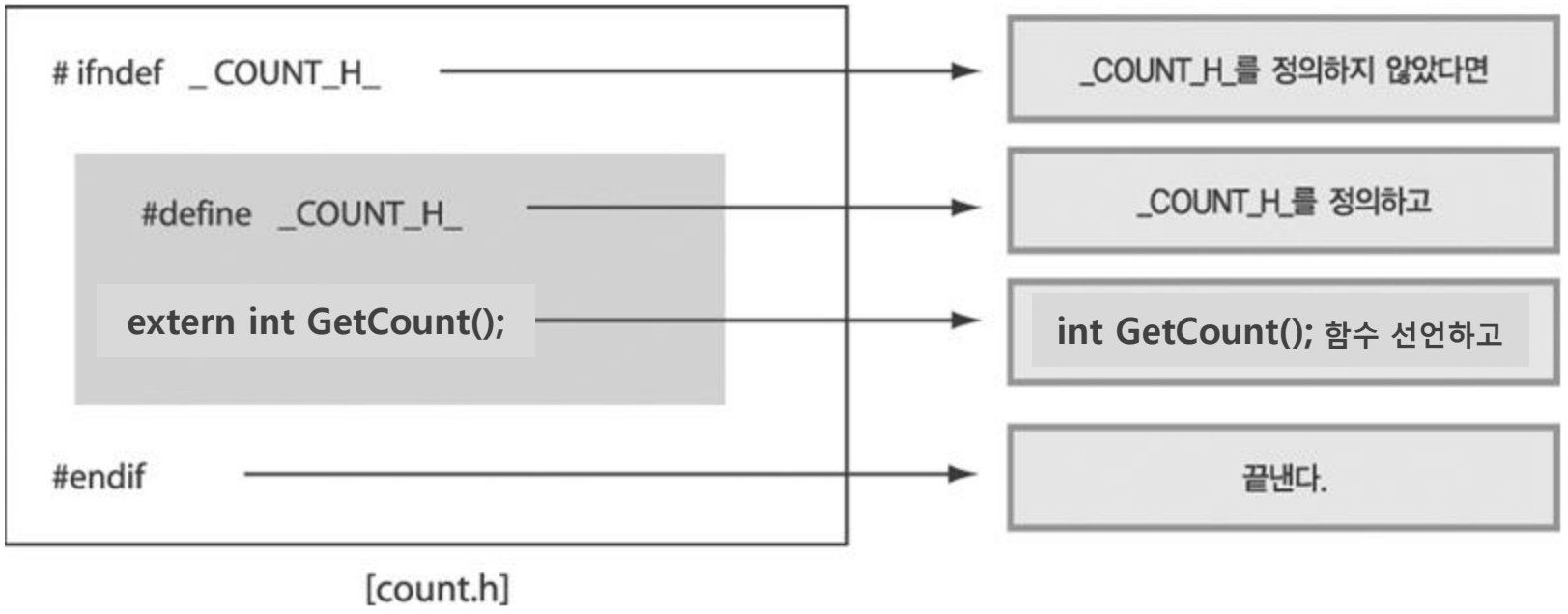
```
#if     CONDITION1
            expression1
#elif   CONDITION2
            expression2
#else
            expression3
#endif
```

27-3 조건부 컴파일

- 헤더 파일 포함 관계에서 발생하는 문제
  - 중첩된 관계로 인해 하나의 헤더 파일을 두 번 이상 포함하면...
  - 중복해서 함수가 정의되거나, 변수가 선언되는 문제점
    - 중복을 피해 헤더 파일을 만들기는 생각보다 어렵다.
  - 조건부 컴파일로 문제 해결!

27-3 조건부 컴파일

- #ifndef, #endif 기반 조건부 컴파일



기본적인 모듈 분리

정답이 있는 것은 아님

MVC 디자인 패턴 참조

- 크게 3 모듈로 분리하고 점차 세분화하여 설계할 것
  - App (Application)
    - main() 함수를 통해 전체적인 프로그램의 흐름 담당
  - Doc (Document)
    - 응용 프로그램에서 처리하는 데이터를 관리하고 business logic 처리
    - 기능별 모듈로 분리
  - View
    - 눈에 보이는 부분 처리
    - 입력 및 출력 (분리 가능)
- 추가 모듈
  - Err (Error)
    - 에러 처리 모듈
  - File 처리 모듈
    - 처리해야 하는 file 종류별로 모듈 구성

모듈 구성 예제

Doc.h

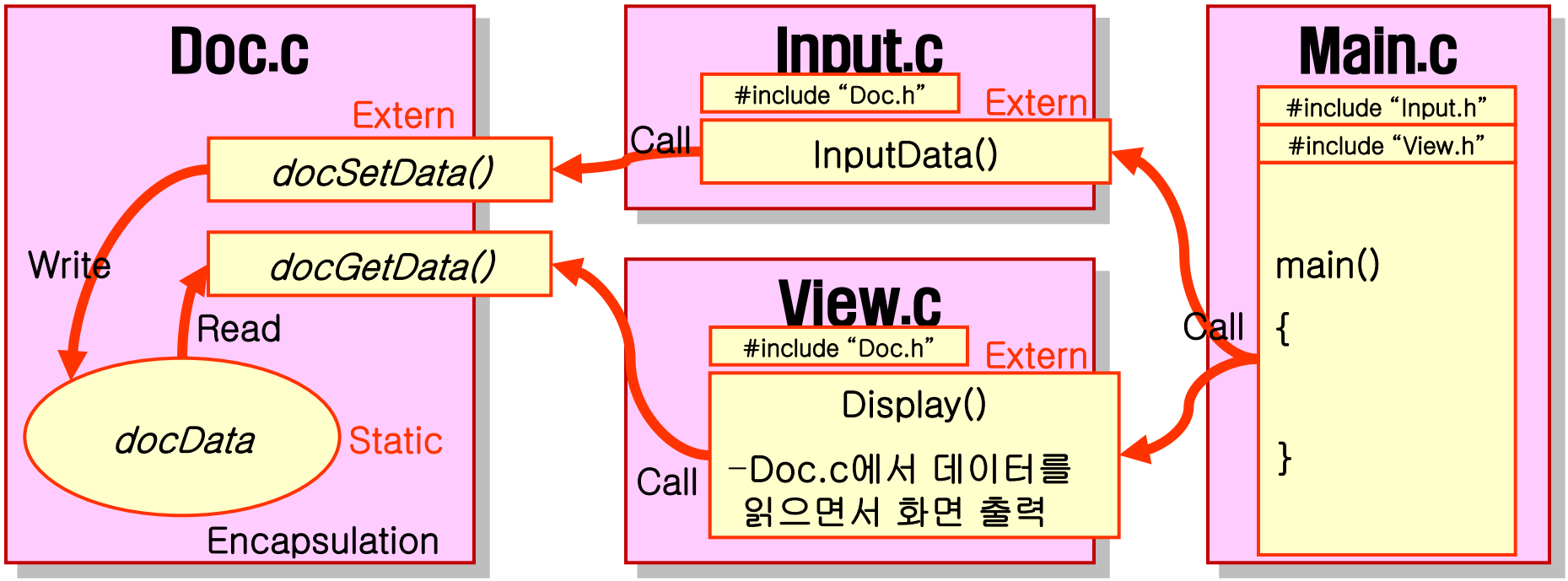
```
#ifndef _DOC_H_
#define _DOC_H_

// struct, typedef, enum 등 ---------

// public functions ---------------
// 모듈 초기화
extern void docInit();

// 모듈 종료 처리
extern void docClose();

// Getter functions
extern int docGetData();

// Setter functions
extern void docSetData(int);

#endif // _DOC_H_
```

Doc.c

```
#include "Doc.h"
// static variables ---------------
static int docData;
// public functions ---------------
// 모듈 초기화
void docInit()
{
    // 변수 초기화 등
}
// 모듈 종료 처리
void docClose()
{
}
// Getter functions
int docGetData()
{
    return docData;
}
// Setter functions
void docSetData(int a)
{
    docData = a;
}
```

함수 호출 예제

- 다른 모듈의 extern 함수를 호출하면서 프로그램 실행
  - 모듈의 데이터는 static으로 만들어서 보호
  - Getter 및 Setter 함수를 extern으로 제공



Naming Convention

정답이 있는 것은 아님

- 변수 및 함수 이름 부여 규칙을 미리 정해 놓을 것
- 공통 (예제)
  - 변수는 명사로 시작, 함수는 동사로 시작
  - 단어가 변경될 때 대문자로 시작
    - 예제, docGetData()
  - Local variable과 global variable을 구분할 수 있도록 이름 붙일 것
    - 예를 들면, local variable은 소문자로 시작, global variable은 대문자로 시작
    - 예를 들면, Global variable은 g로 시작 (예제, gCount)
- 모듈별 이름 부여
  - 모듈을 구분할 수 있도록 함수 앞에 doc, view 등의 prefix를 붙임
    - 예제, docGetData()