

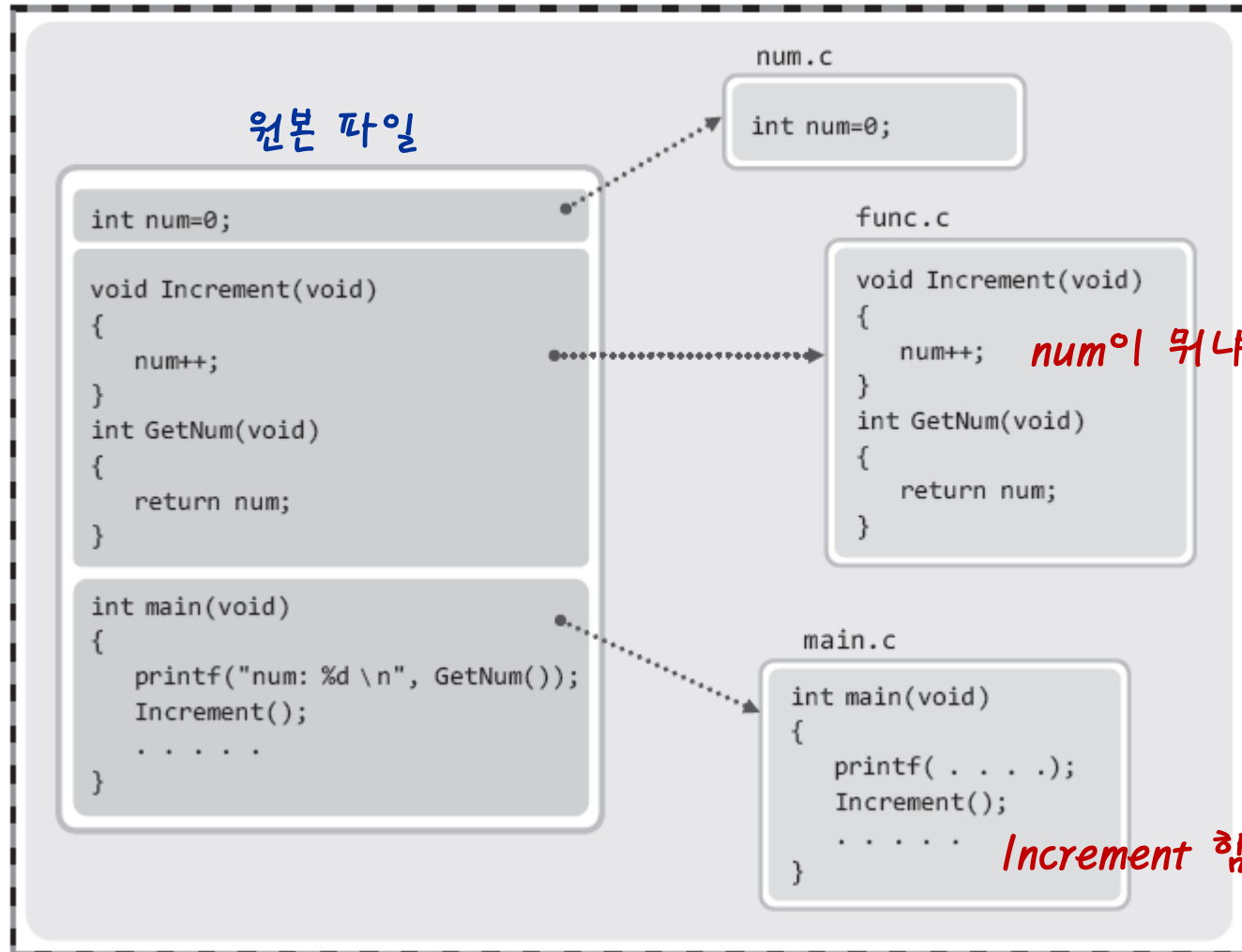
CHAPTER 7

전 처리의 이해 조건 및  
분할 컴파일

# 학습 목표

- 파일의 분할

# 7.1 파일을 그냥 나눠도 될까요?

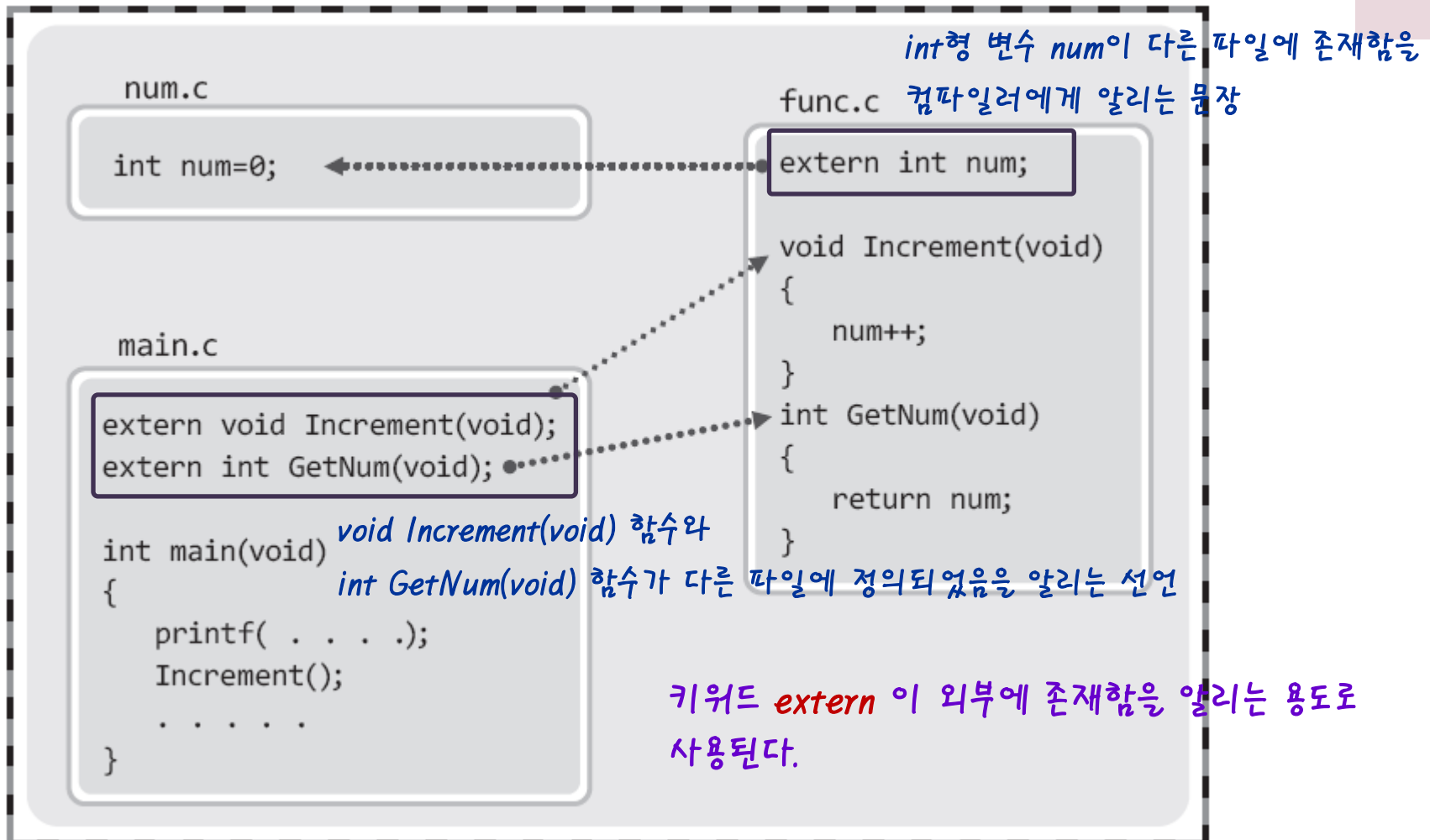


컴파일러는  
파일 단위로 컴파일!

num이 뭐냐?

Increment 함수는 어디 있는 거야?

## 7.2 외부 선언 및 정의 사실을 컴파일러에게 알려줘야 ...



## 7.3 전역변수의 static 선언의 의미

지역변수로 선언되는 경우

```
void  
SimpleFunc(void)  
{  
    static int num=0;  
    ....  
}
```

함수 내에서만 접근이 가능한, 전역변수와 마찬가지로 한번 메모리 공간에 저장되면 종료 시까지 소멸되지 않고 유지되는 변수의 선언

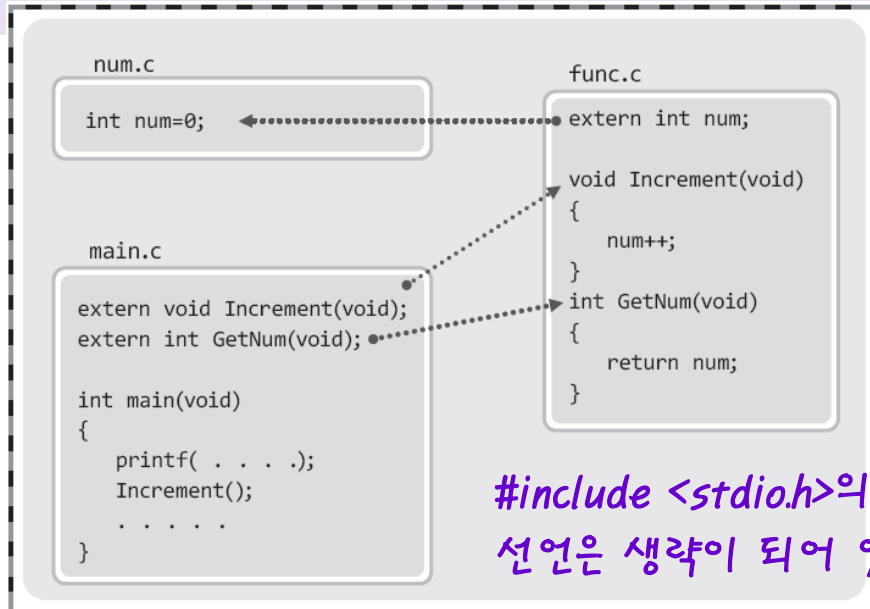
전역변수로 선언되는 경우

```
static int num=0;  
  
void  
SimpleFunc(void)  
{  
    ....  
}
```

이 경우 int num은 전역변수이다. 단 외부 소스파일에서 접근이 불가능한 전역변수가 된다. 즉, 접근의 범위를 파일로 제한하게 된다.

# 둘 이상의 파일을 컴파일하는 방법과 static 에 대한 고찰

## 7.4 파일부터 정리하고 시작합니다.



이 세 개의 파일을 하나의 프로젝트 안에 담아서 하나의 실행파일을 생성해 보는 것이 목적!

다중파일 컴파일 방법 두 가지

- 첫 번째 방법  
→ 파일을 먼저 생성해서 코드를 삽입한 다음에 프로젝트에 추가한다.
- 두 번째 방법  
→ 프로젝트에 파일을 추가한 다음에 코드를 삽입한다.

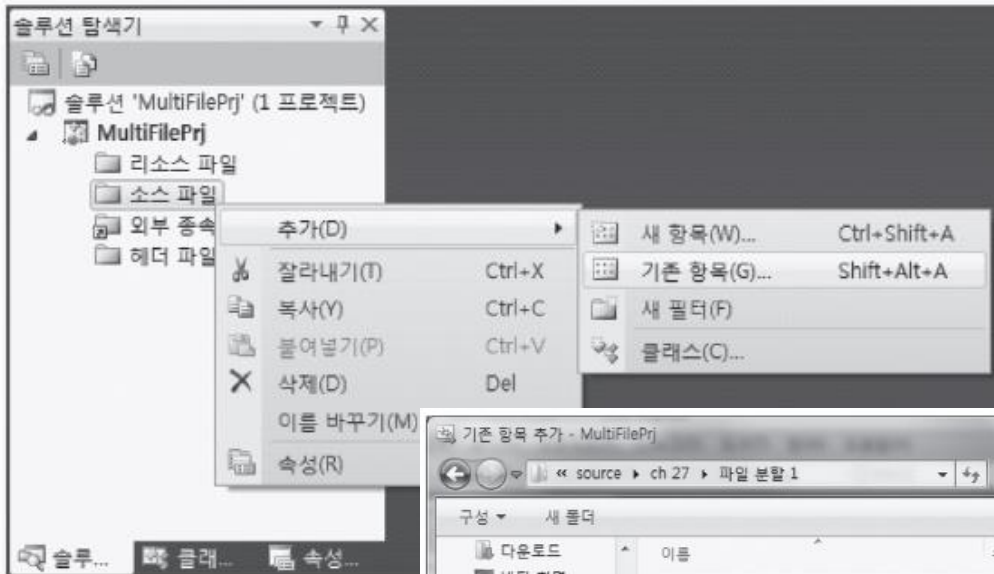
# 7.5 존재하는 파일, 프로젝트에 추가하는 방법

이미 존재하는 소스파일을 추가하는 방법

1단계

아래에서 보이듯이 다수의 파일이 하나의 프로젝트 안에 포함되었음이 솔루션 탐색기에 나타나야 한다.

추가결과 확인



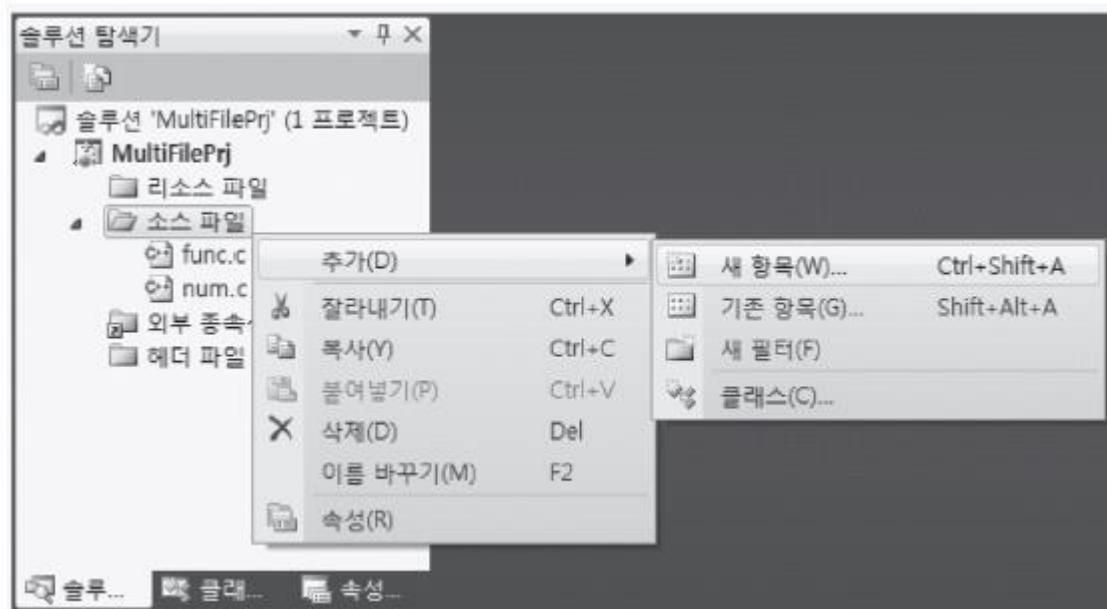
2단계





## 7.6 프로젝트에 새로운 파일을 추가하는 방법

새로운 소스파일을 만들어서 추가하는 방법



이는 기존에 해왔던, 소스파일을 새로 생성해서 프로젝트에 추가하는 방법과 100% 동일하다.  
그 과정을 재차 진행하면 새로운 소스파일을 생성해서 프로젝트 내에 포함시킬 수 있다.

## 7.7 함수에도 static 선언을 할 수 있습니다.

함수를 대상으로 하는 *static* 선언

```
static void MinCnt(void)
{
    cnt--;
}
```

함수의 *static* 선언은 전역변수의 *static* 선언과 그 의미가 동일하다. 즉, 외부 소스파일에서의 접근을(호출을) 허용하지 않기 위한 선언이다.

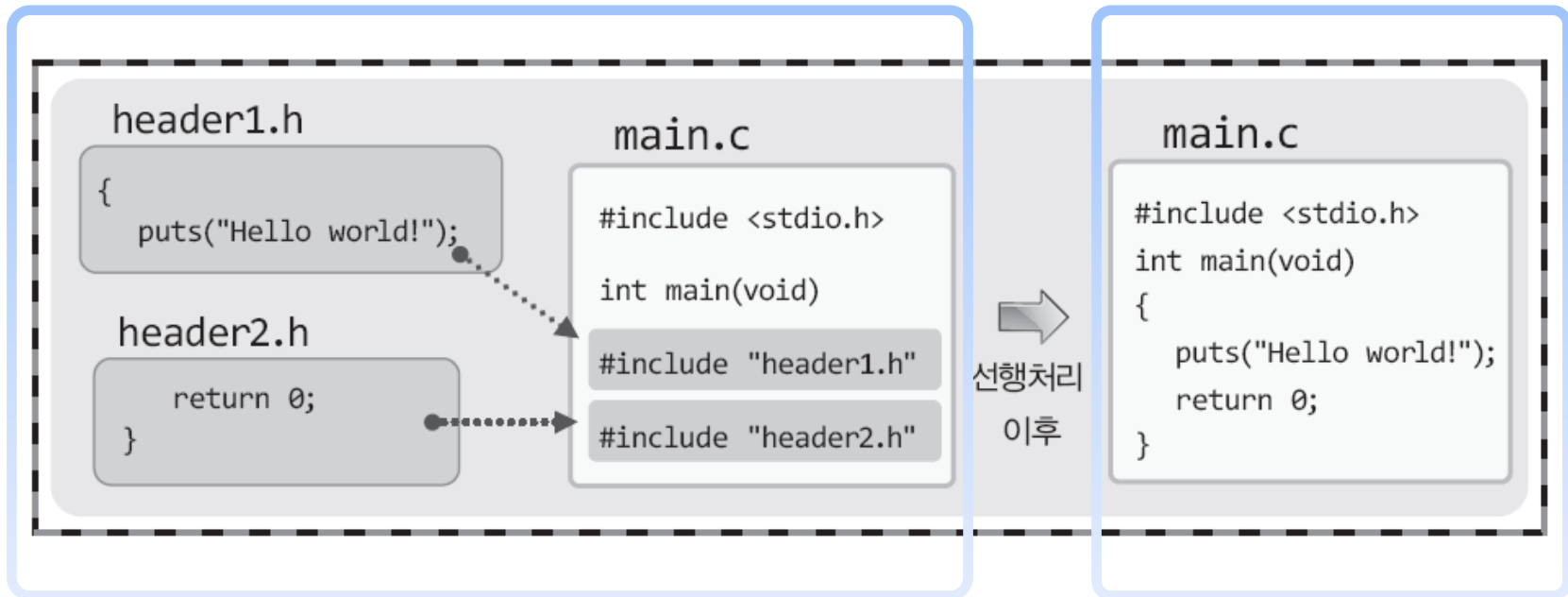
The header features a series of overlapping colored rectangles in shades of red, orange, and purple. A long, light purple rectangle spans across the top, with smaller, darker purple and red rectangles positioned to its right.

# 헤더파일의 디자인과 활용

## 7.8 include 지시자와 헤더파일의 의미

두 개의 헤더파일과 하나의 소스파일로 이뤄진 프로젝트

선행처리 이후의 결과



위의 그림을 통해서 이해할 수 있듯이 `#include` 지시자는 헤더파일을 단순히 포함시키는 기능을 제공한다. 그리고 기본적으로 헤더파일에는 무엇이든 넣을 수 있다. 그러나 아무것이나 넣어서는 안 된다.

## 7.9 헤더 파일을 include 하는 두 가지 방법

### 표준 헤더파일의 포함

```
#include <헤더파일 이름>
```

표준헤더 파일을 포함시킬 때 사용하는 방식이다. 표준헤더 파일이 저장된 디렉터리에서 헤더파일을 찾아서 포함을 시킨다.

### 프로그래머가 정의한 헤더파일의 포함

```
#include "헤더파일 이름"
```

프로그래머가 정의한 헤더파일을 포함시킬 때 사용하는 방식이다. 이 방식을 이용하면 이 문장을 포함하는 소스파일이 저장된 디렉터리에서 헤더파일을 찾게 된다.

## 7.10 절대경로의 지정과 그에 따른 단점

이렇듯 헤더파일의 경로를 명시할 수도 있다.

```
#include "C:\CPower\MyProject\header.h"
```

*Windows의 절대경로 지정방식.*

```
#include "/CPower/MyProject/header.h"
```

*Linux의 절대경로 지정방식*

- ▶ 절대경로를 지정하면 프로그램의 소스파일과 헤더파일을 임의의 위치로 이동시킬 수 없다(동일 운영체제를 기반으로 하더라도).
- ▶ 운영체제가 달라지면 디렉터리의 구조가 달라지기 때문에 경로지정에 대한 부분을 전면적으로 수정해야 한다.

## 7.11 상대경로의 지정 방법

상대경로 기반의 #include 선언

#include "header.h" 이 문장을 포함하는 소스파일이 저장된 디렉터리  
:현재 디렉터리

#include "ReleaseWheader0.h" 현재 디렉터리의 서브인 Release 디렉터리

#include "../CProgWheader1.h" 현재 디렉터리의 상위 디렉터리의  
서브인 Cprog 디렉터리

#include "../../MyHeaderWheader2.h" 현재 디렉터리의 상위 디렉터리의 상위  
디렉터리의 서브인 MyHeader 디렉터리

위와 같은 형태로(상대경로의 지정방식을 기반으로) 헤더파일 경로를 명시하면 프로그램의 소스코드가 저장되어 있는 디렉터를 통째로 이동하는 경우 어디서든 컴파일 및 실행이 가능해진다.

## 7.12 헤더파일에 무엇을 담으면 좋겠습니까?

헤더파일에 삽입이 되는 가장 일반적인 선언의 유형

```
extern int num;  
extern int GetNum(void);    // extern 생략 가능
```

▶ 총 7개의 소스파일과 헤더파일로 이뤄진 예제를 통해서 다음 두 가지에 대한 정보를 얻자!

- 소스파일을 나누는 기준
- 헤더파일을 나누는 기준 및 정의의 형태

▶ 예제의 소스파일과 헤더파일의 구성

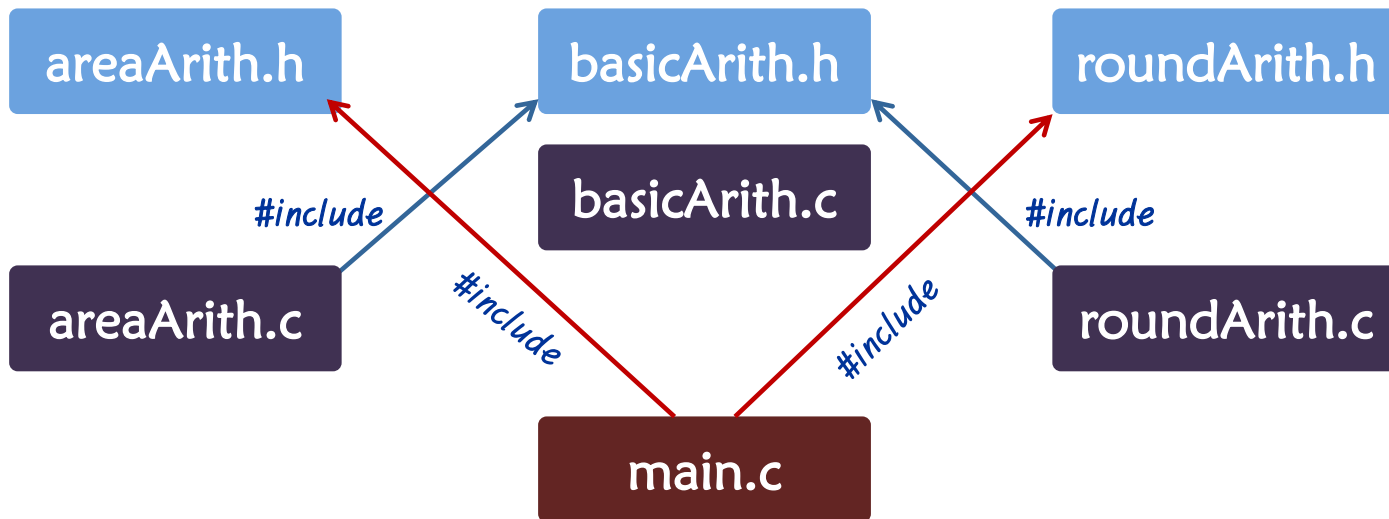
- basicArith.h    basicArith.c
- areaArith.h    areaArith.c
- roundArith.h    roundArith.c
- main.c



## 7.13 헤더파일과 소스파일의 포함관계

### ▶ 예제의 소스파일과 헤더파일의 구성 및 내용

- basicArith.h    basicArith.c    → 수학과 관련된 기본적인 연산의 함수의 정의 및 선언
- areaArith.h    areaArith.c    → 넓이계산과 관련된 함수의 정의 및 선언
- roundArith.h    roundArith.c    → 둘레계산과 관련된 함수의 정의 및 선언
- main.c



## 7.14 basicArith.h & basicArith.c

*basicArith.h* : 기본연산 함수의 선언

```
#define PI 3.1415
double Add(double num1, double num2);
double Min(double num1, double num2);
double Mul(double num1, double num2);
double Div(double num1, double num2);
```

매크로의 정의는 파일단위로 유효하다.  
그래서 PI와 같은 상수의 선언은  
헤더파일에 정의하고, 이를 필요한 모든  
소스파일이 PI가 선언된 헤더파일을  
포함하는 형태를 띈다.

*basicArith.c* : 기본연산 함수의 정의

```
double Add(double num1, double num2)
{
    return num1+num2;
}

double Min(double num1, double num2)
{
    return num1-num2;
}

double Mul(double num1, double num2)
{
    return num1*num2;
}

double Div(double num1, double num2)
{
    return num1/num2;
}
```

## 7.14 areaArith.h & areaArtih.c

```
double TriangleArea(double base, double height);  
double CircleArea(double rad);
```

*areaArith.h : 넓이 계산 함수의 선언*

```
#include "basicArith.h"
```

```
double TriangleArea(double base, double height)  
{  
    return Div(Mul(base, height), 2);  
}  
  
double CircleArea(double rad)  
{  
    return Mul(Mul(rad, rad), PI);  
}
```

*areaArith.c : 넓이 계산 함수의 정의*

## 7.15 roundArith.h & roundArith.c

```
double RectangleRound(double base, double height);  
double SquareRound(double side);
```

*roundArith.h : 둘레계산 함수의 선언*

```
#include "basicArith.h"
```

```
double RectangleRound(double base, double height)  
{  
    return Mul(Add(base, height), 2);  
}  
double SquareRound(double side)  
{  
    return Mul(side, 4);  
}
```

*roundArith.c : 둘레계산 함수의 정의*

## 7.16 main.c

```
#include <stdio.h>
#include "areaArith.h"
#include "roundArith.h"

int main(void)
{
    printf("삼각형 넓이(밑변 4, 높이 2): %g \n",
        TriangleArea(4, 2));
    printf("원 넓이(반지름 3): %g \n",
        CircleArea(3));

    printf("직사각형 둘레(밑변 2.5, 높이 5.2): %g \n",
        RectangleRound(2.5, 5.2));
    printf("정사각형 둘레(변의 길이 3): %g \n",
        SquareRound(3));
    return 0;
}
```

실행결과

```
삼각형 넓이(밑변 4, 높이 2): 4
원 넓이(반지름 3): 28.2735
직사각형 둘레(밑변 2.5, 높이 5.2): 15.4
정사각형 둘레(변의 길이 3): 12
```

## 7.17 구조체의 정의는 어디에? : 문제의 제시

구조체의 정의도 파일 단위로만 그 선언이 유효하다. 따라서 필요하다면 동일한 구조체의 정의를 소스파일마다 추가시켜 줘야 한다.

소스파일 *intdiv.c*

```
typedef struct div
{
    int quotient;    // 몫
    int remainder;   // 나머지
} Div;
```

```
Div IntDiv(int num1, int num2)
{
    Div dval;
    dval.quotient=num1/num2;
    dval.remainder=num1%num2;
    return dval;
}
```

소스파일 *main.c*

```
typedef struct div
{
    int quotient;    // 몫
    int remainder;   // 나머지
} Div;
```

```
extern Div IntDiv(int num1, int num2);

int main(void)
{
    Div val=IntDiv(5, 2);
    printf("몫: %d \n", val.quotient);
    printf("나머지: %d \n", val.remainder);
    return 0;
}
```

같은 구조체 정의를 둘 이상의 소스파일에 직접 추가시킨다는 것 자체가 부담!

# 7.18 구조체의 정의는 어디에? : 해결책의 제시

헤더파일 *stddiv.h*

```
typedef struct div
{
    int quotient;    // 몫
    int remainder;   // 나머지
} Div;
```

#include

```
#include "stddiv.h"
Div IntDiv(int num1, int num2)
{
    Div dval;
    dval.quotient=num1/num2;
    dval.remainder=num1%num2;
    return dval;
}
```

소스파일 *intdiv2.c*

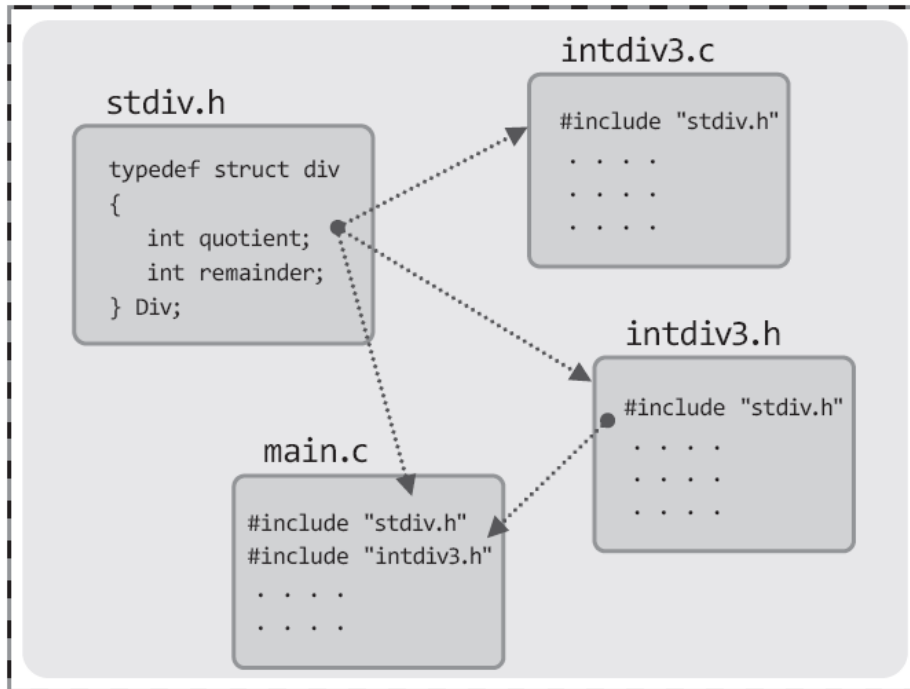
구조체의 정의도 헤더파일에 넣어두고  
필요할 때마다 include 하는 것이  
일반적이다!

#include

```
#include "stddiv.h"
extern Div IntDiv(int num1, int num2);
int main(void)
{
    Div val=IntDiv(5, 2);
    printf("몫: %d \n", val.quotient);
    printf("나머지: %d \n", val.remainder);
    return 0;
}
```

소스파일 *main.c*

## 7.19 헤더파일의 중복삽입 문제



헤더파일을 직접적으로 또는 간접적으로 두 번 이상 포함하는 것 자체는 문제가 아니다. 그러나 두 번 이상 포함시킨 헤더파일의 내용에 따라서 문제가 될 수 있다.

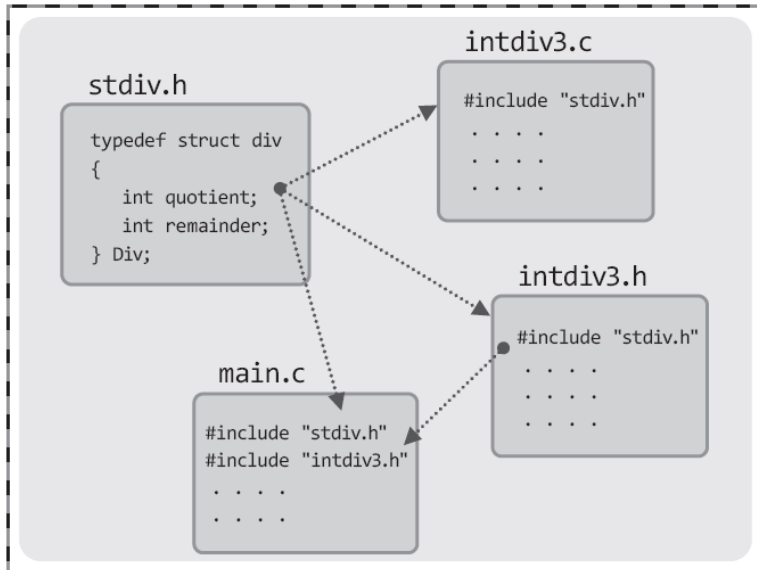
일반적으로 선언(예로 함수의 선언)은 두 번 이상 포함시켜도 문제되지 않는다. 그러나 정의(예로 구조체 및 함수의 정의)는 두 번 이상 포함시키면 문제가 된다.

*main.c는 결과적으로 구조체 Div의 정의를 두 번 포함하는 꼴이 된다! 그런데 구조체의 정의는 하나의 소스파일 내에서 중복될 수 없다!*



## 7.20 조건부 컴파일을 활용한 중복삽입 문제의 해결

### 중복 삽입문제의 해결책



```
#ifndef __STDIV2_H__
#define __STDIV2_H__

typedef struct div
{
    int quotient;    // 몫
    int remainder;  // 나머지
} Div;

#endif
```

매크로 `__STDIV2_H__` 와 `#ifndef`의 효과가 `main.c`에서 어떻게 나타나는지 그려보자!

위와 같은 이유로 모든 헤더파일은 `#ifndef~#endif`로 감싸는 것이 안전하고 또 일반적이다!

# 질문 및 정리

