

## -Part3-

# 제5장 전처리기와 파일 분할 컴파일

## 학습목차

5.1 전처리기

5.2 매크로

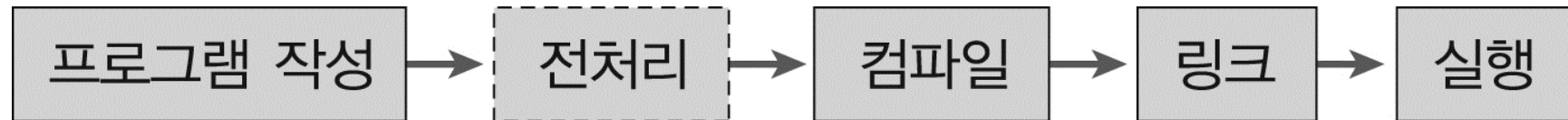
5.3 조건부 컴파일

5.4 파일 분할 컴파일

## 5.1 전처리기

## 5.1 전처리기 (1/2)

### ▶ 전처리와 전처리기



- ✓ **전처리** - 소스파일을 컴파일 하기 전에 먼저 처리해야 하는 일
- ✓ **전처리기** - 전처리를 수행하는 장치(# 문자로 시작)

### 예) 전처리 지시자

#**include** - 헤더 파일을 인클루드

#**define** - 매크로 상수를 정의

## 5.1 전처리기 (2/2)

전처리기 지시자	설명
#include	헤더 파일을 인클루드하는 기능
#define	매크로를 정의하는 기능
#undef	이미 정의된 매크로를 해제하는 기능
#if, #elif, #else, #endif	조건에 따라 컴파일하는 기능
#ifdef	매크로가 정의된 경우에 컴파일하는 기능
#ifndef	매크로가 정의되지 않은 경우에 컴파일하는 기능

## 5.2 매크로

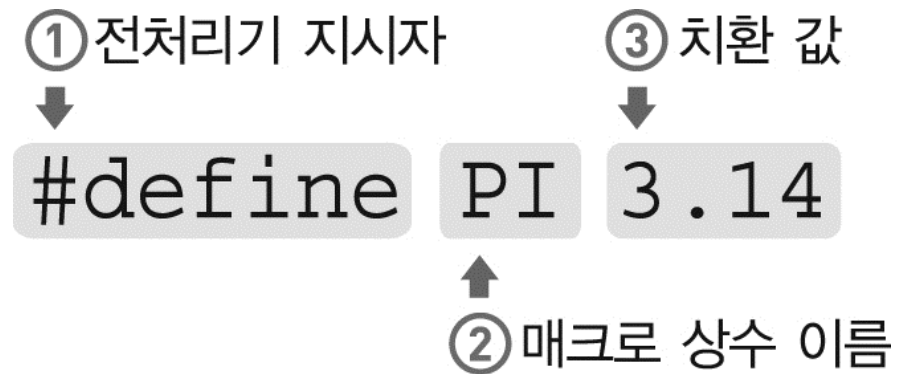
## 5.2 매크로

### ▶ 배울 내용

- ① 매크로 상수
- ② 매크로 해제
- ③ 매크로 함수
- ④ # 연산자와 ## 연산자
- ⑤ 미리 정의된 매크로

## 5.2 매크로 (1/18)

### ▶ 매크로 상수의 정의



- ✓ **전처리기 지시자:** 매크로 상수를 선언하기 위해서 #define를 지정
- ✓ **매크로 상수 이름:** 매크로 상수의 이름을 지정
- ✓ **치환값:** 매크로 상수에 치환되는 값 지정



## 5.2 매크로 (2/18)---[5-2.c 실습]

```
#include <stdio.h>

#define PI 3.14          // 전처리기 지시자

int main(void)
{
    double area, circum, radius;

    fputs("반지름을 입력하세요: ", stdout);
    scanf("%lf", &radius);

    area=PI * radius * radius;
    circum=2 * PI * radius;

    printf("원의 넓이: %lf \n", area);
    printf("원의 둘레: %lf \n", circum);

    return 0;
}
```

## 5.2 매크로 (3/18)---[5-3.c 실습]

```
#include <stdio.h>

#define MAX 100           // 정수형 매크로 상수
#define PI 3.14           // 실수형 매크로 상수
#define STRING "Hello C" // 문자열 매크로 상수
#define OUTPUT printf    // 함수 이름 매크로 상수
#define DATA int        // 자료형 매크로 상수

int main(void)
{
    DATA a=3;
    OUTPUT("%d, %lf, %s, %d\n", MAX, PI, STRING, a);

    return 0;
}
```

## 5.2 매크로 (4/18)

---

### ▶ 매크로 상수의 장점

- ✓ '프로그램 수정이 용이하다.'
- ✓ '숫자들 대신에 직관적인 의미를 갖는 이름을 가진다.'
- ✓ '변수와 달리 추가적인 메모리 공간을 요구하지 않는다.'
- ✓ '코드에 등장하는 상수들을 한곳에 모아서 관리할 수 있다.'

## 5.2 매크로

### ▶ 배울 내용

① 매크로 상수

② 매크로 해제

③ 매크로 함수

④ # 연산자와 ## 연산자

⑤ 미리 정의된 매크로

## 5.2 매크로 (5/18)

### ▶ 매크로 상수의 해제

① 전처리기 지시자



```
#undef PI
```



② 해제할 매크로 이름

- ✓ **전처리기 지시자:** 매크로의 선언을 해제하기 위해서 #undef를 지정
- ✓ **해제할 매크로 이름:** 해제할 매크로 이름 지정(미리 정의된 매크로 상수)

## 5.2 매크로 (6/18)---[5-4.c 실습]

```
#include <stdio.h>

# define MAX 100           // 정수형 매크로 상수
# define PI 3.14           // 실수형 매크로 상수

int main(void)
{
    int a=3;
    printf("변경 전 : %d, %lf \n", MAX, PI);

    #undef MAX              // 매크로 해제
    #undef PI               // 매크로 해제

    #define MAX 1000        // 매크로 상수 재정의
    #define PI 3.141592     // 매크로 상수 재정의
    printf("변경 후 : %d, %lf \n", MAX, PI);

    return 0;
}
```

## 5.2 매크로

### ▶ 배울 내용

① 매크로 상수

② 매크로 해제

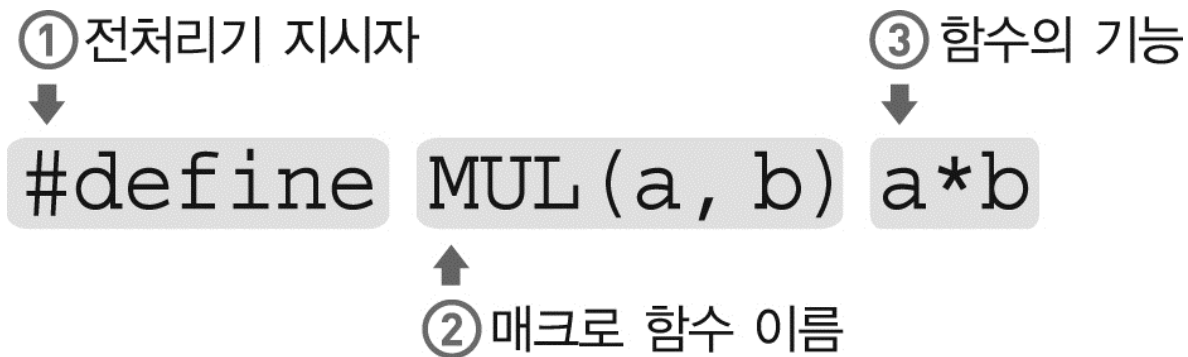
③ 매크로 함수

④ # 연산자와 ## 연산자

⑤ 미리 정의된 매크로

## 5.2 매크로 (7/18)

### ▶ 매크로 함수



- ✓ **전처리기 지시자:** 매크로 함수를 선언하기 위해서 #define을 지정
- ✓ **매크로 함수 이름:** 사용될 매크로 함수의 이름을 지정
- ✓ **함수의 기능:** 매크로 함수 이름에 치환되는 함수의 기능

#### ✓ 특징

- 단순히 치환하기만 하므로 **실제로 함수는 아님**
- **매개변수의 자료형을 신경 쓰지 않음**(자료형의 독립성 보장)



## 5.2 매크로 (8/18)---[5-5.c 실습]

```
#include <stdio.h>

#define MUL(x, y) x*y           // 매크로 함수 정의

int main(void)
{
    int a, b;
    double c, d;

    printf("두개의 정수를 입력하세요: ");
    scanf("%d%d", &a, &b);
    printf("%d * %d = %d \n", a, b, MUL(a, b)); // 매크로 함수 호출

    printf("두개의 실수를 입력하세요: ");
    scanf("%lf%lf", &c, &d);
    printf("%lf * %lf = %lf \n", c, d, MUL(c, d)); // 매크로 함수 호출

    return 0;
}
```

## 5.2 매크로 (9/18)

### ▶ 매크로 함수의 장점

- ✓ 함수의 인자(매개변수)에 대한 자료형의 독립성 보장
- ✓ 속도가 빠름

### ▶ 매크로 함수의 단점

- ✓ 매크로 함수 내부에서 자기 자신을 호출할 수 없음
- ✓ 한 줄이나 두 줄 정도의 간단한 내용만 매크로 함수로 정의해야 함

## 5.2 매크로 (10/18)---[5-6.c 실습]

```
#include <stdio.h>
```

```
#define MUL(x, y) x*y      // 매크로함수  
int mul(x, y);             // 일반함수선언
```

```
int main(void)
```

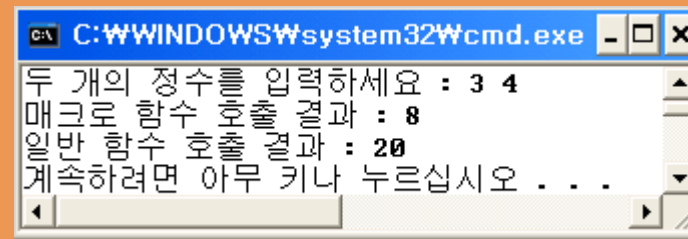
```
{  
    int a, b;  
    printf("두 개의 정수를 입력 하세요: ");  
    scanf("%d %d", &a, &b);
```

```
    printf("매크로함수호출결과: %d \n", MUL(a+1, b+1) ); // 매크로함수호출  
    printf("일반함수호출결과: %d \n", mul(a+1, b+1) );   // 일반함수호출  
    return 0;
```

```
}
```

```
int mul(x, y)
```

```
{  
    return x * y;  
}
```



```
C:\WINDOWS\system32\cmd.exe  
두 개의 정수를 입력하세요 : 3 4  
매크로 함수 호출 결과 : 8  
일반 함수 호출 결과 : 20  
계속하려면 아무 키나 누르십시오 . . .
```

## 5.2 매크로 (11/18)---[5-6.c 분석]

### ▶ 문제점 발생

- ✓  $a+1$ 과  $b+1$ 을  $x$ 와  $y$ 의 위치에 단순히 치환

```
#define MUL(x, y) x*y  
MUL(a+1, b+1)    // 단순 치환, a+1*b+1
```

### ▶ 문제점 해결

- ✓ 괄호를 사용

```
#define MUL(x, y) ((x)*(y))  
MUL(a+1, b+1)    // 의도한 결과, ((a+1)*(b+1))
```

## 5.2 매크로

### ▶ 배울 내용

① 매크로 상수

② 매크로 해제

③ 매크로 함수

④ # 연산자와 ## 연산자

⑤ 미리 정의된 매크로

## 5.2 매크로 (12/18)

### ▶ # 연산자

- ✓ '매크로 함수의 인자를 문자열로 바꾸어 준다.'

### ▶ ## 연산자

- ✓ 토큰(문법 분석의 단위, 예: 숫자, 콤마, 연산자, 식별자 등) 결합 연산자
- ✓ 매크로 함수 안에서 토큰을 결합하는 기능을 수행

## 5.2 매크로 (13/18)---[5-7.c 실습]

```
#include <stdio.h>

#define OUTPUT1(a) a      // 매크로 함수 정의
#define OUTPUT2(a) #a     // 매크로 함수 정의

int main(void)
{
    printf(" %d \n", OUTPUT1(1234) );    // 10진수 1234
    printf(" %s \n", OUTPUT2(1234) );    // 문자열 1234

    return 0;
}
```

## 5.2 매크로 (14/18)---[5-8.c 실습]

```
#include <stdio.h>

#define OUTPUT1(a, b) a + b           // 매크로 함수 정의
#define OUTPUT2(a, b) #a "+" #b      // 매크로 함수 정의

int main(void)
{
    printf(" %d \n", OUTPUT1(11, 22)); // 10진수 덧셈 연산
    printf(" %s \n", OUTPUT2(11, 22)); // 문자열 합치기

    return 0;
}
```



## 5.2 매크로 (15/18)---[5-9.c 실습]

```
#include <stdio.h>

#define OUTPUT(a, b, c) a ## b ## c      // 매크로 함수 정의

int main(void)
{
    int a=3;

    printf(" %d \n", a);
    printf(" %d \n", OUTPUT(a, = , 5));    // 매크로 함수 호출
    printf(" %d \n", a);

    return 0;
}
```

## 5.2 매크로

### ▶ 배울 내용

- ① 매크로 상수
- ② 매크로 해제
- ③ 매크로 함수
- ④ # 연산자와 ## 연산자
- ⑤ 미리 정의된 매크로

## 5.2 매크로 (16/18)

### ▶ 미리 정의된 매크로

✓ C 언어에서 개발자의 편의를 위해 **미리 정의**

미리 정의된 매크로	설명
<code>__FILE__</code>	현재 소스 코드의 파일 이름을 나타내는 매크로, %s 사용
<code>__LINE__</code>	현재 위치의 소스 코드의 행 번호를 나타내는 매크로, %d 사용
<code>__DATE__</code>	현재 소스 코드의 컴파일 날짜를 나타내는 매크로, %s 사용
<code>__TIME__</code>	현재 소스 코드의 컴파일 시간을 나타내는 매크로, %s 사용

## 5.2 매크로 (17/18)---[5-10.c 실습]

```
#include <stdio.h>
int main(void)
{
    printf("파일 이름 : %s \n", __FILE__);
    printf("행 번호 : %d \n", __LINE__);
    printf("컴파일 날짜 : %s \n", __DATE__);
    printf("컴파일 시간 : %s \n", __TIME__);

    return 0;
}
```

## 5.2 매크로 (18/18)---[5-11.c 실습]

```
#include <stdio.h>
int main(void)
{
    double num1, num2, result ;
    printf("실수 두 개를 입력하세요: ");
    scanf("%lf %lf", &num1, &num2);
    result=num1 / num2;

    if(result > 0)
    {
        printf("%lf ₩n", result);
        printf("컴파일 날짜: %s ₩n", __DATE__);
        printf("컴파일 시간: %s ₩n", __TIME__);
        printf("파일 이름: %s ₩n", __FILE__);
    }
    else
    {
        printf("오류 발생₩n");
        printf("행 번호: %d ₩n", __LINE__);
    }
    return 0;
}
```

## 5.3 조건부 컴파일

## 5.3 조건부 컴파일 (1/11)

### ▶ 조건부 컴파일

- ✓ '특정 조건에 만족할 때만 코드가 컴파일되게 한다.'
- ✓ '매크로 상수를 검사하여 조건부 컴파일을 한다.'

### ▶ 배울 내용

① `#if~#endif`

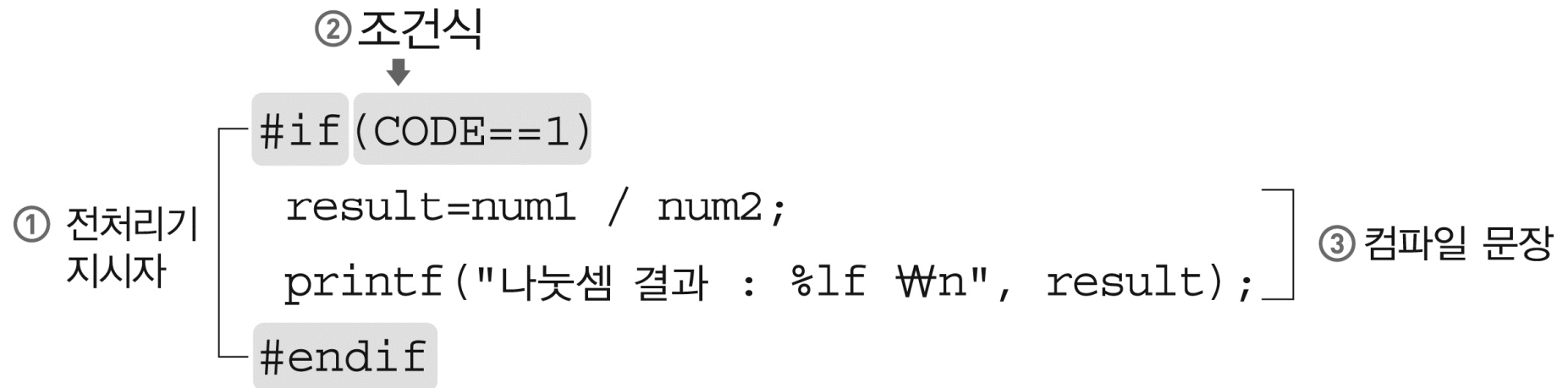
② `#if~#else~#endif`

③ `#if~#elif~#else~#endif`

④ `#ifdef~#endif`와 `#ifndef~#endif`

## 5.3 조건부 컴파일 (2/11)

### ▶ #if ~ #endif 의 기본 형식



- ✓ **전처리 지시자:** 조건부 컴파일 수행 문장을 #if와 #endif로 묶음
- ✓ **조건식:** 컴파일을 수행하기 위한 조건을 지정
- ✓ **컴파일 문장:** 조건식이 '참'일 때 컴파일해야 하는 문장 삽입



## 5.3 조건부 컴파일 (3/11)---[5-12.c 실습]

```
#include <stdio.h>

#define CODE 2

int main(void)
{
    double num1=0, num2=0, result=0;

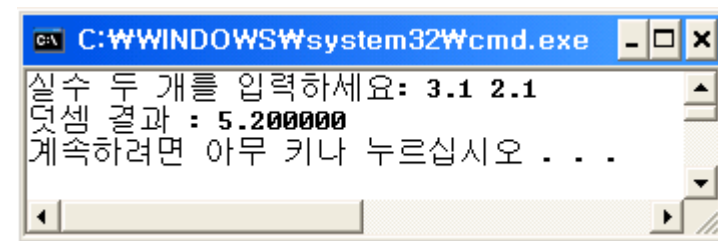
    printf("실수 두개를 입력하세요>>");
    scanf("%lf %lf", &num1, &num2);

    #if(CODE==1)
        result=num1 / num2;
        printf("나눗셈 결과: %lf \n", result);
    #endif

    #if(CODE==2)
        result=num1 + num2;
        printf("덧셈 결과: %lf \n", result);
    #endif
```

```
#if(CODE==3)
    result=num1 * num2;
    printf("곱셈 결과: %lf \n", result);
#endif

#if(CODE==4)
    result=num1 - num2;
    printf("뺄셈 결과: %lf \n", result);
#endif
return 0;
}
```



## 5.3 조건부 컴파일

### ▶ 배울 내용

① `#if~#endif`

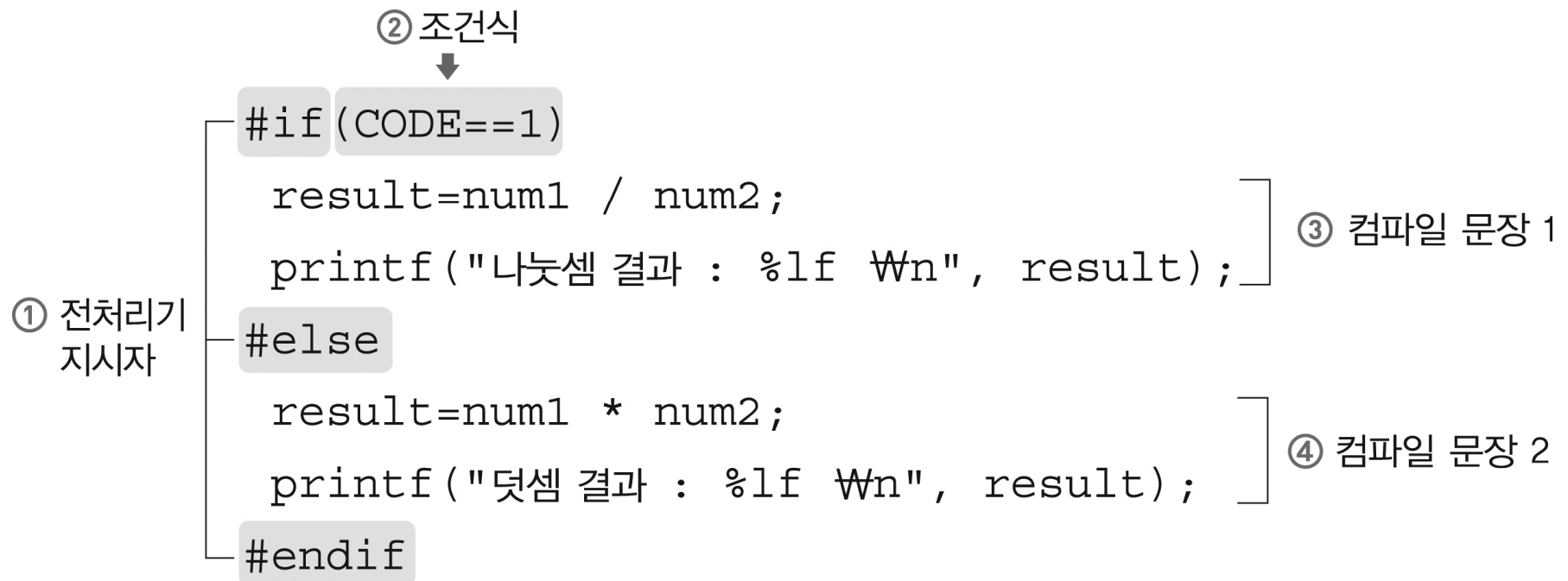
② `#if~#else~#endif`

③ `#if~#elif~#else~#endif`

④ `#ifdef~#endif`와 `#ifndef~#endif`

## 5.3 조건부 컴파일 (4/11)

### ▶ #if~#else~#endif의 기본 형식



## 5.3 조건부 컴파일 (5/11)---[5-13.c 실습]

```
#include <stdio.h>
#define CODE 3
int main(void)
{
    #if(CODE==1)                // 실수의 나눗셈 연산
        double num1=0.0, num2=0.0, result=0.0;
        printf("실수 두개를 입력하세요>>");
        scanf("%lf %lf", &num1, &num2);
        result=num1 / num2;
        printf("나눗셈 결과: %lf \n", result);

    #else                       // 정수의 덧셈 연산
        int num1=0, num2=0, result=0;
        printf("정수 두개를 입력하세요>>");
        scanf("%d %d", &num1, &num2);
        result=num1 + num2;
        printf("덧셈 결과: %d \n", result);
    #endif
    return 0;
}
```

## 5.3 조건부 컴파일

### ▶ 배울 내용

① `#if~#endif`

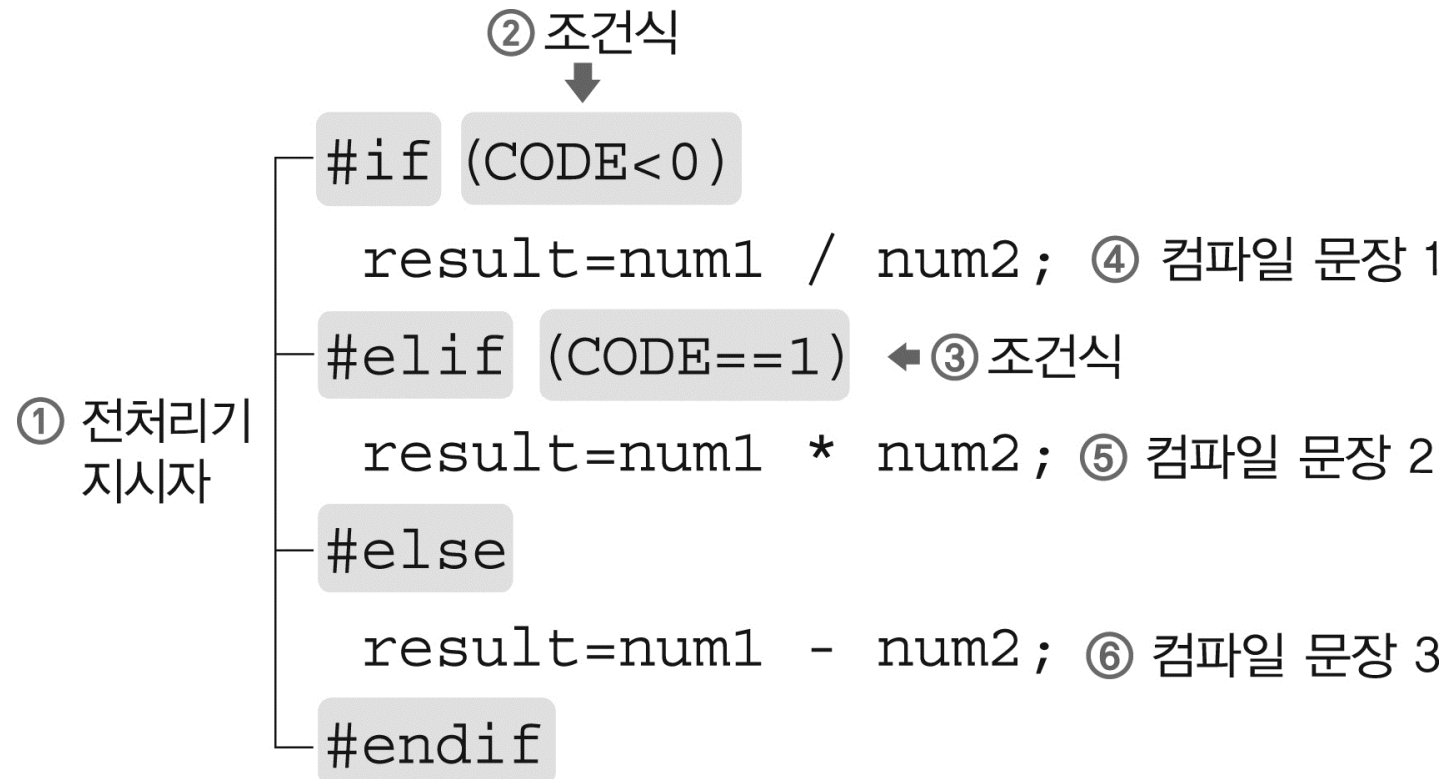
② `#if~#else~#endif`

③ `#if~#elif~#else~#endif`

④ `#ifdef~#endif`와 `#ifndef~#endif`

## 5.3 조건부 컴파일 (6/11)

### ▶ #if~#elif~#else~#endif의 기본 형식



## 5.3 조건부 컴파일 (7/11)---[5-14.c 실습]

```
# include <stdio.h>

#define CODE 3

int main(void)
{
    double num1=3.3, num2=1.1;
    double result=0.0;

    #if(CODE<0)
        result=num1+num2;
        printf("덧셈결과: %lf \n", result);

    #elif(CODE==1)
        result=num1 / num2;
        printf("나눗셈결과: %lf \n", result);
```



```
#elif(CODE==2)
    result=num1 * num2;
    printf("곱셈결과: %lf \n", result);

    #elif(CODE==3)
        result=num1 - num2;
        printf("뺄셈결과: %lf \n", result);

    #else
        printf("프로그램종료\n");

    #endif
    return 0;
}
```

## 5.3 조건부 컴파일

### ▶ 배울 내용

① `#if~#endif`

② `#if~#else~#endif`

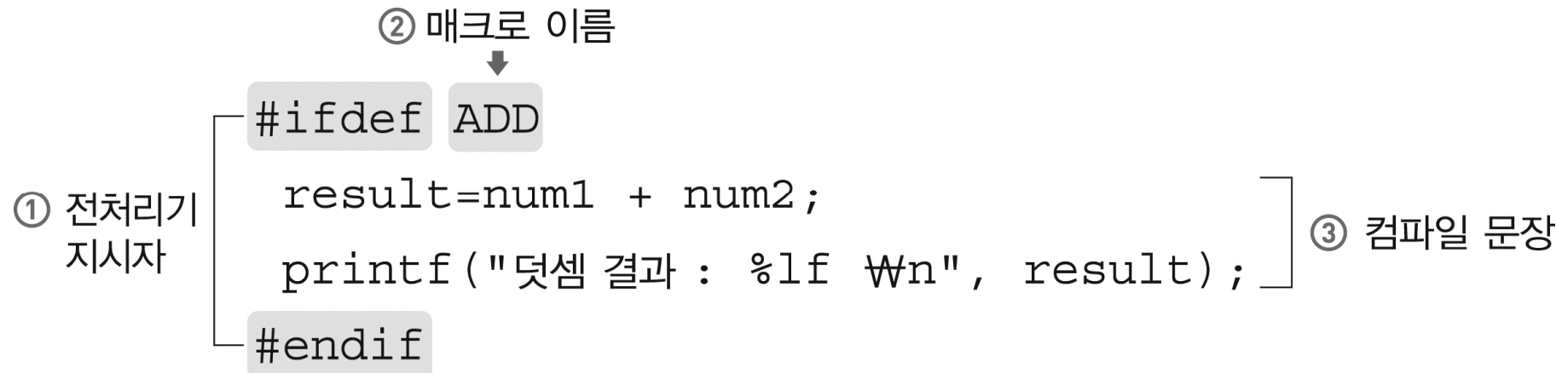
③ `#if~#elif~#else~#endif`

④ `#ifdef~#endif`와 `#ifndef~#endif`



## 5.3 조건부 컴파일 (8/11)

### ▶ #ifdef ~ #endif의 기본 형식



‘매크로 상수 ADD가 정의 되어 있다면 조건부 컴파일을 수행한다.’

## 5.3 조건부 컴파일 (9/11)---[5-15.c 실습]

```
#include <stdio.h>
#define ADD
#define MUL
int main(void)
{
    double num1=3.3, num2=1.1;
    double result=0.0;

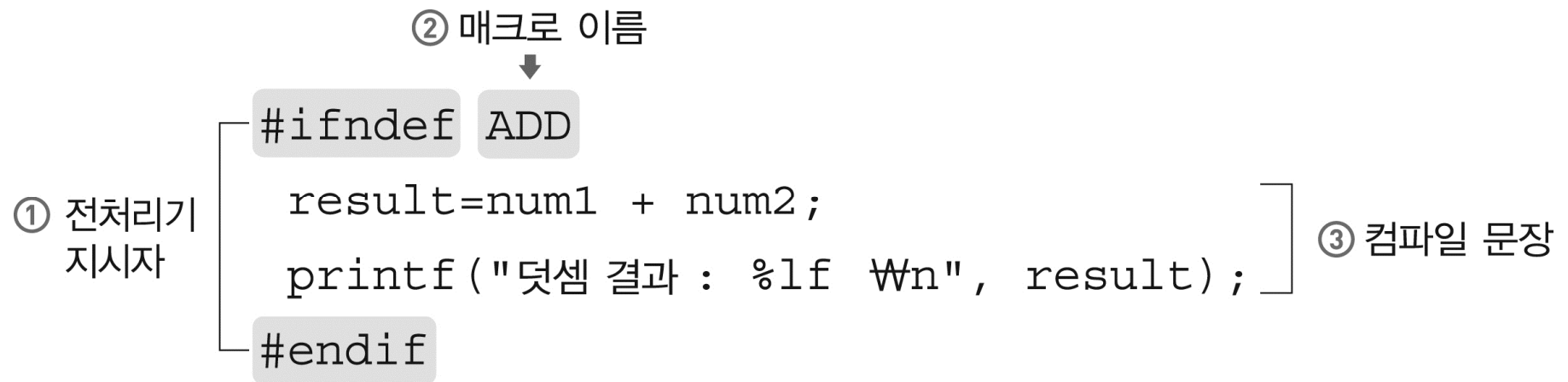
    #ifdef ADD
        result=num1 + num2;
        printf("ADD(덧셈) 결과: %lf \n", result);
    #endif

    #ifdef MUL
        result=num1 * num2;
        printf("MUL(곱셈) 결과: %lf \n", result);
    #endif

    return 0;
}
```

## 5.3 조건부 컴파일 (10/11)

### ▶ #ifndef ~ #endif의 기본 형식



‘매크로 상수 ADD가 정의 되어 있지 않다면 조건부 컴파일을 수행한다.’

## 5.3 조건부 컴파일 (11/11)---[5-16.c 실습]

```
#include <stdio.h>

#ifndef ADD
    #define ADD
#endif

#ifndef MUL
    #define MUL
#endif

int main(void)
{
    double num1=3.3, num2=1.1;
    double result=0.0;

    #ifdef ADD
        result=num1 + num2;
        printf("ADD(덧셈) 결과: %lf \n", result);
    #endif
```

```
#ifdef MUL
    result=num1 * num2;
    printf("MUL(곱셈) 결과: %lf \n", result);
#endif
    return 0;
}
```



```
C:\WINDOWS\system32\cmd...
ADD<덧셈> 결과 : 4.400000
MUL<곱셈> 결과 : 3.630000
계속하려면 아무 키나 누르십시오...
```

## 5.4 파일 분할 컴파일

## 5.4 파일 분할 컴파일 (1/19)

### ▶ 파일 분할 컴파일

✓ '여러 개의 파일로 분할된 프로그램을 실행할 때 수행 한다.'

#### ✓ 파일 분할의 장점

- '프로그램의 생산성이 높아진다.'
- '파일 단위로 에러를 수정할 수 있다.'
- '기능의 응집도가 높아져 유지 보수 용이하다.'

## 5.4 파일 분할 컴파일

### ▶ 배울 내용

① 파일 분할

② 접근 금지 static 키워드

③ #include를 이용한 사용자 헤더 파일 만들기

## 5.4 파일 분할 컴파일 (2/19)---[5-17.c 실습]

```
#include <stdio.h>

int a=6, b=3;           // 전역변수 a와 b를 선언

int main(void)
{
    int result=0;
    result=a + b;
    printf("덧셈 결과: %d \n",result);

    return 0;
}
```



## 5.4 파일 분할 컴파일 (3/19)---[5-17.c 분석]

```
int a=6, b=3;    // 전역 변수 a와 b를 선언
```

파일 5-17-1.c

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int result=0;
```

```
    result=a + b;
```

```
    printf("덧셈 결과 : %d \n", result);
```

```
    return 0;
```

```
}
```

파일 5-17-2.c

에러 : 컴파일러가 main( ) 함수의 변수 a, b를  
인식하지 못한다.

## 5.4 파일 분할 컴파일 (4/19)---[5-17.c 분석]

```
int a=6, b=3;    // 전역 변수 a와 b를 선언
```

파일 5-17-1.c

```
#include <stdio.h>
```

```
extern int a, b;
```

파일 5-17-2.c

```
int main(void)
```

```
{
```

```
    int result=0;
```

```
    result=a + b;
```

```
    printf(":덧셈 결과 %d \n", result);
```

```
    return 0;
```

```
}
```

정상 : 컴파일러가 main( ) 함수의 변수 a, b를 인식한다.

## 5.4 파일 분할 컴파일 (5/19)---[5-18.c 실습]

```
#include <stdio.h>

int num1=10, num2=20;           // 전역 변수 선언

void add(num1, num2)           // 함수 정의
{
    printf("덧셈 연산: %d \n", num1 + num2);
}

int main(void)
{
    add(num1, num2);           // 함수 호출

    return 0;
}
```

## 5.4 파일 분할 컴파일 (6/19)---[5-18.c 분석]

```
int num1=10, num2=20;
void add(num1, num2)
{
    printf("덧셈 연산 : %d \n", num1 + num2);
}
```

파일 5-18-1.c

```
#include <stdio.h>
int main(void)
{
    add(num1, num2);
    return 0;
}
```

파일 5-18-2.c

**에러** : 컴파일러가 main( ) 함수의 add( ) 함수와  
변수 num1, num2를 인식하지 못한다.

## 5.4 파일 분할 컴파일 (7/19)---[5-18.c 분석]

```
int num1=10, num2=20;
void add(num1, num2)
{
    printf("덧셈 연산 : %d Wn", num1 + num2);
}
```

파일 5-18-1.c

```
#include <stdio.h>
extern int num1, num2;
extern void add(int num1, int num2);
int main(void)
{
    add(num1, num2);
    return 0;
}
```

파일 5-18-2.c

**정상** : 컴파일러가 main( ) 함수의 add( ) 함수와  
변수 num1, num2를 인식한다.

## 5.4 파일 분할 컴파일

### ▶ 배울 내용

① 파일 분할

② 접근 금지 static 키워드

③ #include를 이용한 사용자 헤더 파일 만들기

## 5.4 파일 분할 컴파일 (8/19)

### ▶ 접근 금지 static 키워드

- ✓ 'extern' 키워드로 외부의 변수나 함수를 참조할 수 없도록 한다.'

```
static int num1=10, num2=20;    // static 전역 변수 선언
static void add(num1, num2)     // static 함수 선언
{
    printf("덧셈 연산: %d %n", num1 + num2);
}
```

5-19-1.c

```
#include <stdio.h>
```

5-19-2.c

```
extern int num1, num2;           // 에러
extern void add(int num1, int num2); // 에러

int main(void)
{
    add(num1, num2);
    return 0;
}
```



## 5.4 파일 분할 컴파일

### ▶ 배울 내용

① 파일 분할

② 접근 금지 static 키워드

③ #include를 이용한 사용자 헤더 파일 만들기



## 5.4 파일 분할 컴파일 (9/19)

### ▶ #include를 이용한 사용자 헤더 파일 만들기

#### ✓ #include <표준 라이브러리>

- 예) #include <stdio.h>, #include <string.h>, #include <stdlib.h>

#### ✓ #include "사용자 정의 라이브러리"

- 예) #include "myheader.h"

- 상대 경로(헤더파일을 현재 소스 코드가 있는 디렉터리에서 찾아 포함)

- 예) #include "D:\mylib\myheader.h"

- 절대 경로(헤더파일을 설정된 경로에서 찾아 포함)

## 5.4 파일 분할 컴파일 (10/19)---[5-20.c 실습]

```
#include <stdio.h>

#define PI 3.14
double circle(int radius);    // 원의 둘레 함수 선언(2 파이r)
double area(int radius);      // 원의 넓이 함수 선언(파이r 제곱)

int main(void)
{
    printf("반지름 3의 원의 둘레: %lf \n", circle(3));
    printf("반지름 3의 원의 넓이: %lf \n", area(3));
    return 0;
}

double circle(int radius)      // 원의 둘레 정의
{
    double result=2 * PI * radius;
    return result;
}

double area(int radius)        // 원의 넓이 정의
{
    double result=PI * radius * radius;
    return result;
}
```

## 5.4 파일 분할 컴파일 (11/19)---[5-20.c 분할]

```
#define PI 3.14
double circle(int radius)    // 원의 둘레 정의
{
    double result=2 * PI * radius;
    return result;
}
double area(int radius)      // 원의 넓이 정의
{
    double result=PI * radius * radius;
    return result;
}
```

5-20-1.c

```
#include <stdio.h>
```

```
extern double circle(int radius); // extern 생략 가능
extern double area(int radius);   // extern 생략 가능
```

```
int main(void)
{
    printf("반지름 3의 원의 둘레: %f \n", circle(3) );
    printf("반지름 3의 원의 넓이: %f \n", area(3) );
    return 0;
}
```

5-20-2.c

이 부분을 헤더파일로  
만들자!!!

## 5.4 파일 분할 컴파일 (12/19)---[파일 분할 실습]

```
#define PI 3.14
double circle(int radius)    // 원의 둘레 정의
{
    double result=2 * PI * radius;
    return result;
}
double area(int radius)      // 원의 넓이 정의
{
    double result=PI * radius * radius;
    return result;
}
```

importance.c

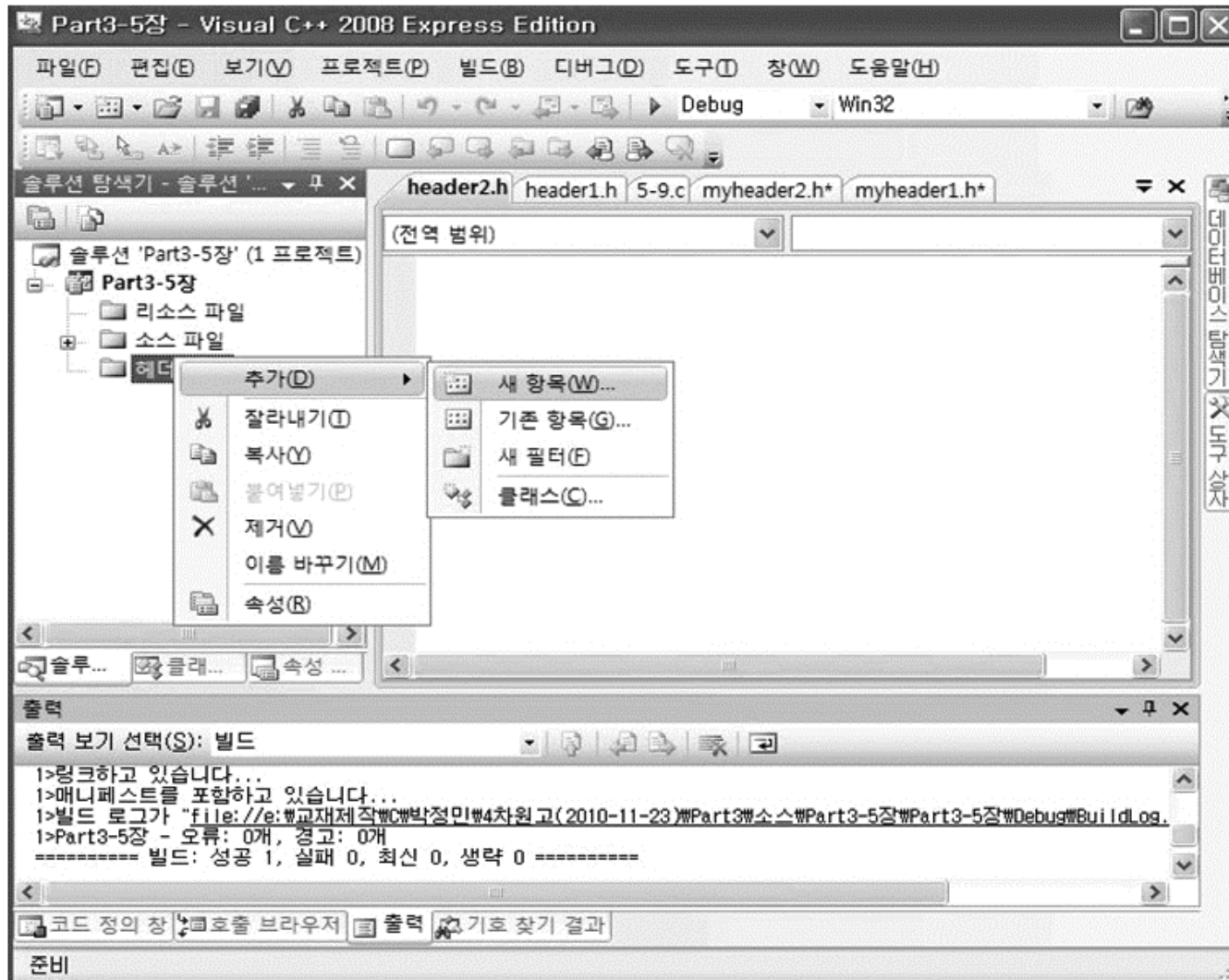
```
extern double circle(int radius); // extern 생략 가능
extern double area(int radius);   // extern 생략 가능
```

importance.h

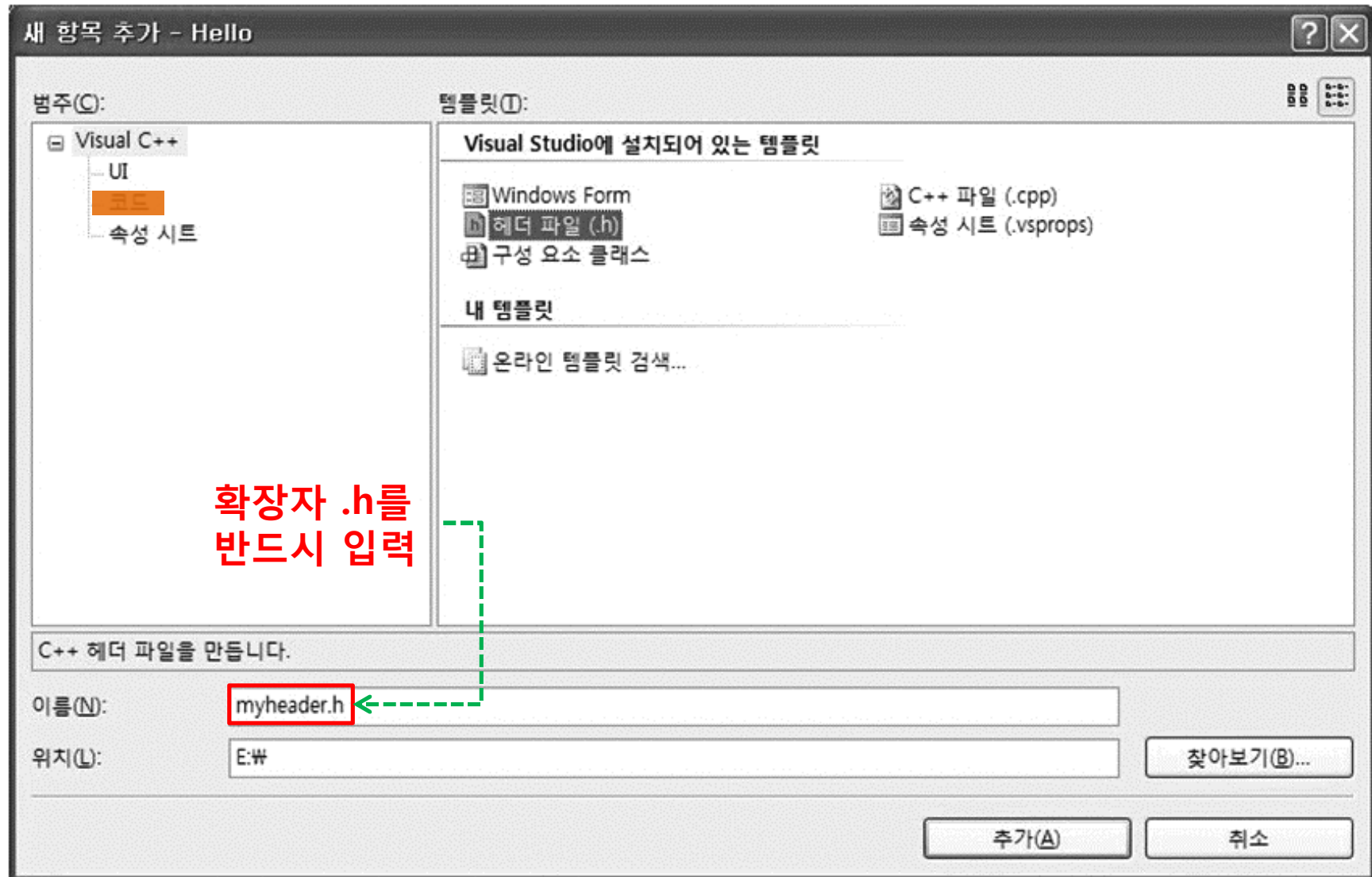
```
#include <stdio.h>
#include "importance.h"
int main(void)
{
    printf("반지름 3의 원의 둘레: %lf \n", circle(3));
    printf("반지름 3의 원의 넓이: %lf \n", area(3));
    return 0;
}
```

main.c

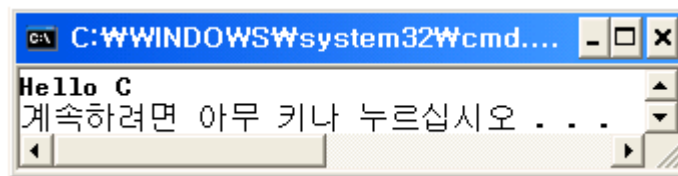
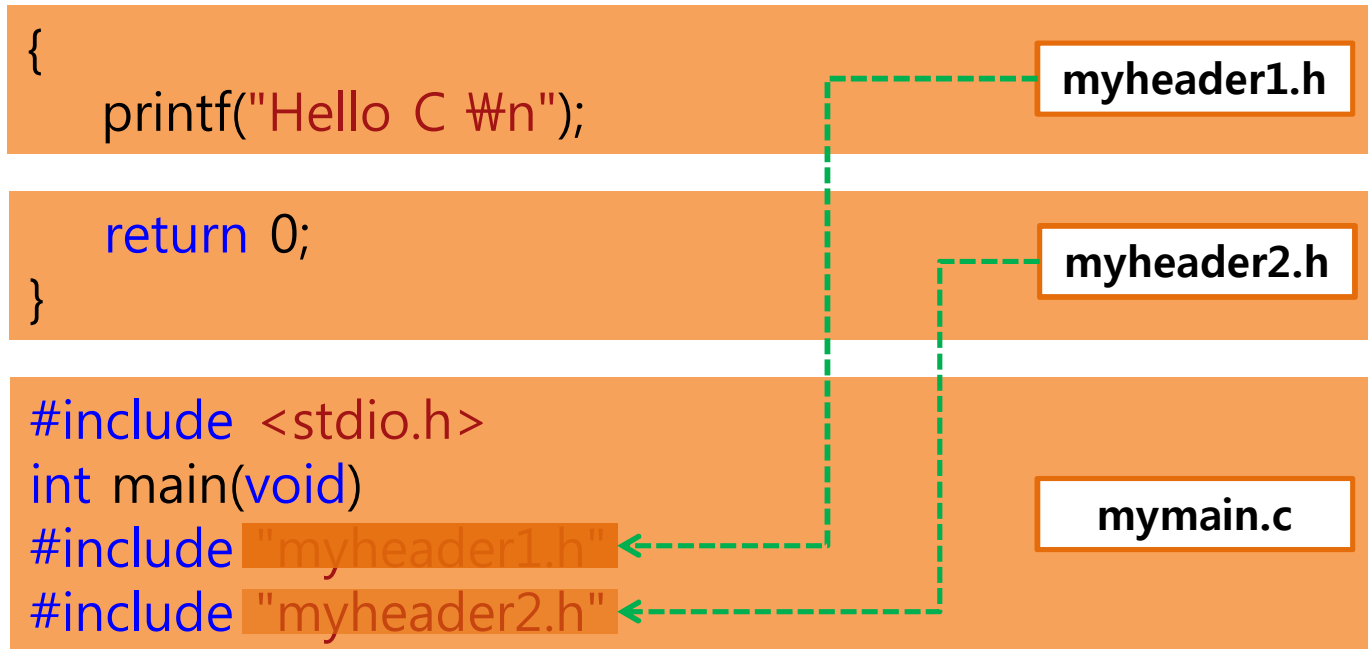
## 5.4 파일 분할 컴파일 (13/19)---[헤더파일 생성(1)]



## 5.4 파일 분할 컴파일 (14/19)---[헤더파일 생성(2)]



## 5.4 파일 분할 컴파일 (15/19)---[파일 분할 실습]



정상



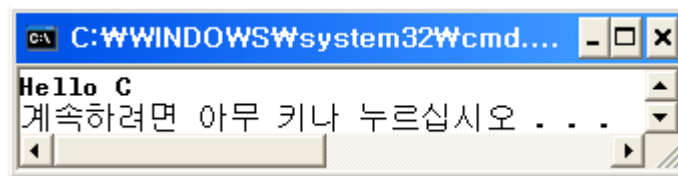
## 5.4 파일 분할 컴파일 (16/19)---[파일 분할 분석]

### ▶ #include 전처리 후

```
#include <stdio.h>
int main(void)
#include "myheader1.h"
#include "myheader2.h"
```

전처리 후

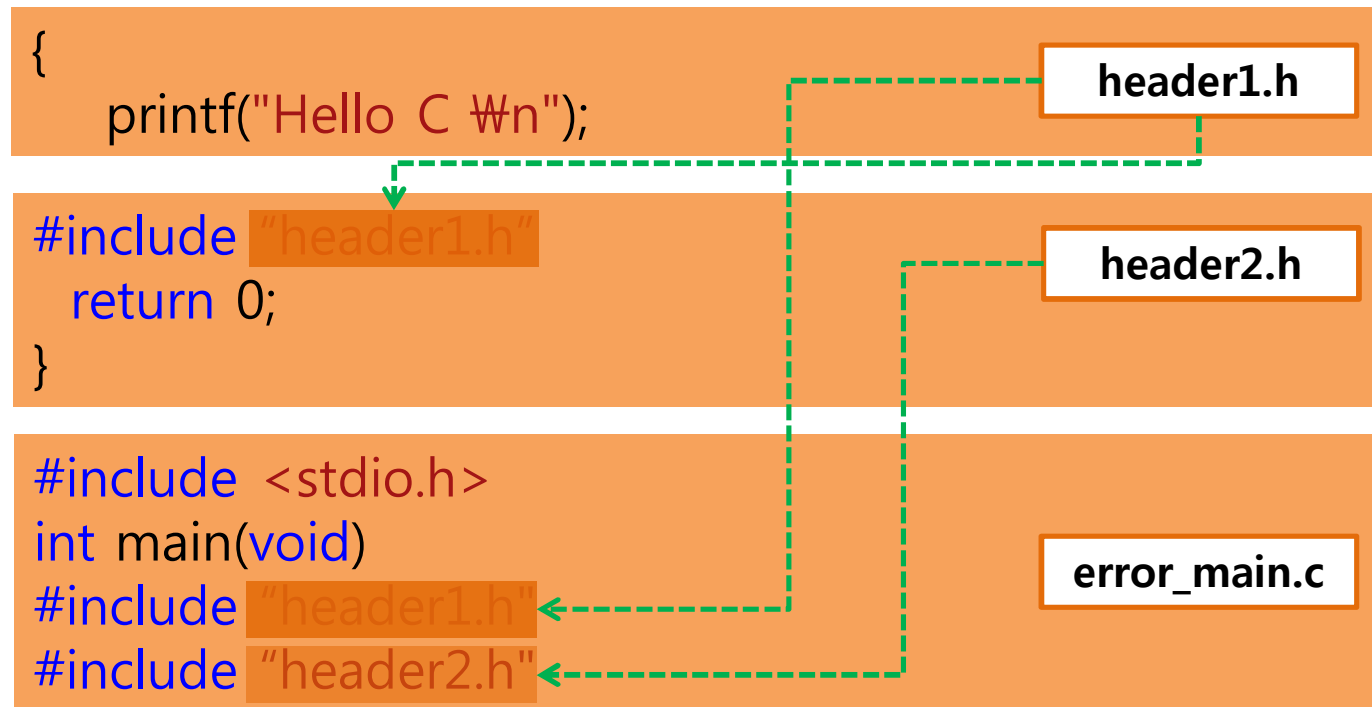
```
#include <stdio.h>
int main(void)
{
    printf("Hello C \n");
    return 0;
}
```



정상



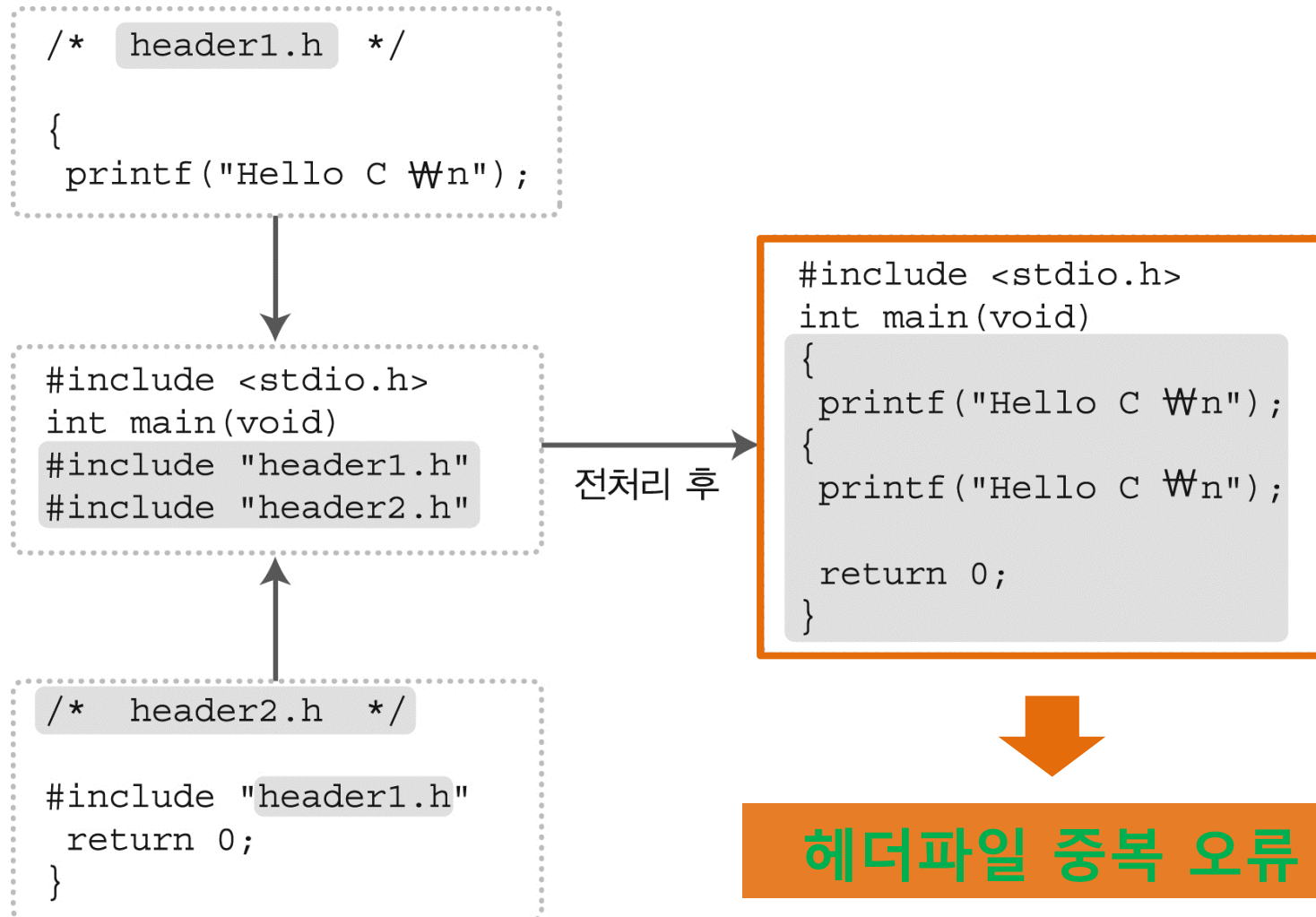
## 5.4 파일 분할 컴파일 (17/19)---[파일 분할 실습]



헤더파일 중복 오류

## 5.4 파일 분할 컴파일 (18/19)---[파일 분할 분석]

### ▶ #include 전처리 후



## 5.4 파일 분할 컴파일 (19/19)---[파일 분할 실습]

```
#ifndef HEADER
#define HEADER
{
    printf("Hello C \n");
#endif
```

```
#include "header1.h"
return 0;
}
```

```
#include <stdio.h>
int main(void)
#include "header1.h"
#include "header2.h"
```

header1.h

header2.h

error\_main.c

↓

헤더파일 중복 오류 해결

## 공부한 내용 떠올리기

---

- ▶ 전처리기 **의 의미**
- ▶ 전처리기 **지시자**의 종류
- ▶ 매크로 **를 사용하는 방법**
- ▶ 조건부 **컴파일**
- ▶ 파일 **분할 컴파일**

## 미국 명문대 입학조건 (출처: 사랑과 지혜의 탈무드)

