# 전처리기와 분할 컴파일

## - Chapter 11 -

# 학습목차

I. 전처리기

II. 매크로

III. 조건부 컴파일

IV. 파일 분할 컴파일

# 전처리기

## 전처리기 개요

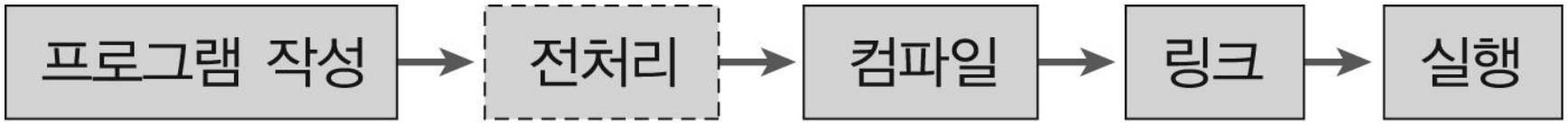
### 전처리

- 소스파일을 컴파일 하기 전에 먼저 처리해야 하는 일

### 전처리기

- 전처리를 수행하는 장치(# 문자로 시작)

### 결국 전처리기는 소스 코드가 컴파일되기 전에 먼저 처리해야 하는 작업을 수행하는 장치




### 예) 전처리 지시자

- #include - 헤더 파일을 인클루드
- #define - 매크로 상수를 정의

# 전처리기

## 전처리기의 사용

### 전처리기는 #문자로 시작

#### #문자로 시작하기 때문에 구별이 쉬움

### 뒤에 세미콜론을 사용하지 않음

#### C언어에서는 구문의 끝을 분류하기 위하여 세미콜론(;)을 붙이지만 전처리기는 붙이지 않음

#### 일반 명령코드는 소스상에서 한 줄에 여러 구문을 작성하고 세미콜론으로 구분하지만 전처리기는 한 줄에 하나의 지시자만 사용

```
#include <stdio.h>
#define MAX 100

int main(int argc, char *argv[]) {
        int i = MAX;
        printf("i : %d MAX : %d₩n", i, MAX);
        return 0;
}
```

```
i : 100 MAX : 100

--------------------------------
```

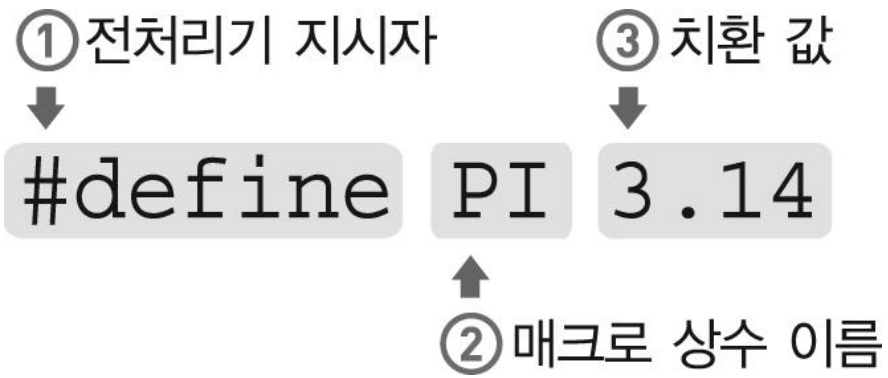
# 전처리기

- 전처리 지시자의 종류
  - 헤더 파일을 include하거나
  - 특정 문자열(예:MAX)을 상수(예:100)나 문자로 치환하거나
  - 조건에 따라서 코드의 일부를 컴파일 하거나, 컴파일 하지 못하게 하는 선택 기능을 제공

| 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 |

# 매크로

- 매크로(Macro)
  - #define으로 시작되는 전처리 문장을 매크로라고 함
  - 매크로 상수와 매크로 함수로 분류
- 매크로 상수
  - 전처리기 지시자:  매크로 상수를 선언하기 위해서 #define를 지정
  - 매크로 상수 이름:  매크로 상수의 이름을 지정
  - 치환값: 매크로 상수에 치환되는 값 지정



# 매크로

- 원의 넓이를 계산하는 소스 코드
  - 아래 코드에서는 원주율을 3.14로 사용
  - 정밀도를 높이기 위하여 원주율을 3.141592로 변경하려면?

```
#include <stdio.h>
int main(void)
{
        double area, circum, radius;
        printf("반지름을 입력하시오 : ");
        scanf("%lf", &radius);
        area=3.14 * radius * radius;
        circum=2 * 3.14 * radius;
        printf("원의 넓이 : %lf ₩n", area);
        printf("원의 둘레 : %lf ₩n", circum);
        return 0;
}
```

```
반지름을 입력하시오 : 2.5
원의 넓이 : 19.625000
원의 둘레 : 15.700000
```

# 매크로

- 원의 넓이를 계산하는 소스 코드
  - 매크로 상수의 값만 변경해주면 됨

```
#include <stdio.h>
#define PI 3.141592                                    //#define PI 3.14
int main(void)
{
                        double area, circum, radius;
                        printf("반지름을 입력하세요 :");
                        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;
}
```

```
반지름을 입력하세요 : 2.5
원의 넓이 : 19.634950
원의 둘레 : 15.707960

--------------------------------
```

# 매크로

## 매크로 상수의 다양한 사용

### 실수형 상수, 문자열 상수, 함수이름이나 자료형도 매크로 상수로 정의가능

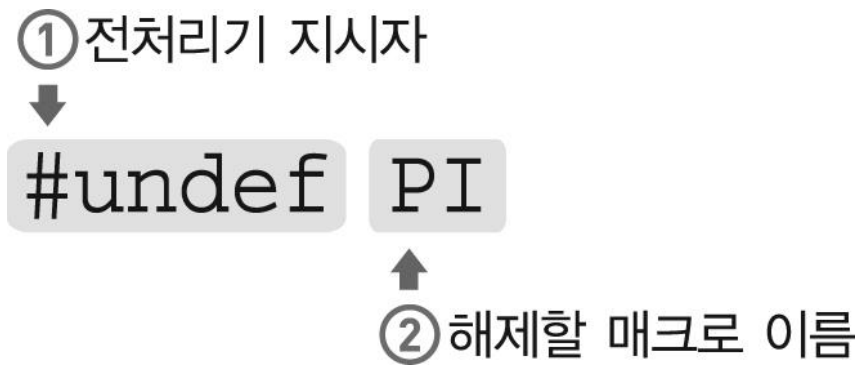
```
#include <stdio.h>

#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;
}
```

```
100, 3.140000, Hello C, 3

--------------------------------
```

# 매크로

## 매크로 해제

- 매크로의 선언을 해제하기 위해서 #undef를 지정
  - 보통 매크로는 한번 정의하면 소스코드 전체에 일관되게 적용하기 때문에 #undef를 사용하는 경우는 흔치 않음
  - #undef를 사용하여 기존 매크로를 재정의할 수 있음
- #define과 반대로 정의된 매크로를 해제하고 #undef를 사용한 이후부터 치환을 중지
- 해제할 매크로 이름 지정
  - 미리 정의된 매크로 상수이름을 지정



# 매크로

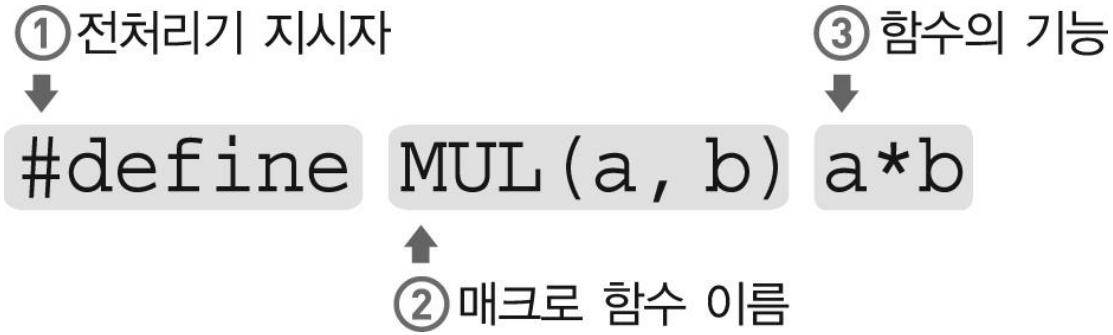
▶매크로 해제 예제

```
#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;
}
```

```
변경 전 : 100, 3.140000
변경 후 : 1000, 3.141592
```

# 매크로

- 매크로 함수
  - 함수처럼 인자를 설정할 수 있는 매크로이며 매크로 함수라고 불리지만 단순히 치환하기만 하므로 실제로 함수는 아님
  - 매개변수의 자료형을 신경 쓰지 않음
    - 자료형의 독립성 보장
- 매크로 함수 구조
  - 전처리기 지시자
    - 매크로 함수를 선언하기 위해서 #define을 지정
  - 매크로 함수 이름
    - 사용될 매크로 함수의 이름을 지정
  - 함수의 기능
    - 매크로 함수 이름에 치환되는 함수의 기능



# 매크로

## 매크로 함수 예제

### 자료형의 독립성 보장

```
#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;
}
```

```
두개의 정수를 입력하세요: 10 20
10 * 20 = 200
두개의 실수를 입력하세요: 2.5 3.8
2.500000 * 3.800000 = 9.500000
```

# 매크로

- 매크로 함수의 특징
  - 매크로 함수의 장점
    - 함수의 인자(매개변수)에 대한 자료형의 독립성 보장
      - int형이나 double형의 변수를 별도로 선언하지 않아도 됨
    - 속도가 빠름
  - 매크로 함수의 단점
    - 매크로 함수 내부에서 자기 자신을 호출할 수 없음
    - 한 줄 정도의 간단한 내용만 매크로 함수로 정의해야 함

# 매크로

▶매크로 함수의 문제점 확인

```
#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;
}
```

```
두 개의 정수를 입력 하세요: 10 20
매크로함수호출결과: 31
일반함수호출결과: 231

--------------------------------
```

# 매크로

## 매크로 함수의 문제점

- 매크로 함수는 단순히 치환만 함으로 곱셈의 우선순위에 의해 a+1 * b+1 ➔ a + b + 1
  - a+1과 b+1을 x와 y의 위치에 단순히 치환
- 일반 함수는 (a+1)*(b+1)로 연산

```
#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))
```

# 매크로

## # 연산자

### 매크로 함수의 인자를 문자열로 변경해줌

```
#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;
}
```

```
1234
1234
```

# 매크로

## # 연산자 예제

### 인자가 문자열로 처리되는 것을 확인

```
#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;
}
```

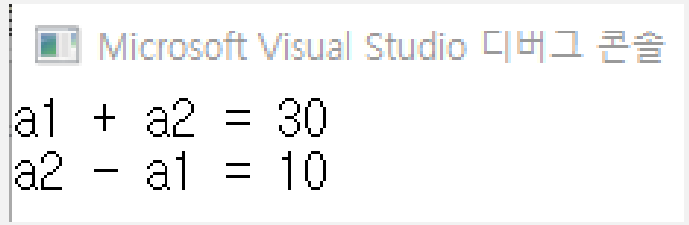
```
33
11+22
```

# 매크로

## ## 연산자

- 토큰(문법 분석의 단위, 예: 숫자, 콤마, 연산자, 식별자 등) 결합 연산자
  - C언어에서 토큰은 컴파일러가 인식하는 문자나 문자열의 최소 단위
  - 예를 들어 **int num = x + y;** 라는 코드는 int, num, =, x, +, y, ;로 나뉘어서 7개의 토큰으로 분류됨
- 매크로 함수 안에서 토큰을 결합하는 기능을 수행

```
#include <stdio.h>

#define PRINT_EXPR(x) printf(#x " = %d₩n", x)
#define NAME_CAT(x, y) (x ## y)

int main(void)
{
        int a1, a2;

        NAME_CAT(a, 1) = 10;        // (a1) = 10;
        NAME_CAT(a, 2) = 20;        // (a2) = 20;
        PRINT_EXPR(a1 + a2);        // printf("a1 + a2" " = %d₩n", a1 + a2);
        PRINT_EXPR(a2 -  a1);       // printf("a2 - a1" " = %d₩n", a2 - a1);

        return 0;
}
```



# 매크로

## 미리 정의된 매크로

### C언어에서 개발자의 편의를 위해 미리 정의된 매크로

| 미리 정의된 매크로 | 설명 |
|---|---|
| _ _FILE_ _ | 현재 소스 코드의 파일 이름을 나타내는 매크로, %s 사용 |
| _ _LINE_ _ | 현재 위치의 소스 코드의 행 번호를 나타내는 매크로, %d 사용 |
| _ _DATE_ _ | 현재 소스 코드의 컴파일 날짜를 나타내는 매크로, %s 사용 |
| _ _TIME_ _ | 현재 소스 코드의 컴파일 시간을 나타내는 매크로, %s 사용 |
| _ _FUNCTION_ _ | 현재 위치의 함수명을 나타내는 매크로, %s 사용 |

# 매크로

- 미리 정의된 매크로 실습
  - _ _는 실제코드에서 붙여서 사용 ➔ __
  - 아래 예제에서는 확인하기 쉽도록 띄어씀 __

```c
#include <stdio.h>

void func(void);

int main(void)
{
    printf("컴파일 날짜와 시간 : %s, %s\n\n", __DATE__, __TIME__);
    printf("파일명 : %s\n", __FILE__);
    printf("함수명 : %s\n", __FUNCTION__);
    printf("행번호 : %d\n", __LINE__);
#line 100 "macro.c"     // 행 번호를 100부터 시작, 파일명은 macro.c로 표시
    func();             // 여기부터 행 번호는 100으로 시작

    return 0;
}

void func(void)
{
    printf("\n");
    printf("파일명 : %s\n", __FILE__);
    printf("함수명 : %s\n", __FUNCTION__);
    printf("행번호 : %d\n", __LINE__);
}
```

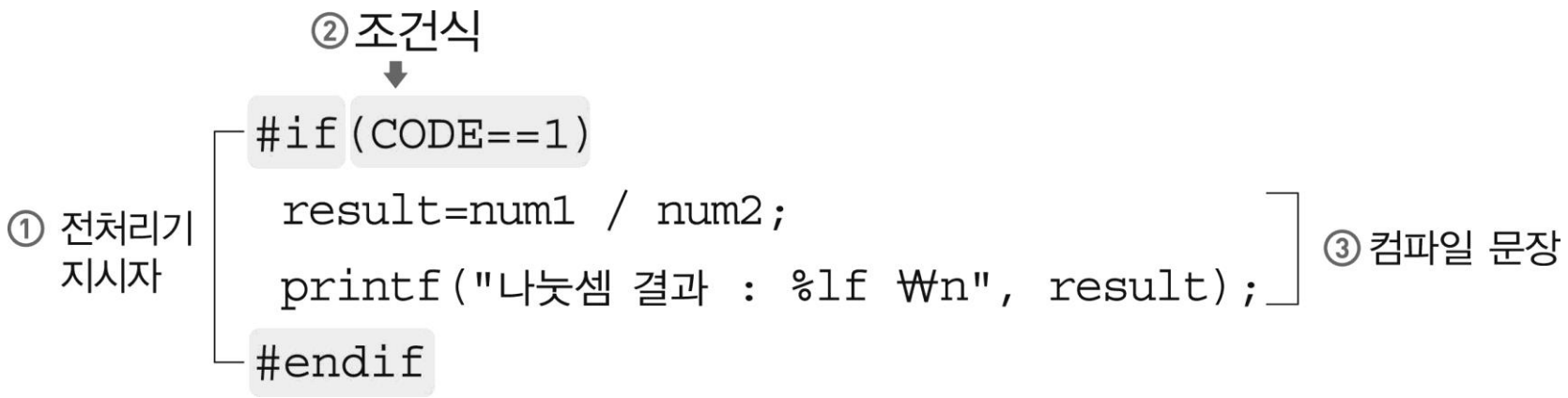
```
Microsoft Visual Studio 디버그 콘솔
컴파일 날짜와 시간 : Dec 16 2019, 08:36:37

파일명 : c:₩users₩choe₩source₩repos₩helloworld₩helloworld.c
함수명 : main
행번호 : 10

파일명 : macro.c
함수명 : func
행번호 : 110
```

# 조건부 컴파일

- 조건부 컴파일
  - 프로그램의 실행 환경을 고려하지 않고 프로그래밍을 하게 되면 유지보수가 어려울 수 있음
    - C언어는 다양한 OS에서 사용되면 각 운영체제마다 제공되는 표준 함수나 동작 방식이 다를 수 있음
    - 동일한 운영체제라도 사용하는 컴파일러와 라이브러리에 따라서 일부 함수가 없는 경우도 있음
  - 이러한 문제점을 해결하는 것이 조건부 컴파일
    - 특정 조건에 만족할 때만 코드가 컴파일
    - 주로 매크로 상수의 존재 유무나 매크로 상수 값을 검사하여 수행
    - 조건에 따라 특정 코드를 컴파일 하도록 만들 수 있어서 이식성있는 코드를 개발하는데 도움을 줌

# 조건부 컴파일

- #if ~ #endif문
  - 전처리 지시자
    - 조건부 컴파일 수행 문장을 #if와 #endif로 묶음
  - 조건식
    - 컴파일을 수행하기 위한 조건을 지정
    - 산술 연산자, 관계 연산자, 논리 연산자 등을 사용할 수 있음
  - 컴파일 문장
    - 조건식이 '참'일 때 컴파일해야 하는 문장 삽입



# 조건부 컴파일

## #if ~ #endif문 예제

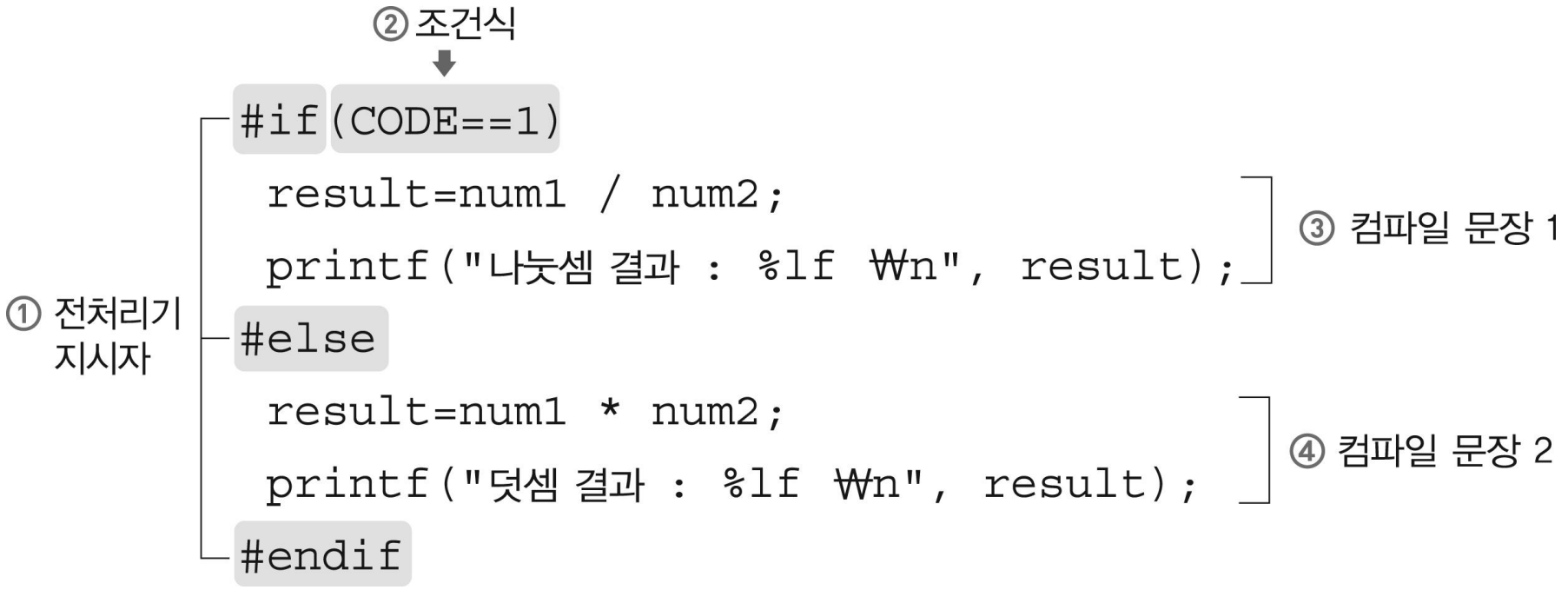
### 매크로 상수인 CODE의 값에 따라서 컴파일 되는 구역 선택

```
#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;
}
```

```
실수 두개를 입력하세요>>1.1 2.2
덧셈 결과: 3.300000
```

# 조건부 컴파일

- #if~#else~#endif
  - if ~ else문과 유사하게 사용



# 조건부 컴파일

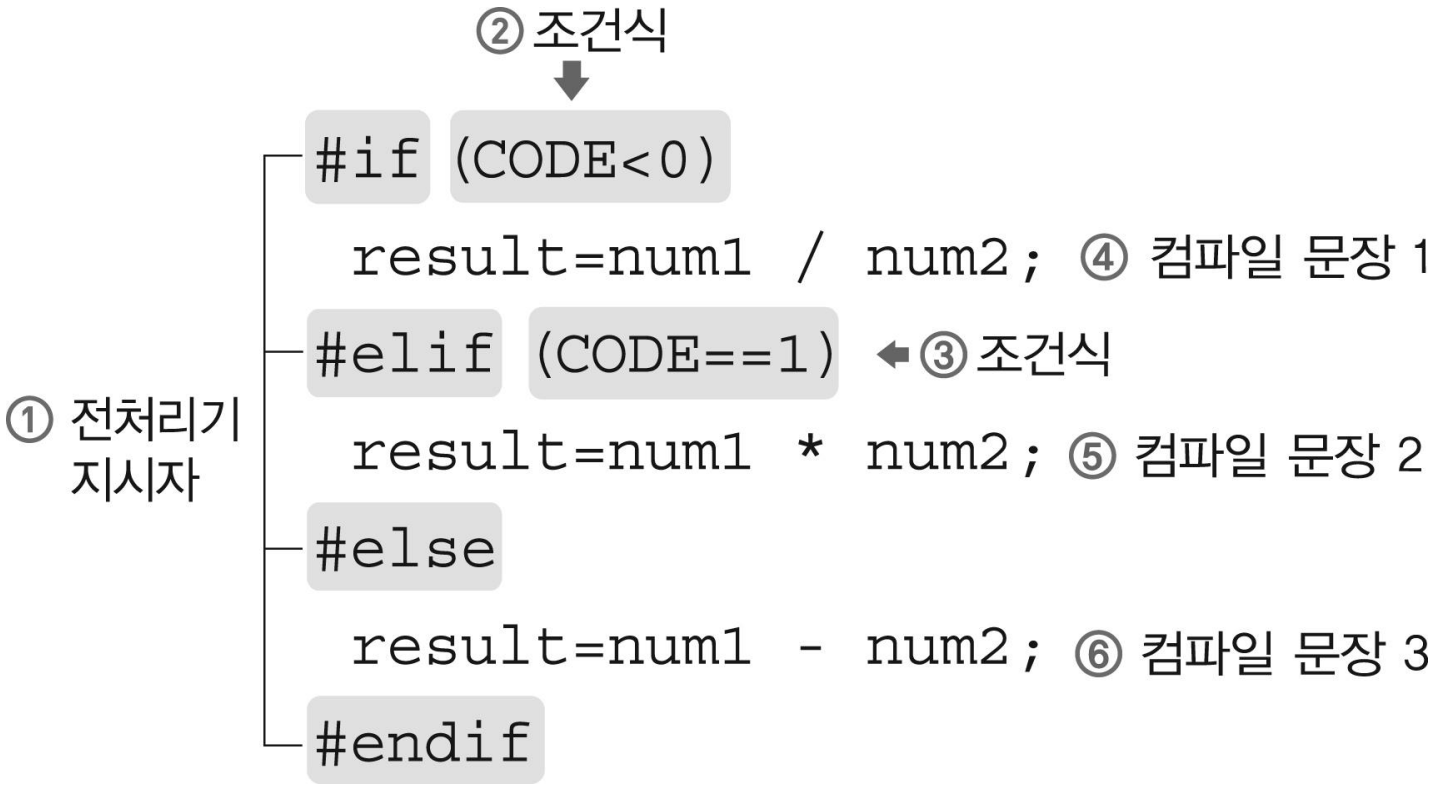
## #if~#else~#endif 예제

### 매크로 상수에 따라 컴파일되는 코드가 달라짐

```
#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;
}
```

# 조건부 컴파일

- #if~#elif~#else~#endif
  - 컴파일 조건이 추가됨



# 조건부 컴파일

## #if~#else~#endif 예제

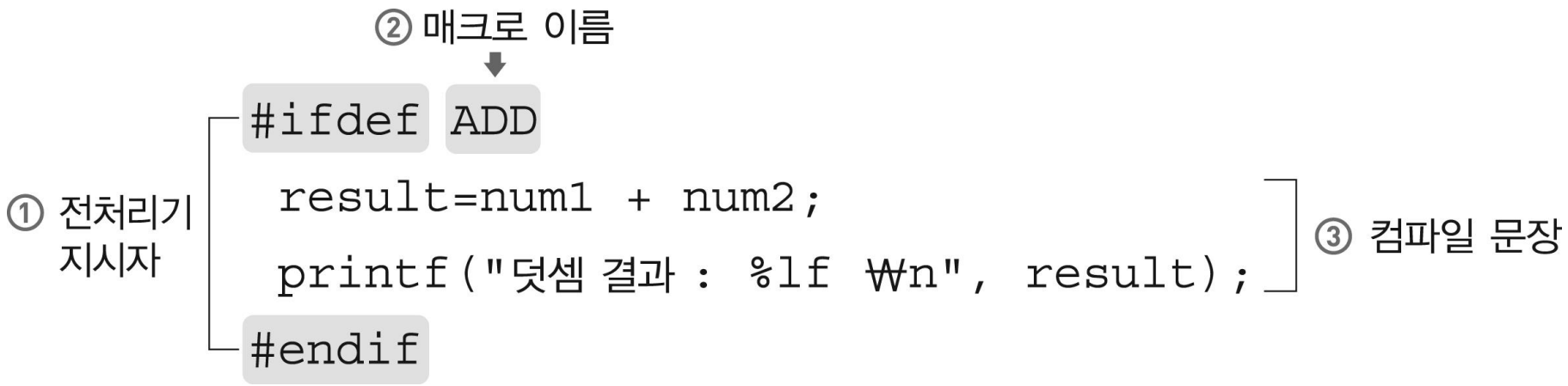
### 매크로 상수에 따라 컴파일되는 코드가 달라짐

```
# 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;
}
```

```
실수 두개를 입력하세요>>1.2 2.3
행 번호 : 18
덧셈 결과: 3.500000
```

# 조건부 컴파일

- #ifdef ~ #endif
  - ifdef문 뒤에 지정된 매크로 이름이 선언되어 있다면 컴파일을 수행




  - 위의 예제는 매크로 상수 ADD가 정의 되어 있다면 조건부 컴파일을 수행

# 조건부 컴파일

▶#ifdef ~ #endif 예제

```
#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;
}
```

```
ADD(덧셈) 결과: 4.400000
MUL(곱셈) 결과: 3.630000
```

# 파일 분할 컴파일

- 파일 분할 컴파일
  - 지금까지 실습 시에는 하나의 소스파일에서 프로그래밍을 진행하였음
    - 실제로 프로그램을 개발 할 시 프로그램의 내용이 복잡해져서 관리가 쉽지 않고 컴파일 시간이 오래 걸림
  - 파일 분할 컴파일은 여러 개의 소스 파일로 분할된 프로그램을 실행할 때 사용
- 파일 분할의 장점
  - 프로그램의 생산성이 높아짐
    - 여러 사람이 각각 맡은 소스코드를 수정
  - 파일 단위로 에러를 수정
    - 기능별로 분류되어 있기때문에 오류가 발생된 곳을 찾기 쉬움
    - 수정 후에는 프로젝트 전체를 다시 컴파일하지 않아도 되므로 빌드 시간도 절약됨
  - 기능의 응집도가 높아져 유지 보수 용이
    - 특정 동작에 연관된 코드들만 따로 분류해 놓음으로써 유지 보수 및 재사용성이 높아짐

# 파일 분할 컴파일

- 파일 분할
  - 아래 예제 파일을 분할

```
#include <stdio.h>

int num1 = 6, num2 = 3;                    // 전역변수 a와 b를 선언

int main(void)
{
        int result=0;
        result= num1 + num2;
        printf("덧셈 결과: %d ₩n",result);

        return 0;
}
```

# 파일 분할 컴파일

- 소스코드 추가
  - 프로젝트에서 마우스 우 클릭 후 var_test.c 파일 추가
  - 아래와 같이 각각의 소스 파일에 아래와 같이 코드 입력
  - 컴파일러가 main.c 파일에서 main()함수의 변수 a, b를 인식하지 못함

```
//main.c
#include <stdio.h>

int main(void)
{
        int result=0;
        result= num1 + num2;
        printf("덧셈 결과: %d ₩n",result);

        return 0;
}
```

```
//var_test.c
int num1 = 6, num2 = 3;
```

```
In function 'main':
[Error] 'a' undeclared (first use in this function)
[Note] each undeclared identifier is reported only once for each function it appears in
[Error] 'b' undeclared (first use in this function)
recipe for target 'main.o' failed
```

# 파일 분할 컴파일

- extern 키워드
  - 외부 파일에서 선언된 내용(변수, 함수)을 참조
  - 함수는 extern 키워드를 생략 가능
  - extern 키워드를 이용하여 var_test.c에 선언된 a, b변수를 참조

```
//main.c
#include <stdio.h>

extern num1, num2;

int main(void)
{
        int result=0;
        result= num1 + num2;
        printf("덧셈 결과: %d ₩n",result);

        return 0;
}
```

```
//var_test.c
int num1=6, num2=3;
```

# 파일 분할 컴파일

- extern 키워드 추가 예제
  - 아래 코드(main.c)에서 num1, num2 변수와 add 함수를 다른 소스 파일에 따로 분리

```
#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;
}
```

# 파일 분할 컴파일

- extern 키워드 추가 예제
  - 분리된 add함수를 var_test.c 파일로 이동
  - 함수는 extern 키워드 생략가능
    - extern add(int num1, int num2); ➔ add(int num1, int num2);

```
//main.c
#include <stdio.h>

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

        return 0;
}
```

```
//var_test.c
#include <stdio.h>

int num1=10, num2=20;    // 전역 변수 선언

void add(int a, int b)      // 함수 정의
{
        printf("덧셈 연산: %d ₩n", a + b);
}
```

# 파일 분할 컴파일

- static 키워드
  - static 키워드는 지역변수와 전역변수에 따라 각각 다른 기능을 함
- 지역 변수의 static 키워드
  - 지역변수에 static 키워드를 사용하면 프로그램의 실행과 동시에 메모리에 할당
  - 해당 지역(블록)을 벗어나도 메모리에서 사라지지 않음
  - 따로 초기화하지 않을 경우 0으로 초기화

# 파일 분할 컴파일

▶static 변수 예제

```
//일반 변수 예제
void testFunc();
int main(){
        testFunc();
        testFunc();
        testFunc();
        return 0;
}
void testFunc(){
        int i = 1;
        printf("i : %d₩n",i);
        i++;
}
```

```
i : 1
i : 1
i : 1
```

```
//static 변수 예제
void testFunc();
int main(){
        testFunc();
        testFunc();
        testFunc();
        return 0;
}
void testFunc(){
        static int i = 1;
        printf("i : %d₩n",i);
        i++;
}
```

```
i : 1
i : 2
i : 3
```

# 파일 분할 컴파일

- 전역 변수의 static 키워드
  - 접근금지의 의미
  - 시스템 프로그램과 관련되거나 중요한 변수나 함수를 extern 키워드로 참조한다면 문제가 발생할 수 있음
  - 외부 변수를 extern 키워드로 참조할 수 없도록, 해당 변수를 static 키워드로 선언
- static키워드를 추가한 예제
  - 오류 발생

```
//main.c
#include <stdio.h>

extern num1, num2;
extern add(int num1, int num2);
int main(void)
{
        add(num1, num2);            // 함수 호출
        return 0;
}
```

```
//var_test.c
#include <stdio.h>

static int num1=10, num2=20;

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

# 파일 분할 컴파일

- #include를 이용한 사용자 헤더 파일 만들기
  - 전처리 지시자인 #include를 통하여 헤더 파일(.h)이 소스코드 내부에 포함됨
- 사용 방법(<>, “”)
  - #include <표준 라이브러리>
    - 예) #include <stdio.h>, #include <string.h>, #include <stdlib.h>
  - #include “사용자 정의 라이브러리”
    - 상대 경로(헤더파일을 현재 소스 코드가 있는 디렉터리에서 찾아 포함)
      - 예) #include “myheader.h”
    - 절대 경로(헤더파일을 설정된 경로에서 찾아 포함)
      - 예) #include “D:₩mylib₩myheader.h”

# 파일 분할 컴파일

▶#include를 이용한 사용자 헤더 파일 만들기 예제(1)

▶아래 예제를 분할하고 헤더파일을 생성

```
#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;
}
```

```
반지름 3의 원의 둘레: 18.840000
반지름 3의 원의 넓이: 28.260000
```

# 파일 분할 컴파일

## #include를 이용한 사용자 헤더 파일 만들기 예제(2)

### 외부 함수 참조 부분을 헤더파일로 생성

```c
//main.c
#include <stdio.h>

extern double circle(int radius);
extern double area(int radius);

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

```c
//circleFunc.c
#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;
}
```

# 파일 분할 컴파일

▶#include를 이용한 사용자 헤더 파일 만들기 예제(3)

```
//circleHeader.h
double circle(int radius);
double area(int radius);
```

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

```
//circleFunc.c
#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;
}
```

```
반지름 3의 원의 둘레: 18.840000
반지름 3의 원의 넓이: 28.260000

--------------------------------
```

# 분할 컴파일

- 분할 컴파일
  - 여러 개의 소스코드 파일로 나누어 구현한 후 각각 컴파일하여 하나의 프로그램(실행 파일)을 만드는 일
- 분할 컴파일과 데이터 공유

| 구분 | 사용 예 | 기능 |
|---|---|---|
| 분할 컴파일 과정 | main.c sub.c | 2개의 소스 파일 작성 |
| | main.obj sub.obj | 각각 컴파일 후 개체 파일 생성 |
| | yuni.exe | 링크로 실행 파일 생성 |
| 전역 변수의 공유 | extern int cnt; | 다른 파일의 전역 변수 cnt 사용 |
| | static int tot; | 다른 파일에서 공유할 수 없도록 제한 |
| 헤더 파일의 중복 | #ifndef _POINT_H_ | 매크로명이 정의되어 있지 않으면 |
| | #define _POINT_H_ | 매크로명을 정의하고 |
| | Point 구조체 선언 | 헤더 파일의 내용 처리 |
| | #endif | #ifndef의 끝 표시 |

# 분할 컴파일

## 분할 컴파일 예제

### main.c

```
#include <stdio.h>

void input_data(int *, int *);        // 두 정수를 입력하는 함수 선언
double average(int, int);             // 평균을 구하는 함수 선언

int main(void)
{
        int a, b;
        double avg;

        input_data(&a, &b);                                  // 두 정수 입력
        avg = average(a, b);                                 // 평균 계산
        printf("%d와 %d의 평균 : %.1lf₩n", a, b, avg);   // 입력값과 평균 출력

        system("pause");

        return 0;
}
```

```
두 정수 입력 : 5 8
5와 8의 평균 : 6.5
계속하려면 아무 키나 누르십시오 . . .
```

# 분할 컴파일

## 분할 컴파일 예제

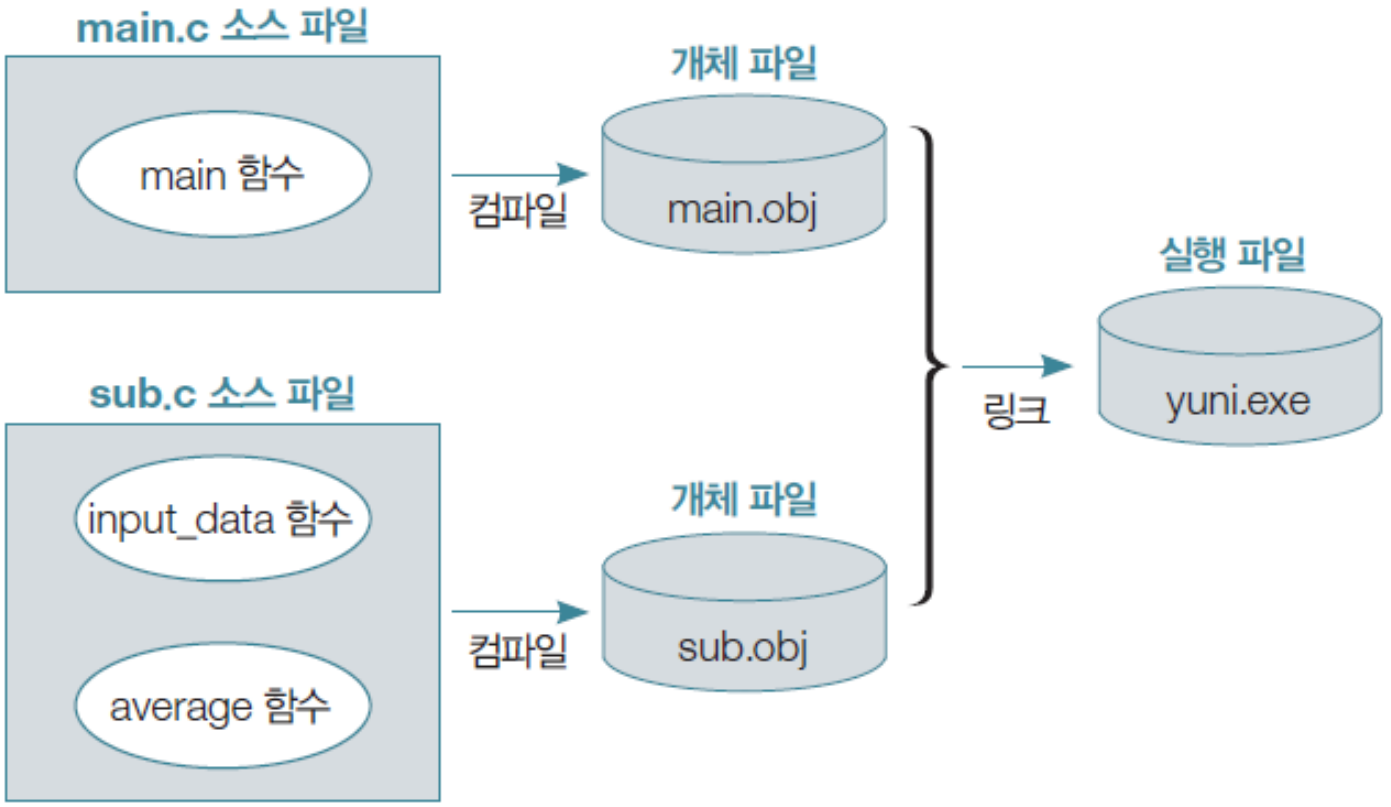
### sub.c

```
#include <stdio.h>                    // printf, scanf 함수 사용을 위해 필요

void input_data(int *pa, int *pb)    // 두 정수 입력 함수
{
        printf("두 정수 입력 : ");
        scanf("%d %d", pa, pb);
}

double average(int a, int b)         // 평균을 구하는 함수
{
        int tot;
        double avg;

        tot = a + b;
        avg = tot / 2.0;

        return avg;
}
```

```
두 정수 입력 : 5 8
5와 8의 평균 : 6.5
계속하려면 아무 키나 누르십시오 . . .
```

# 분할 컴파일

- 분할 컴파일 방법
  - 각 파일은 개별적으로 컴파일된 후 링크 단계에서 합쳐져 하나의 실행 파일이 됨
  - 개체 파일명 - 각 소스 파일과 같은 이름으로 만들어짐
  - 실행 파일명 - 프로젝트 이름으로 만들어짐



# 분할 컴파일

- 헤더 파일의 필요성
  - 분할 컴파일할 때는 사용자 정의 헤더 파일 필요
    - 각 소스 파일을 독립적으로 컴파일할 수 있도록 필요한 변수와 함수의 선언을 포함해야 함
  - 헤더 파일은 텍스트 파일로 소스코드의 일부를 따로 만들어 필요한 파일에서 인클루드
  - 헤더 파일은 각 파일에 공통으로 필요한 코드 모아 작성
  - 수정된 내용을 빠르고 편리하게 반영 가능
  - 구조체의 중복선언 같은 문제가 아니면 편리하게 사용 가능
    - 구조체를 중복 선언하는 경우 오류 발생
      - 멤버의 구성은 다르지만 동일한 이름의 구조체를 중복하여 선언하는 경우 사용시 어떤 구조체인지 구분할 수 없으므로 오류 발생

# 분할 컴파일

- 헤더파일의 중복 문제
  - point.h

```
typedef struct
{
        int x;          // x좌표
        int y;          // y좌표
} Point;
```

# 분할 컴파일

- 헤더파일의 중복 문제
  - line.h

```
#include "point.h"   // Point 구조체를 위해 포함

typedef struct
{
        Point first;     // 첫 번째 점
        Point second;    // 두 번째 점
} Line;
```

# 분할 컴파일

## 헤더파일의 중복 문제

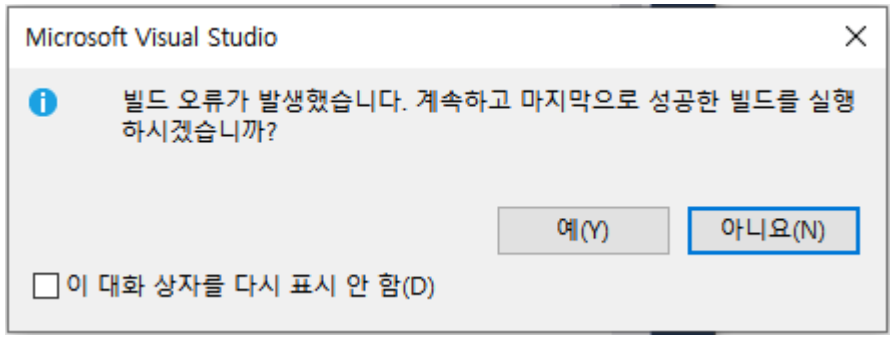
### main.c

```
#include <stdio.h>
#include "point.h"                    // Point 구조체 선언
#include "line.h"                     // Line 구조체 선언

int main(void)
{
        Line a = { {1, 2}, {5, 6} };        // Line 구조체 변수 초기화
        Point b;                            // 가운데 점의 좌표 저장

        b.x = (a.first.x + a.second.x) / 2;  // 가운데 점의 x좌표 계산
        b.y = (a.first.y + a.second.y) / 2;  // 가운데 점의 y좌표 계산
        printf("선의 가운데 점의 좌표 : (%d, %d)₩n", b.x, b.y);

        return 0;
}
```

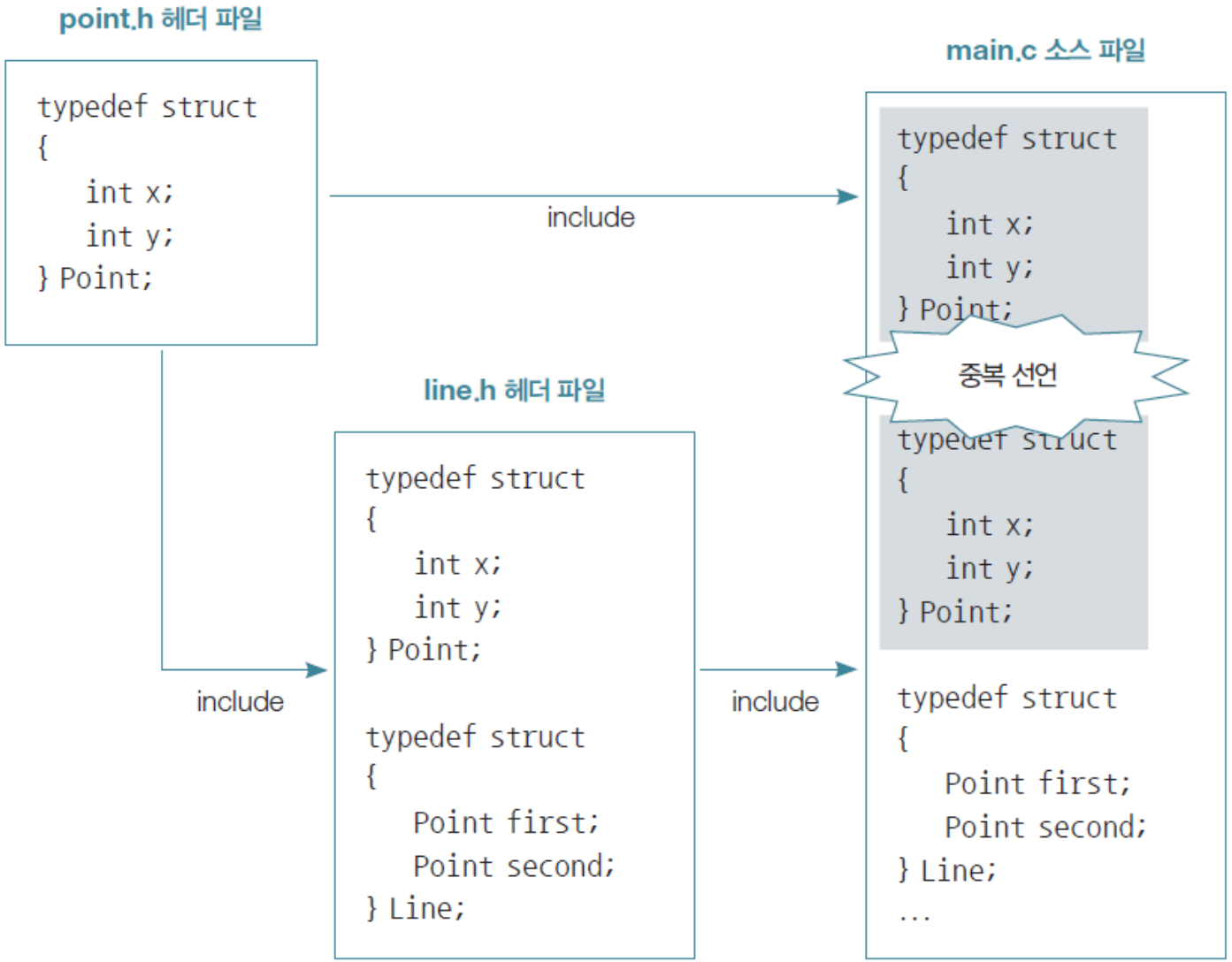
# 분할 컴파일

## 헤더파일의 중복 문제

### Point 구조체의 중복된 선언으로 인한 컴파일 오류




```
error C2371: 'Point': 재정의. 기본 형식이 다릅니다.
note: 'Point' 선언을 참조하십시오.
```



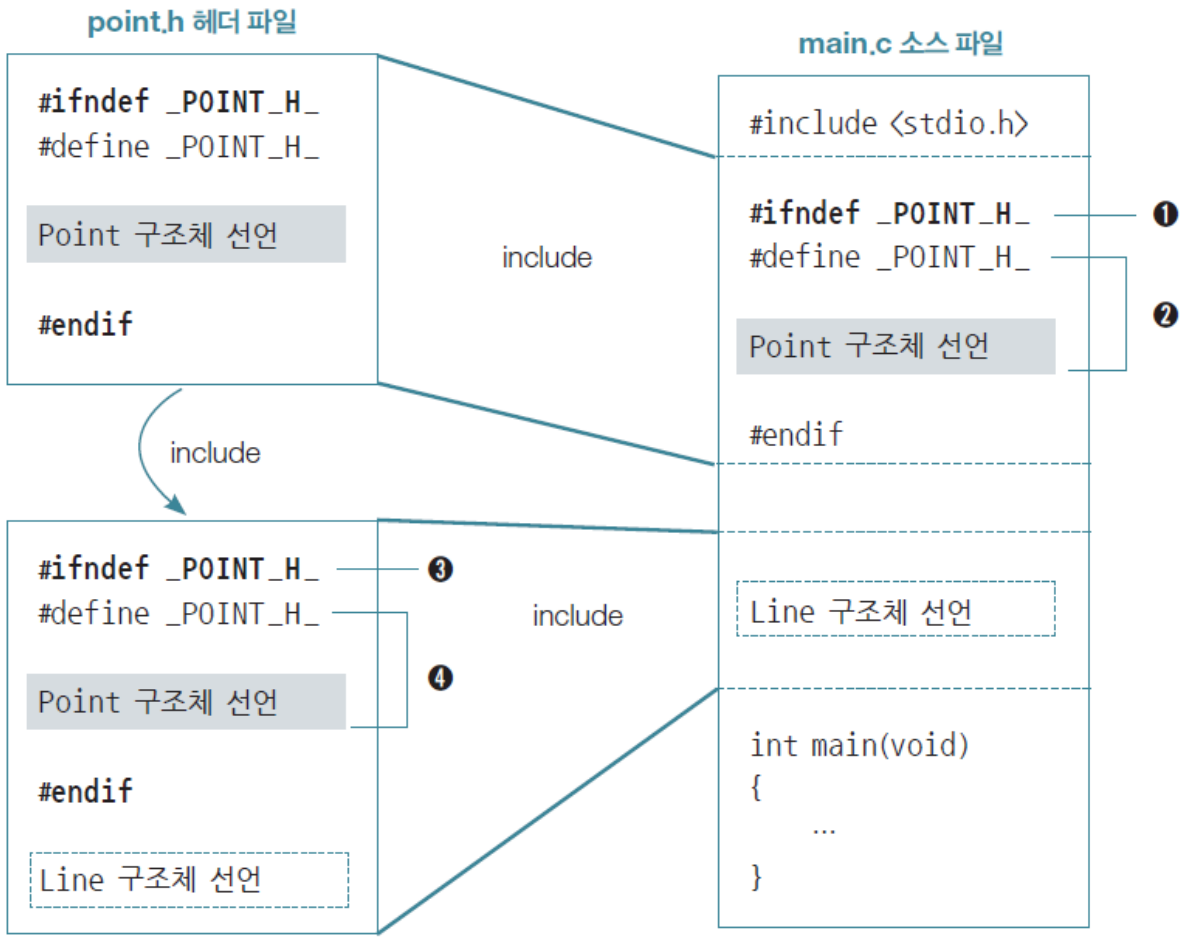
# 분할 컴파일

- 헤더파일의 중복 문제 해결
  - point.h

```
#ifndef _POINT_H_   // _POINT_H_ 매크로명이 정의되어 있지 않으면
#define _POINT_H_   // _POINT_H_ 매크로명 정의

typedef struct
{
        int x;          // x좌표
        int y;          // y좌표
} Point;

#endif              // #ifndef _POINT_H_의 끝
```

# 분할 컴파일

- 헤더파일의 중복 문제 해결
  - 조건부 컴파일 전처리 과정
  - point.h가 main.c에 처음 인클루드 될 경우에 ❶에서 매크로명 _POINT_H_ 정의되어 있지 않으므로 ❷가 포함
    - 매크로명 _POINT_H_가 정의되고 Point 구조체 선언
  - 두 번째 인클루드 될 때는 매크로명 _POINT_H_가 정의되어 있으므로 ❸에서 조건이 거짓
    - ❹는 자연스럽게 추가되지 않음



# 연습문제 1

- 두 개의 숫자와 사칙연산 기호를 입력하면 실행하여 결과를 출력하는 코드를 작성하시오.
  - 입력과 출력을 담당하는 코드는 iofunc.c에 작성
    - 입출력함수는 각각 myInput(), myOutput()
  - 사칙연산에 대한 코드는 myfunc.c에 작성
  - 필요한 각 함수에 대한 선언은 myheader.h로 구성
  - main()에서 사칙연산 함수를 호출해서 결과를 확인하는 main.c를 작성
  - 연산 결과는 소수점 두 번째 자리까지 표기

```
숫자와 연산자를 입력하시오 : 2 3 +
5.00
```

# 연습문제 2

- 국어 영어 수학 점수를 입력 받아서 평균과 총점을 계산하고 학점을 표기하는 프로그램을 작성하시오.
  - 프로그램은 score.h  score.c main.c로 구성
  - 입력과 출력(평균, 총점)은 실수로 처리
    - 출력은 소수점 두 번째 자리까지 표기
  - 학점 기준
    - A : 100 ~ 90
    - B : ~ 80
    - C : ~ 70
    - D : ~ 60
    - F : 나머지

# Q & A