

19장 전처리와 분할 컴파일



❖ 파일을 포함하는 #include

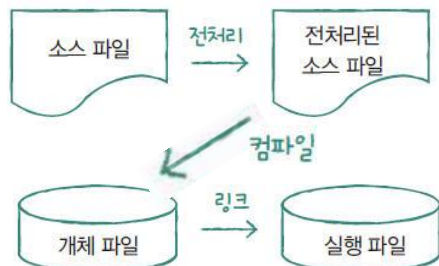
사용자 정의 헤더 파일을 사용하는 프로그램

소스 코드 student.h

```

01 // 사용자 정의 헤더 파일
02 typedef struct
03 {
04     int num;
05     char name[20];
06 } Student;

```



소스 코드 main.c

```

01 // 소스 파일
02 #include <stdio.h>
03 #include "student.h"
04
05 int main(void)
06 {
07     Student a = {315, "홍길동"};
08
09     printf("학번 : %d, 이름 : %s\n", a.num, a.name);
10
11     return 0;
12 }

```

```

typedef struct
{
    int num;
    char name[20];
} Student;

int main(void)
{
    ...
}

```

전처리가 끝나면
student.h
의 내용이
이곳에 복사된다.

실행결과

학번 : 315, 이름 : 홍길동

❖ 매크로명을 만드는 #define

다양한 매크로명의 사용 **소스 코드** 예제19-2.c

```

01 #include <stdio.h>
02 #define PI 3.14159                // 상수를 매크로명으로 정의
03 #define LIMIT 100.0              // 상수를 매크로명으로 정의
04 #define MSG "passed!"            // 문자열을 매크로명으로 정의
05 #define ERR_PRN printf("범위를 벗어났습니다!\n")    // 출력문을 매크로명으로 정의
06
07 int main(void)
08 {
09     double radius, area;          // 반지름과 면적 변수
10
11     printf("반지름을 입력하세요(100 이하) : ");
12     scanf("%lf", &radius);        // 반지름 입력
13     area = PI * radius * radius;  // 면적 계산
14     if (radius > LIMIT) ERR_PRN;   // 반지름이 100을 초과하면 오류 메시지 출력
15     else printf("원의 면적 : %.2lf (%s)\n", area, MSG); // 면적과 메시지 출력
16
17     return 0;
18 }

```

실행결과 1

반지름을 입력하세요(100 이하) : 5
원의 면적 : 78.54 (passed!)

실행결과 2

반지름을 입력하세요(100 이하) : 101
범위를 벗어났습니다!

❖ #define을 사용한 매크로 함수

매크로 함수를 사용한 프로그램

소스 코드 예제19-3.c

```

01 #include <stdio.h>
02 #define SUM(a, b) ((a) + (b)) // 두 값을 더하는 매크로 함수
03 #define MUL(a, b) ((a) * (b)) // 두 값을 곱하는 매크로 함수
04
05 int main(void)
06 {
07     int a = 10, b = 20;
08     int x = 30, y = 40;
09     int res;
10
11     printf("a + b = %d\n", SUM(a, b)); // a와 b의 합
12     printf("x + y = %d\n", SUM(x, y)); // x와 y의 합
13     res = 30 / MUL(2, 5); // 30을 2와 5의 곱으로 나눔
14     printf("res : %d\n", res);
15
16     return 0;
17 }

```

매크로 함수 정의

#define MUL(a, b) a * b

매크로 함수 사용

```

res = 30 / MUL(2, 5);
           ↓ 전처리 후
res = 30 / 2 * 5;

```

실행결과

```

a + b = 30
x + y = 70
res : 3

```

19- 1

전처리 지시자

❖ 이미 정의된 매크로

이미 정의된 매크로의 기능

소스 코드 예제19-4.c

```

01 #include <stdio.h>
02
03 void func(void);
04
05 int main(void)
06 {
07     printf("컴파일 날짜와 시간 : %s, %s\n\n", __DATE__, __TIME__);
08     printf("파일명 : %s\n", __FILE__);
09     printf("함수명 : %s\n", __FUNCTION__);
10     printf("행번호 : %d\n", __LINE__);
11
12 #line 100 "macro.c"
13     func();
14
15     return 0;
16 }
17
18 void func(void)
19 {
20     printf("\n");
21     printf("파일명 : %s\n", __FILE__);
22     printf("함수명 : %s\n", __FUNCTION__);
23     printf("행번호 : %d\n", __LINE__);
24 }

```

실행결과

컴파일 날짜와 시간 : Apr 4 2019, 17:22:15

파일명 : c:\studyc\19-4\예제19-4.c
 함수명 : main
 행번호 : 10

파일명 : macro.c
 함수명 : func
 행번호 : 110

이미 정의된 매크로	기능
__FILE__	전체 디렉터리 경로를 포함한 파일명
__FUNCTION__	매크로명이 사용된 함수 이름
__LINE__	매크로명이 사용된 행 번호
__DATE__	컴파일을 시작한 날짜
__TIME__	컴파일을 시작한 시간

19- 1

전처리 지시자

❖ 매크로 연산자 #과

#과 ##을 사용한 매크로 함수

소스 코드 예제19-5.c

```
01 #include <stdio.h>
02 #define PRINT_EXPR(x) printf(#x " = %d\n", x)
03 #define NAME_CAT(x, y) (x ## y)
04
05 int main(void)
06 {
07     int a1, a2;
08
09     NAME_CAT(a, 1) = 10;    // (a1) = 10;
10     NAME_CAT(a, 2) = 20;    // (a2) = 20;
11     PRINT_EXPR(a1 + a2);    // printf("a1 + a2" " = %d\n", a1 + a2);
12     PRINT_EXPR(a2 - a1);    // printf("a2 - a1" " = %d\n", a2 - a1);
13
14     return 0;
15 }
```

#define PRINT_EXPR(x) printf(#x " = %d\n", x)

a1 + a2

"a1 + a2"

실행결과1

×

a1 + a2 = 30

a2 - a1 = 10

19- 1

전처리 지시자

❖ 조건부 컴파일 지시자 (1/2)

#if, #ifdef, #else, #endif를 사용한 조건부 컴파일

소스 코드 예제 19-6.c

```
01 #include <stdio.h>
02 #define VER 7
03 #define BIT16
04
05 int main(void)
06 {
07     int max;
08
09     #if VER >= 6
10         printf("버전 %d입니다.\n", VER);
11     #endif
12
13     #ifdef BIT16
14         max = 32767;
15     #else
16         max = 2147483647;
17     #endif
18
19     printf("int형 변수의 최댓값 : %d\n", max);
20
21     return 0;
22 }
```

실행결과1

버전 7입니다.

int형 변수의 최댓값 : 32767

❖ 조건부 컴파일 지시자 (2/2)

- 조건부 컴파일의 다양한 사용법



- 전처리 연산자 defined와 !defined

```
#if (defined(BIT16) && (VER >= 6))
    컴파일할 문장
#endif
```

```
#if !defined BIT16
    컴파일할 문장
#endif
```



```
#ifndef BIT16
    컴파일할 문장
#endif
```


❖ #pragma 지시자

#pragma 를 사용한 바이트 얼라인먼트 변경

소스 코드 예제 19-7.c

```
01 #include <stdio.h>
02 #pragma pack(push, 1)
03 typedef struct
04 {
05     char ch;
06     int in;
07 } Sample1;
08
09 #pragma pack(pop)
10 typedef struct
11 {
12     char ch;
13     int in;
14 } Sample2;
15
16 int main(void)
17 {
18     printf("Sample1 구조체의 크기 : %d바이트\n", sizeof(Sample1));
19     printf("Sample2 구조체의 크기 : %d바이트\n", sizeof(Sample2));
20
21     return 0;
22 }
```

실행결과

Sample1 구조체의 크기 : 5바이트
Sample2 구조체의 크기 : 8바이트

#pragma warning(disable:4101) // 4101번 경고 메시지는 모두 표시하지 않음

키워드로 끝내는 핵심 포인트

- ❖ **#include**는 지정한 파일을 소스 코드에 적절하게 포함시킨다.
- ❖ **#define**은 매크로 상수와 매크로 함수를 만들 때 쓴다.
- ❖ **#if, #else, #elif, #ifdef, #ifndef, #endif**는 조건부 컴파일을 위해 사용하는 **조건부 컴파일 지시자**다. 그 외에도 **#pragma, #error, #line** 등 컴파일 과정을 돕는 다양한 지시자가 있다.
- ❖ **defined, #, ##**은 전처리 지시자와 함께 사용하는 전처리 연산자다.

표로 정리하는 핵심 포인트

표 19-1 다양한 전처리 지시자

지시자	사용 예	기능
#include	#include <stdio.h>	include 디렉터리에서 stdio.h를 찾아 그 내용 복사
	#include "myhdr.h"	소스 파일이 있는 디렉터리에서 myhdr.h를 찾아 그 내용 복사
#define	#define PI 3.14	PI는 상수 3.14로 바뀜
	#define SUM(x, y) ((x)+(y))	SUM(10, 20)은 ((10)+(20))으로 바뀜
#if ~ #endif	#if (VER >= 6) max = 1; #endif	VER이 6 이상이면 max = 1; 컴파일
#ifdef ~ #endif	#ifdef DEBUG printf("%d", a); #endif	DEBUG가 정의되어 있으면 printf 문장 컴파일
기타	#ifndef, #else, #elif, #undef, #pragma, #error, #line 등 사용 가능	

❖ 분할 컴파일 방법 (1/5)

main 함수 정의

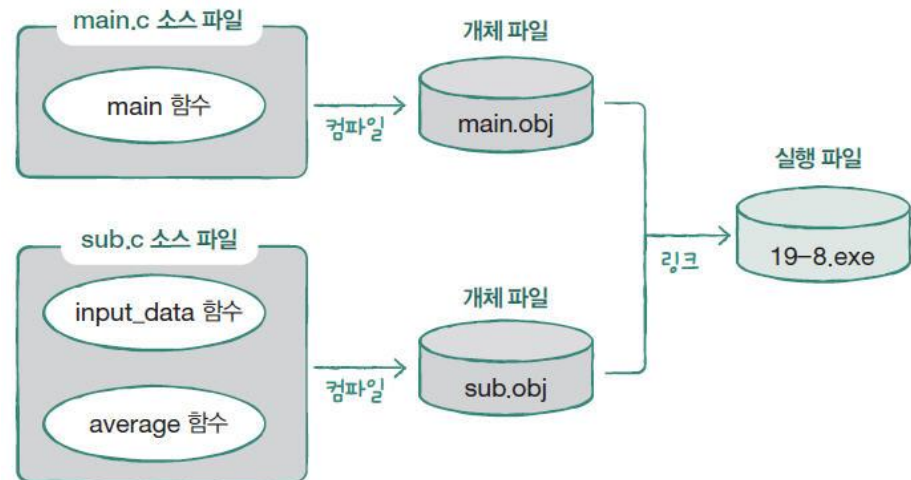
소스 코드 main.c

두 정수의 평균을 구하는 프로그램

```

01 #include <stdio.h>
02
03 void input_data(int *, int *);
04 double average(int, int);
05
06 int main(void)
07 {
08     int a, b;
09     double avg;
10
11     input_data(&a, &b);           // 두 정수 입력
12     avg = average(a, b);         // 평균 계산
13     printf("%d와 %d의 평균 : %.1lf\n", a, b, avg); // 입력값과 평균 출력
14
15     return 0;
16 }

```



❖ 분할 컴파일 방법 (2/5)

input_data, average 함수 정의

소스 코드 sub.c

두 정수의 평균을 구하는 프로그램

```
01 #include <stdio.h>                // printf, scanf 함수 사용을 위해 필요
02
03 void input_data(int *pa, int *pb)    // 두 정수 입력 함수
04 {
05     printf("두 정수 입력 : ");
06     scanf("%d%d", pa, pb);
07 }
08
09 double average(int a, int b)         // 평균을 구하는 함수
10 {
11     int tot;
12     double avg;
13
14     tot = a + b;
15     avg = tot / 2.0;
16
17     return avg;
18 }
```

실행결과

✕

두 정수 입력 : 5 8

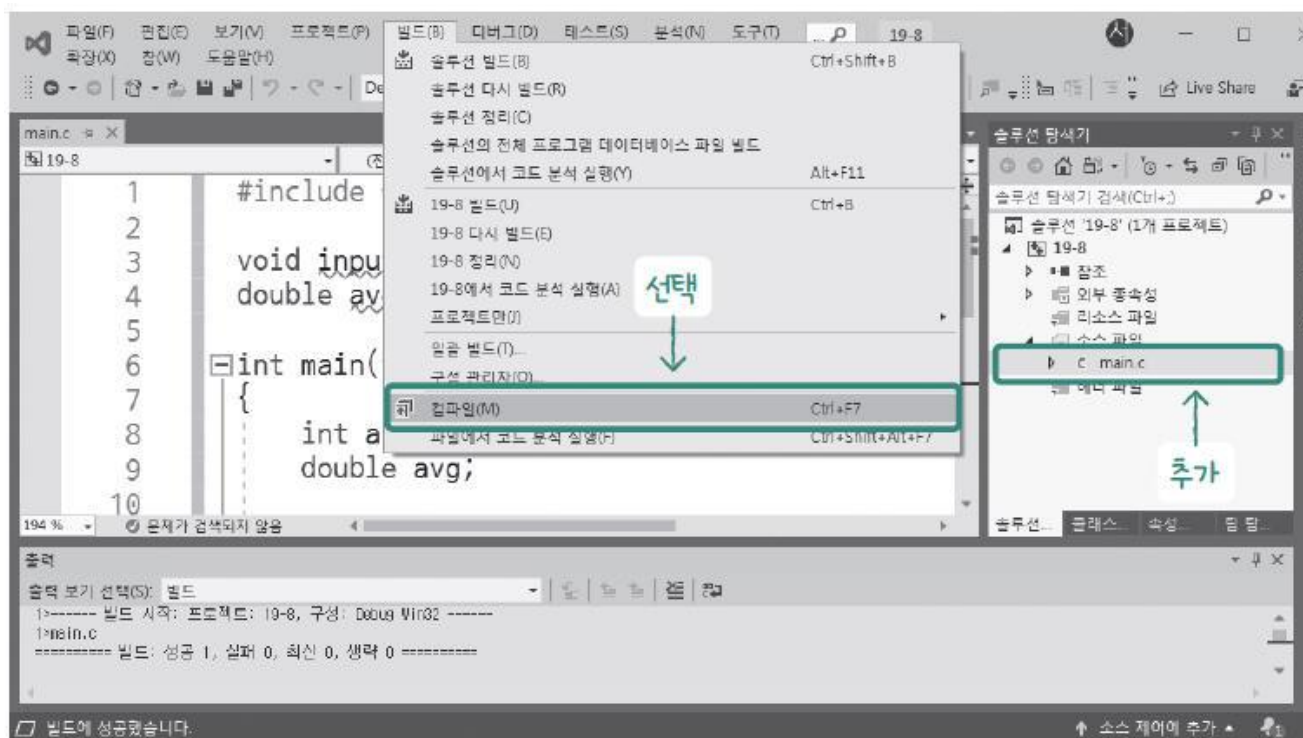
5와 8의 평균 : 6.5

19- 2

분할 컴파일

❖ 분할 컴파일 방법 (3/5)

- 프로젝트에 main.c 파일을 추가하고 컴파일한다.

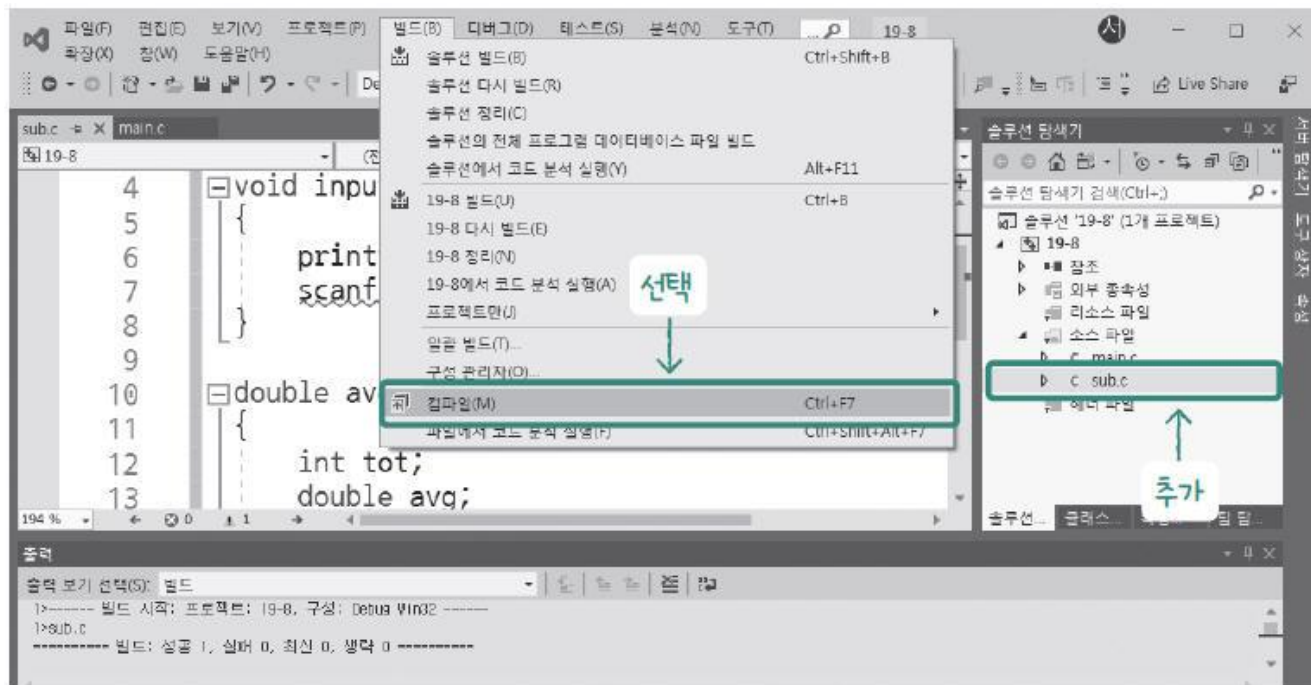


19- 2

분할 컴파일

❖ 분할 컴파일 방법 (4/5)

- 프로젝트에 sub.c 파일을 추가하고 컴파일한다.

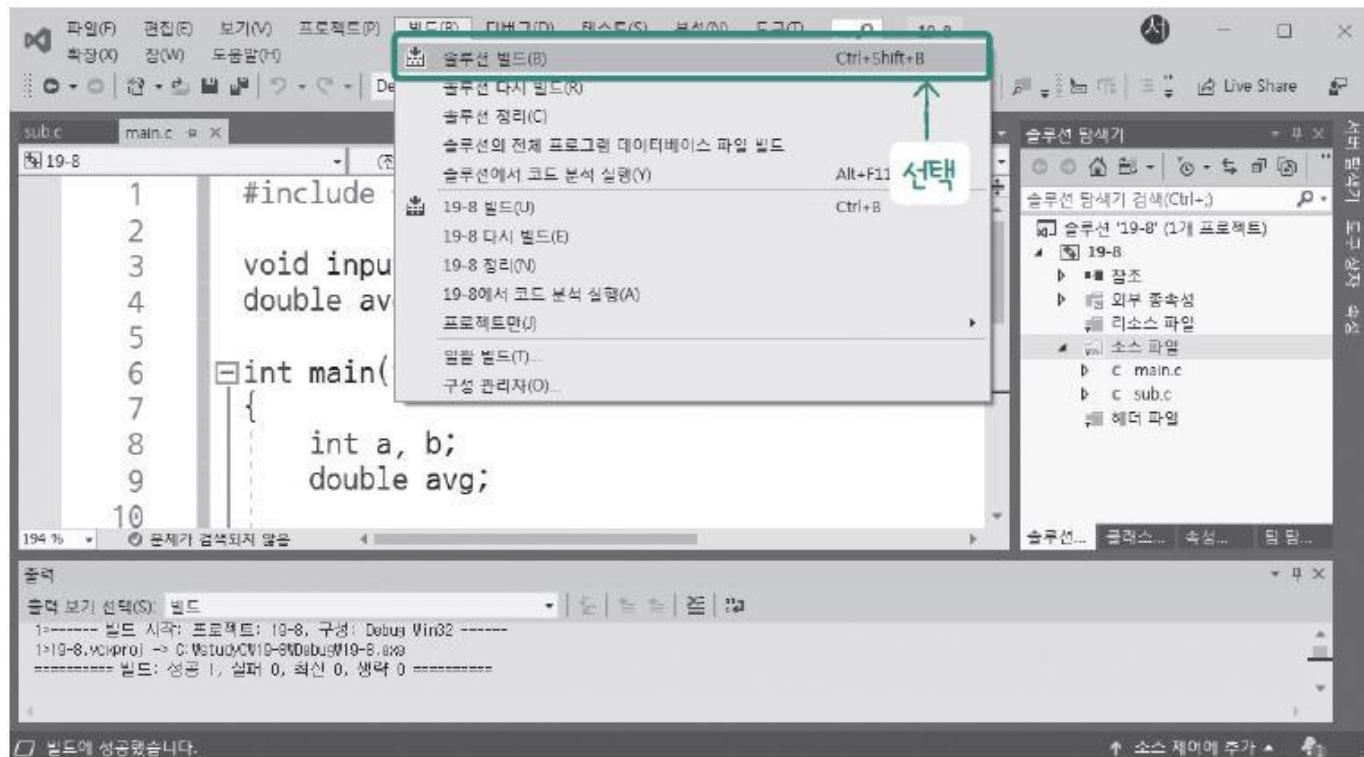


19- 2

분할 컴파일

❖ 분할 컴파일 방법 (5/5)

- 링크([빌드]-[솔루션 빌드])를 수행하여 실행 파일을 만든다.



❖ 분할 컴파일에서 extern과 static의 용도 (1/3)

main, print_data 함수 정의

소스 코드 main.c

전역 변수에 extern과 static을 사용한 프로그램

```

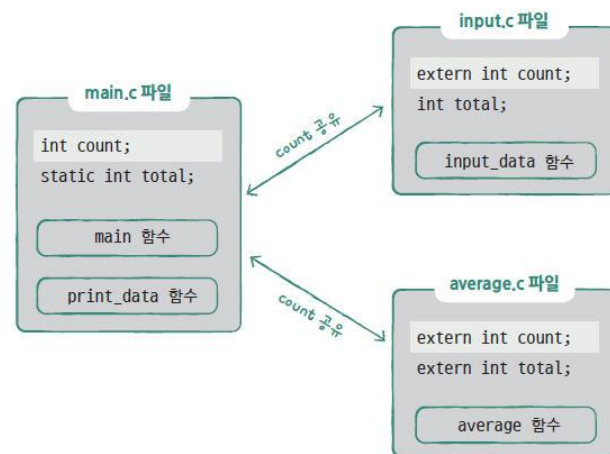
01 #include <stdio.h>
02
03 int input_data(void);
04 double average(void);
05 void print_data(double);
06
07 int count = 0;
08 static int total = 0;
09

```

```

10 int main(void)
11 {
12     double avg;
13
14     total = input_data();
15     avg = average();
16     print_data(avg);
17
18     return 0;
19 }
20
21 void print_data(double avg)
22 {
23     printf(" 입력한 양수의 개수 : %d\n", count);
24     printf(" 전체 합과 평균 : %d, %.1lf\n", total, avg);
25 }

```



❖ 분할 컴파일에서 extern과 static의 용도 (2/3)

input_data 함수 정의

소스 코드 input.c

전역 변수에 extern과 static을 사용한 프로그램

```
01 #include <stdio.h>
02
03 extern int count;
04 int total = 0;
05
06 int input_data(void)
07 {
08     int pos;
09
```

```
10     while (1)
11     {
12         printf("양수 입력 : ");
13         scanf("%d", &pos);
14         if (pos < 0) break;
15         count++;
16         total += pos;
17     }
18
19     return total;
20 }
```


❖ 분할 컴파일에서 extern과 static의 용도 (3/3)

average 함수 정의

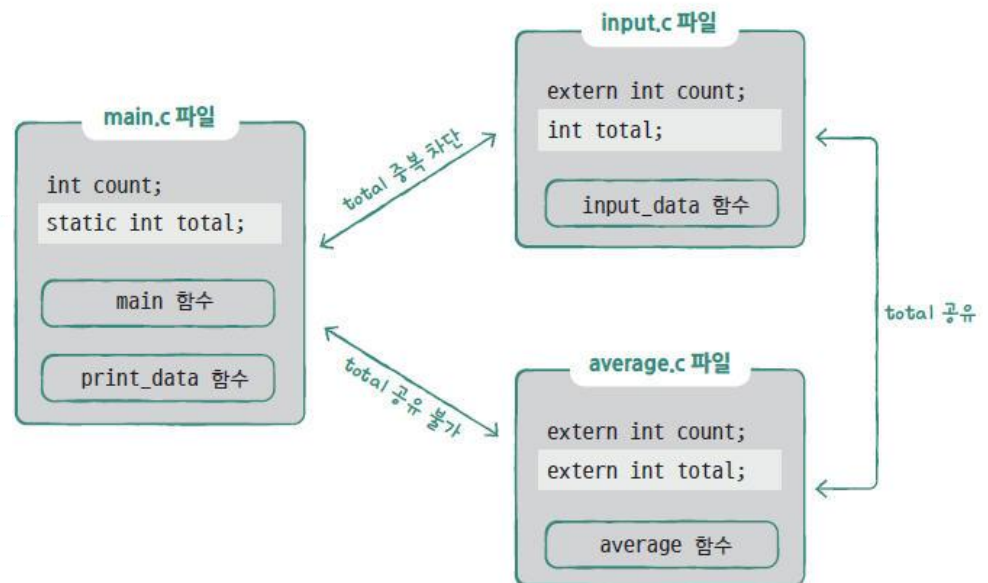
소스 코드 average.c

전역 변수에 extern과 static을 사용한 프로그램

```
01 extern int count;
02 extern int total;
03
04 double average(void)
05 {
06     return total / (double)count;
07 }
```

실행결과

양수 입력 : 8 ↵
양수 입력 : 3 ↵
양수 입력 : 6 ↵
양수 입력 : -1 ↵
입력한 양수의 개수 : 3
전체 합과 평균 : 17, 5.7



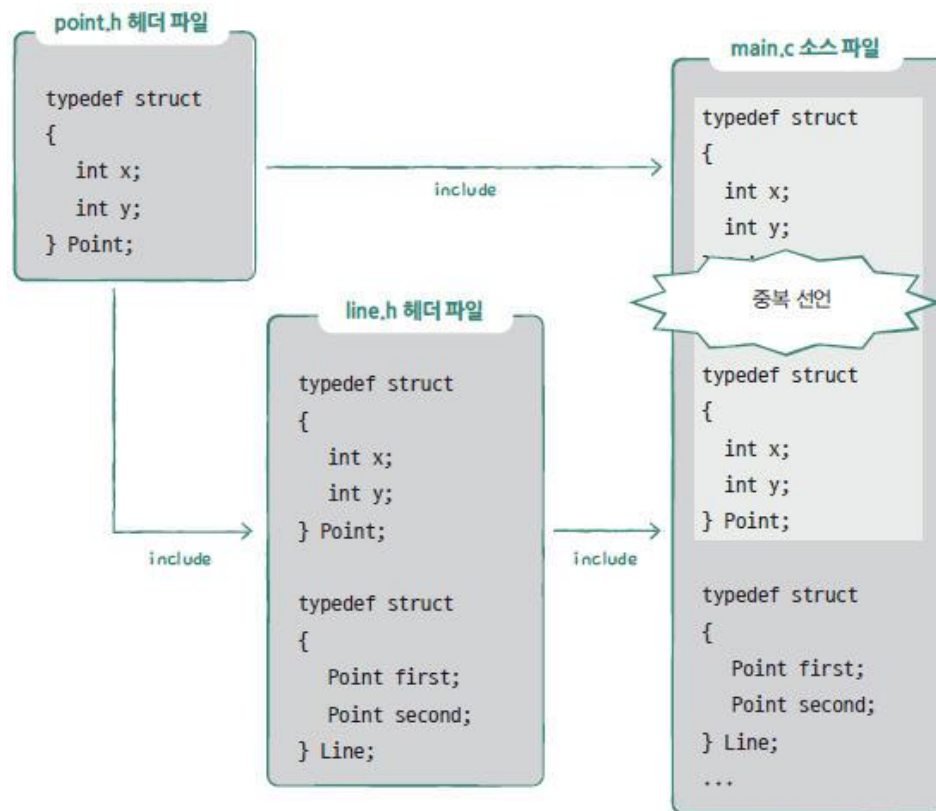
❖ 헤더 파일의 필요성과 중복 문제 해결 방법 (1/3)

Point 구조체 선언 소스 코드 point.c

```
01 typedef struct
02 {
03     int x;
04     int y;
05 } Point;
```

Line 구조체 선언 소스 코드 line.c

```
01 #include "point.h"
02
03 typedef struct
04 {
05     Point first;
06     Point second;
07 } Line;
```




❖ 헤더 파일의 필요성과 중복 문제 해결 방법 (2/3)

Point와 Line 구조체 모두 사용

소스 코드 main.c

```
01 #include <stdio.h>
02 #include "point.h"           // Point 구조체 선언
03 #include "line.h"           // Line 구조체 선언
04
05 int main(void)
06 {
07     Line a = { {1, 2}, {5, 6} };    // Line 구조체 변수 초기화
08     Point b;                       // 가운데 점의 좌표 저장
09
10     b.x = (a.first.x + a.second.x) / 2; // 가운데 점의 x좌표 계산
11     b.y = (a.first.y + a.second.y) / 2; // 가운데 점의 y좌표 계산
12     printf("선의 가운데 점의 좌표 : (%d, %d)\n", b.x, b.y);
13
14     return 0;
15 }
```

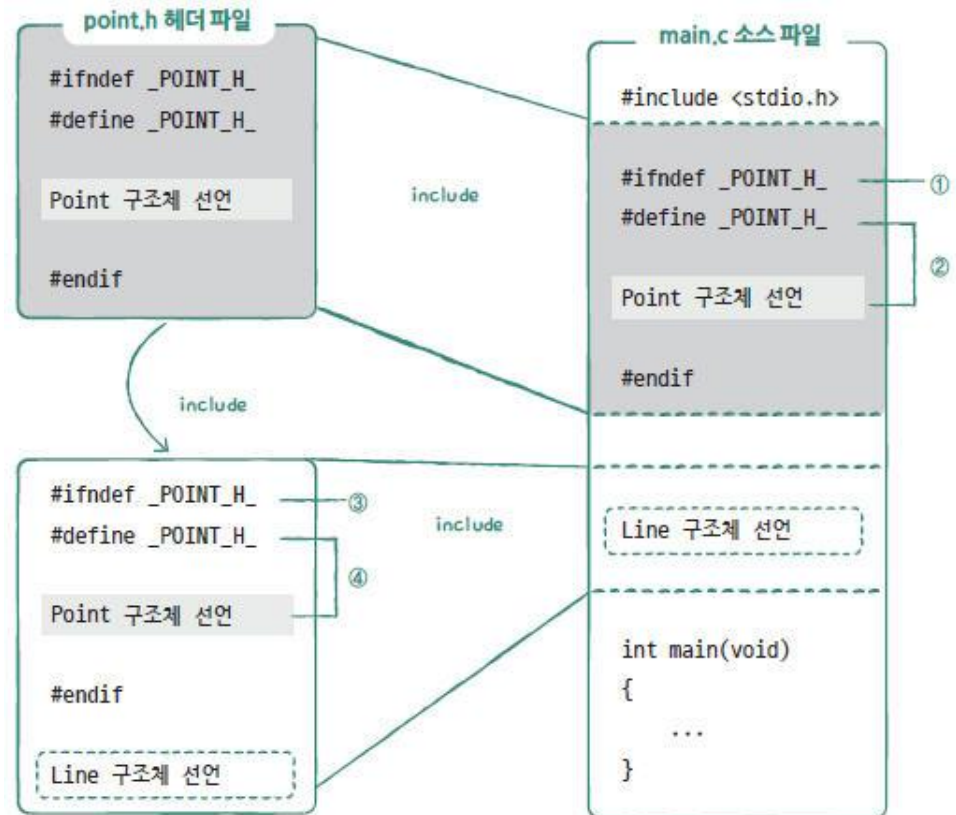
 실행결과 ×

선의 가운데 점의 좌표 : (3, 4)

❖ 헤더 파일의 필요성과 중복 문제 해결 방법 (3/3)

Point 구조체 선언 소스 코드 point.c

```
01 #ifndef _POINT_H_
02 #define _POINT_H_
03
04 typedef struct
05 {
06     int x;
07     int y;
08 } Point;
09
10 #endif
```



키워드로 끝내는 핵심 포인트

- ❖ **모듈**은 독립적으로 컴파일과 디버깅이 가능한 하나의 파일이다.
- ❖ **분할 컴파일**된 개체 파일은 **링크**^{link}되어 하나의 프로그램(파일)이 된다.
- ❖ 파일 간에 전역 변수를 공유할 때는 **extern**을 쓰고 공유를 제한할 때는 **static**을 쓴다.
- ❖ **헤더 파일의 중복 포함 문제**는 조건부 컴파일 지시자로 해결할 수 있다.

표로 정리하는 핵심 포인트

표 19-3 분할 컴파일과 데이터의 공유 방법

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
	19-8.exe	링크로 실행 파일 생성
전역 변수의 공유	extern int count;	다른 파일의 전역 변수 count 사용
	static int total;	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