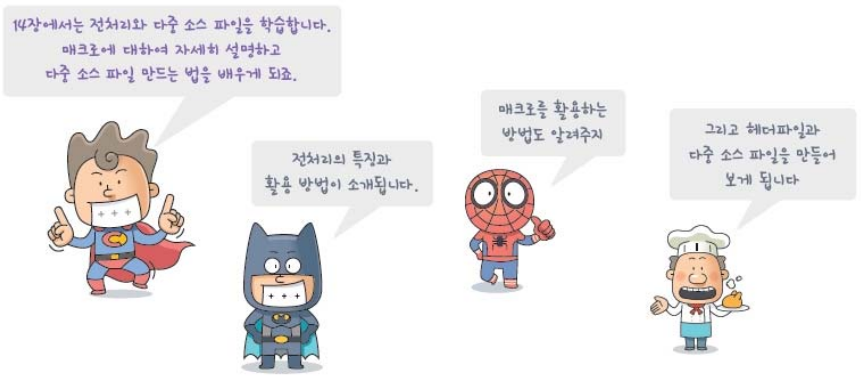
CHAPTER

14 전처리와 다중 소스 파일

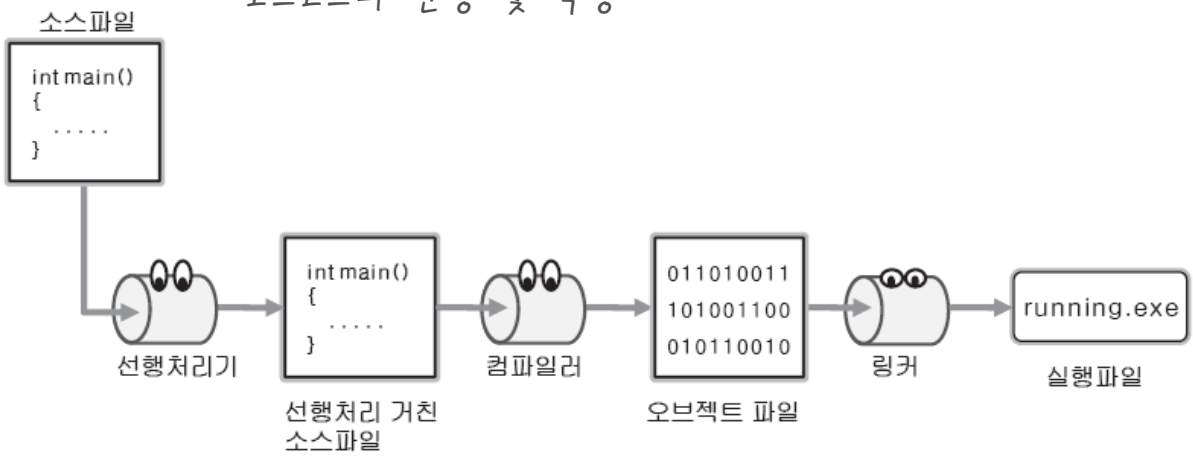
학습목차

- 14.1 전처리와 매크로
- 14.2 조건부 컴파일
- 14.3 다중 소스 파일



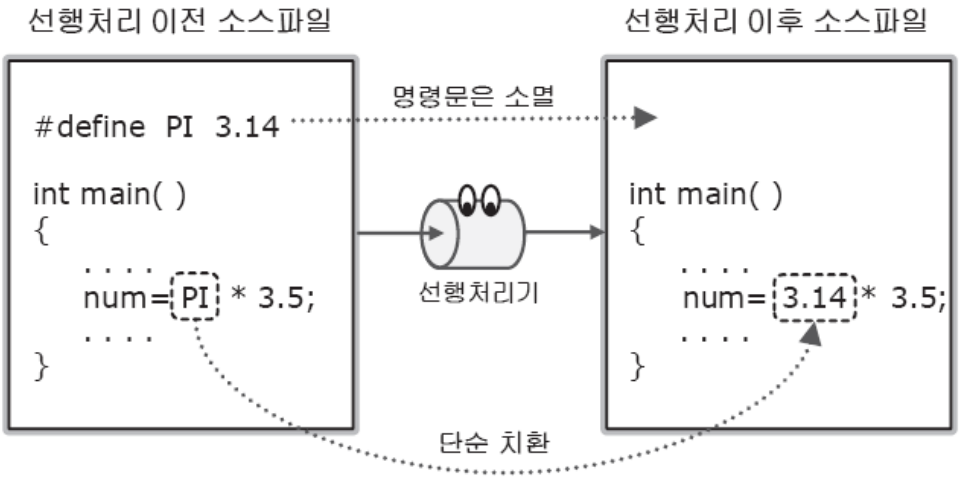
■ 전처리는 선행처리를 의미합니다.

컴파일 이전의 선행처리를 통한
소스코드의 변경 및 확장



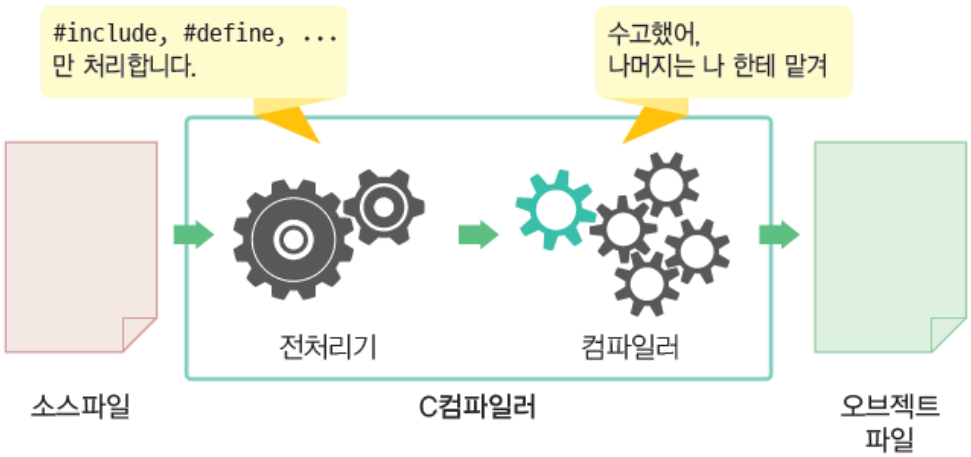
■ 전처리기가 하는 일은?

전처리기의 주 역할은
단순
치환(substitution)



전처리기란?

- *전처리기(preprocessor)*는 컴파일하기에 앞서서 소스 파일을 처리하는 컴파일러의 한 부분



전처리기의 요약

지시어	의미
#define	매크로 정의
#include	파일 포함
#undef	매크로 정의 해제
#if	조건이 참일 경우
#else	조건이 거짓일 경우
#endif	조건 처리 문장 종료
#ifdef	매크로가 정의되어 있는 경우
#ifndef	매크로가 정의되어 있지 않은 경우
#line	행번호 출력
#pragma	컴파일 지시, 시스템에 따라 의미가 다름

■ 헤더파일을 include하는 두 가지 방법

#include <헤더파일 이름>
  표준 헤더파일이 저장되어 있는 디렉터리에서 파일을 찾는다.

#include "헤더파일 이름"
  소스파일이 저장된 디렉터리에서 헤더파일을 찾는다.
  때문에 프로그래머가 정의한 헤더파일의 포함에 사용된다.

#include "C:₩CPower₩MyProject₩header.h"
  → Windows상에서의 절대 경로 지정

#include "/Cpower/MyProject/header.h"
  → Linux상에서의 절대 경로 지정

■ 상대 경로의 지정 방법

상대 경로의 지정이 훨씬 유용한 방법

#include "Realease₩header0.h"

소스파일이 있는 디렉터리의 하위 디렉터리인 Release 디렉터리에 존재하는 header0.h 포함

#include "..₩CProg₩header1.h"

한 단계 상위 디렉터리의 하위 디렉터리인 CProg에 존재하는 header1.h 포함

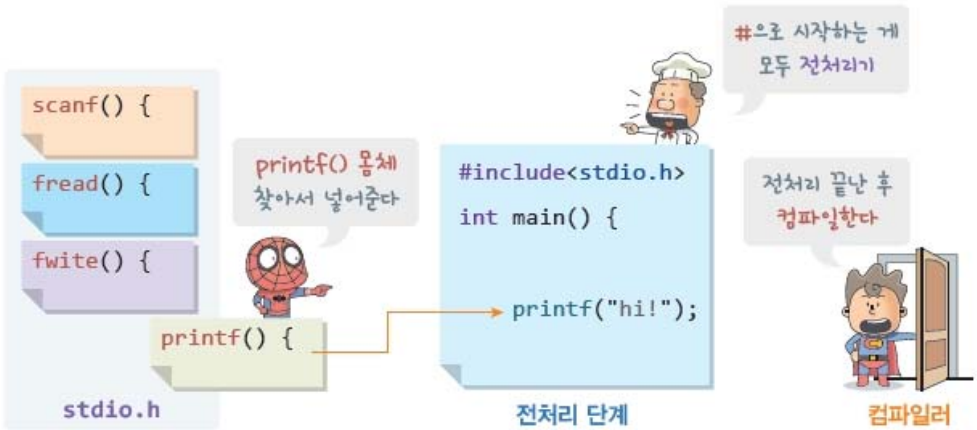
#include "..₩..₩MyHeader₩header2.h"

두 단계 상위 디렉터리의 하위 디렉터리인 MyHeader에 존재하는 header2.h를 포함하라.

14.1 전처리와 매크로

1 전처리 이해하기

- 전처리란 컴파일을 하기전에 먼저 처리한다는 의미
- printf() 함수를 사용하는 경우, #include <stdio.h>를 코드의 상단에 넣어야 함
- #include <stdio.h>에서 #include는 전처리의 의미
  - 컴파일을 하기 전에 stdio.h 헤더 파일로 가서 printf()의 몸통 코드를 가지고 옴
  - 그리고 난 후에 컴파일러는 제대로 된 전체 코드를 가지고 컴파일 작업



14.1 전처리와 매크로

- 전처리의 특징을 이용하면 코드를 좀 더 편하게 만들 수 있음 - 대표적인 예가 매크로
- #define MAX 5을 정의하면 전처리 과정에서 코드내의 모든 MAX를 5로 바꿔줌
- 전처리에 해당하는 키워드는 모두 #으로 시작함
- 전처리문의 끝에 세미콜론(;)을 쓰지 않음

```c
#include <stdio.h>

#define MAX 5                  // 매크로 값을 바꾸면 코드 내 관련 값이 동시에 변경됨

int main() {
   int k;
   double arr[MAX], sum = 0.0;

   printf("%d개 값을 입력하시오 :", MAX);
   for (k = 0; k < MAX; k++) {              // k는 0부터 MAX보다 작을 때까지
      scanf("%lf", &arr[k]);
      sum = sum + arr[k];
   }
   printf("평균 = %.1lf", sum / MAX);

   return 0;
}
```

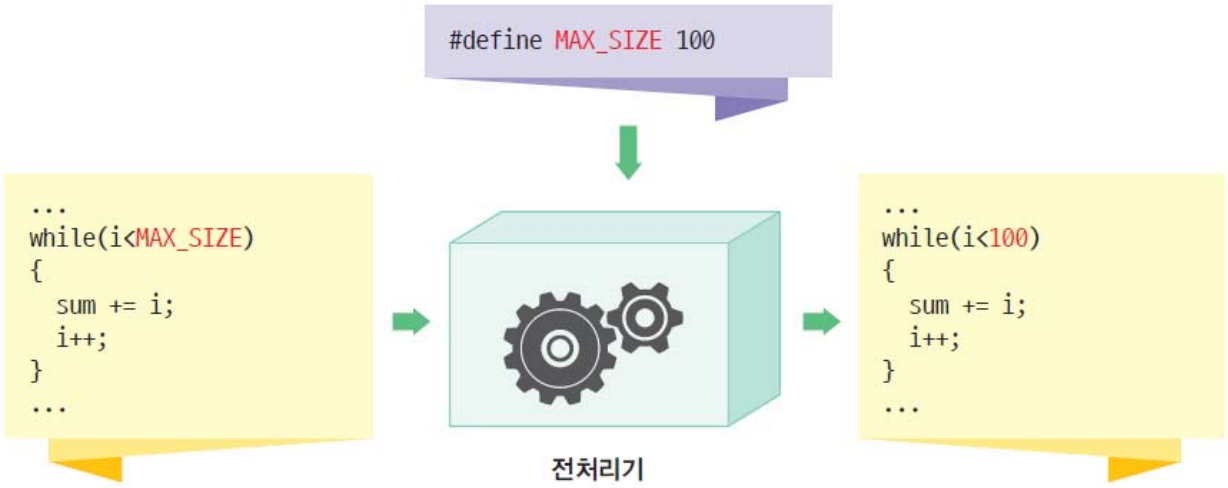

단순 매크로

Syntax: 단순 매크로 정의

예 `#define MAX_SIZE 100`

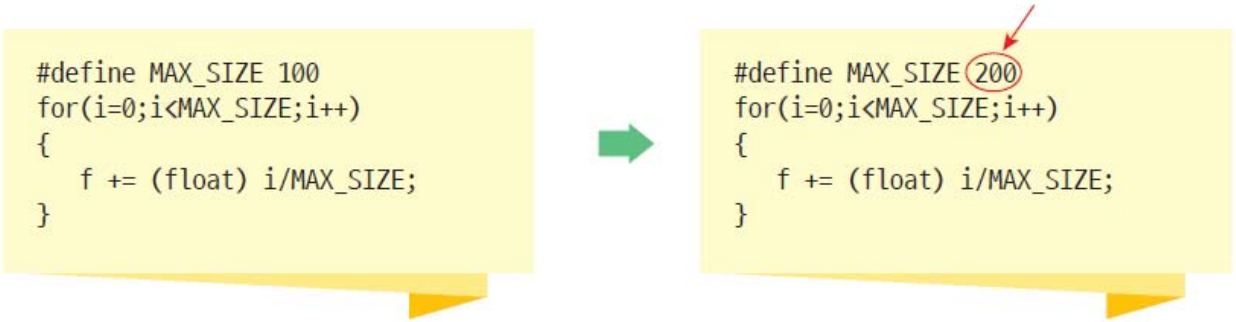
기호상수 MAX_SIZE를 100으로 정의한다.

단순 매크로



단순 매크로의 장점

- 프로그램의 가독성을 높인다.
- 상수의 변경이 용이하다.



단순 매크로의 예

```
#define PI              3.141592            // 원주율
#define TWOPI           (3.141592 * 2.0)    // 원주율의 2배
#define MAX_INT         2147483647          // 최대정수
#define EOF             (-1)                // 파일의 끝표시
#define MAX_STUDENTS            2000        // 최대 학생수
#define EPS             1.0e-9              // 실수의 계산 한계
#define DIGITS          "0123456789"        // 문자 상수 정의
#define BRACKET         "(){}[]"            // 문자 상수 정의
#define getchar()       getc(stdin)         // stdio.h에 정의
#define putchar()       putc(stdout)        // stdio.h에 정의
```

getc(FILE *fp)는 매크로로 구현되어 있고 fgetc(FILE *fp)는 함수로 구현되어 있어서 getc()는 인수로 수식을 사용하면 문제가 생길 수 있지만 더 빠르고 fgetc()는 함수이므로 함수 포인터로 연결하여 사용할 수 있다

```
#define getc(_Stream)          _getc_nolock(_Stream)
#define putc(_Ch, _Stream)     _putc_nolock(_Ch, _Stream)
```


함수 매크로

- *함수 매크로(function-like macro)*란 매크로가 함수처럼 매개 변수를 가지는 것

Syntax: 함수매크로

예

```
#define SQUARE(x) ((x) * (x))
```

매크로 / 인수 / SQUARE(X)는 이것과 같다.

함수 매크로의 예

#define SUM(x, y)	((x) + (y))
#define AVERAGE(x, y, z)	(((x) + (y) + (z)) / 3)
#define MAX(x,y)	((x) > (y)) ? (x) : (y)
#define MIN(x,y)	((x) < (y)) ? (x) : (y)

2 함수 매크로

매크로 선언 `#define 매크로_이름 매크로_값`

- 매크로 선언- #define 옆에 "매크로_이름"과 "매크로_값"을 써주면 됨
- 불린 자료형을 사용하고 싶다면 [예제 14-2]와 같이 매크로로 정의 하면 됨

```c
#include <stdio.h>

#define TRUE 1
#define FALSE 0                    // enum { FALSE, TRUE };도 같은 의미

int main() {
   int love;

   printf("이거를 마신다? YES=1, NO=0: ");
   scanf("%d", &love);
   if (love == TRUE)                          // 불린 변수 사용 가능
       printf("둘이 사귄다\n");
   else
       printf("둘이 헤어진다\n");

}
```

실행 화면

```
잔을 비우시겠습니까? YES=1, NO=0: 1
둘이 사귄다
```

14.1 전처리와 매크로

- 함수도 매크로 형태로 정의할 수 있음
- 매크로_이름에 함수 이름과 매개변수를 쓰고 매크로_값에 함수에 사용할 코드를 쓰면 됨

```
#define MUL(a, b) a * b
```

MUL() 함수 매크로 선언

```
01  #include <stdio.h>
02
03  #define MUL(a, b) a * b
04
05  int main() {
06
07     printf("5와 7 곱하기 결과 %d\n", MUL(5, 7));
08     printf("2.7과 4.2 곱하기 결과 %f\n", MUL(2.7, 4.2));
09     printf("2+3과 3+4 곱하기 결과 %d\n", MUL(2+3, 3+4));
10
11     return 0;
12  }
```

오류 고치기

실행 화면

```
5와 7 곱하기 결과 35
2.7과 4.2 곱하기 결과 11.340000
2+3과 3+4 곱하기 결과 15
```


14.1 전처리와 매크로

- 9번 줄의 MUL(2+3, 3+4)의 결과는 15로 틀린 답을 출력
- 해결하기 위해서는 다음과 같이 괄호를 묶어주면 됨

```
#define MUL(a, b) (a) * (b)
```

연산자의 우선순위 때문에 괄호 사용

3 여러 실행문을 가진 매크로

- 실행문이 여러 개인 함수를 매크로로 정의할 경우 중괄호{ }를 사용, 중괄호 생략 가능
- 두 수를 바꾸는 SWAP() 함수를 매크로 형태로 만들면 다음과 같음

```
#define SWAP(a, b) {int t; t=a; a=b; b=t;}
```

SWAP() 매크로 선언

```
#define SWAP(a, b) {      \
   int t;                 \
   t = a;                 \
   a = b;                 \
   b = t; }
```

\를 사용하면 여러 줄에 걸쳐 매크로를 정의할 수 있음

[예제 14-4] 함수 매크로를 사용한 버블 정렬 코드

```
01 #include <stdio.h>
02
03 #define SWAP(a, b) {int t; t=a; a=b; b=t;}   // 두 수 바꿔주는 매크로 SWAP
04
05 void bubble_sort(int *pd, int size) {        // 배열을 포인터 *pd로 받음
06    int j, k;
07
08    for (k = 0; k < size; k++) {              // for k - 카드 수만큼 반복
09       for (j = 1; j < size; j++) {           // for j - 두 수 비교
10          if (pd[j-1] > pd[j])
11          SWAP(pd[j-1], pd[j]);
12       }                                      // End for j
13    }                                         // End for k
14 }                                            // End bubble sort
15
16 int main() {
17    int data[9] = {2, 7, 3, 5, 9, 8, 6, 1, 4}, k;
18
19    bubble_sort(data, 9);
20    for (k = 0; k < 9; k++)
21       printf("%d ", data[k]);                // 배열 값 출력
22    return 0;
23 }
```

중괄호 생략가능

실행 화면

```
1 2 3 4 5 6 7 8 9
```

4 매크로 선언 위치

- [예제 14-5]는 구조체 코드 중 출동 순서를 정하는 코드
- 4번 줄 #define SWAP(a, b) {hero t; t=a; a=b; b=t;}에서 hero 구조체를 사용
- hero 구조체 정의는 6번 줄부터 시작
- SWAP( ) 매크로의 구조체 hero는 정의 이전에 사용하는 것처럼 보임
- SWAP() 매크로는 4번 줄에 나타나지만, 19번 줄에 도달해야만 SWAP() 매크로가 사용됨 -> 문제 없음

```
01  #include <stdio.h>
02
03  #define MAX 4
04  #define SWAP(a, b) {hero t; t=a; a=b; b=t;}  // 두 구조체를 바꿔주는 매크로
05
06  typedef struct hero {              // typedef를 사용하여 hero 구조체 정의
07     char name[10];                     // hero 구조체 멤버: 이름(문자열)
08     int age;                           // hero 구조체 멤버: 나이(정수)
09     float power;                       // hero 구조체 멤버: 힘(실수)
10  } hero;                            // hero 구조체 별명 hero
11
12  int main() {
13     hero ho[MAX] = { {"gildong", 22, 8.2}, {"hyungwook", 28, 9.3},
                        {"changsik", 21, 7.6}, {"hosik", 29, 8.1} };
14     int j, k;
15
16     for (k = 0; k < MAX; k++) {         // for k - 구조체 배열 수만큼 반복
17        for (j = 1; j < MAX; j++) {         // for j - 두 수 비교
18           if (ho[j-1].age > ho[j].age)             // if 나이가 많으면
19              SWAP(ho[j-1], ho[j]);                 // 매크로: 두 수 바꾸기
20        }                                    // End for j
21     }                                    // End for k
22     for (k = 0; k < MAX; k++)
23        printf("%d:%s %d %.1f\n", k, ho[k].name, ho[k].age,
                                                        ho[k].power);
24     return 0;
25  }
```

실행 화면

```
0:changsik 21 7.6
1:gildong 22 8.2
2:hyungwook 28 9.3
3:hosik 29 8.1
```


5 문자열 변환

- 만약 #define SPRINT(s) printf("%s₩n", s)와 같이 선언을 하고, SPRINT(I LOVE YOU)로 매크로를 사용하게 되면 오류가 발생
  - 문자열 형태인 SPRINT("I LOVE YOU")로 써야 함
  - 변수 s를 문자열로 취급하고 싶다면 #s라 쓰면 됨

```
01  #include <stdio.h>
02
03  #define SPRINT(s) printf("%s\n", s)
04  #define NPRINT(s) printf("%s\n", #s)
05
06  int main() {
07
08     SPRINT("I LOVE YOU");
09     NPRINT(I LOVE YOU);
10
11     return 0;
12  }
```

실행 화면

```
I LOVE YOU
I LOVE YOU
```

- 변수 앞에 샵(#) 연달아 붙이면, 뒤의 변수를 앞의 변수에 이어 붙임

[예제 14-7] 매크로 이어 붙이기 코드

```c
#include <stdio.h>

#define CPRINT(a, b) printf("%d\n", a##b);

int main() {

   CPRINT(5, 7);

   return 0;
}
```

실행 화면

```
57
```


- **매크로 연산자 #과 ##**

#과 ##을 사용한 매크로 함수 소스 코드 예제19-5.c

```c
#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;
}
```

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

a1 + a2 "a1 + a2"

실행결과1

```
a1 + a2 = 30
a2 - a1 = 10
```

6 내장 매크로

- 내장 매크로(predefined macros)란 컴퓨터의 안에 내장되어 있는 정보를 알려주는 매크로
- 대표적인 내장 매크로는 다음의 표와 같음

표 14-1 대표적인 내장 매크로

내장 매크로	의미	반환 값 형태
__DATE__	현재 코드가 컴파일된 날짜	문자열
__TIME__	현재 코드가 컴파일된 시간	문자열
__TIMESTAMP__	컴파일 날짜 및 시간	문자열
__FILE__	코드가 들어 있는 파일이름(경로 포함)	문자열
__LINE__	코드에서 실행된 라인 번호	정수

[예제 14-8] 내장 매크로 코드

```c
#include <stdio.h>

int main() {

   printf("컴파일 날짜: %s\n", __DATE__);
   printf("컴파일 시간: %s\n", __TIME__);
   printf("컴파일 날짜 시간: %s\n", __TIMESTAMP__);
   printf("파일위치: %s\n", __FILE__);
   printf("라인번호: %d\n", __LINE__);

   return 0;
}
```

실행 화면

```
컴파일 날짜: Dec 15 2020
컴파일 시간: 17:18:39
컴파일 날짜 시간: Tue Dec 15 17:18:37 2020
파일위치: C:\Users\zoch\source\repos\Project1\macro5.c
라인번호: 9
```

01 다음 중 전처리에 해당하는 키워드는 어떤 기호로 시작하는가?

① < ② # ③ ; ④ ?

02 다음 중 전처리에 키워드를 사용할 때 문장의 끝에 생략할 수 있는 기호는 무엇인가?

① < ② # ③ ; ④ ?

03 다음 중 매크로 전처리에 사용되는 키워드는 무엇인가?

① `define` ② `object` ③ `include` ④ `link`

04 다음 중 매크로를 여러 줄에 걸쳐서 쓸 때 사용하는 기호는 무엇인가?

① < ② # ③ ; ④ \

05 다음 중 함수 매크로에서 변수를 문자열로 취급하려 할 때 변수 앞에 붙이는 기호는 무엇인가?

① < ② # ③ ; ④ \

14.2 조건부 컴파일

1 조건부 컴파일

- 오류를 찾거나 서로 다른 운영체제에서 동작하는 코드를 만들 때 편리한 조건부 매크로

- "#ifdef 식별자"는 식별자가 정의되어 있으면 코드 1과 코드 2가 실행
- #endif는 조건부 매크로의 끝을 나타냄

조건부 매크로

```
#ifdef 조건
    코드 1;
    코드 2;
#endif
```

14.2 조건부 컴파일

[예제 14-9] 조건부 매크로를 이용한 코드

```c
#include <stdio.h>

// #define DEBUG                                  // 맨 앞 주석(//) 지우면 작동

int main() {
   int test = 7;

#ifdef DEBUG                                      // DEBUG 정의되면 printf()실행
   printf("디버그 켜졌음: test 값은 %d\n", test);
#endif
   printf("프로그램 작동\n");

   return 0;
}
```

실행 화면

```
프로그램 작동
```

3번 줄 주석 제거 후 실행 화면

```
디버그 켜졌음: test 값은 7
프로그램 작동
```


14.2 조건부 컴파일

- "#ifndef 식별자"는 식별자가 정의되지 않을 때만 작동
- 만약 MAX가 선언되지 않았을 경우에만 MAX를 선언하는 코드는 다음과 같음

```c
#ifndef MAX
#define MAX 100
#endif
```

- "#undef는 매크로 선언을 취소
- 지금까지 사용했던 매크로 선언을 취소하고 새로운 매크로를 선언할 때 사용하는 방법은 다음과 같음

```c
#define MAX 100
#undef MAX
#define MAX 200
```

2 다중 조건 매크로

- '#ifdef 식별자'는 식별자가 선언되었는지 안 되었는지 만으로 컴파일 되는 코드를 결정
- 만약 값을 사용하고 싶다면 다음과 같은 "#if (조건)"을 사용

조건부 매크로	

```
#if (조건)
   코드 1;
   코드 2;
#endif
```


[예제 14-10] 조건부 매크로를 이용한 코드

```
01  #include <stdio.h>
02
03  #define DEBUG 2
04
05  int main() {
06     int test = 7, temp = 10;
07
08  #if (DEBUG == 1)
09     printf("디버그 켜졌음: test 값은 %d\n", test);
10  #endif
11
12     printf("프로그램 작동\n");
13
14  #if (DEBUG == 2)
15     printf("디버그 켜졌음: temp 값은 %d\n", temp);
16  #endif
17
18     return 0;
19  }
```

실행 화면

```
프로그램 작동
디버그 켜졌음: temp 값은 10
```

14.2 조건부 컴파일

- #if가 있다면 당연히 #else도 있음
- 조건에 따라 서로 다르게 실행하는 조건부 매크로를 만들고 싶다면 다음과 같이 정의

다중 조건 매크로

```
#ifdef 조건
   코드 1;
#else
   코드 2;
#endif
```

```
#if (조건)
   코드 1;
#else
   코드 2;
#endif
```


14.2 조건부 컴파일

[예제 14-11] 조건부 매크로를 사용한 UFO 코드

```
01  #include <stdio.h>
02  #include <stdlib.h>                          // system()이 있는 라이브러리
03
04  // #define UNIX
05
06  int main() {
07     int k, m;
08
09     for (k = 0; k < 70; k++) {
10  #ifdef UNIX
11        system("clear");                        // 유닉스일 때 실행되는 코드
12  #else
13        system("cls");                          // 윈도우일 때 실행되는 코드
14  #endif
15        for (m = 0; m < k; m++) printf(" ");          // 빈칸을 만든다
16        printf("<#=#>\n");                            // UFO 모양 출력
17     }
18     return 0;
19  }
```

- 조건을 여러 개 만들고 싶다면 "#elif (조건)"을 사용
- #elif는 else if를 줄인 말
- #elif의 형식은 다음과 같음

다중 조건 매크로

```
#if (조건 1)
   코드 1;
#elif (조건 2)
   코드 2;
#else
   코드 3;
#endif
```


• 조건부 컴파일 지시자

- 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 예제19-7.c

```c
#include <stdio.h>
#pragma pack(push, 1)
typedef struct
{
    char ch;
    int in;
} Sample1;

#pragma pack(pop)
typedef struct
{
    char ch;
    int in;
} Sample2;

int main(void)
{
    printf("Sample1 구조체의 크기 : %d바이트\n", sizeof(Sample1));
    printf("Sample2 구조체의 크기 : %d바이트\n", sizeof(Sample2));

    return 0;
}
```

실행결과

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

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


■ 컴파일러에게 정보를 전달해 줍시다 : #pragma

```
#pragma warning(disable:4996)
```

↓ ↓ ↓

"4996번 경고 메시지는 그냥 뿌리지 마세요."

#pragma 명령문의 구성은 컴파일러마다 차이를 보인다.

중간 점검

1. 전처리기 지시자 #ifdef을 사용하여 TEST가 정의되어 있는 경우에만 화면에 "TEST"라고 출력하는 문장을 작성하여 보자.

06 다음 중 조건이 없는 조건부 매크로를 시작하는 단어는 무엇인가?

① `if` ② `#ifdef` ③ `#if` ④ `#defif`

07 다음 중 조건부 매크로의 끝에 사용하는 단어는 무엇인가?

① `#else` ② `#end` ③ `#ifend` ④ `#endif`

08 다음 중 목적 파일의 확장자는 무엇인가?

① `.exe` ② `.obj` ③ `.dll` ④ `.bin`

09 다음 중 조건이 있는 조건부 매크로를 시작하는 단어는 무엇인가?

① `if` ② `#ifdef` ③ `#if` ④ `#defif`

10 다음 중 다중 조건 조건부 매크로에서 2번째 조건에 사용하는 단어는 무엇인가?

① `#elif` ② `#else` ③ `#elseif` ④ `#ifelse`

마무리

❖ 다음 중에서 설명이 옳지 않은 문장을 수정하시오.

① 소스 파일과 전처리 후의 파일은 형태가 같은 텍스트 파일이다.

② 전처리 지시자 include는 파일의 내용을 단순히 복사한다.

③ 매크로 함수를 많이 사용하면 실행 속도는 빨라지나 프로그램의 크기가 커진다.

④ #ifdef 지시자는 수식과 매크로명을 모두 조건식으로 쓸 수 있다.

마무리

❖ 다음과 같이 매크로 함수가 정의되어 있을 때 출력문의 결과를 적으시오.

```
#define SUM(x, y) x + y
printf("%d", SUM(20, 6) * 3);
```

20 + 6 * 3

마무리

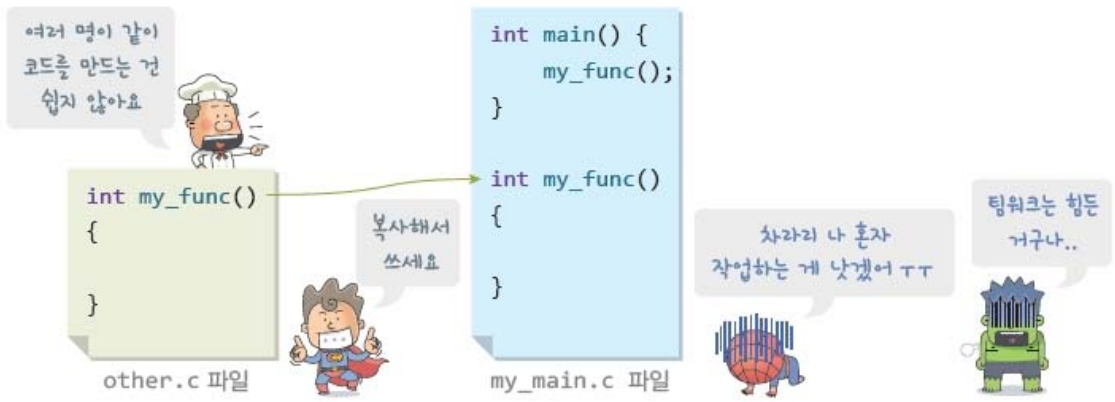
다음 프로그램의 결과를 적으시오.

```
#define DEBUG
#define LEVEL 2
int main(void) {
        int flag;
        #ifndef DEBUG
                flag = 0;
        #elif LEVEL == 1
                flag = 1;
        #elif defined(MAX_LEVEL) && (LEVEL == 2)
                flag = 2;
        #else   flag = 3;
        #endif
                printf("%d", flag);
        return 0; }
```


14.3 다중 소스 파일

1 다중 소스 파일 이해하기

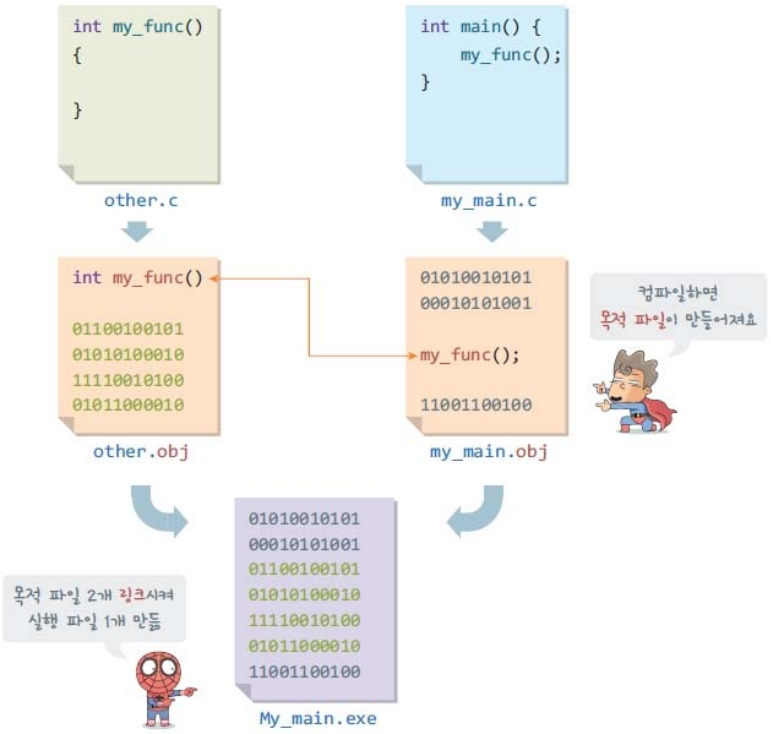
- 다중 소스(multiple source) 파일이란 여러 개의 소스 파일을 사용하여 하나의 코드를 완성하는 것
- 커다란 코드를 2명이 나누어 작성하는 팀 프로젝트를 수행한다고 가정
  - 2명이 따로 만든 코드를 하나로 합쳐서 컴파일을 하면 더 많은 오류가 나올 것
- 다중 소스 파일이 필요한 또 다른 경우도 있음
- 여러 명이 나누어 작업 할 때 공동으로 사용하는 함수가 있는 경우, 매번 사용할 때 마다 복사-붙여 넣기를 하는 것은 번거로움



14.3 다중 소스 파일

2 실행 파일이 만들어지는 과정

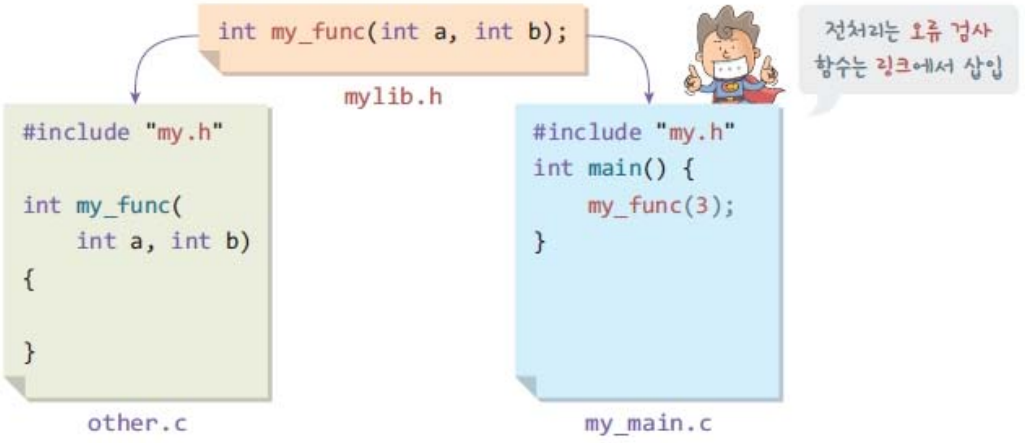
- 우리가 mycode.c를 컴파일 하면 실행파일인 mycode.exe가 바로 만들어 지는 것이 아님
- 실행파일이 만들어 지기 전에 객체(object) 파일인 "mycode.obj"가 만들어 짐
  - 객체파일이란 오류검사를 거쳐 만든 실행파일 직전의 파일
  - 외부함수가 비어 있는 상태
- my_main.c를 만든 사람은 other.obj와 my_main.obj를 같이 컴파일
- 이 단계를 링크(link) 단계라 부름
- my_main.obj 비어 있던 my_func()함수의 실행코드가 채워지면서 실행 파일인 my_main.exe가 만들어짐



14.3 다중 소스 파일

3 다중 소스 파일 설계

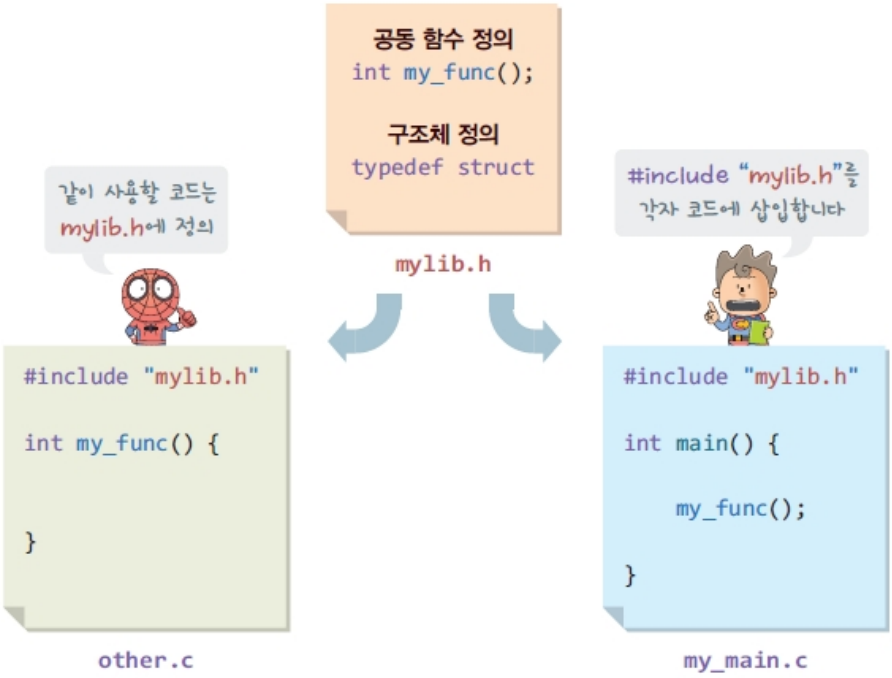
- #include<....h>로 컴파일 전에 .h 파일을 포함하는 이유 - 함수에서 발생할 수 있는 오류를 원형 점검하기 위함
- 라이브러리에 들어 있는 함수일 경우, 함수의 선언부를 전처리하여 오류를 점검하고, 실제의 실행코드가 삽입되는 것은 링크단계



14.3 다중 소스 파일

> 다중 소스 만드는 법

- 가장 먼저 다중 소스에서 사용할 공동의 함수이름과 매개변수를 정의하고, 해당 함수가 어떻게 작업 할 지를 결정
- 정의된 함수와 구조체를 헤더파일에 모아 놓은 후 각자 헤더파일을 포함 시켜 작업



■ 헤더파일에 무엇을 담으면 좋겠습니까?

```
extern int num;
extern int GetNumber(void);       //extern 생략 가능
#define PI 3.14

typedef struct __div
{
    int quotient;
    int remainder;
} div;
```

- 필요 시마다 소스파일에 반복해서 추가해야 하는 extern유형의 선언!
- 소스파일단위로 인식되는 매크로의 정의
- 소스파일단위로 추가해야 하는 구조체의 정의

■ 헤더파일의 중복 삽입 문제

stdiv.h

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

intdiv3.h

```
#include "stdiv.h"

div IntDiv(int num1, int num2);
```

main.c

```
#include <stdio.h>
#include "stdiv.h"
#include "intdiv3.h"

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

구조체 정의의 중복 삽입은

컴파일 에러로 이어진다!

함수의 선언이나 extern 선언은 중복 선언이 가능하지만 구조체는 중복 안됨

■ 헤더파일 중복 삽입 문제의 해결

stdiv2.h

```
#ifndef __STDIV2_H__
#define __STDIV2_H__

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

#endif
```

또는 #pragma once의 삽입

stdiv4.h

```
#ifndef __INTDIV4_H__
#define __INTDIV4_H__

#include "stdiv2.h"
div IntDiv(int num1, int num2);

#endif
```

main.c

```
#include <stdio.h>
#include "stdiv2.h"
#include "intdiv4.h"

int main(void)
{
    div val=IntDiv(5, 2);
    printf("몫 : %d ₩n", val.quotient);
    printf("나머지 : %d ₩n", val.remainder);

    return 0;
}
```

4 다중 소스 파일 만들기

- 다중 소스파일 헤더 - round.h

```
01  int round(char *str, int size);      // size는 널 문자를 제외한 문자 총 개수
```

- 다중 소스파일 - round.c

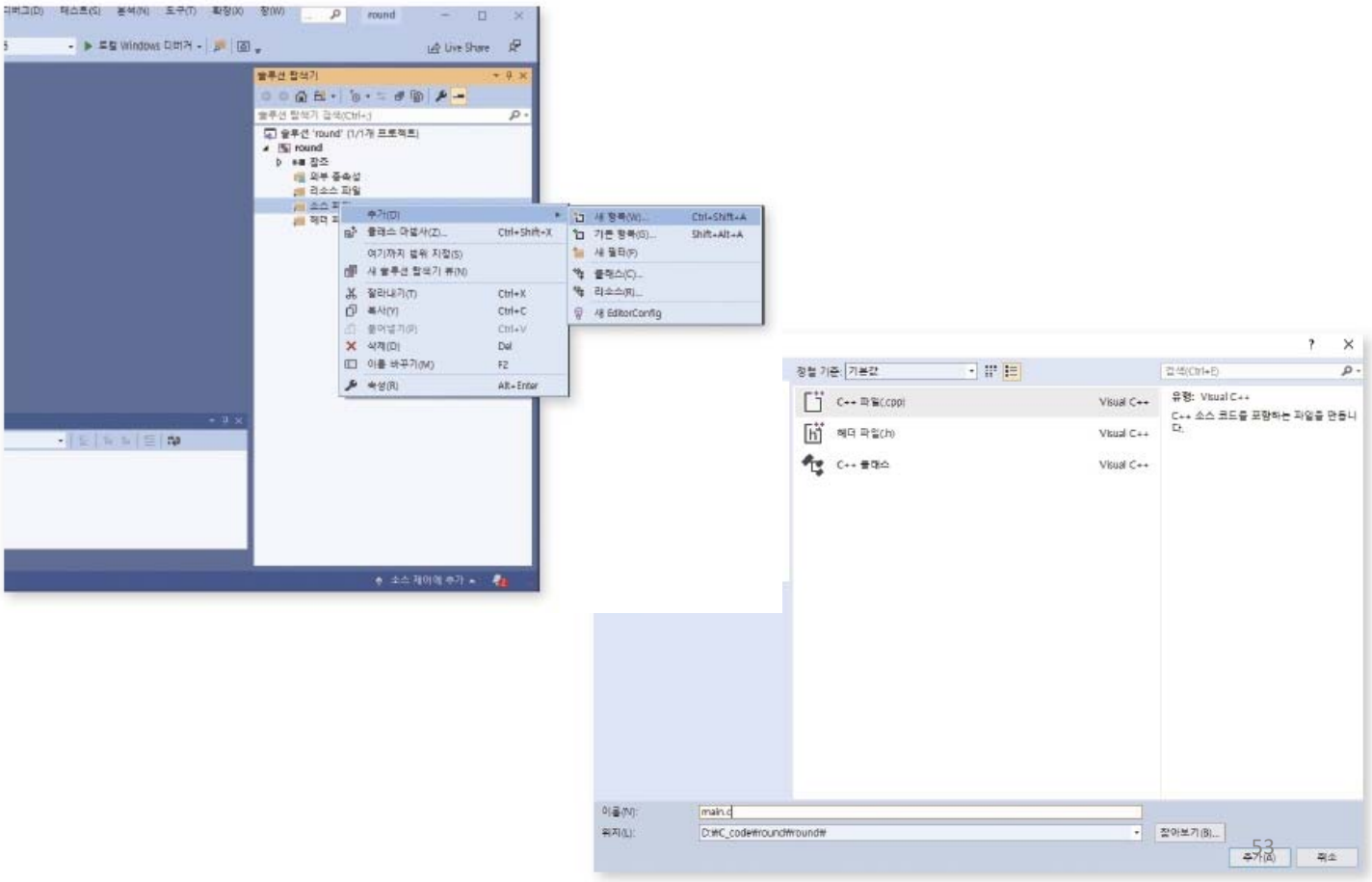
```
01  #include <stdio.h>
02  #include <stdlib.h>                          // system()이 있는 라이브러리
03  #include "round.h"                           // 사용자 헤더는 큰따옴표
04
05  int round(char *str, int size) {
06     int k, m;
07
08     for (k = 0; k < 30; k++) {
09        system("cls");
10        for (m = 0; m < size; m++)
11           putchar(str[(m + k) % size]);
12     }
13     return 0;
14  }
```


- 다중 소스파일 - main.c

```
01  #include "round.h"
02
03  int main() {
04     round("LOVE AGAIN", 10);
05     round("GOOD BYE", 8);
06     return 0;
07  }
```

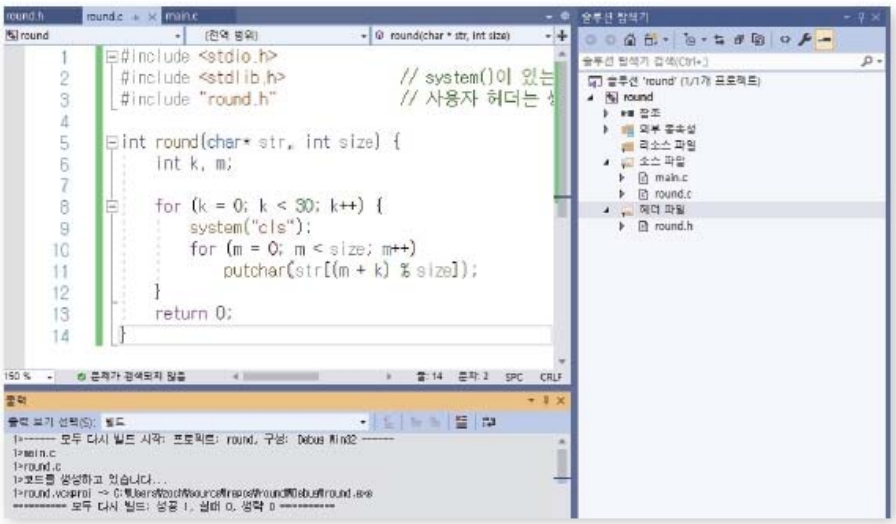
14.3 다중 소스 파일

비주얼 스튜디오에서 다중 소스 파일 실행하기



14.3 다중 소스 파일

비주얼 스튜디오에서 다중 소스 파일 실행하기



```
#include <stdio.h>
#include <stdlib.h>          // system()이 있는
#include "round.h"           // 사용자 헤더는

int round(char* str, int size) {
    int k, m;

    for (k = 0; k < 30; k++) {
        system("cls");
        for (m = 0; m < size; m++)
            putchar(str[(m + k) % size]);
    }
    return 0;
}
```

5 정적 변수

- 전역변수 - 블록 안에서만 존재하는 지역변수와 달리 코드 전체가 공유 할 수 있는 변수
- static 변수는 지역변수 형태와 전역변수 형태로 만들어 질 수 있음
- static을 지역변수로 선언하면 전역변수처럼 사용할 수 있는 특징이 있음

[예제 14-14] 전역 변수 테스트 코드

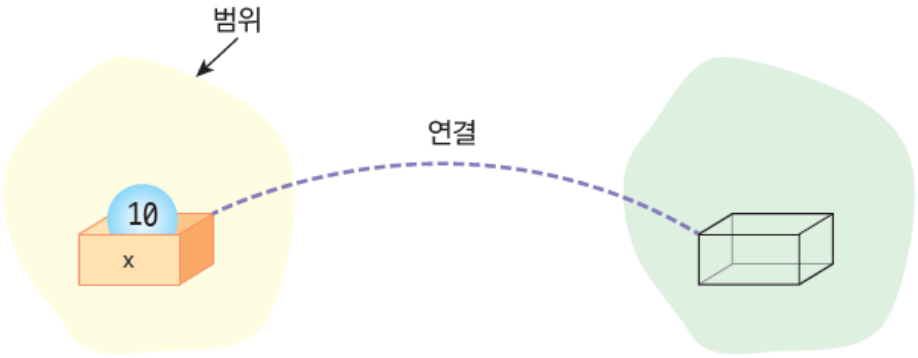
```c
#include <stdio.h>

void static_test(int num) {                 // 매개 변수 num
    static int order;                       // order는 정적 변수

    order = order + num;
    printf("전역 변수 값 = %d\n", order);
}                                           // return이 없음으로 void

int main() {

    static_test(3);                         // static_test(3) 함수 호출
    static_test(4);                         // static_test(4) 함수 호출

    return 0;
}
```

실행 화면

```
전역 변수 값 = 3
전역 변수 값 = 7
```

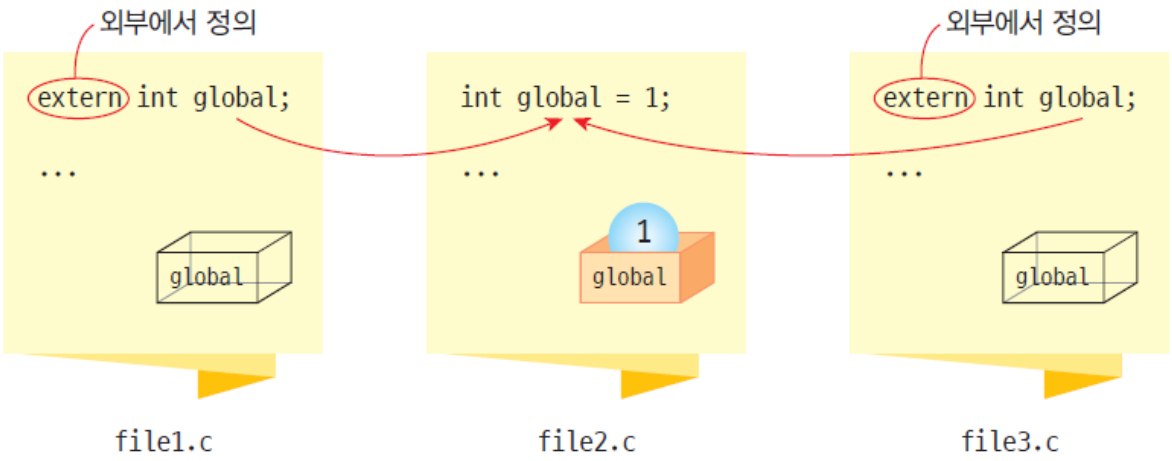
연결

- *연결(linkage):* 다른 범위에 속하는 변수들을 서로 연결하는 것
  - 외부 연결
  - 내부 연결
  - 무연결
- 전역 변수만이 연결을 가질 수 있다.



외부 연결

- 전역 변수를 extern을 사용해서 서로 연결



연결 예제

linkage1.c

```c
#include <stdio.h>
int all_files;      // 다른 소스 파일에서도 사용할 수 있는 전역 변수
static int this_file; // 현재의 소스 파일에서만 사용할 수 있는 전역 변수
extern void sub();

int main(void)
{
        sub();
        printf("%d\n", all_files);
        return 0;
}
```

연결

linkage2.c

```c
extern int all_files;
void sub(void)
{
        all_files = 10;
}
```

```
10
```

함수앞의 static

main.c

```c
#include <stdio.h>

extern void f2();
int main(void)
{
        f2();
        return 0;
}
```

static이 붙는 함수는 파일 안에서만 사용할 수 있다.

sub.c

```c
static void f1()
{
        printf("f1()이 호출되었습니다.\n");
}
void f2()
{
        f1();
        printf("f2()가 호출되었습니다.\n");
}
```

14.3 다중 소스 파일

6 외부 변수

- 파일 외부에서도 변수를 공유하고 싶다면 extern을 사용
- extern int turn은 새로운 변수를 만들지는 못하고, 이미 선언된 변수를 사용하겠다는 의미
- extern을 사용하여 static 전역변수를 접근하려하면 오류 발생
- 내 파일에서만 참조가 되고, 다른 파일에서는 볼 수 없는 변수를 만들고 싶다면 static 전역변수를 만들면 됨

```c
#include "round.h"

int turn;

int main() {
   turn = 50;
   round("GOOD BYE", 8);
   return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>                    // system()이 있는 라이브러리
#include "round.h"                     // 사용자 헤더는 큰따옴표

extern int turn;                       // main.c 변수 turn 사용

int round(char *str, int size) {
   int k, m;

   for (k = 0; k < turn; k++) {
      system("cls");
      for (m = 0; m < size; m++)
         putchar(str[(m + k) % size]);
   }
   return 0;
}
```

저장 유형 정리

- 일반적으로는 *자동 저장 유형* 사용 권장
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *정적 지역 변수*를 사용
- 많은 파일과 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

저장 유형	키워드	정의되는 위치	범위	생존 시간
자동	auto	함수 내부	지역	임시
레지스터	register	함수 내부	지역	임시
정적 지역	staic	함수 내부	지역	영구
전역	없음	함수 외부	모든 소스 파일	영구
정적 전역	static	함수 외부	하나의 소스 파일	영구
외부 참조	extern	함수 외부	모든 소스 파일	영구

11 다음 중 링크 단계에 사용되는 파일은 무엇인가?

① 소스 파일 ② 데이터 파일 ③ 실행 파일 ④ 목적 파일

12 다음 중 컴파일 이후에 라이브러리가 비어 있는 채로 만들어진 파일은 무엇인가?

① 소스 파일 ② 데이터 파일 ③ 실행 파일 ④ 목적 파일

13 다음 중 확장자가 .obj인 파일은 무엇인가?

① 소스 파일 ② 데이터 파일 ③ 실행 파일 ④ 목적 파일

14 다음 중 C 컴파일러 제작자가 제공하는 라이브러리에 사용하는 기호는 무엇인가?

① <> ② {} ③ () ④ ""

15 다음 중 사용자가 만든 헤더 파일에 사용하는 기호는 무엇인가?

① <> ② {} ③ () ④ ""

16 다음 중 블록 안에서만 살아있는 변수를 가리키는 단어는 무엇인가?

① 지역 변수 ② 전역 변수 ③ 정적 변수 ④ 외부 변수

17 다음 중 블록 안에서 선언되었으나 블록이 끝나도 사라지지 않는 변수는 무엇인가?

① 지역 변수 ② 전역 변수 ③ 정적 변수 ④ 외부 변수

19 다음 중 해당 파일에서만 볼 수 있고 외부 파일에서는 볼 수 없는 변수는 무엇인가?

① 지역 변수 ② 전역 변수 ③ 정적 변수 ④ 외부 변수

마무리

사칙연산 계산기 프로그램

❖ 사칙연산 수식을 입력하면 그 결과를 출력하는 프로그램을 작성하라. 단, 수식을 입력하는 함수와 사칙연산 함수를 매크로 함수로 작성하라.

// 실행예
수식 입력(종료 Ctrl+Z) : 10 + 20
10 + 20 = 30