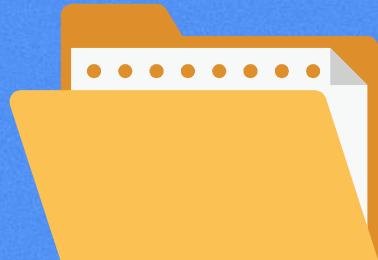




제04장

전처리와 입출력



01. 전처리



전처리 개요

- **전처리기 역할**
 - 컴파일(compile) 전에 전처리기(preprocessor)의 전처리(preprocess) 과정이 필요
 - 결과인 전처리 출력파일을 만들어 컴파일러에게 보내는 작업을 수행
- **전처리 지시자(preprocess directives)**
 - #include, #define과 같은 전처리 지시자는 항상 #으로 시작
 - 마지막에 세미콜론 ; 이 없는 등 일반 C 언어 문장과는 구별
 - 기타 조건 지시자로 ~~#if, #elif, #else, #endif, #ifndef, #undef~~ 등이 있음

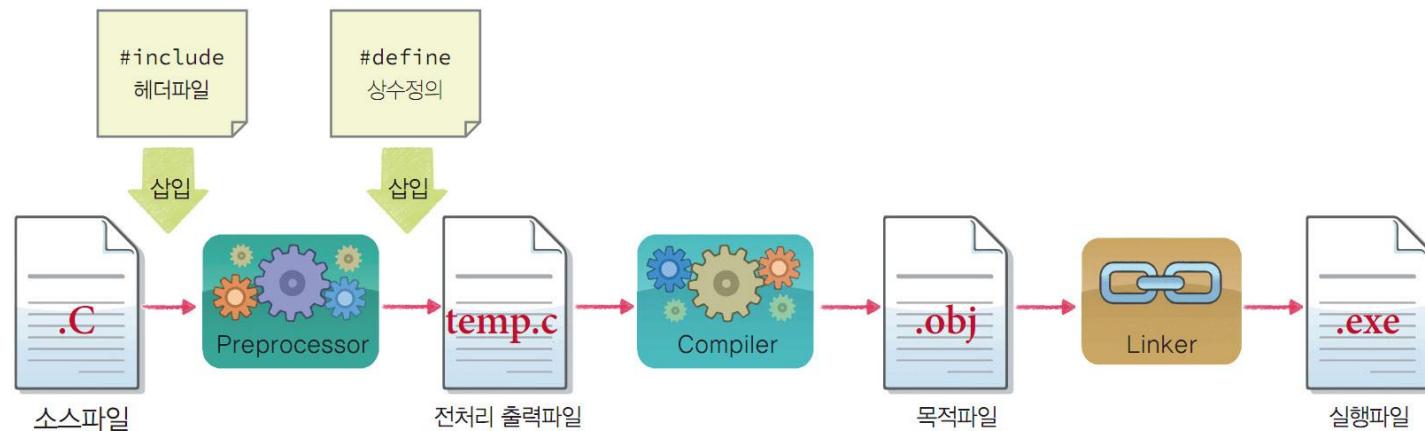


그림 4-1 전처리기와 컴파일러

전처리 지시자 #include

• 헤더파일

- `#include <stdio.h>`
 - `#include, #define` 등
 - 자료형의 재정의(`typedef`), 함수원형(`prototype`) 정의 등과 같은 문장이 있는 텍스트 파일
- 대표적인 헤더파일, 확장자 *.h
 - `stdio.h`
- 헤더파일 직접 보기

소스파일

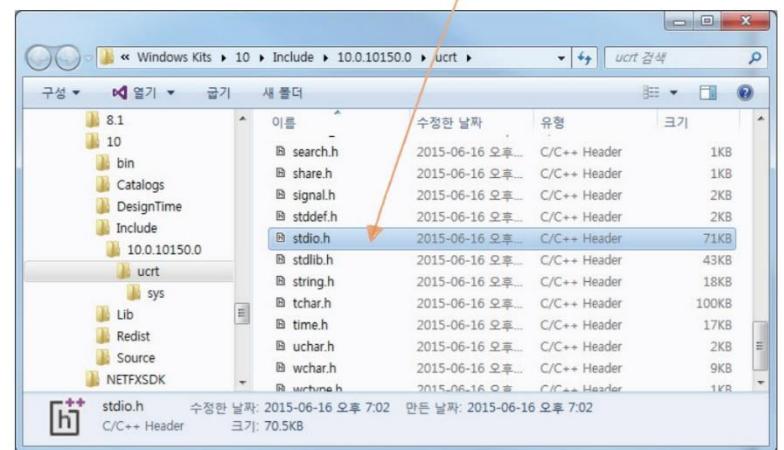
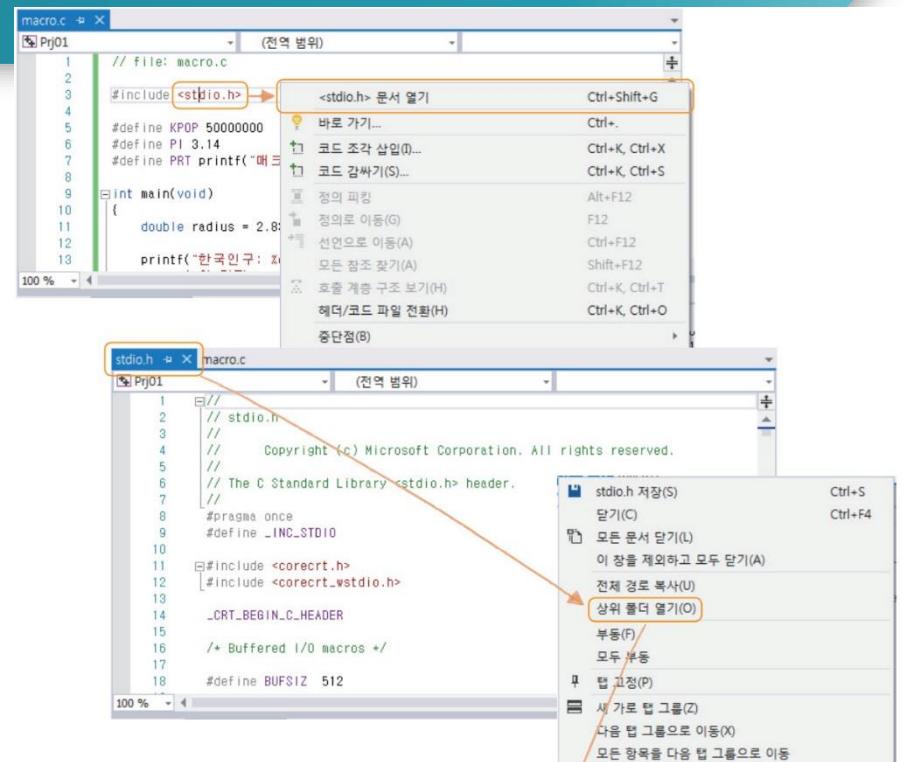
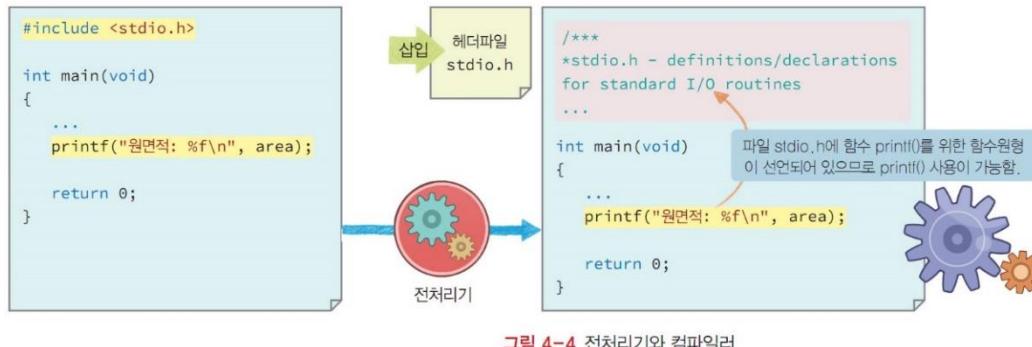


그림 4-3 비주얼 스튜디오에서 헤더파일 열기

전처리 지시자 #define

- **매크로 상수**
 - 전처리기(preprocessor)는 소스에서 정의된 매크로 상수를 모두 #define 지시자에서 정의된 문자열로 대체(replace)
- **#define name value**
 - #define에 정의된 name은 전처리기에 의해 모두 value로 대체되어 컴파일
 - #define은 하나의 텍스트(정수, 실수 또는 문자열 등)를 또 다른 텍스트(KPOP, PI, PRT등)로 정의
 - PI라는 매크로 상수
 - 전처리 과정에서 모두 3.14라는 실수로 값이 바뀐 소스로 컴파일
 - 단 매크로 상수는 문자열 내부 또는 주석 부분에서는 대체되지 않음

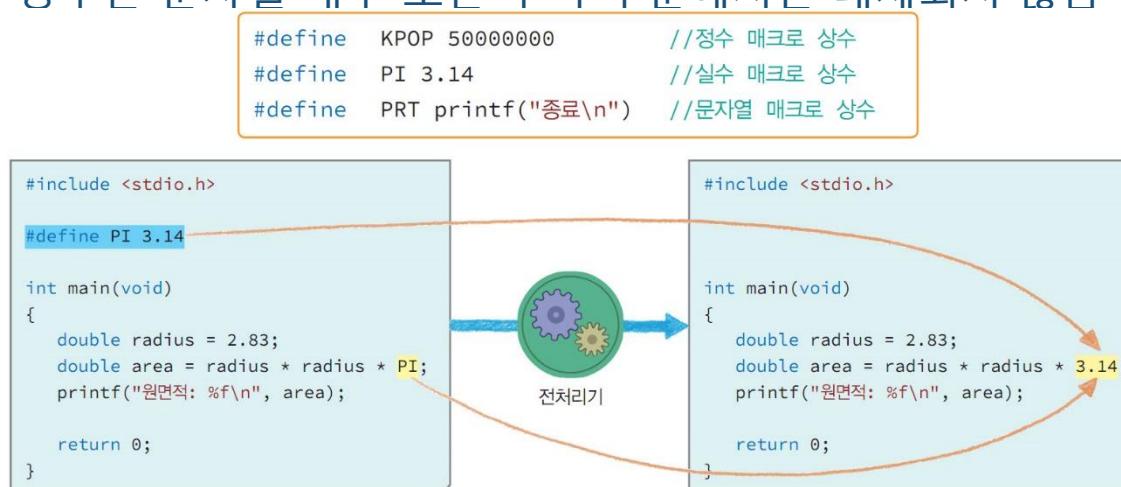


그림 4-5 매크로 상수와 전처리 과정

```

#define KPOP 50000000          //정수 매크로 상수
#define PI 3.14                //실수 매크로 상수
#define PRT printf("종료\n")    //문자열 매크로 상수

```

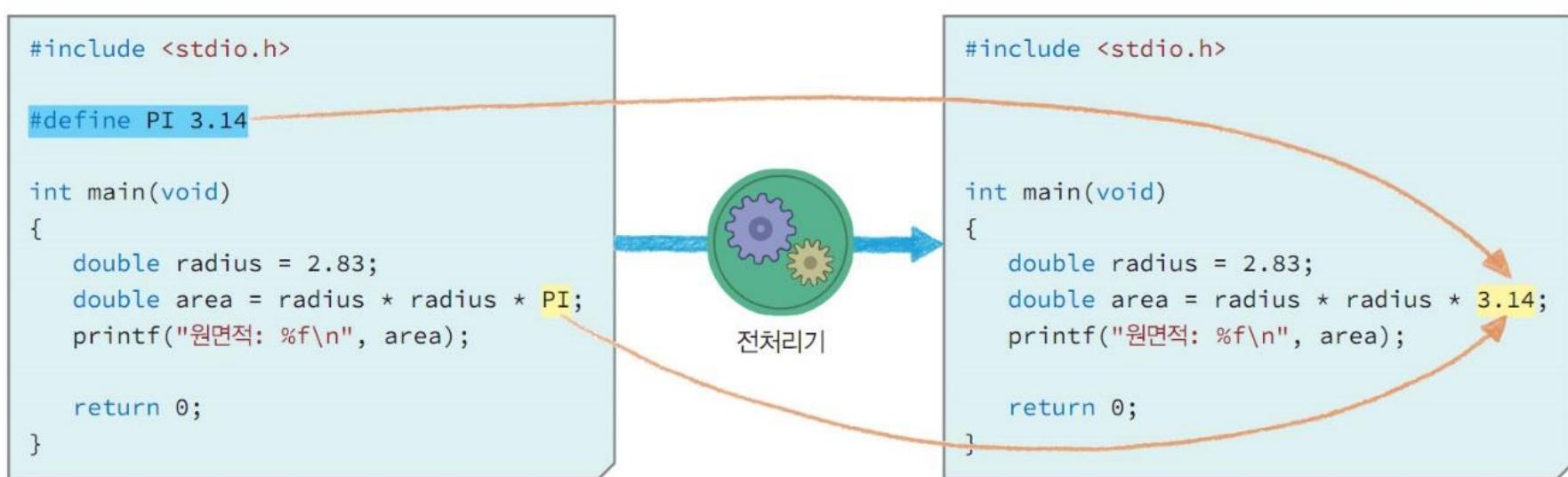


그림 4-5 매크로 상수와 전처리 과정

참고 : 인자를 사용한 매크로

예제 advancemacro.c

- #define에서 그 활용도를 높이기 위한 방안이 함수와 같이 인자(parameter)를 이용하는 방법

인자인 x부분이 실제 사용한 수로 대체된다.

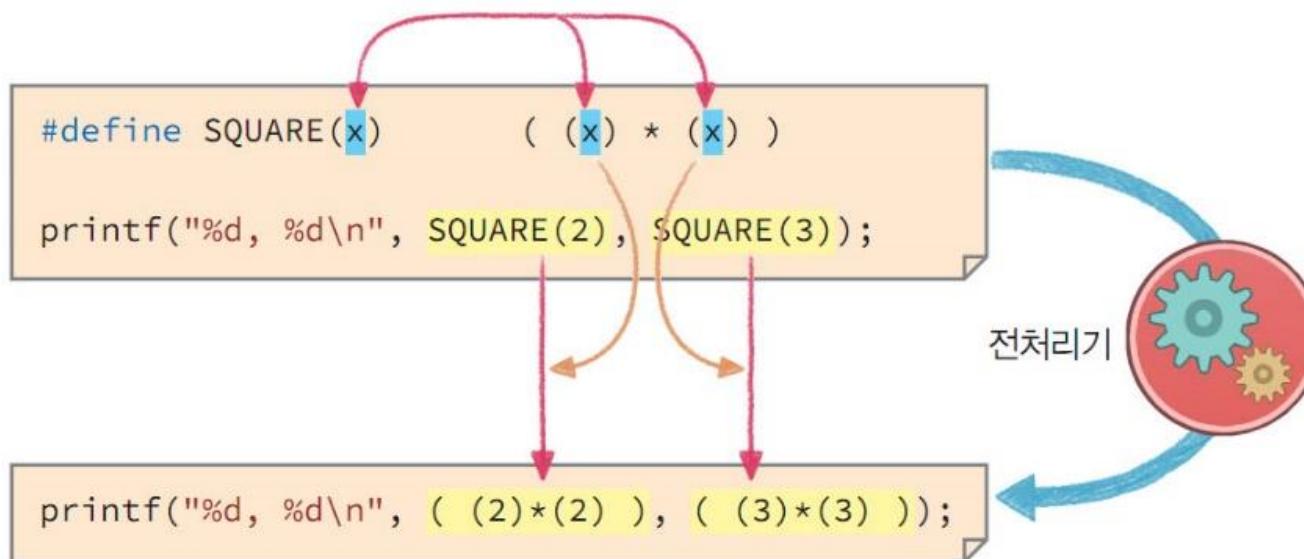
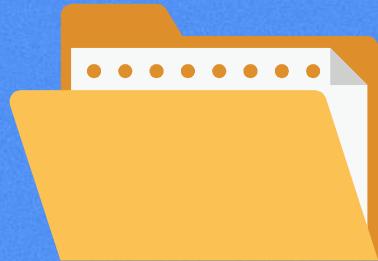


그림 4-7 인자가 있는 매크로의 치환



02. 출력함수 printf()



출력함수 printf()에서 형식지정자의 이해

• printf()의 인자

- 형식문자열과 출력할 목록으로 구분
 - 형식문자열에서 %d와 같이 %로 시작하는 형식지정자 순서대로 서식화하여 그 위치에 출력



그림 4-10 출력 함수 printf() 개요

• 함수 printf()의 첫 번째 인자인 형식문자열(format string)

- 일반문자와 이스케이프 문자
 - 이스케이프 문자는 `\\"`와 `\'`같이 `\`로 시작하는 문자
- 형식 지정자(format specification)로 구성
 - %d와 %s와 같이 %로 시작하는 형식지정자
- 형식지정자 %d 위치에 바로 20이라는 정수가 출력
 - 이스케이프 문자 `\\"`는 문자 "이 그대로 출력

```
printf("%d대 연애에서 가장 중요한 것은 \"밀당\"이다.", 20);
```

정수의 십진수, 8진수, 16진수 출력

- 함수 `printf()`에서 정수 출력을 위한 형식 지정자
 - 정수의 십진수 출력을 위한 형식 지정자는 `%d`와 `%i`
 - 8진수로 출력하려면 `%o`를 이용
 - 앞 부분에 숫자 0이 붙는 출력을 하려면 `%#o`를 이용
 - 소문자의 십육진수로 출력하려면 `%x`와 대문자로 출력하려면 `%X`를 이용
 - 16진수 앞에 `0x` 또는 `0X`를 붙여 출력하려면 `#`을 삽입하여 `%#x`와 `%#X`를 이용
- 함수 `printf()`
 - 반환값은 출력한 문자 수이며,
 - 오류가 발생하면 음수를 반환

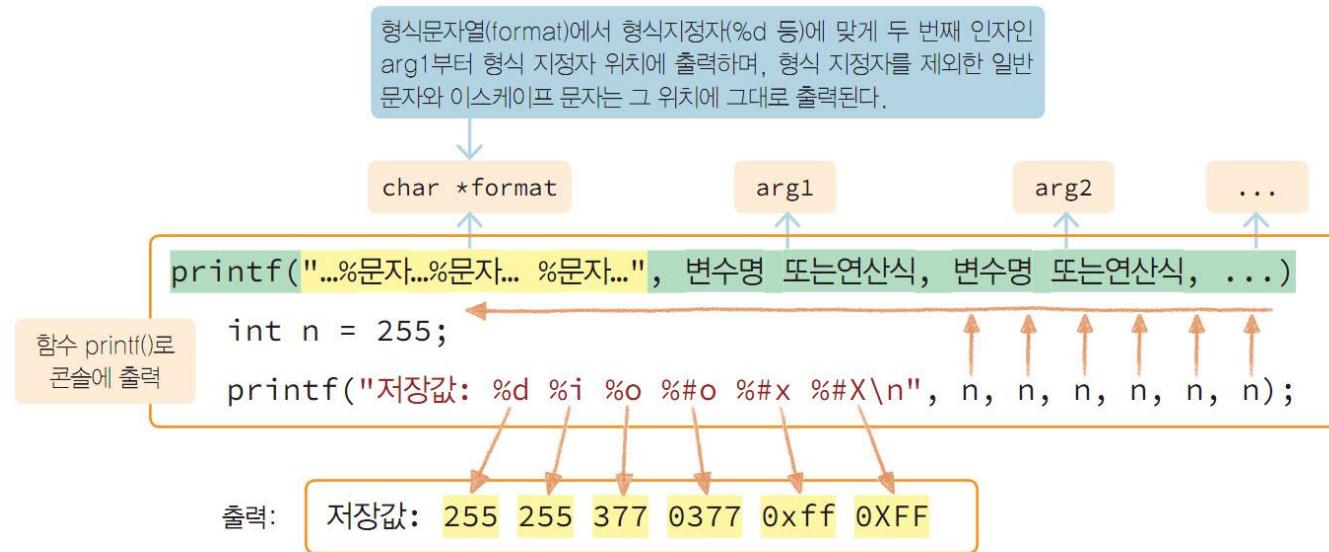


그림 4-12 콘솔출력을 위한 함수 `printf()`

실수를 위한 출력 %f

- **실수의 간단한 출력을 위한 형식 지정자는 %f**
 - 형식지정자 %f는 실수를 기본적으로 3.400000와 같이 소수점 6자리까지 출력
 - 함수 printf()에서 실수 출력으로 %f와 함께 %lf도 사용
- **출력 폭의 지정**
 - 출력 필드 폭이 출력 내용의 폭보다 넓으면 정렬은 기본이 오른쪽
 - 필요하면 왼쪽으로 지정

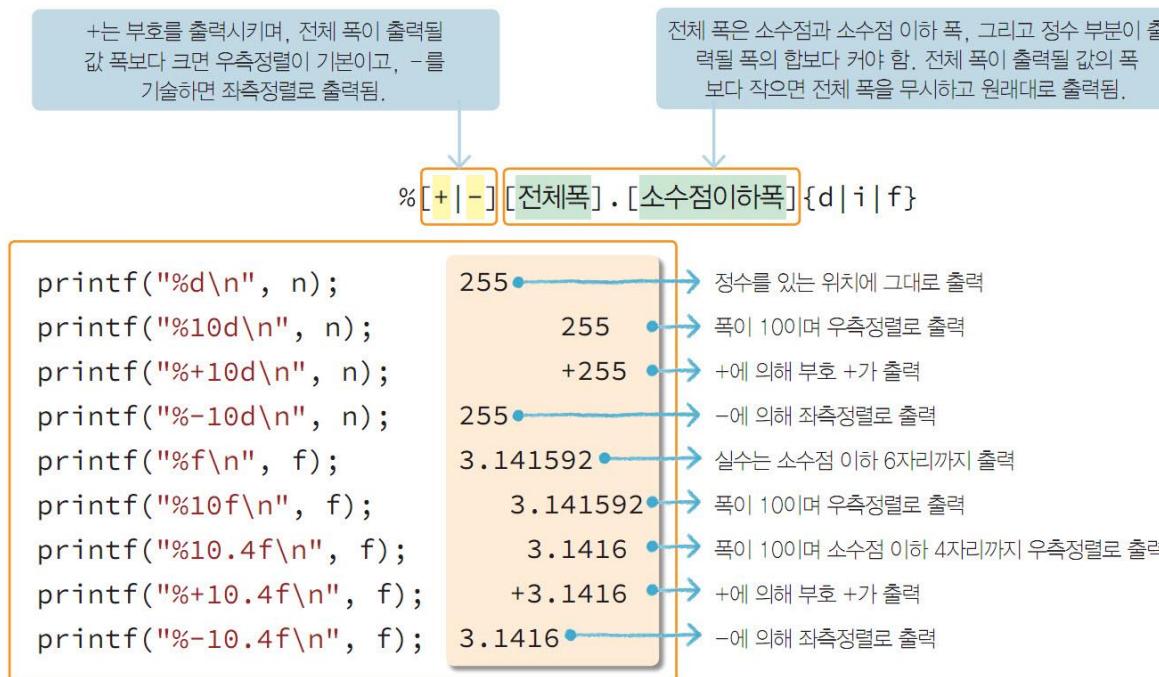


그림 4-13 폭과 정렬을 지정하는 형식 지정자

형식문자(type field characters) 종류

• 형식 문자

표 4-3 형식문자(type field characters) 종류

서식문자	자료형	출력 양식
c	char, int	문자 출력
d, i	int	부호 있는 정수 출력으로, lf는 long int, lld는 long long int형 출력
o	unsigned int	부호 없는 팔진수로 출력
x, X	unsigned int	부호 없는 십육진수 출력 x는 3ff와 같이 소문자 십육진수로, X는 3FF와 같이 대문자로 출력, 기본으로 앞에 0이나 0x, 0X는 표시되지 않으나 #이 앞에 나오면 출력
u	unsigned int	부호 없는 십진수(unsigned decimal integer)로 출력
e, E	double	기본으로 m.dddddExxx의 지수 형식 출력(정수 1자리와 소수점 이하 6자리, 지수승 3자리), 즉 123456.7890라면 1.234568e+005로 출력
f, lf	double	소수 형식 출력으로 m.123456 처럼 기본으로 소수점 6자리 출력되며, 정밀도에 의해 지정 가능, lf는 long double 출력
g, G	double	주어진 지수 형식의 실수를 e(E) 형식과 f 형식 중에서 짧은 형태(지수가 주어진 정밀도 이상이거나 -4보다 작으면 e나 E 사용하고, 아니면 f를 사용)로 출력, G를 사용하면 E가 대문자로
s	char *	문자열에서 '\0'가 나올 때 까지 출력되거나 정밀도에 의해 주어진 문자 수만큼 출력
p	void *	주소값을 십육진수 형태로 출력
%		%를 출력

• 옵션 지정

표 4-4 옵션지정 문자(flags) 종류

문자	기본(없으면)	의미	예와 설명
-	우측정렬	수는 지정된 폭에서 좌측정렬	%-10d
+	음수일 때만 - 표시	결과가 부호가 있는 수이면 부호 +, -를 표시	%+10d
0	0을 안 채움	우측정렬인 경우, 폭이 남으면 수 앞을 모두 0으로 채움	%010x %-0처럼 좌측정렬과 0 채움은 함께 기술해도 의미가 없음
#	리딩 문자 0, 0x, 0X가 없음	서식문자가 o(서식문자 octal)인 경우 0이 앞에 붙고, x(서식문자 hexa)인 경우 ox가 붙으며, X인 경우 0X가 앞에 붙음	수에 앞에 붙는 0이나 0x는 0으로 채워지는 앞 부분에 출력

표 4-3 형식문자(type field characters) 종류

서식문자	자료형	출력 양식
c	char, int	문자 출력
d, i	int	부호 있는 정수 출력으로, lf는 long int, lld는 long long int형 출력
o	unsigned int	부호 없는 팔진수로 출력
x, X	unsigned int	부호 없는 십육진수 출력 x는 3ff와 같이 소문자 십육진수로, X는 3FF와 같이 대문자로 출력, 기본으로 앞에 0이나 0x, 0X는 표시되지 않으나 #이 앞에 나오면 출력
u	unsigned int	부호 없는 십진수(unsigned decimal integer)로 출력
e, E	double	기본으로 m.dddddExxx의 지수 형식 출력(정수 1자리와 소수점 이하 6자리, 지수승 3자리), 즉 123456.7890라면 1.234568e+005로 출력
f, lf	double	소수 형식 출력으로 m.123456 처럼 기본으로 소수점 6자리 출력되며, 정밀도에 의해 지정 가능, lf는 long double 출력
g, G	double	주어진 지수 형식의 실수를 e(E) 형식과 f 형식 중에서 짧은 형태(지수가 주어진 정밀도 이상이거나 -4보다 적으면 e나 E 사용하고, 아니면 f를 사용)로 출력, G를 사용하면 E가 대문자로
s	char *	문자열에서 '\0'가 나올 때 까지 출력되거나 정밀도에 의해 주어진 문자 수만큼 출력
p	void *	주소값을 십육진수 형태로 출력
%		%를 출력

ld

LAB 정수와 실수, 문자와 문자열의 출력

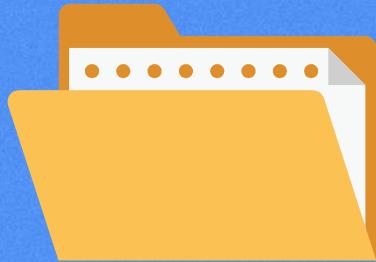
- 개인의 성별과 이름, 나이, 성적 등 개인 정보 출력 프로그램
 - 자료형 int 형인 나이와 double 형 성적 평균평점을 등을 출력
 - 성별, 나이, 몸무게, 평균평점을 다음과 같이 출력
- 결과
 - 성별: M
 - 이름: 안 병훈
 - 나이: 20
 - 몸무게: 62.49
 - 평균평점(GPA): 3.880

Lab 4-2 basictoutput.c

```
01 // basictoutput.c:  
02  
03 #include <stdio.h>  
04  
05 int main(void)  
06 {  
07     int age = 20;  
08     double gpa = 3.88f;  
09     char gender = 'M';  
10     float weight = 62.489f;  
11  
12     printf("성별: ____\n", _____);  
13     printf("이름: %s\n", "안 병훈");  
14     printf("나이: ____\n", _____);  
15     printf("몸무게: %.2f\n", weight);  
16     printf("평균평점(GPA): ____\n", _____);  
17  
18     return 0;  
19 }
```

정답

```
12     printf("\n성별: %c\n", gender);  
14     printf("나이: %d\n", age);  
16     printf("평균평점(GPA): %.3f\n", gpa);
```



03. 입력함수 `scanf()`



함수 `scanf()`와 정수 입력

- **함수 `scanf()`**

- 대표적인 입력함수. `%s`와 `%d`같은 동일한 형식 지정자를 사용
 - 표준 입력으로부터 여러 종류의 자료값을 훑어 주소연산자 `&`가 붙은 변수 목록에 저장
- 첫 번째 인자는 형식문자열(format string)
 - 형식지정자(format specification)는 `%d`, `%c`, `%lf`, `%f`와 같이 `%`로 시작
- 두 번째 인자부터는 키보드 입력값이 복사 저장되는 입력변수 목록
 - 변수이름 앞에 반드시 주소연산자 `&`를 붙여 나열
- 반환 유형은 `int`로
 - 표준입력으로 변수에 저장된 입력 개수를 반환, 다음 문장의 반환값은 3

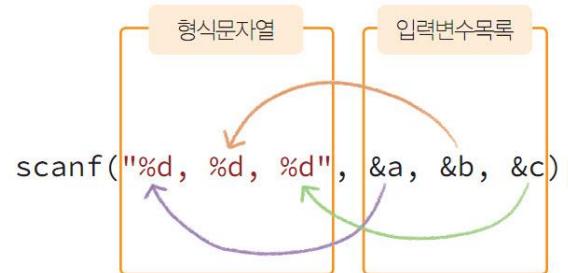


그림 4-15 CT 촬영과 같이 표준입력의 자료를 스캔(scan)하여 주소가 지정된 변수에 저장

- **int 형 변수 `year`에 키보드 입력값을 저장**

- `scanf("%d", &year)`
 - `year`로 기술하면 입력값이 저장될 주소를 찾지 못해 오류가 발생

정수 입력

예제 scanf.c

- 지정된 형식지정자에 맞게 키보드로 적당한 값을 입력한 후 [Enter] 키를 누르기 전까지는 실행을 멈춰 입력을 기다림
- 만일 년, 월, 일을 2017-4-29과 같이 중간에 -를 넣어 입력 받으려면 함수 `scanf("%d - %d - %d", ...)` 처럼 형식문자열에 입력 형식을 명시
- 여러 입력값을 구분해주는 구분자(separator)
 - , / , 콤마(,) 등을 사용할 수 있는데, 입력된 구분자는 형식만 체크하고 저장하지 않음

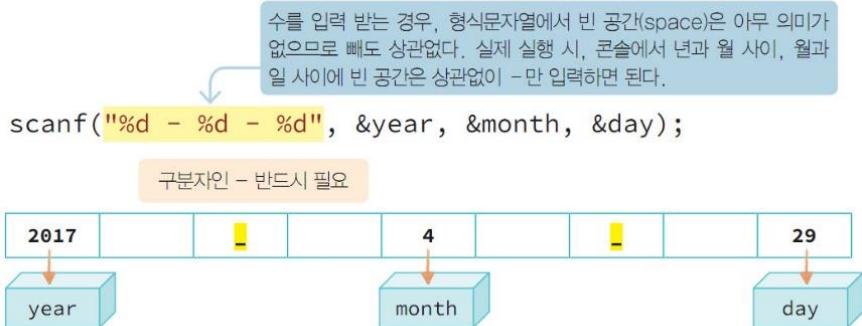


그림 4-17 입력값 사이에 특정 형식 지정

```
05 int main(void)
06 {
07     int year = 0;
08     printf("당신의 입학년도는? ");
09     scanf("%d", &year); //scanf("당신의 입학년도는 ? %d", &year); 문제 발생
10     printf("입학년도: %d\n\n", year);
11
12     int month, day;
13     printf("당신의 생년월일은? ");
14     scanf("%d - %d - %d", &year, &month, &day);
15     printf("생년월일: %d-%d-%d\n", year, month, day);
16
17     return 0;
18 }
```

설명

09 `scanf()` 바로 이전에 입력에 대한 정보를 주는 문자열인 프롬프트(prompt)를 출력하는데, 이러한 정보가 출력되지 않으면, 실행시 커서만 깜빡거리 사용자가 진행을 못하는 경우가 발생
10 간혹 `scanf("당신의 입학년도는 ? %d", &year);` 문장으로 값을 입력받으려 하는데, 잘못된 문장으로 `scanf()`의 형식문자열에 표시된 문자는 꼭 입력이 되어야 하는 형식이며, 콘솔에서 입력년도를 입력한 후 반드시 `[return]` 키 입력이 필요
15 입력값이 정수이므로 형식문자열 "%d-%d-%d"도 같은 기능을 수행

실행결과

당신의 입학년도는? 2016
입학년도: 2016

붉은색인 입학년도를 입력한 후 [Enter] 키
를 눌러야 프로그램이 진행됨

당신의 생년월일은? 2000-4-15
생년월일: 2000-4-15

2000과 4 사이, 4와 15 사이에는 반드시 -를 입력해야 함

실습예제 4-9

scanf.c

```
01 // file: scanf.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
```

```

05
06 int main(void)
07 {
08     int year = 0;
09     printf("당신의 입학년도는? ");
10     scanf("%d", &year); //scanf("당신의 입학년도는 ? %d", &year); 문제 발생
11     printf("입학년도: %d\n\n", year);
12
13     int month, day;
14     printf("당신의 생년월일은? ");
15     scanf("%d - %d - %d", &year, &month, &day);
16     printf("생년월일: %d-%d-%d\n", year, month, day);
17
18     return 0;
19 }

```

설명

- 09 scanf() 바로 이전에 입력에 대한 정보를 주는 문자열인 프롬프트(prompt)를 출력하는데, 이러한 정보가 출력되지 않으면, 실행시 커서만 깜박거려 사용자가 진행을 못하는 경우가 발생
- 10 간혹 scanf("당신의 입학년도는 ? %d", &year); 문장으로 값을 입력받으려 하는데, 잘못된 문장으로 scanf()의 형식문자열에 표시된 문자는 꼭 입력이 되어야 하는 형식이며, 콘솔에서 입학년도를 입력한 후 반드시 [return] 키 입력이 필요
- 15 입력값이 정수이므로 형식문자열 "%d-%d-%d"도 같은 기능을 수행

실행결과

당신의 입학년도는? 2016
입학년도: 2016

붉은색인 입학년도를 입력한 후 [Enter] 키
를 눌러야 프로그램이 진행됨

당신의 생년월일은? 2000-4-15
생년월일: 2000-4-15

2000과 4 사이, 4와 15 사이에는 반드시 -를 입력해야 함

실수와 문자의 입력

- 제어문자 %f와 %lf, %c

- 입력자료를 실수 float형 변수에 저장: 형식 지정자 %f를 사용
 - 실수 double형 변수에 저장: 형식 지정자 %lf를 사용
- 입력 자료를 문자 char형 변수에 저장: 제어문자 %c를 사용
- 출력 printf()에서 실수의 출력을 위한 형식 지정자
 - %f와 %lf를 모두 사용 가능

scanf() 오류를 방지하기 위한 상수 정의

- #define _CRT_SECURE_NO_WARNINGS
- 위 scanf() 오류방지를 위한 매크로 상수 지시자가 없으면 error C4996이 발생

실수와 문자 입력

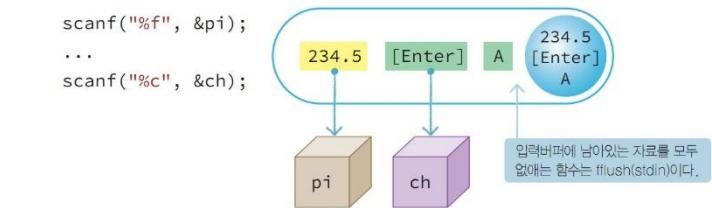
예제 floatcharscan.c

- 두 번의 scanf() 호출로, 콘솔에서 실수 234.5하나를 입력한 후 [Enter] 키를 누르고 다음 줄에 문자 'A'를 입력하여 변수 ch에 저장

- 입력버퍼에는 [Enter] 키가 남아 있어, 두 번째 scanf()에서 char형 변수 ch에는 순서대로 [Enter]인 문자 '\n'가 저장되고, 실제 문자 'A'는 저장되지 않는 문제가 발생

- 이러한 문제를 해결하는 방법

- 문자를 입력 받는 형식지정자 %c 앞에 "공백문자를 넣어" 형식문자열을 "%c"로 지정
- 아직 입력버퍼에 남아있는 [Enter] 키가 %c 앞에 공백문자로 인식되어 무시되고,
- 이어 커서 위치에 입력되는 'A'가 변수 ch에 저장



실습예제 4-11 floatcharscan.c

```
01 // file: floatcharscan.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     float pi;
09     printf("원주율을 입력하세요.\n");
10     scanf("%f", &pi);
11     printf("%f\n", pi);
12
13     char ch1, ch2;
14     printf("구분자를 공백으로 두 문자를 입력하세요.\n");
15
16     //가장 앞에 공백을 두어 enter를 제거, 구분자로 공백(여러 개도 가능)을 사용
17     scanf(" %c %c", &ch1, &ch2);
18     printf("ch1=%c ch2=%c\n", ch1, ch2);
19
20 }
```

설명

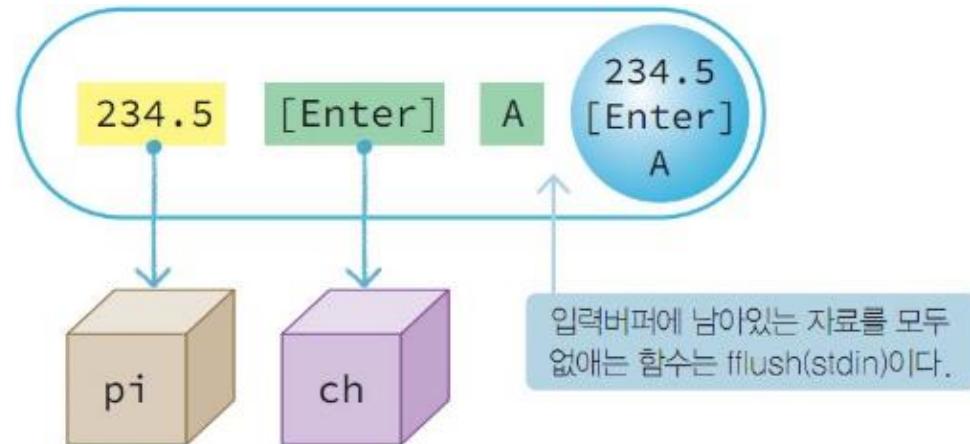
10 scanf()에서 원주율을 입력 받고, 이후 입력한 Enter키도 버퍼에 남아 있으므로 다음에 문자를 계속 입력 받으면 주의가 필요
16 scanf()에서 이전에 입력된 Enter키를 처리하기 위해 반드시 " %c %c"와 같이 %c 앞에 공백이 필요하며, 이후 문자 입력하는 문자와 문자 사이에도 공백이 필요함

실행결과

원주율을 입력하세요.
3.14159265
3.141593
구분자를 공백으로 두 문자를 입력하세요.
A &
ch1=A ch2=&

그림 4-19 문자 입력의 문제와 해결 방법

```
scanf("%f", &pi);  
...  
scanf("%c", &ch);
```



해결방법 1

```
scanf("%f", &pi);  
  
fflush(stdin); //입력 버퍼를 모두 비움  
  
scanf("%c", &ch);
```

해결방법 2

```
scanf("%f ", &pi);  
  
scanf(" %c", &ch);
```

그림 4-19 문자 입력의 문제와 해결 방법

실습예제 4-11

floatcharscanf.c

```
01 // file: floatcharscanf.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     float pi;
09     printf("원주율을 입력하세요.\n");
10     scanf("%f", &pi);
11     printf("%f\n", pi);
12
13     char ch1, ch2;
14     printf("구분자를 공백으로 두 문자를 입력하세요.\n");
15
16     //가장 앞에 공백을 두어 enter를 제거, 구분자로 공백(여러 개도 가능)을 사용
17     scanf(" %c %c", &ch1, &ch2);
18     printf("ch1=%c ch2=%c\n", ch1, ch2);
19
20 }
```

설명

- 10 `scanf()`에서 원주율을 입력 받고, 이후 입력한 Enter키도 버퍼에 남아 있으므로 다음에 문자를 계속 입력 받으려면 주의가 필요
- 16 `scanf()`에서 이전에 입력된 Enter키를 처리하기 위해 반드시 " %c %c"와 같이 %c 앞에 공백이 필요하며, 이후 문자 입력하는 문자와 문자 사이에도 공백이 필요함

실행결과

원주율을 입력하세요.
3.14159265
3.141593
구분자를 공백으로 두 문자를 입력하세요.
A &
ch1=A ch2=&

다양한 형식지정자

예제 radixscan.c

- 함수 scanf()에서 정수의 콘솔입력값
- 8진수로 인지하려면 %0를 사용
- %x는 16진수로 인지
- 함수 scanf()에서 이용되는 다양한 형식지정자

표 4-8 함수 scanf()의 형식 지정자

형식 지정자	콘솔 입력값의 형태	입력 변수 인자 유형
%d	십진수로 인식	정수형 int 변수에 입력값 저장
%i	십진수로 인식하며, 단 입력값에 0이 앞에 붙으면 8진수로 0x가 붙으면 16진수로 인식하여 저장	정수형 int 변수에 입력값 저장
%u	unsigned int로 인식	정수형 unsigned int 변수에 입력값 저장
%o	8진수로 인식	정수형 int 변수에 입력값 저장
%x, %X	16진수로 인식	정수형 int 변수에 입력값 저장
%f	부동소수로 인식	부동소수형 float 변수에 입력값 저장
%lf	부동소수로 인식	부동소수형 double 변수에 입력값 저장
%e, %E	지수 형태의 부동소수로 인식	부동소수형 float 변수에 입력값 저장
%c	문자로 인식	문자형 char 변수에 입력값 저장
%s	일련의 문자인 문자열(string)로 인식	문자열을 저장할 배열에 입력값 저장
%p	주소(address) 값으로 인식	정수형 unsigned int 변수에 입력값 저장

실습예제4-12 radixscan.c

```
01 // file: radixscan.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int a, b, c;
09     printf("십진수, 팔진수, 십육진수를 각각 입력하세요.\n");
10     scanf("%d %o %x", &a, &b, &c);
11     printf("%d %#o %#x\n\n", a, b, c);
12
13     printf("십진수, 팔진수(0리딩 표현), 십육진수(0x리딩 표현)를 각각 입력하세요.\n");
14     scanf("%i %i %i", &a, &b, &c);
15     printf("%d %d %d\n", a, b, c);
16
17     return 0;
18 }
```

설명

형식지정자 %d는 십진수, %o는 팔진수, %x는 십육진수 정수를 입력
형식지정자 %i인 경우, 입력값이 03과 같이 0이 리딩하는 수는 팔진수로 인식하며, 0x1f과 같이 0x로 리딩하는 수는 십육진수로 인식

실행결과

십진수, 팔진수, 십육진수를 각각 입력하세요.
82 67 1F ← 067 0x1f로 입력도 가능
82 067 0x1f

십진수, 팔진수(0리딩 표현), 십육진수(0x리딩 표현)를 각각 입력하세요.
82 067 0x1f
82 55 31

```

01 // file: radixscan.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int a, b, c;
09     printf("십진수, 팔진수, 십육진수를 각각 입력하세요.\n");
10     scanf("%d %o %#x", &a, &b, &c);
11     printf("%d %#o %#x\n\n", a, b, c);
12
13     printf("십진수, 팔진수(0리딩 표현), 십육진수(0x리딩 표현)를 각각 입력하세요.\n");
14     scanf("%i %i %i", &a, &b, &c);
15     printf("%d %d %d\n", a, b, c);
16
17     return 0;
18 }
```

설명

- 10 형식지정자 %d는 십진수, %o는 팔진수, %x는 십육진수 정수를 입력
 14 형식지정자 %i인 경우, 입력값이 03과 같이 0이 리딩하는 수는 팔진수로 인식하며, 0x1f과 같이 0x로 리딩하는 수는 십육진수로 인식

실행결과

십진수, 팔진수, 십육진수를 각각 입력하세요.

82 67 1F ← 067 0x1f로 입력도 가능

82 067 0x1f

십진수, 팔진수(0리딩 표현), 십육진수(0x리딩 표현)를 각각 입력하세요.

82 067 0x1f

82 55 31

표 4-8 함수 scanf()의 형식 지정자

형식 지정자	콘솔 입력값의 형태	입력 변수 인자 유형
%d	십진수로 인식	정수형 int 변수에 입력값 저장
%i	십진수로 인식하며, 단 입력값에 0이 앞에 붙으면 8진수로 0x가 붙으면 16진수로 인식하여 저장	정수형 int 변수에 입력값 저장
%u	unsigned int로 인식	정수형 unsigned int 변수에 입력값 저장
%o	8진수로 인식	정수형 int 변수에 입력값 저장
%x , %X	16진수로 인식	정수형 int 변수에 입력값 저장
%f	부동소수로 인식	부동소수형 float 변수에 입력값 저장
%lf	부동소수로 인식	부동소수형 double 변수에 입력값 저장
%e , %E	지수 형태의 부동소수로 인식	부동소수형 float 변수에 입력값 저장
%c	문자로 인식	문자형 char 변수에 입력값 저장
%s	일련의 문자인 문자열(string)로 인식	문자열을 저장할 배열에 입력값 저장
%p	주소(address) 값으로 인식	정수형 unsigned int 변수에 입력값 저장

함수 getchar()와 putchar()

예제 putchar.c

문자의 입출력 함수

- 함수 getchar()는 'get character'의 의미로 문자 하나를 입력하는 매크로 함수
 - putchar()는 'put character'로 반대로 출력하기 위한 매크로 함수
 - 헤더파일 stdio.h 가 필요
- 함수 getchar()는 인자 없이 함수를 호출
- 입력된 문자값을 자료형 char나 정수형으로 선언된 변수에 저장
 - char a = getchar();
- 함수호출 putchar('a')
- 인자인 'a'를 출력하는 함수로 사용



그림 4-20 함수 getchar()와 putchar()

실습예제 4-13 putchar.c

```
01 // file: putchar.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     char a = '\0';
08
09
10     a = getchar(); ← 함수 getchar()는 인자가 필요 없으며 반환값으로 입력값을 저장한다.
11     putchar(a); putchar('\n'); ↑ 문자의 출력은 함수 putchar()의 인자에 출력하려는 문자를 기술하여 출력한다.
12
13     return 0;
14 }
```

설명

10 함수 getchar()는 반환값으로 입력 문자를 전달하므로 반환값을 대입할 대입문이 필요함
11 함수 putchar()는 putchar('문자')와 같이 출력할 문자를 인자로 전달하여 출력

실행결과

문자 하나 입력:

#

실습예제 4-13

putchar.c

```
01 // file: putchar.c  
02  
03 #include <stdio.h>  
04  
05 int main(void)  
06 {  
07     char a = '\0';  
08 }
```

```
10     a = getchar(); ← 함수 getchar()는 인자가 필요 없으며 반환값으로 입력값을 저장한다.  
11     putchar(a); putchar('\n');  
12  
13     return 0;         문자의 출력은 함수 putchar()의 인자에  
14 }                 출력하려는 문자를 기술하여 출력한다.
```

설명

- 10 함수 getchar()는 반환값으로 입력 문자를 전달하므로 반환값을 대입할 대입문이 필요함
- 11 함수 putchar()는 putchar('문자')와 같이 출력할 문자를 인자로 전달하여 출력

실행결과

문자 하나 입력:

```
#  
#
```

LAB 십진수, 팔진수, 십육진수인 세 정수를 입력 받아 적절히 출력

- **십진수, 팔진수, 십육진수인 세 정수를 입력 받아 다음 조건을 만족하도록 적절히 출력되는 프로그램**
 - 세 정수를 '십진수 – 팔진수 – 십육진수'의 형식으로 입력
 - 입력과 출력
 - 세 개의 정수를 각각 다음과 같이 입력하세요. 십진수 - 팔진수 - 십육진수
 - 100 - 65 - f3
 - 입력한 수는 다음과 같습니다.
 - 100 - 65 - f3
 - 100 - 53 - 243

Lab 4-3 basictio.c

```
01 // file: bsicio.h
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int dec = 30, oct = 012, hex = 0x5E;
09     printf("세 개의 정수를 각각 다음과 같이 입력하세요. ");
10     printf("십진수 - 팔진수 - 십육진수\n");
11
12     scanf("%d - %o - %x", _____);
13     printf("\n입력한 수는 다음과 같습니다.\n");
14     printf("_____ \n", dec, oct, hex);
15     printf("_____ \n", dec, oct, hex);
16
17     return 0;
18 }
```

정답

```
12 scanf("%d - %o - %x", &dec, &oct, &hex);
14 printf("%d - %o - %x\n", dec, oct, hex);
15 printf("%d - %d - %d\n", dec, oct, hex);
```

Lab 4-3

basictio.c

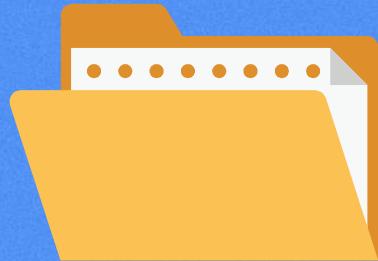
```
01 // file: bsicio.h
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int dec = 30, oct = 012, hex = 0x5E;
09     printf("세 개의 정수를 각각 다음과 같이 입력하세요. ");
10     printf("십진수 - 팔진수 - 십육진수\n");
11
12     scanf("%d - %o - %x", _____);
13     printf("\n입력한 수는 다음과 같습니다.\n");
14     printf("_____ \n", dec, oct, hex);
15     printf("_____ \n", dec, oct, hex);
16
17     return 0;
18 }
```

정답

```
12 scanf("%d - %o - %x", &dec, &oct, &hex);
14 printf("%d - %o - %x\n", dec, oct, hex);
15 printf("%d - %d - %d\n", dec, oct, hex);
```



제05장 연산자



01. 연산식과 다양한 연산자



연산식과 연산자 분류

• 연산자와 피연산자, 연산식과 연산값

- 연산식 = 수식 (expression)
 - $y = x + 1;$
 - 수식은 반드시 하나의 결과값을 가짐;

x = 1;
y = x + 1;

- 연산자(operator)와 피연산자(operand)

• 수식 (연산식) 3 + 4

- '+'는 연산자이고, 3과 4는 피연산자
- 항상 하나의 결과값: 7

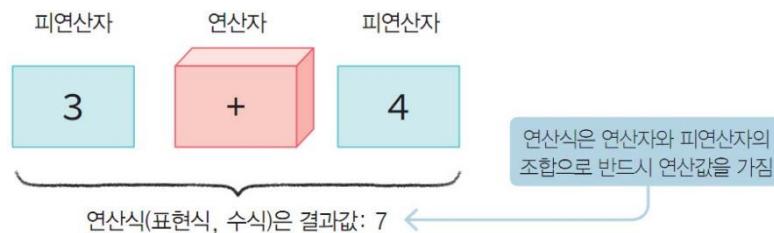


그림 5-1 연산식과 연산식 값(연산식 평가 또는 결과값)

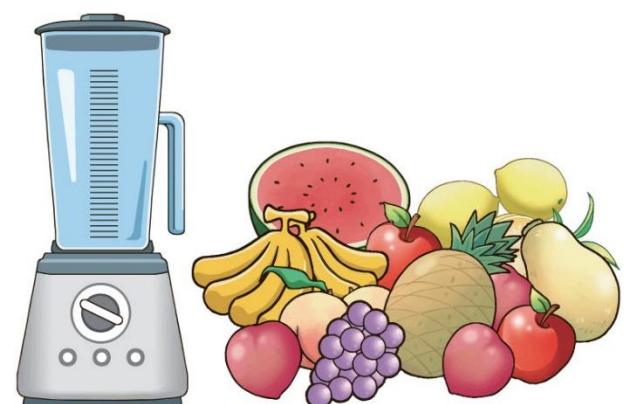


그림 5-2 믹서기(연산자)와 야채(피연산자)

다양한 연산자

- 단(일)항(unary), 이항(binary), 삼항(ternary) 연산자

- 부호를 표시하는 +, -는 단항연산자
- 덧셈, 뺄셈의 +, -, *, / 등의 연산은 이항연산자
- 삼항연산자는 조건연산자 '? :'가 유일

- 단항연산자

- 연산자의 위치에 따라 전위와 후위로 나뉨
 - `++a`처럼 연산자가 앞에 있으면 전위(prefix) 연산자
 - `a++`와 같이 연산자가 뒤에 있으면 후위(postfix) 연산자

단항연산자

연산자 피연산자 (전위: prefix)

`++a`: 전위 증가연산자
`sizeof (int)`: sizeof 연산자
`(int) 3.46`: 자료형 변환연산자
`-3`: 부호연산자

피연산자 연산자 (후위: postfix)

`b--`: 후위 감소연산자
`a++`: 후위 증가연산자

이항연산자

피연산자 1 연산자 피연산자 2

`a % 3`
`3 && 4`
`5 >= 7`



삼항연산자

피연산자 1 ? 피연산자 2 : 피연산자 3

`3 ? 4 : 5`
`(5 >= 7) ? (3+5) : (10/2)`

그림 5-3 단항, 이항, 삼항연산자

산술연산자와 부호연산자

• 산술연산자는 +, -, *, /, %

- % 나머지(remainder, modulus) 연산자

- 피연산자로 정수만 가능
 - 연산식 $10 / 4$ 는 연산값이 2

- 실수끼리의 연산 $10.0 / 4.0$

- 결과는 정상적으로 2.5

- 나머지 연산식 $a \% b$

- 결과는 a 를 b 로 나눈 나머지 값
 - %의 피연산자는 반드시 정수
 - 실수이면 오류가 발생

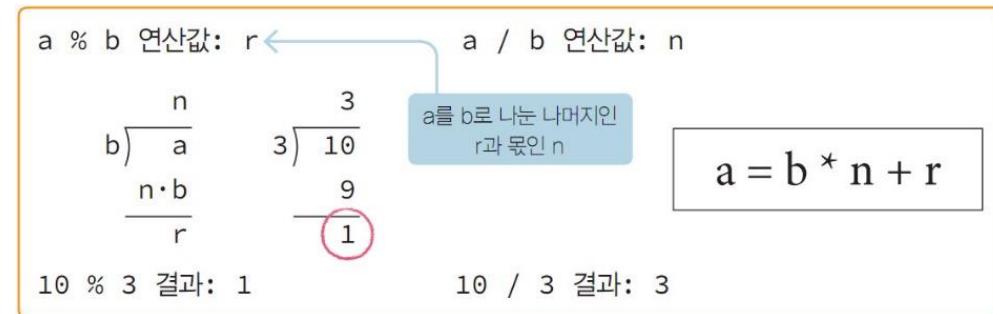


그림 5-4 나누기 연산과 나머지 연산의 이해



그림 5-5 산술연산에서 결합성

부호연산자와 다양한 연산식

- **부호 연산자: 단항 연산자**
 - 연산식 $+3, -4.5, -a$
 - 수, 변수의 부호로 표기하는 연산자
- **다양한 연산식**

표 5-1 다양한 산술 연산식

연산식	설명	연산값	연산식	설명	연산값
$-a$	부호연산자	10	$-b + 2.5$	부호연산자	0.0
$2 - a$	빼기연산자	-3	$a / b + b * 2$	$(a / b) + (b * 2)$	5.0
$2 * a + 3$	$(2 * a) + 3$	13	$a * 2 / b - a$	$((a * 2) / b) - a$	-1.0
$10 - a / 2$	$10 - (a / 2)$	8	$a + b * 2 / a$	$a + ((b * 2) / a)$	6.0
$10 \% a + 10 / a$	$(10 \% a) + (10 / a)$	2	$a \% b$	실수는 %에 오류	오류

대입연산자 =

예제 assignment.c

- 대입연산자(assignment operator) =
 - 연산자 오른쪽의 연산값을 변수에 저장하는 연산자
- $a = b = c = 5$ 와 같은 중첩된 대입문
 - 변수 a, b, c 모두 5가 저장
 - 연산값은 마지막으로 a에 저장된 값인 5

실습예제 5-2 assignment.c

```
01 // file: assignment.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a, b, c;
08     a = b = c = 5;    //(a = (b = (c = 5)))
09
10    printf("a = a + 2 ==> %d\n", a = a + 2);
11    printf("a ==> %d\n", a);
12    printf("a = b + c ==> %d\n", a = b + c);
13    printf("a ==> %d\n", a);
14
15    return 0;
16 }
```

설명

10 대입연산자의 결합성은 오른쪽에서 왼쪽으로 수행
11 대입연산식의 연산값은 대입된 저장값

실행결과

```
a = a + 2 ==> 7
a ==> 7
a = b + c ==> 10
a ==> 10
```

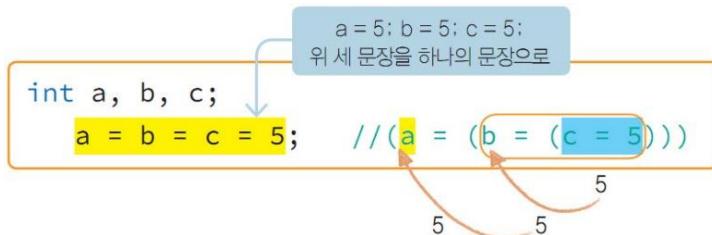


그림 5-8 중첩된 대입문

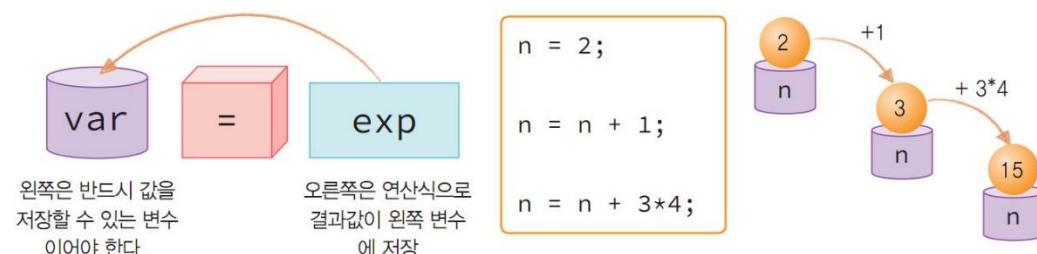


그림 5-7 대입연산자 수행 방법

실습예제 5-2

assignment.c

```
01 // file: assignment.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a, b, c;
08     a = b = c = 5;    //(a = (b = (c = 5)))
09
10     printf("a = a + 2 ==> %d\n", a = a + 2);
11     printf("a ==> %d\n", a);
12     printf("a = b + c ==> %d\n", a = b + c);
13     printf("a ==> %d\n", a);
14
15     return 0;
16 }
```

설명

10 대입연산자의 결합성은 오른쪽에서 왼쪽으로 수행

11 대입연산식의 연산값은 대입된 저장값

실행결과

```
a = a + 2 ==> 7
a ==> 7
a = b + c ==> 10
a ==> 10
```

축약 대입연산자 =

예제 assignment.c

- 대입연산식 $a = a + b$
 - 중복된 a 를 생략하고 간결하게 $a += b$
- $-=$, $*=$, $/=$, $%=$ 을 축약 대입연산자
 - 산술연산자와 대입연산자를 이어 붙인 연산자 $+=$
 - 즉 $a += 2$ 는 $a = a + 2$ 의 대입연산을 의미

연산식 $a + exp$ 의
결과가 변수 a 에 저장
피연산자 exp 는 변수뿐만
아니라 모든 연산식이 가능

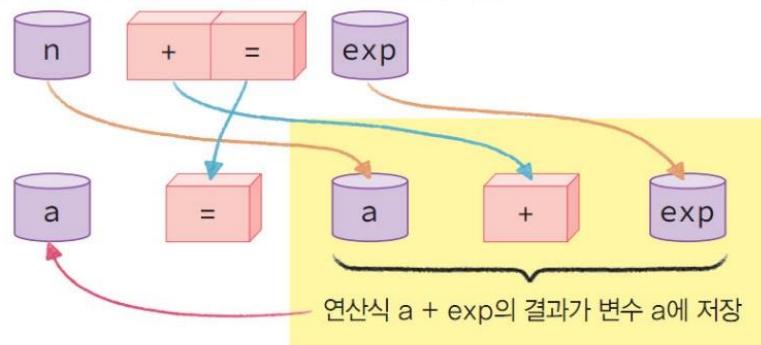


그림 5-9 축약 대입연산자의 연산 방법

실습예제 5-3

compoundassign.c

```
01 // file: compoundassign.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int x = 5, y = 10;
09
10     printf("두 정수를 입력 >> ", &x, &y);
11     scanf("%d%d", &x, &y);
12
13     printf("The addition is: %d\n", x += y);
14     printf("x = %d, y = %d\n", x, y);
15     printf("The subtraction is: %d\n", x -= y);
16     printf("x = %d, y = %d\n", x, y);
17     printf("The multiplication is: %d\n", x *= y);
18     printf("x = %d, y = %d\n", x, y);
19     printf("The division is: %d\n", x /= y);
20     printf("x = %d, y = %d\n", x, y);
21     printf("The remainder is: %d\n", x %= y);
22     printf("x = %d, y = %d\n", x, y);
23     printf("x *= x + y is: %d\n", x *= x + y);
24     printf("x = %d, y = %d\n", x, y);
25
26     return 0;
27 }
```

설명

11 두 정수를 입력하기 위한 scanf()로 변수 앞에 반드시 & 삽입
13 연산식 $x += y$ 결과는 x 에 대입된 값으로, 14행의 x 값과 동일

실행결과

```
두 정수를 입력 >> 10 5
The addition is: 15
x = 15, y = 5
The subtraction is: 10
x = 10, y = 5
The multiplication is: 50
x = 50, y = 5
The division is: 10
x = 10, y = 5
The remainder is: 0
x = 0, y = 5
x *= x + y is: 0
x = 0, y = 5
```

증가 감소 연산자(1)

- **연산자 ++, --**
 - 증가연산자 ++ : 변수값을 각각 1 증가시키고
 - 감소연산자 -- : 1 감소
 - n++와 ++n
 - 모두 $n=n+1$ 의 기능 수행
 - n--와 --n
 - $n=n-1$ 의 기능 수행
- **n++: 후위(postfix)**
 - 1 증가되기 전 값이 연산 결과값
- **++n: 전위(prefix)**
 - 1 증가된 값이 연산 결과값

$n++ \leftarrow$ 수식, 수
식값은? n 의 값
 $\rightarrow n$ 의 값을 증가
전 또는 후?

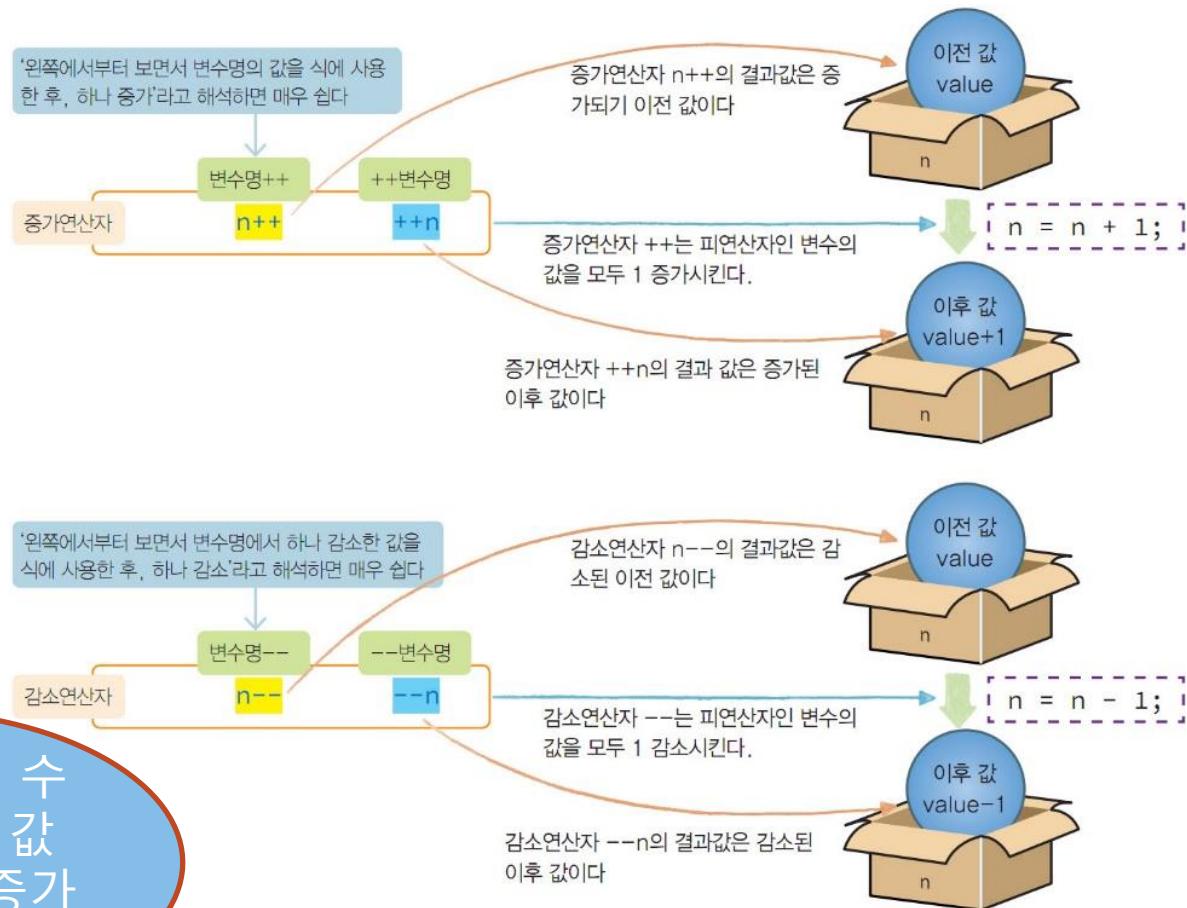


그림 5-11 증가연산자 ++, 감소연산자 --

증가 감소 연산자(2)

• 주의

- 연산자 기호 중간에 공백이 들어갈 수 없으며
- 증감연산자는 변수만을 피연산자로 사용할 수 있으며
- 상수나 일반 수식을 피연산자로 사용할 수 없음

• 사용 예

```
int n = 10;
```

출력

```
printf("%d\n", n++);  
printf("%d\n", n);
```

10

11

```
int n = 10;
```

출력

```
printf("%d\n", ++n);  
printf("%d\n", n);
```

10

11

```
int n = 10;
```

출력

```
printf("%d\n", n--);  
printf("%d\n", n);
```

10

9

```
int n = 10;
```

출력

```
printf("%d\n", --n);  
printf("%d\n", n);
```

9

9

잘못된 사용 예

```
int a = 10;
```

```
++300;
```

//상수에는 증감연산자를 사용할 수 없다.

```
(a+1)--;
```

//일반 수식에는 증감연산자를 사용할 수 없다.

//하나의 연산식에 동일한 변수의 증감연산자는 사용하지 말자.

```
a = ++a * a--;
```

그림 5-12 증감연산자 사용

LAB 표준입력된 두 실수의 산술연산 출력

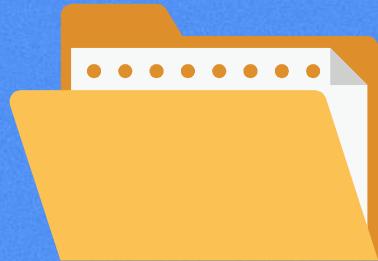
- 다음 정보를 이용하여 두 실수의 합, 차, 곱, 나누기의 과정과 결과를 출력하는 프로그램
 - 자료형 double의 변수 a와 b에 표준입력으로 받아 저장
 - 다음 결과 창과 같은 서식(실수는 모두 폭 8, 정밀도는 2로)으로 출력
- 실행결과
 - 산술연산을 수행할 두 실수를 입력하세요
 - 54.987, 4.87654
 - 54.99 + 4.88 ==> 59.86
 - 00054.99 - 00004.88 ==> 50.11
 - +54.99 * +4.88 ==> 268.15
 - 54.99 / 4.88 ==> 11.28

Lab 5-1 tworealop.c

```
01 // file: tworealop.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     double a = 0, b = 0;
09
10     printf("산술연산을 수행할 두 실수를 입력하세요\n");
11     scanf("_____", &a, &b);
12
13     printf("%8.2f + %8.2f ==> %8.2f\n", a, b, a + b);
14     printf("%8.2f - %8.2f ==> %8.2f\n", a, b, a - b);
15     printf("_____= ==> %8.2f\n", _____);
16     printf("_____= ==> %8.2f\n", _____);
17
18     return 0;
19 }
```

정답

```
11     %lf, %lf
15     %+8.2f * %+8.2f           a, b, a * b
16     %-8.2f / %-8.2f          a, b, a / b
```



02. 관계와 논리, 조건과 비트연산자



두 피연산자의 크기 비교

예제 relation.c

- 두 피연산자의 크기를 비교하기 위한 연산자: 관계연산자
 - 비교 결과가 참이면 1, 거짓이면 0

- 피연산자가 문자인 경우, 문자 코드값에 대한 비교의 결과

- 문자 'a'는 코드값이 97이고 문자 'Z'는 코드값이 90이므로 연산식 ('Z' < 'a')은 1인 참을 의미
- 예: 문자 '0' < '1' < '2' < '3' < ... < '9'인 관계

실습예제 5-5 relation.c

```
01 // file: relation.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     printf("(3 > 4) 결과값: %d\n", (3 > 4));
08     printf("(3 < 4.0) 결과값: %d\n", (3 < 4.0));
09     printf("('a' <= 'b') 결과값: %d\n", ('a' <= 'b'));
10     printf("('Z' <= 'a') 결과값: %d\n", ('Z' <= 'a'));
11     printf("(4.27 >= 4.35) 결과값: %d\n", (4.27 >= 4.35));
12     printf("(4 != 4.0) 결과값: %d\n", (4 != 4.0));
13     printf("(4.0F == 4.0) 결과값: %d\n", (4.0F == 4.0));
14
15     return 0;
16 }
```

설명

10 소문자는 대문자보다 코드값이 크므로, 관계연산도 참을 의미
13 4와 4.0은 같은 것으로 평가

실행결과

(3 > 4) 결과값: 0
(3 < 4.0) 결과값: 1
('a' <= 'b') 결과값: 1
('Z' <= 'a') 결과값: 0
(4.27 >= 4.35) 결과값: 0
(4 != 4.0) 결과값: 0
(4.0F == 4.0) 결과값: 1

표 5-2 관계연산자의 종류와 사용

연산자	연산식	의미	예제	연산(결과)값
>	x > y	x가 y보다 큰가?	3 > 5	0(거짓이면)
>=	x >= y	x가 y보다 큰거나 같은가?	5-4 >= 0	1(참이면)
<	x < y	x가 y보다 작은가?	'a' < 'b'	1(참이면)
<=	x <= y	x가 y보다 작거나 같은가?	3.43 <= 5.862	1(참이면)
!=	x != y	x와 y가 다른가?	5-4 != 3/2	0(거짓이면)
==	x == y	x가 y가 같은가?	'%' == 'A'	0(거짓이면)

```

01 // file: relation.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     printf("(3 > 4) 결과값: %d\n", (3 > 4));
08     printf("(3 < 4.0) 결과값: %d\n", (3 < 4.0));
09     printf("('a' <= 'b') 결과값: %d\n", ('a' <= 'b'));
10     printf("('Z' <= 'a') 결과값: %d\n", ('Z' <= 'a'));
11     printf("(4.27 >= 4.35) 결과값: %d\n", (4.27 >= 4.35));
12     printf("(4 != 4.0) 결과값: %d\n", (4 != 4.0));
13     printf("(4.0F == 4.0) 결과값: %d\n", (4.0F == 4.0));
14
15     return 0;
16 }
```

설명

- 10 소문자는 대문자보다 코드값이 크므로, 관계연산도 참을 의미
 13 4와 4.0은 같은 것으로 평가

실행결과

```
(3 > 4) 결과값: 0
(3 < 4.0) 결과값: 1
('a' <= 'b') 결과값: 1
('Z' <= 'a') 결과값: 0
(4.27 >= 4.35) 결과값: 0
(4 != 4.0) 결과값: 0
(4.0F == 4.0) 결과값: 1
```

논리연산자

예제 logic.c

- 세 가지 논리연산자 $\&\&$, $\|$, $!$
 - 논리연산자 $\&\&$, $\|$, $!$ 은 각각 and, or, not 의미
- C 언어에서 참과 거짓의 논리형은 따로 없음
 - 0, 0.0, $\text{\wedge}0$ 은 거짓을 의미
 - 0이 아닌 모든 정수와 실수 (그리고 널(null) 문자 ' $\text{\wedge}0$ '가 아닌 모든 문자와 문자열)은/는 모두 참을 의미
- 논리연산자 $\&\&$
 - 두 피연산자가 모두 참이면 결과가 1(참)이며
 - 나머지 경우는 모두 0
- 논리연산자 $\|$
 - 두 피연산자 중에서 하나만 참이면 1이고,
 - 모두 0(거짓)이면 0
- 논리연산자 $!$
 - 단항연산자로 피연산자가 0이면 결과는 1이고
 - 참이면 결과는 0

실습예제 5-6 logic.c

```
01 // file: logic.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     char null = '\0', a = 'a';
08     int zero = 0, n = 10;
09     double dzero = 0.0, x = 3.56;
10
11     printf("%d ", !zero);
12     printf("%d ", zero && x);
13     printf("%d\n", dzero || null);
14     printf("%d ", n && x);
15     printf("%d ", a || null);
16     printf("%d\n", "java" && "C Lang");
17
18     return 0;
19 }
```

설명

16 문자열에 문자가 있으면 참을 의미하므로 1 출력

실행결과

```
1 0 0
1 1 1
```

x	y	x && y	x y	!x
0(거짓)	0(거짓)	0	0	1
0(거짓)	0(0이 아닌 값)	0	1	1
0(0이 아닌 값)	0(거짓)	0	1	0
0(0이 아닌 값)	0(0이 아닌 값)	1	1	0

21 && 3	1
!2 && 'a'	0
3>4 && 4>=2	0
1 '\0'	1
2>=1 3 <=0	1
0.0 2-2	0
!0	1

그림 5-13 논리연산자의 연산 결과

실습예제 5-6

logic.c

```
01 // file: logic.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     char null = '\0', a = 'a';
08     int zero = 0, n = 10;
09     double dzero = 0.0, x = 3.56;
10
11     printf("%d ", !zero);
12     printf("%d ", zero && x);
13     printf("%d\n", dzero || null);
14     printf("%d ", n && x);
15     printf("%d ", a || null);
16     printf("%d\n", "java" && "C Lang");
17
18     return 0;
19 }
```

설명

16 문자열에 문자가 있으면 참을 의미하므로 1 출력

실행결과

1 0 0

1 1 1

단축 평가

예제 shorteval.c

• 단축 평가(short circuit evaluation)

- 연산자 `&&`와 `||`는 피연산자 두 개 중에서 왼쪽 피연산자만으로 논리연산 결과가 결정된다면, 오른쪽 피연산자는 평가하지 않음

• 예를 들어 ($x \&& y$) 연산식

- x 의 값이 0(거짓)이라면 y 의 값을 평가하지 않고 연산 ($x \&& y$) 결과는 0

• ($x \parallel y$) 수식

- x 가 0이 아니(참)라면 더 이상 y 의 값을 평가하지 않고 결과 1이라고 평가

실습예제 5-7 shorteval.c

```
01 // file: shorteval.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 10, b = 5, m = 1;
08     int result;
09
10    result = (a < b) && (m++ == 1);
11    printf("m=%d result=%d\n", m, result);
12
13    result = (a > b) || (--m == 0);
14    printf("m=%d result=%d\n", m, result);
15
16    return 0;
17 }
```

설명

10 &&의 왼쪽 ($a < b$)가 0이므로 오른쪽 연산을 하지 않고 0으로 판별되며, m 은 변하지 않음
13 ||의 왼쪽 ($a > b$)가 1이므로 오른쪽 연산을 하지 않고 1로 판별되며, m 은 변하지 않음

실행결과

m=1 result=0
m=1 result=1

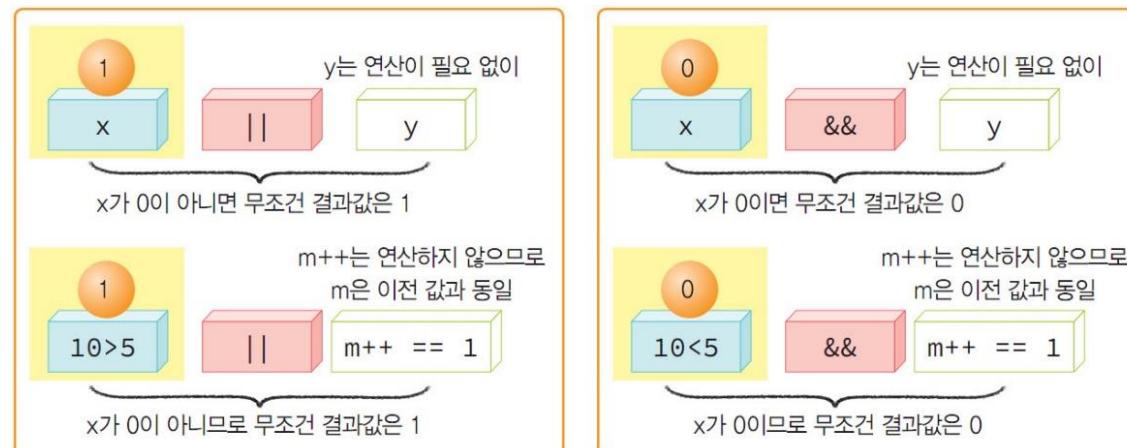


그림 5-14 단축 평가

조건 연산자

예제 condition.c

• 연산자 ?: :

- 조건연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자
- 연산식 ($x ? a : b$)에서 피연산자는 x, a, b 세 개
 - 피연산자인 x 가 참이면(0이 아니면) 결과는 a 이며,
 - x 가 0이면(거짓) 결과는 b

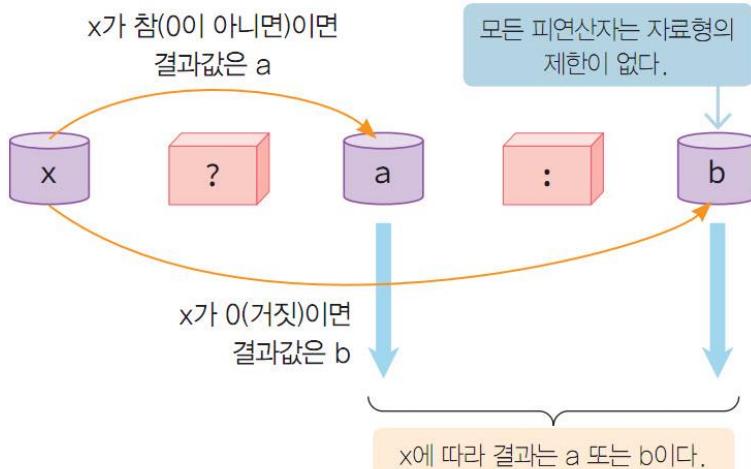


그림 5-15 조건연산자 계산 방법

실습예제 5-8 condition.c

```
01 // file: condition.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int a = 0, b = 0;
09     printf("두 정수를 입력 >> ");
10
11     scanf("%d%d", &a, &b);
12
13     printf("최대값: %d ", (a > b) ? a : b);
14     printf("최소값: %d\n", (a < b) ? a : b);
15     printf("절대값: %d ", (a > 0) ? a : -a);
16     printf("절대값: %d\n", (b > 0) ? b : -b);
17
18     ((a % 2) == 0) ? printf("짝수 ") : printf("홀수 ");
19
20     printf("%s\n", ((b % 2) == 0) ? "짝수" : "홀수");
21 }
```

설명

17 조건 삼항연산자의 두 번째와 세 번째 피연산자는 문장도 가능
18 조건 삼항연산자의 두 번째와 세 번째 피연산자는 문자열을 비롯하여 모든 자료형도 가능

실행결과

두 정수를 입력 >> 8 -9
최대값: 8 최소값: -9
절대값: 8 절대값: 9
짝수 홀수

비트 연산자

- **비트 논리 연산자**
 - 피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자

표 5-4 각 비트 연산 방법

x(비트1)	y(비트2)	$x \& y$	$x y$	$x ^ y$	$\sim x$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- **&, |, ^, ~ 4가지**
 - 연산자 ~
 - ~ 5 와 같이 연산자 \sim 가 피연산자인 5 앞에 위치하는 전위 단항 연산자
 - 나머지는 모두 ($3 | 4$)처럼 피연산자가 두 개인 이항 연산자
 - 피연산자의 자료형은 정수형에 해당하는 char, int, long, long long
 - 각 피연산자를 int 형으로 변환하여 연산하며 결과도 int 형

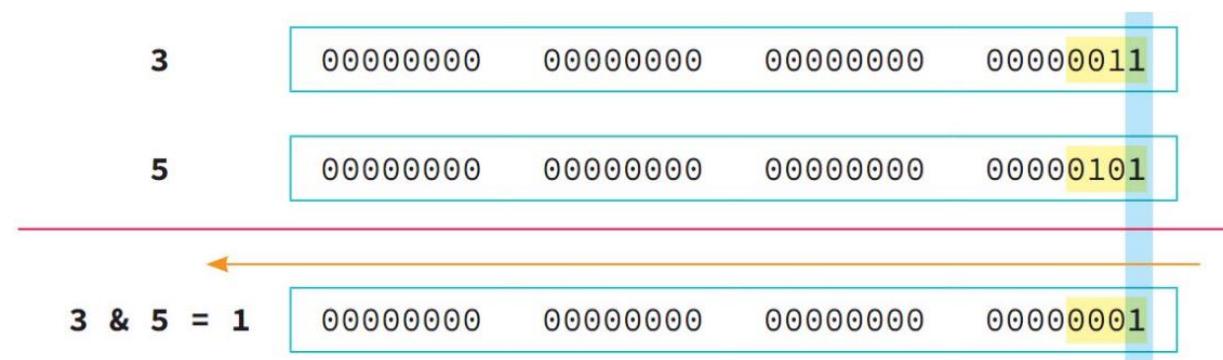


그림 5-17 비트 논리 연산 3 & 5

비트 연산자

예제 bitlogic.c

• 컴퓨터에서 정수의 음수 표현 방법

- 보수를 이용하는 방법을 사용
- 즉 양수 정수 a에서 음수인 -a의 비트 표현은 2의 보수 표현인 ($\sim a + 1$)
- 즉 -1 은 $((\sim 1) + 1)$ 로, 정수 -1 을 비트로 표현하면 32비트가 모두 1인 정수

• 비트 논리 연산식 $x \& -1$

- 정수 x 를 -1 로 논리 and연산을 수행하는 식으로 결과는 x

• 비트 논리 연산식 $x | 0$

- 정수 x 를 0 으로 논리 or연산을 수행하는 식으로 결과는 x

표 5-6 다양한 비트 논리연산식

연산식	설명	연산값
$1 \& 2$	0001 $\&$ 0010	0
$3 4$	0011 $ $ 0100	7
$3 ^ 4$	0011 $^$ 0100	7
~ 2	$-2 == \sim 2 + 1$	-3

실습예제 5-9 bitlogic.java

```
01 // bitlogic.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int x = 127;
08
09     printf("%5d -> %08x\n", x, x);
10    printf("x & 1 -> %08x\n", x & 1);
11    printf("x | 1 -> %08x\n", x | 1);
12    printf("x ^ 1 -> %08x\n", x ^ 1);
13    printf("~(-1) -> %08x\n", ~(-1));
14    printf(" ~1 -> %08x\n", ~1);
15
16    return;
17 }
```

설명

07 정수 127은 이진수로 1111111
09 int은 32비트로 8개의 십육진수로 표현이 가능하고 7f
10 비트 AND 연산은 1이 가장 오른쪽만 1이고, 127도 가장 오른쪽은 1이므로 결과는 1
11 비트 OR 연산의 결과는 그대로 127
12 두 피연산자의 가장 오른쪽 비트도 1과 1로 같으므로 0이 되어 126

실행결과

127 -> 0000007f
x & 1 -> 00000001
x | 1 -> 0000007f
x ^ 1 -> 0000007e
~(-1) -> 00000000
~1 -> fffffffe

연산식	설명	연산값
$1 \& 3$	0001 $\&$ 0011	1
$3 \& 5$	0011 $ $ 0101	7
$3 ^ 5$	0011 $ $ 0101	6
~ 3	$-3 == \sim 3 + 1$	-4

실습예제 5-9

bitlogic.java

```
01 // bitlogic.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int x = 127;
08
09     printf("%5d -> %08x\n", x, x);
10    printf("x & 1 -> %08x\n", x & 1);
11    printf("x | 1 -> %08x\n", x | 1);
12    printf("x ^ 1 -> %08x\n", x ^ 1);
13    printf("~(-1) -> %08x\n", ~(-1));
14    printf(" ~1 -> %08x\n", ~1);
15
16    return;
17 }
```

설명

- 07 정수 127은 이진수로 1111111
- 09 int은 32비트로 8개의 십육진수로 표현이 가능하고 7f
- 10 비트 AND 연산은 10이 가장 오른쪽만 10이고, 127도 가장 오른쪽은 10이므로 결과는 1
- 11 비트 OR 연산의 결과는 그대로 127
- 12 두 피연산자의 가장 오른쪽 비트도 1과 1로 같으므로 0이 되어 126

실행결과

```
127 -> 0000007f
x & 1 -> 00000001
x | 1 -> 0000007f
x ^ 1 -> 0000007e
~(-1) -> 00000000
~1 -> fffffffe
```

비트 이동 연산자

- **비트 이동연산자(bit shift operators) >>, <<**
 - 연산자의 방향인 왼쪽이나 오른쪽으로, 비트 단위로 줄줄이 이동시키는 연산자
 - <<
 - 오른쪽(LSB: Least Significant Bit) 빈 자리가 생겨
 - 모두 0으로 채워짐
 - >>
 - 왼쪽 빈 자리 MSB는
 - 원래의 부호비트에 따라 0또는 1이 채워짐
 - 실제 연산에서는 int 형의 크기인 32비트의 비트에서 연산을 수행

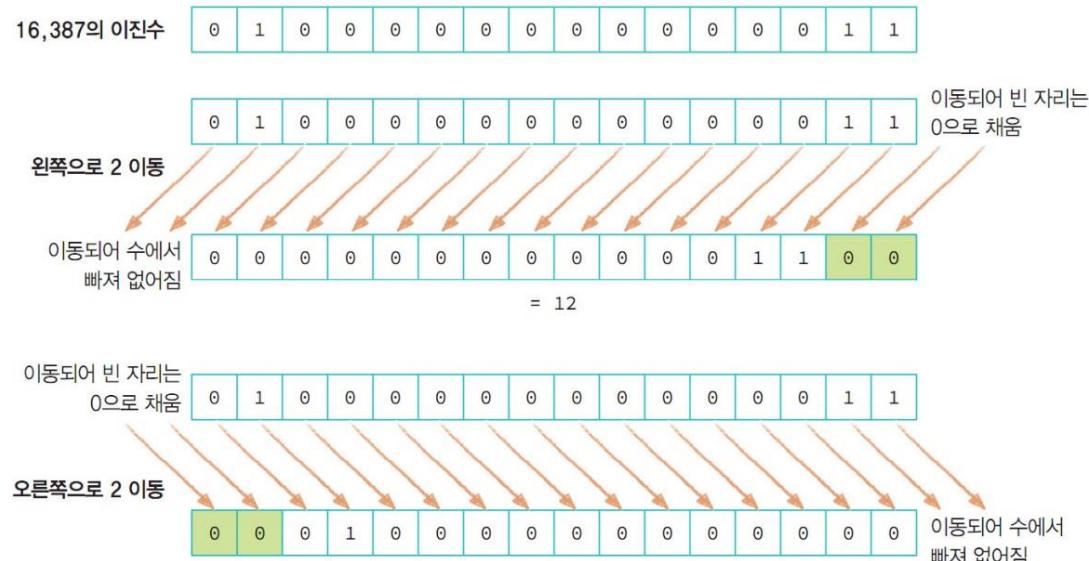


그림 5-20 비트 논리 연산 $16387 \ll 2$ 와 $16387 \gg 2$

표 5-7 비트 이동 연산자

연산자	이름	사용	연산 방법	새로 채워지는 비트
>>	right shift	op1 >> op2	op1을 오른쪽으로 op2 비트만큼 이동	가장 왼쪽 비트인 부호 비트는 원래의 부호 비트로 채움
<<	left shift	op1 << op2	op1을 왼쪽으로 op2 비트만큼 이동	가장 오른쪽 비트를 모두 0으로 채움

비트 이동 연산자

예제 shift.c

- 왼쪽 이동연산자에 의해 최상위 부호 비트가 0에서 1로, 1에서 0으로 바뀔 수 있으므로 왼쪽 이동연산자에 의해 항상 2배씩 커지지는 않음
- 음수 홀수에 대한 오른쪽 이동연산자 $>>$ 도 -1 을 한 짹수를 2로 나눈 결과

실습예제 5-10 shift.c

```
01 // shift.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int x = 16391;
08
09     printf("%6d --> %08x\n", x, x);
10    printf("x >> 1 --> %d, %08x\n", x >> 1, x >> 1);
11    printf("x >> 2 --> %d, %08x\n", x >> 2, x >> 2);
12    printf("x >> 2 --> %d, %08x\n", x >> 2, x >> 2); 9
13    printf("x << 2 --> %d, %08x\n", x << 2, x << 2);
14    printf("x << 2 --> %d, %08x\n", x << 3, x << 3); 9
15
16    return;
17 }
```

설명

09 정수 16391은 십육진수로 4007
10~12 정수 $>>$ n은 정수를 n번 2로 나눈 효과
13~14 정수 $<<$ n은 정수를 n번 2로 곱한 효과

실행결과

16391 --> 00004007
x >> 1 --> 8195, 00002003
x >> 2 --> 4097, 00001001
x >> 2 --> 2048, 00000800 *9*
x << 2 --> 65564, 0001001c *9*
x << 2 --> 131128, 00020038

표 5-8 다양한 비트 이동연산식

연산식	설명	연산값
$15 << 1$	$0000\ 1111 << 1$ $0001\ 1110$	30
$0x30000000 << 2$	최상위 4비트: $0011\ ... << 2$ $1100\ ...$	-1073741824 0XC0000000
$-30 >> 1$	짝수이면 2로 나누기	-15

연산식	설명	연산값
$15 << 2$	$0000\ 1111 << 2$ $0011\ 1100$	60
$-30 << 2$	음수로 4배 커짐	-120
$-30 >> 2$	한 비트 이동마다. 음수 홀수는 $-(a+1)$ 한 수를 2로 나누기	-8

실습예제 5-10

shift.c

```
01 // shift.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int x = 16391;
08
09     printf("%6d --> %08x\n", x, x);
10    printf("x >> 1 --> %d, %08x\n", x >> 1, x >> 1);
11    printf("x >> 2 --> %d, %08x\n", x >> 2, x >> 2);
12    printf("x >> 2 --> %d, %08x\n", x >> 3, x >> 3); // 오류
13    printf("x << 2 --> %d, %08x\n", x << 2, x << 2);
14    printf("x << 2 --> %d, %08x\n", x << 3, x << 3);
15
16    return;
17 }
```

설명

- 09 정수 16391은 십육진수로 4007
- 10~12 정수 >> n은 정수를 n번 2로 나눈 효과
- 13~14 정수 << n은 정수를 n번 2로 곱한 효과

실행결과

```
16391 --> 00004007
x >> 1 --> 8195, 00002003
x >> 2 --> 4097, 00001001
x >> 2 --> 2048, 00000800 // 오류
x << 2 --> 65564, 0001001c
x << 2 --> 131128, 00020038
```

LAB 표준입력으로 받은 두 정수의 비트 연산 수행 출력

- 표준입력으로 받은 두 정수의 6개 비트 연산을 수행하여 결과를 출력하는 프로그램
 - 비트 연산은 &, |, ^, ~, >>, << 연산을 수행
 - 단항 연산은 첫 번째 변수에 대한 연산 수행
- 결과
 - 비트 연산이 가능한 두 정수를 입력하세요
 - 40 3
 - 40 & 3 ==> 0
 - 40 | 3 ==> 43
 - 40 ^ 3 ==> 43
 - ~ 40 ==> -41
 - 40 >> 3 ==> 5
 - 40 << 3 ==> 320

Lab 5-2 twobitop.c

```
01 // twobitop.c:  
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의  
03  
04 #include <stdio.h>  
05  
06 int main(void)  
07 {  
08     int a = 0, b = 0;  
09  
10     printf("비트 연산이 가능한 두 정수를 입력하세요\n");  
11     scanf("%d %d", _____);  
12  
13     printf("%3d & %3d ==> %3d\n", a, b, a & b);  
14     printf("%3d | %3d ==> %3d\n", a, b, a | b);  
15     printf("%3d ^ %3d ==> %3d\n", _____);  
16     printf(~%3d ==> %3d\n", a, ~a);  
17     printf("%3d >> %3d ==> %3d\n", a, b, a >> b);  
18     printf("%3d << %3d ==> %3d\n", a, b, _____);  
19  
20     return 0;  
21 }
```

정답

```
11 scanf("%d %d", &a, &b);  
15 printf("%3d ^ %3d ==> %3d\n", a, b, a ^ b);  
18 printf("%3d << %3d ==> %3d\n", a, b, a << b);
```

Lab 5-2

twobitop.c

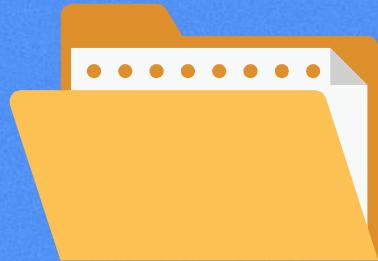
```

01 // twobitop.c:
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     int a = 0, b = 0;
09
10     printf("비트 연산이 가능한 두 정수를 입력하세요\n");
11     scanf("%d %d", _____);
12
13     printf("%3d & %3d ==> %3d\n", a, b, a & b);
14     printf("%3d | %3d ==> %3d\n", a, b, a | b);
15     printf("%3d ^ %3d ==> %3d\n", _____);
16     printf(" ~%3d ==> %3d\n", a, ~a);
17     printf("%3d >> %3d ==> %3d\n", a, b, a >> b);
18     printf("%3d << %3d ==> %3d\n", a, b, _____);
19
20     return 0;
21 }
```

정답

```

11 scanf("%d %d", &a, &b);
15 printf("%3d ^ %3d ==> %3d\n", a, b, a ^ b);
18 printf("%3d << %3d ==> %3d\n", a, b, a << b);
```



03. 형변환 연산자와 연산자 우선순위



내림변환과 올림변환

- 자료형 변환은 크기 자료형의 범주 변화에 따른 구분
- 올림변환

- 작은 범주의 자료형(int)에서 보다 큰 범주인 형(double)으로의 형변환
- 올림변환은 정보의 손실이 없으므로 컴파일러에 의해 자동으로 수행 가능
 - 컴파일러가 자동으로 수행하는 형변환: 묵시적 형변환(implicit type conversion)

내림변환

- 큰 범주의 자료형(double)에서 보다 작은 범주인 형(int)으로의 형변환
- 대입연산 `int a = 3.4;`에서 내림변환이 필요
 - 컴파일러가 스스로 시행하는 묵시적 내림변환의 경우 정보의 손실이 일어날 수 있으므로 경고를 발생
 - 프로그래머의 명시적 형변환 필요

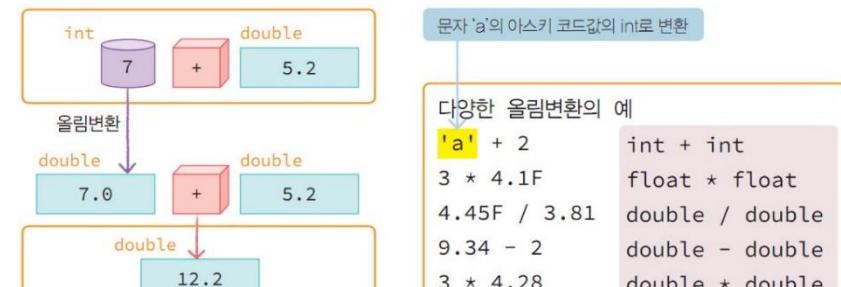


그림 5-21 피연산자의 자동 올림변환

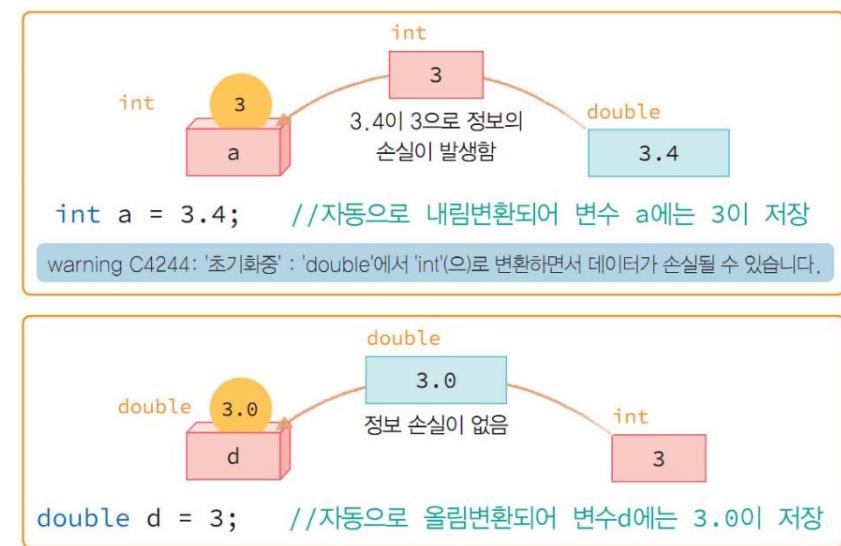


그림 5-22 대입연산에서의 내림변환과 올림변환

형변환 연산자

예제 shift.c

- 명시적 형변환(explicit type conversion)

- 형변환 연산자 '(type) 피연산자'는 뒤에 나오는 피연산자의 값을 괄호에서 지정한 자료형으로 변환하는 연산자

올림변환

- 상수나 변수의 정수값을 실수로 변환하려면 올림변환을 사용
- 연산식 (double) 7의 결과는 7.0

내림변환

- 실수의 소수부분을 없애고 정수로 사용하려면 내림변환을 사용
- (int) 3.8의 결과는 3
- 단항연산자인 형변환 연산자는 모든 이항연산자보다 먼저 계산

실습예제 5-11

typecast.c

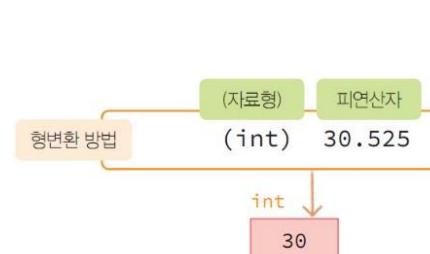
```
01 // file: typecast.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 3.4;           //자동으로 내림변환되어 변수 a에는 3이 저장
08     double d = 3;          //자동으로 올림변환되어 변수 d에는 3.00이 저장
09
10    printf("%5d %10f ", a, d);
11    printf("%10f\n", 3 + 4.5);
12
13    printf("%5d ", 10 / 4);
14    printf("%10f ", (double)10 / 4);
15    printf("%10f ", 10 / (double)4);
16    printf("%10f\n", (double)(10 / 4));
17
18    printf("%5d ", (int)(3.4 + 7.8));
19    printf("%10d ", (int) 3.4 + (int) 7.8);
20    printf("%10f ", (int) 3.4 + 7.8);
21    printf("%10f\n", 3.4 + (int) 7.8);
22
23    return 0;
24 }
```

설명

- (정수/정수)의 결과는 나머지 버린 결과
- 자료형 변환연산자 (double)은 다른 이항 연산자보다 먼저 계산
- 실수와 정수가 함께 하는 연산은 정수를 실수로 자동 변환하여 실수 연산으로

실행결과

```
03 3.000000 7.500000
2 2.500000 2.500000 2.000000
11 10 10.800000 10.400000
```



다양한 형변환연산 예

(int) 'A'	65
(int) 3.14	3
(double) 9	9.0
(double) 3.4F	3.4
(double) 7/2	3.5

연산자 sizeof와 콤마연산자

예제 comma.c

연산자 sizeof

- 연산값 또는 자료형의 저장장소의 크기를 구하는 연산자
 - 결과값은 바이트 단위의 정수
 - 피연산자가 int와 같은 자료형인 경우 괄호를 사용, 피연산자가 상수나 변수 또는 연산식이면 괄호는 생략 가능

콤마연산자 ,

- 콤마연산자 ,는 왼쪽과 오른쪽 연산식을 각각 순차적으로 계산
 - 결과값은 가장 오른쪽에서 수행한 연산의 결과
 - $3+4, 2*5$ 의 결과값은 10
 - 연속으로 나열된 식에서는 마지막에 수행된 가장 오른쪽 연산식의 결과가 전체 식의 결과값

실습예제 5-12 comma.c

```
01 // file: comma.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 100, b = 50, c;
08
09     printf("%d ", sizeof(short));
10    printf("%d ", sizeof a);
11    printf("%d ", sizeof 3.5F);
12    printf("%d\n", sizeof 3.14);
13
14    c = ++a, b++;
15    printf("%d %d %d\n", a, b, c);
16    c = (3 + a, b * 2);
17    printf("%d %d %d\n", a, b, c);
18
19    return 0;
20 }
```

설명

09 연산자 sizeof (자료형)에서 괄호는 필수
14 (c = ++a), (b++); 이므로 변수 a, b, c에는 각각 101, 51, 101
16 변수 C에 저장되는 값은 b * 2 이므로 102

실행결과

02 4 4 8
101 51 101
101 51 102



sizeof (int)	결과는 4
sizeof (3.14)	결과는 8
sizeof a	결과는 2 (short a;)

그림 5-25 sizeof 연산자의 연산방법



3 + 4, 5 - 10	결과는 -5
3 + 4, 5 - 10, 2 * 3	결과는 6

그림 5-26 콤마연산자의 연산방법

실습예제 5-12

comma.c

```
01 // file: comma.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 100, b = 50, c;
08
09     printf("%d ", sizeof(short));
10    printf("%d ", sizeof a);
11    printf("%d ", sizeof 3.5F);
12    printf("%d\n", sizeof 3.14);
13
14    c = ++a, b++;
15    printf("%d %d %d\n", a, b, c);
16    c = (3 + a, b * 2);
17    printf("%d %d %d\n", a, b, c);
18
19    return 0;
20 }
```

설명

- 09 연산자 `sizeof` (자료형)에서 괄호는 필수
- 14 (`c = ++a`), (`b++`); 이므로 변수 `a`, `b`, `c`에는 각각 101, 51, 101
- 16 변수 `C`에 저장되는 값은 `b * 2` 이므로 102

실행결과

```
02 4 4 8
101 51 101
101 51 102
```

복잡한 표현식의 계산

- 연산자 우선순위와 결합성
- 규칙 3개를 적용
 - 첫 번째 규칙은 괄호가 있으면 먼저 계산
 - 두 번째 규칙으로 연산의 우선순위(priority)
 - 즉 '곱하기와 나누기는 더하기와 빼기보다 먼저 계산한다.'라는 규칙
 - 세 번째 규칙은 동일한 우선순위인 경우
 - 연산을 결합하는 방법인 결합성(또는 결합규칙)
 - 연산자 결합성: '괄호가 없고 동일한 우선 순위라면, 덧셈과 뺄셈, 곱셈과 나눗셈과 같은 일반적인 연속된 연산은 왼쪽부터 오른쪽으로 차례로 계산'

$$3 + 8 / 2 * 4$$

다음 수식 연산 절차와 방법

1. $3 + \textcircled{1}$.
2. $\textcircled{1} = \textcircled{2} * 4$, $\textcircled{2} = 8 / 2 = 4$
3. $\textcircled{1} = 4 * 4 = 16$
4. $3 + \textcircled{1} = 3 + 16 \Rightarrow 19$

다음 수식 연산 절차와 방법

1. $8 / 2 * 4$ 를 먼저 계산해야 하므로 $3 + \textcircled{1}$.
2. $\textcircled{1}$ 은 결합성에 따라 $8/2$ 를 먼저 계산하여 4
3. 다시 결합성에 따라 위 2의 결과인 4와 다음 4를 곱하면 결과는 16
4. 그러므로 최종 $3 + 16$ 으로 결과는 19

$$3 + (8 / 2) * 4$$

그림 5-27 다양한 연산자가 있는 연산 방식

연산자의 우선순위(priority)

우선순위가 1위

- 함수호출과 괄호로 사용되는 ()
- 후위 증감연산자 a++와 a-- 등의 단항연산자
- 여러 개 있으면 결합성에 따라 왼쪽에서 오른쪽 순으로 계산

우선순위가 2위

- 전위 증감연산자 ++a와 -a
- 주소연산자 &
- 오른쪽에서 왼쪽으로 차례로 계산
- 모든 단항연산자는 우선순위가 1, 2위이며

우선순위가 16위

- 콤마연산자

표 5-9 C 언어의 연산자 우선순위

우선 순위	연산자	설명	분류	결합성(계산방향)
1	() [] . . -> a++ a--	함수 호출 및 우선 지정 인덱스 필드(유니온) 멤버 지정 필드(유니온) 포인터 멤버 지정 후위 증가, 후위 감소	단항	-> (좌에서 우로)
	++a --a ! ~ sizeof - + & *	전위 증가, 전위 감소 논리 NOT, 비트 NOT(보수) 변수, 자료형, 상수의 바이트 단위 크기 음수 부호, 양수 부호 주소 간접, 역참조		<- (우에서 좌로)
3	(형변환)	형변환		
4	* / %	곱하기 나누기 나머지	산술	-> (좌에서 우로)
5	+ -	더하기 빼기		-> (좌에서 우로)
6	<< >>	비트 이동	이동	-> (좌에서 우로)
7	< > <= >=	대소 비교		-> (좌에서 우로)
8	== !=	동등 비교	관계	-> (좌에서 우로)
9	&	비트 AND 또는 논리 AND		-> (좌에서 우로)
10	^	비트 XOR 또는 논리 XOR	비트	-> (좌에서 우로)
11		비트 OR 또는 논리 OR		-> (좌에서 우로)
12	&&	논리 AND(단락 계산)	논리	-> (좌에서 우로)
13		논리 OR(단락 계산)		-> (좌에서 우로)
14	? :	조건	조건	<- (우에서 좌로)
15	= += -= *= /= %= <= >= &= = ^=	대입	대입	<- (우에서 좌로)
16	,	콤마	콤마	-> (좌에서 우로)

LAB 섭씨(celsius) 온도를 화씨(fahrenheit) 온도로 출력

- 표준입력으로 받은 섭씨(celsius) 온도를 화씨(fahrenheit) 온도로 출력하는 프로그램
 - 다음과 같은 입력과 출력
 - 섭씨(C) 온도를 화씨 온도(F)로 변환하는 식: $F = 9/5*C + 32$
- 결과
 - 변환할 섭씨온도를 입력하세요. >> 34.765879
 - 입력된 34.77도는 화씨온도로 94.58도 입니다.

Lab 5-3

celtofar.c

```
01 // celtofar.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03
04 #include <stdio.h>
05
06 int main(void)
07 {
08     double fahrenheit, celsius;
09     printf("변환할 섭씨온도를 입력하세요. >> ");
10     scanf("%lf", &celsius);
11
12     _____;
13     printf("\n입력된 %.2f도는 화씨온도로 %.2f도 입니다.\n", _____);
14
15     return 0;
16 }
```

정답

```
12     fahrenheit = (9.0 / 5.0) * celsius + 32.0;
13     printf("\n입력된 %.2f도는 화씨온도로 %.2f도 입니다.\n", celsius, fahrenheit)
```