



제05장 연산자

단원 목표

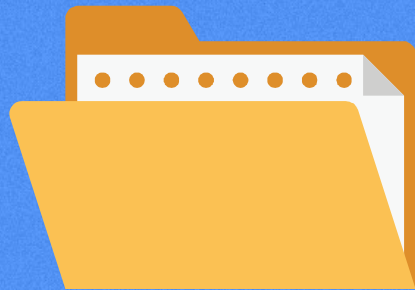
학습목표

- ▶ 연산자의 기본 개념인 다음 용어를 이해하고 설명할 수 있다.
 - 연산자, 피연산자, 연산식
 - 단항연산자, 이항연산자, 삼항연산자
- ▶ 다음 연산자의 사용 방법과 연산식의 결과를 알 수 있다.
 - 산술연산자 $+$, $-$, $*$, $/$, $\%$
 - 대입연산자 $=$, $+=$, $-=$, $*=$, $/=$, $\% =$
 - 증감연산자 $++$, $--$, 조건연산자 $?:$
 - 관계연산자 $>$, $<$, $>=$, $<=$, $!=$, 논리연산자 $\&\&$, $\|\$, $!$
 - 형변환연산자 (type cast)
 - sizeof 연산자, 콤마연산자
- ▶ 연산자 우선순위에 대하여 이해하고 설명할 수 있다.
 - 괄호, 단항, 이항, 삼항연산자의 순위
 - 산술연산자의 순위
 - 관계와 논리연산자 순위
 - 조건, 대입, 콤마연산자 순위

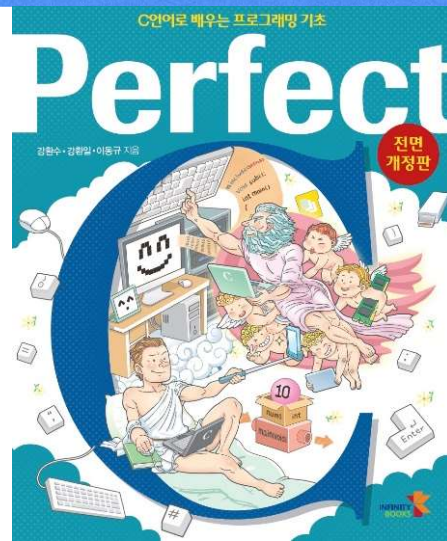
학습목차

- 5.1 연산식과 다양한 연산자
- 5.2 관계와 논리, 조건과 비트연산자
- 5.3 형변환 연산자와 연산자 우선순위





01. 연산식과 다양한 연산자



연산식과 연산자 분류

- 연산자와 피연산자, 연산식과 연산값

- 연산식

- 일상에서 사용하는 $(3 + 4 * 5)$ 와 같은 간단한 수식, 반드시 하나의 결과값인 연산값
 - 변수와 다양한 리터럴 상수, 함수의 호출 등으로 구성되는 식을 연산식

- 연산자(operator)와 피연산자(operand)

- 산술연산자 $+$, $-$, $*$ 기호와 같이 이미 정의된 연산을 수행하는 문자 또는 문자조합 기호
 - 연산(operation)에 참여하는 변수나 상수

- 믹서기는 연산자로, 믹서기 속에 들어가는 재료는 피연산자

- '오렌지 + 딸기'라는 수식

- 오렌지와 딸기는 피연산자가 되고 '+'는 믹서기가 된다고 이해

- 연산식 $3 + 4$

- '+'는 연산자이고, 3과 4는 피연산자

- 항상 하나의 결과값: 7

- 연산식의 결과값을 간단히 '연산값'

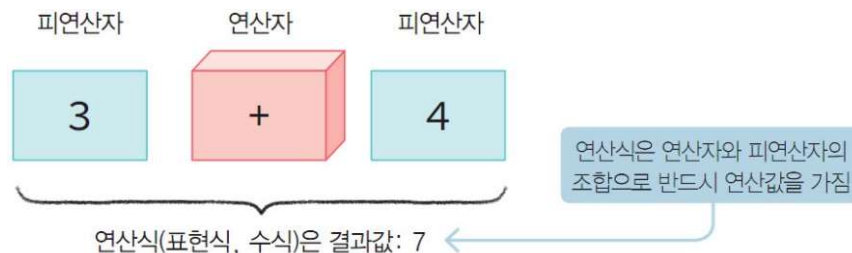


그림 5-1 연산식과 연산식 값(연산식 평가 또는 결과값)

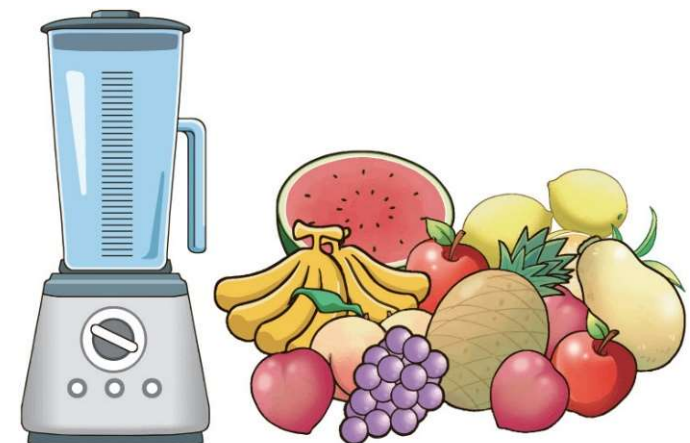


그림 5-2 믹서기(연산자)와 야채(피연산자)

다양한 연산자

- 단(일)항(unary), 이항(binary), 삼항(ternary) 연산자
 - 연산자는 연산에 참여하는 피연산자(operand)의 갯수에 따라 구분
 - 부호를 표시하는 +, -는 단항연산자
 - 덧셈, 뺄셈의 +, -, *, / 등의 연산은 이항연산자
 - 삼항연산자는 조건연산자 '? : '가 유일
- 단항연산자
 - 연산자의 위치에 따라 전위와 후위로 나뉨
 - ++a처럼 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

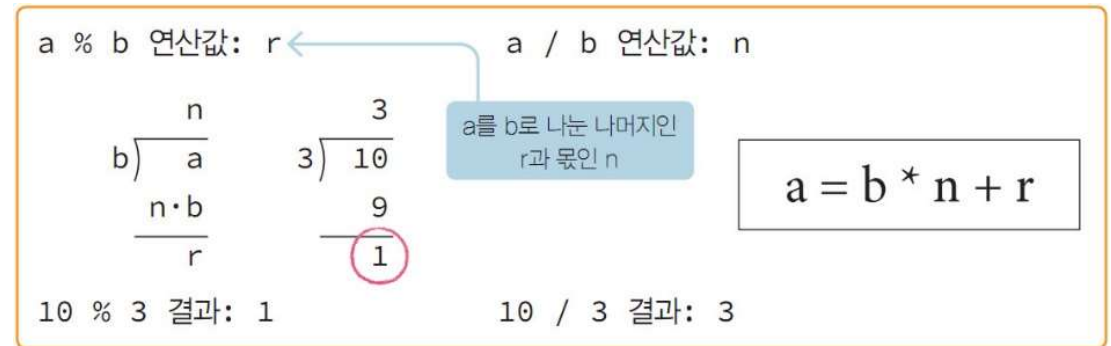


그림 5-4 나누기 연산과 나머지 연산의 이해



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
- 연산자 오른쪽에 위치한 연산식 exp를 계산
 - 오른쪽을 의미하는 right 단어에서 r-value
- 그 결과를 왼쪽 변수 var에 저장
 - 대입연산자의 왼쪽 부분에는 반드시 하나의 변수만이 올 수 있음
 - 이 하나의 변수를 왼쪽을 의미하는 left 단어에서 l-value
- $a = b = c = 5$ 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
```

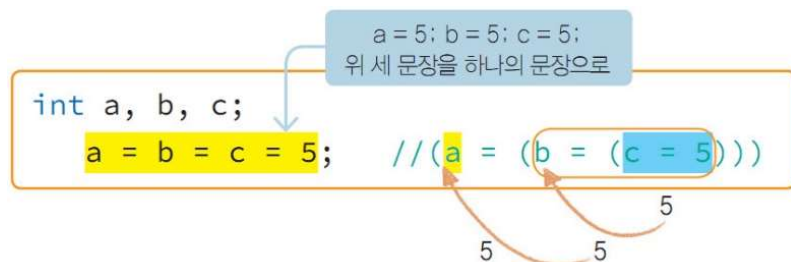


그림 5-8 중첩된 대입문

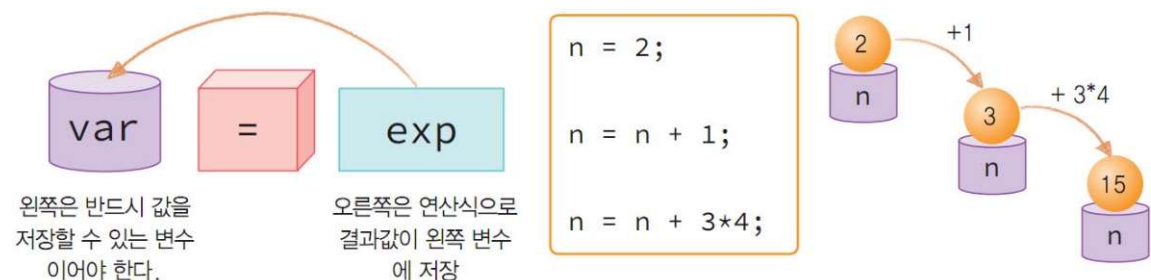


그림 5-7 대입연산자 수행 방법

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

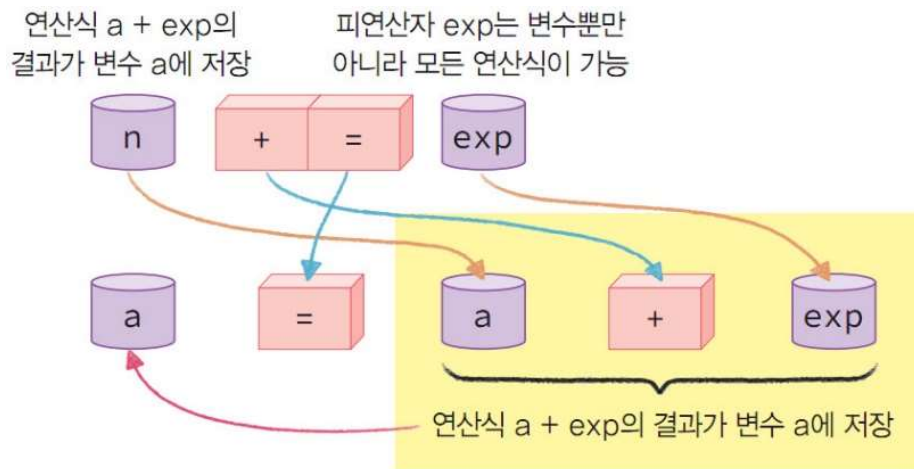


그림 5-9 축약 대입연산자의 연산 방법

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

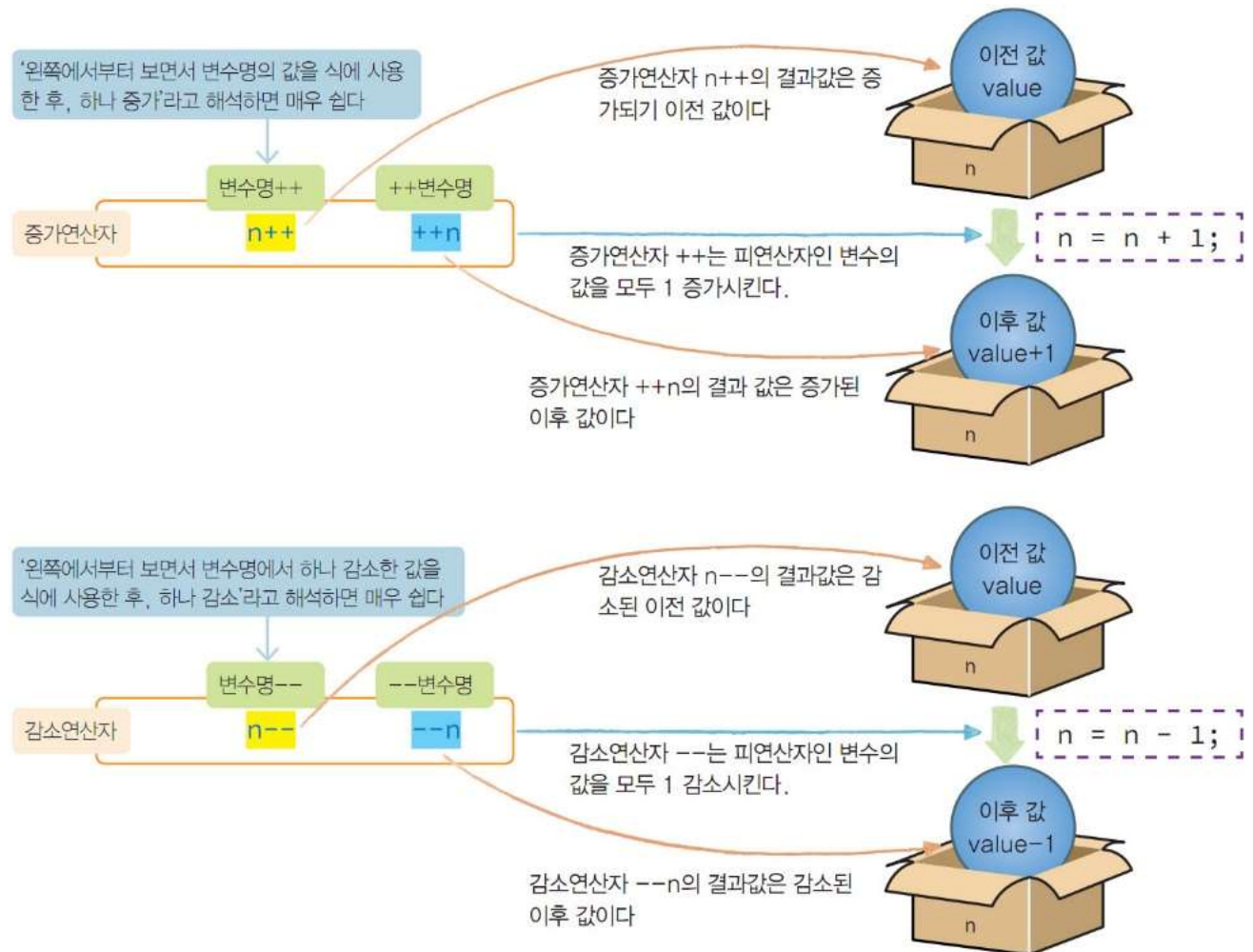


그림 5-11 증가연산자 ++, 감소연산자 --

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++);
```

10

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

11

```
int n = 10;
```

출력

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

10

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

11

```
int n = 10;
```

출력

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

10

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

9

```
int n = 10;
```

출력

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

9

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

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.

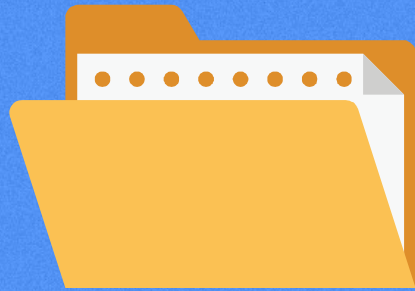
```
a = ++a * a--;
```

그림 5-12 증감연산자 사용

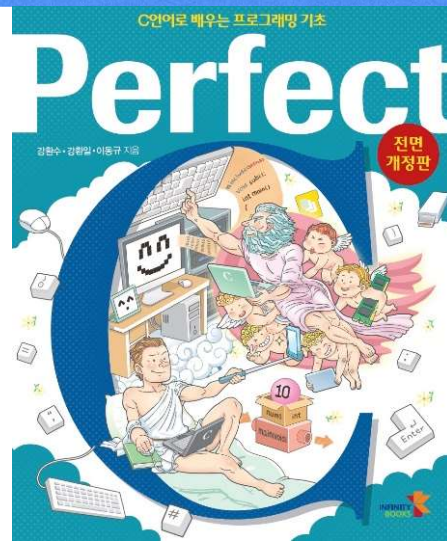
LAB 표준입력된 두 실수의 산술연산 출력

- 다음 정보를 이용하여 두 실수의 합, 차, 곱, 나누기의 과정과 결과를 출력하는 프로그램
 - 자료형 double의 변수 a와 b 에 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
	<pre>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("%.2f + %.2f ==> %.2f\n", a, b, a + b); 14 printf("%.2f - %.2f ==> %.2f\n", a, b, a - b); 15 printf("_____ ==> %.2f\n", _____); 16 printf("_____ ==> %.2f\n", _____); 17 18 return 0; 19 }</pre>
정답	<pre>11 %lf, %lf 15 %.2f * %.2f a, b, a * b 16 %-.2f / %-.2f a, b, a / b</pre>



02. 관계와 논리, 조건과 비트연산자



두 피연산자의 크기 비교

예제 relation.c

- 두 피연산자의 크기를 비교하기 위한 연산자: 관계연산자
 - 비교 결과가 참이면 0, 거짓이면 1
- 관계연산자 `!=`, `>=`, `<=`는 연산 기호의 순서가 명확
 - 정수형, 실수형, 문자형 등이 피연산자가 될 수 있음
- 피연산자가 문자인 경우, 문자 코드값에 대한 비교의 결과
 - 문자 'a'는 코드값이 97이고 문자 'Z'는 코드값이 90이므로 연산식 `('Z' < 'a')`는 1인 참을 의미
 - 즉 문자 '0' < '1' < '2' < '3' < ... < '9'인 관계가 있으며
 - 'a' < 'b' < 'c' < ... < 'x' < 'y' < 'z' 관계가 있고,
 - 대문자도 마찬가지로이며, 'Z' < 'a'로 소문자는 대문자보다 모두 큼

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: 1
(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
>	<code>x > y</code>	x가 y보다 큰가?	<code>3 > 5</code>	0 (거짓이면)
>=	<code>x >= y</code>	x가 y보다 크거나 같은가?	<code>5-4 >= 0</code>	1 (참이면)
<	<code>x < y</code>	x가 y보다 작은가?	<code>'a' < 'b'</code>	1 (참이면)
<=	<code>x <= y</code>	x가 y보다 작거나 같은가?	<code>3.43 <= 5.862</code>	1 (참이면)
!=	<code>x != y</code>	x와 y가 다른가?	<code>5-4 != 3/2</code>	0 (거짓이면)
==	<code>x == y</code>	x가 y가 같은가?	<code>'%' == 'A'</code>	0 (거짓이면)

논리연산자

예제 logic.c

- 세 가지의 논리연산자 &&, ||, !을 제공
 - 논리연산자 &&, ||, !은 각각 and, or, not 의미
 - 결과가 참이면 1 거짓이면 0을 반환
- C 언어에서 참과 거짓의 논리형은 따로 없음
 - 0, 0.0, ₩0은 거짓을 의미
 - 0이 아닌 모든 정수와 실수, 그리고 널(null) 문자 '₩0'가 아닌 모든 문자와 문자열은 모두 참을 의미
- 논리연산자 &&
 - 두 피연산자가 모두 참(0이 아니어야)이면 결과가 1(참)이며
 - 나머지 경우는 모두 0
- 논리연산자 ||
 - 두 피연산자 중에서 하나만 참(0이 아니어야)이면 1이고,
 - 모두 0(거짓)이면 0
- 논리연산자 !
 - 단항연산자로 피연산자가 0이면 결과는 1이고
 - 참(0이 아닌 값)이면 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

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

실습예제 5-7

shorteval.c

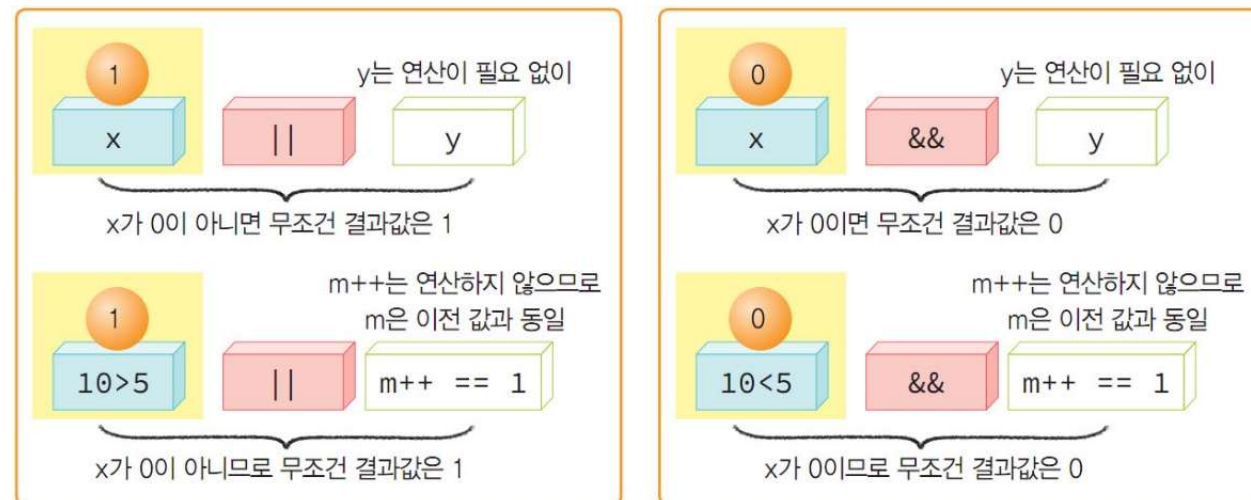
```
01 // file: shorteval.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 10, b = 5, m = 1;
08     int result;
09
10     result = (a < b) && (m++ == 1);
11     printf("m=%d result=%d\n", m, result);
12
13     result = (a > b) || (--m == 0);
14     printf("m=%d result=%d\n", m, result);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

설명

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

실행결과

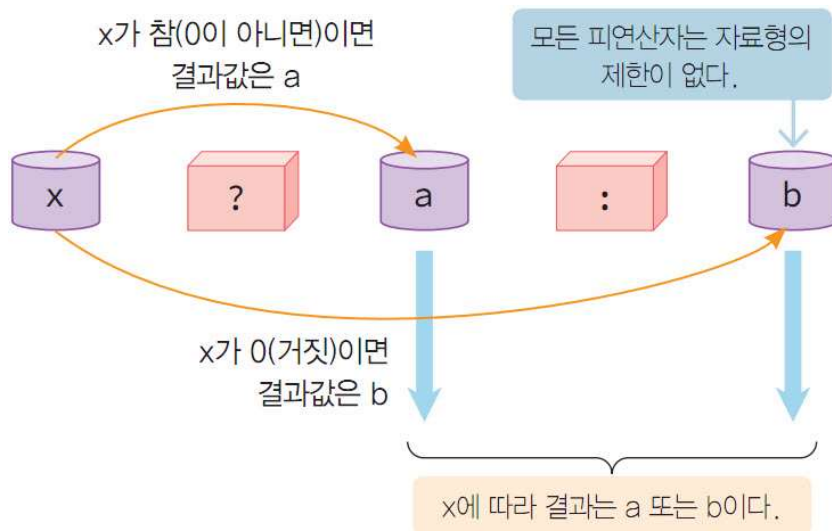
m=1 result=0
m=1 result=1



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 x가 참이면(0이 아니면) 결과는 a이며,
 - x가 0이면(거짓) 결과는 b



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     printf("%s\n", ((b % 2) == 0) ? "짝수" : "홀수");
20
21     return 0;
22 }
```

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

그림 5-15 조건연산자 계산 방법

비트 연산자

- 정수의 비트 중심(bitwise) 연산자를 제공

- 비트 논리연산자와 이동연산자가 제공

- 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	~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 ~

- ~5와 같이 연산자 ~가 피연산자인 4 앞에 위치하는 전위인 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

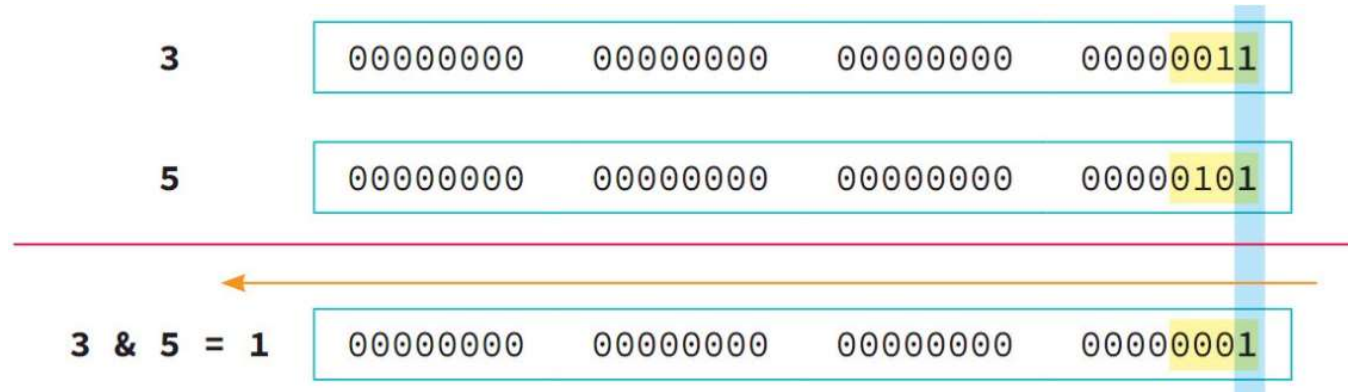


그림 5-17 비트 논리 연산 3 & 5

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 -1은 $((\sim 1) + 1)$ 로, 정수 -1을 비트로 표현하면 32비트가 모두 1인 정수
- 비트 논리 연산식 $x \& -1$
 - 정수 x를 -1로 논리 and 연산을 수행하는 식으로 결과는 x
- 비트 논리 연산식 $x | 0$
 - 정수 x를 0으로 논리 or 연산을 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 == ~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 10이 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 -> ffffffff
    
```

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

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

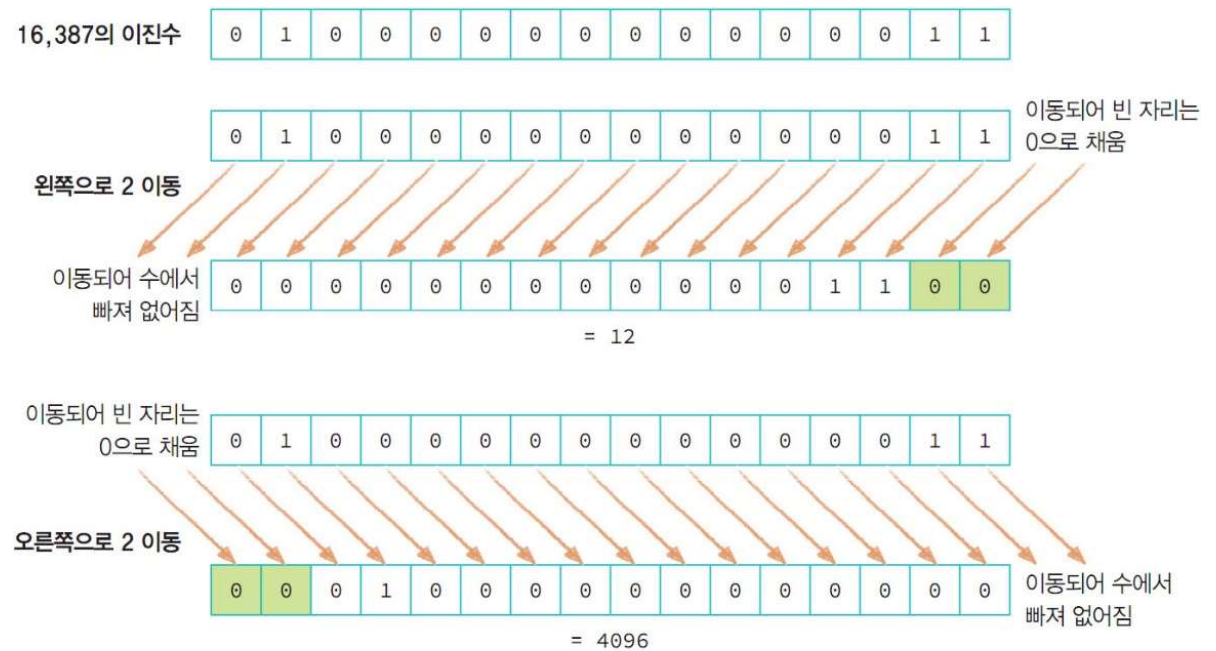


그림 5-20 비트 논리 연산 16387 << 2와 16387 >> 2

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 -1을 한 짝수를 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 >> 3, x >> 3);
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
x << 2 --> 65564, 0001001c
x << 2 --> 131128, 00020038
```

표 5-8 다양한 비트 이동연산식

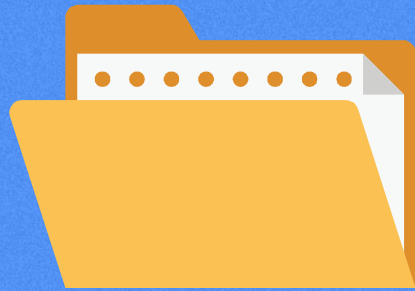
연산식	설명	연산값
15 << 1	0000 1111 << 1 0001 1110	30
0x30000000 << 2	최상위 4비트: 0011 0... << 2 1100 0...	-1073741824 0XC0000000
-30 >> 1	짝수이면 2로 나누기	-15

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

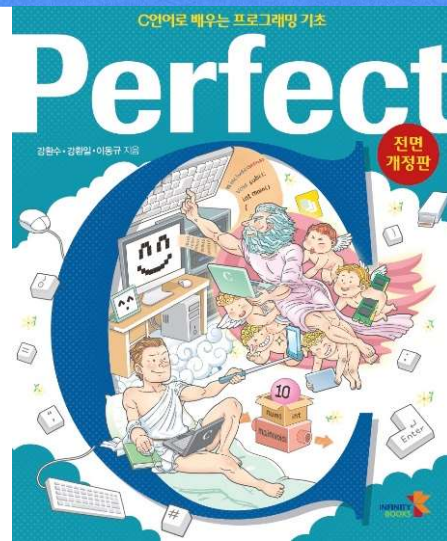
LAB 표준입력으로 받은 두 정수의 비트 연산 수행 출력

- 표준입력으로 받은 두 정수의 6개 비트 연산을 수행하여 결과를 출력하는 프로그램
 - 비트 연산은 $\&$, $|$, $\wedge$, $\sim$, $\gg$, $\ll$ 연산을 수행
 - 단항 연산은 첫 번째 변수에 대한 연산 수행
- 결과
 - 비트 연산이 가능한 두 정수를 입력하세요
 - 40 3
 - $40 \& 3 \implies 0$
 - $40 | 3 \implies 43$
 - $40 \wedge 3 \implies 43$
 - $\sim 40 \implies -41$
 - $40 \gg 3 \implies 5$
 - $40 \ll 3 \implies 320$

Lab 5-2	twobitop.c
	<pre>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", &a, &b); 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", a, b, a ^ b); 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, a << b); 19 20 return 0; 21 }</pre>
정답	<pre>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);</pre>



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 형 넓히기라고도 부름
- 올림변환은 정보의 손실이 없으므로 컴파일러에 의해 자동으로 수행 가능
 - 컴파일러가 자동으로 수행하는 형 변환: 묵시적 형변환(implicit type conversion)

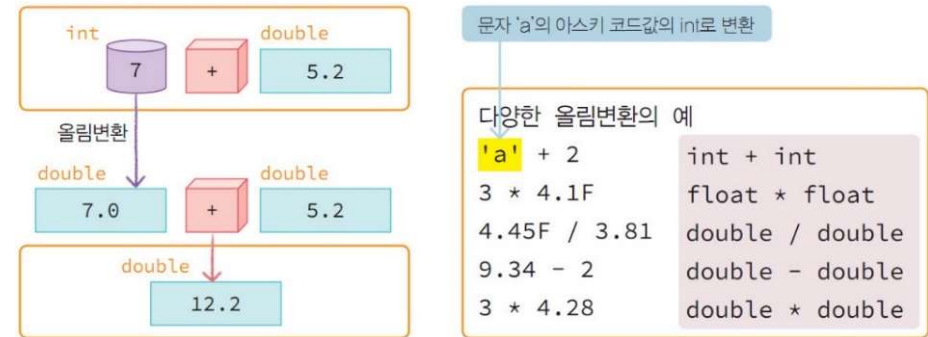


그림 5-21 피연산자의 자동 올림변환

- 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 형변환이 필요

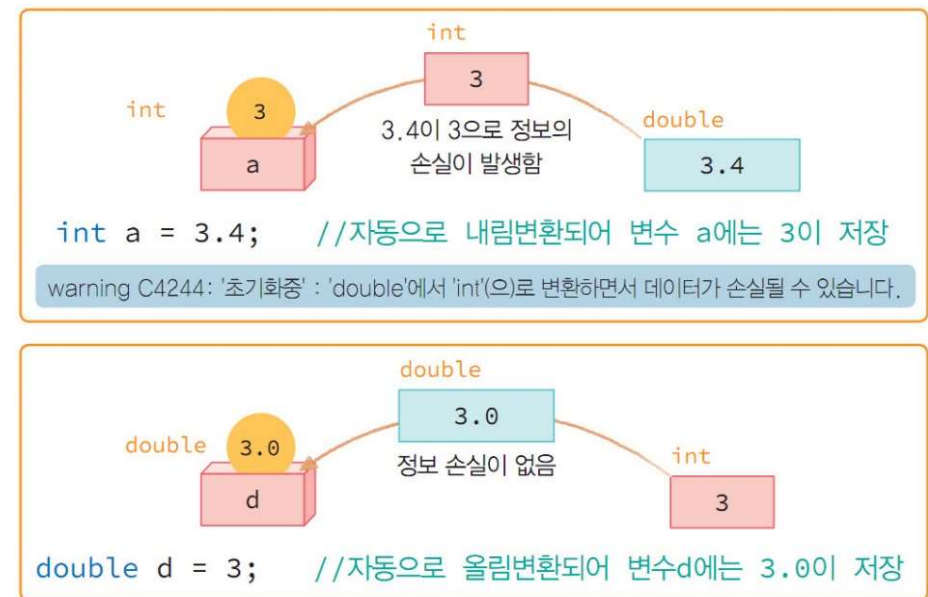


그림 5-22 대입연산에서의 내림변환과 올림변환

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 3.0이 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

13 (정수/정수)의 결과는 나머지 버린 결과
 14 자료형 변환연산자 (double)은 다른 이항 연산자보다 먼저 계산
 15 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
```

형변환 방법

(자료형) 피연산자
 (int) 30.525

int
 30

다양한 형변환연산 예

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

25

그림 5-23 형변환 연산자와 예

25

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

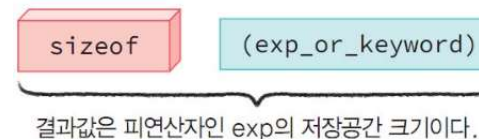
```
01 // file: comma.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 100, b = 50, c;
08
09     printf("%d ", sizeof(short));
10     printf("%d ", sizeof a);
11     printf("%d ", sizeof 3.5F);
12     printf("%d\n", sizeof 3.14);
13
14     c = ++a, b++;
15     printf("%d %d %d\n", 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)	결과값은 4
sizeof (3.14)	결과값은 8
sizeof a	결과값은 2 (short a;)

그림 5-25 sizeof 연산자의 연산방법



$3 + 4, 5 - 10$	결과값은 -5
$3 + 4, 5 - 10, 2 * 3$	결과값은 6

그림 5-26 콤마연산자의 연산방법

복잡한 표현식의 계산

- 연산자 우선순위와 결합성
- 규칙 3개를 적용
 - 첫 번째 규칙은 괄호가 있으면 먼저 계산
 - 괄호 우선 규칙: '괄호가 있으면 먼저 계산한다'라는 규칙
 - 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'
 - 다만 제곱승과 같은 정해진 연산은 오른쪽에서 왼쪽으로 차례로 계산한다'라는 규칙

$$3 + 8 / 2 * 4$$

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 $3 + 16$ 이므로 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	관계	-> (좌에서 우로)
8	== !=	동등 비교		-> (좌에서 우로)
9	&	비트 AND 또는 논리 AND	비트	-> (좌에서 우로)
10	^	비트 XOR 또는 논리 XOR		-> (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)

연산자 우선순위

예제 priority.c

- 다양한 연산식의 사용

우선순위 요약

이항연산자

콤마 < 대입 < 조건(삼항) < 논리 < 관계 < 산술 < 단항 < 괄호와 대괄호

- 괄호와 대괄호는 무엇보다도 가장 먼저 계산한다.
- 모든 단항연산자는 어느 이항연산자보다 먼저 계산한다.
- 산술연산자 *, /, %는 +, -보다 먼저 계산한다.
- 산술연산자는 이항연산자 중에서 가장 먼저 계산한다.
- 관계연산자는 논리연산자보다 먼저 계산한다.
- 조건 삼항연산자는 대입연산자보다 먼저 계산하나, 다른 대부분의 연산보다는 늦게 계산한다.
- 조건 > 대입 > 콤마연산자 순으로 나중에 계산한다.

그림 5-28 연산자 우선순위 요약

실습예제 5-13

priority.c

```
01 // file: priority.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int a = 4, b = 6;
08     double x = 3.3, y = 4.7;
09
10     printf("%d ", a + b > y && x < y); //산술 > 관계 > 논리
11     printf("%d ", a++ - --b * 2); //단항 > 곱셈 > 뺄셈
12     printf("%f ", a > b ? x + 1 : y * 2); //산술 > 관계 > 조건
13     printf("%f ", x += 3 && y + 2); //산술 > 논리 > 대입
14     printf("%f\n", (x = x + 1, y = y + 1)); //괄호 > 산술 > 대입 > 콤마
15
16     return 0;
17 }
```

설명

- 연산식 ((a + b) > y) && (x < y)로 실행
- 연산식 ((a++) - (--b * 2))로 실행
- 연산식 (a > b) ? (x + 1) : (y * 2)로 실행
- 연산식 x += (3 && (y + 2))로 실행
- 연산식 ((x = x + 1), (y = y + 1))로 실행

실행결과

1 -6 9.400000 4.300000 5.700000

표 5-10 연산 우선순위 예

변수값	표현식	설명	해석	결과
x = 3 y = 3	x >> 1 + 1 > 1 & y	산술 > 이동 > 관계 > 비트	((x >> (1 + 1)) > 1) & y	0
	x - 3 y & 2	산술 > 비트 > 논리	(x - 3) (y & 2)	1
	x & y && y >= 4	관계 > 비트 > 논리	(x & y) && (y >= 4)	0
	x && x y++	증가 > 비트 > 논리	x && (x (y++))	1

결합성

예제 association.c

- 동일한 우선순위 계산 순서

결합법칙

- 대부분 좌에서 우로 수행
- 우에서 좌(d)로 수행
 - 우선 순위가 2위인 전위의 단항 연산자, 우선 순위 14위인 조건연산자 그리고 우선순위 15위인 대입연산자
- 산술연산식 $10 * 3 / 2$
 - $((10 * 3) / 2)$, 결과는 15
- 축약 대입연산자로 구성
 - 연산식 $n += m /= 3$
 - 우에서 좌(d)로 먼저 결합하여 식 $(n += (m /= 3))$ 을 수행

실습예제 5-14

association.c

```
01 // file: association.c
02
03 #include <stdio.h>
04
05 int main(void)
06 {
07     int m = 5, n = 10;
08
09     //우측에서 좌측으로 결합
10     printf("%d ", n += m /= 3);
11     m = 5; n = 10;
12     printf("%d\n", (n += (m /= 3)));
13
14     printf("%d ", 10 * 3 / 2); //좌측에서 우측으로 결합
15     printf("%d\n", 10 * (3 / 2)); //우측에서 좌측으로 결합
16
17     //우측에서 좌측으로 결합
18     printf("%d ", 3>4 ? 3-4 : 3>4 ? 3+4 : 3*4);
19     printf("%d\n", 3>4 ? 3-4 : (3>4 ? 3+4 : 3*4));
20
21     return 0;
22 }
```

설명

10~12 연산식 $n += m /= 3$ 은 $(n += (m /= 3))$ 이므로 모두 결과값인 11 출력
14~15 연산식 $10 * 3 / 2$ 은 $(10 * 3) / 2$ 이므로 $10 * (3 / 2)$ 과 결과가 다름
18~19 조건연산자는 결합성이 오른쪽에서 왼쪽으로

실행결과

```
11 11
15 10
12 12
```

```
int n = 10, m = 5;
n += m /= 3;
```

n += m /= 3

① $m = m / 3$ 의 결과 1

② $n = n + 1$ 의 결과 11

그림 5-29 우에서 좌(←)로 먼저 결합하는 축약 대입연산자

다양한 연산식

- 수학이나 공학에서의 다양한 수식을 표현
 - 연산의 우선순위를 고려하여 괄호의 사용이 필요
 - 제곱근을 구하는 함수 sqrt()를 활용

표 5-11 다양한 연산식 표현

수학식	C 연산식
$(a + b)(x + y)^2$	<code>(a + b) * (x + y) * (x + y)</code>
$4x^3 + 3x^2 - 5x + 10$	<code>4*x*x*x + 3*x*x - 5*x + 10</code>
$\frac{a + b}{a - b}$	<code>(a + b) / (a - b)</code>
$\frac{5}{9}(F - 32)$	<code>(5.0 / 9) * (F - 32)</code>
$\frac{9}{5}C + 32$	<code>9.0 / 5 * C + 32</code>
$\frac{1 - x}{x^2}$	<code>(1 - x) / (x * x)</code>
$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$	<code>(-b + sqrt(b*b - 4*a*c)) / 2*a</code> sqrt(x)는 x의 제곱근을 구하는 함수

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	celftofar.c
	<pre>01 // celftofar.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 }</pre>
정답	<pre>12 fahrenheit = (9.0 / 5.0) * celsius + 32.0; 13 printf("\n입력된 %.2f도는 화씨온도로 %.2f도 입니다.\n", celsius, fahrenheit</pre>



Thank you