



C 프로그래밍 및 실습

C Programming

이지민



CHAP 04 수식과 연산자

학습목표

수식의 개념과 연산자, 피연산자에 대해서 알아본다.

C의 연산자의 종류를 알아본다.

연산자의 우선 순위와 결합 방향에 대하여 알아본다..

목차



연산자의 기본 개념

- 수식
- 연산자와 피연산자



연산자의 종류

- 산술연산자 / 증감연산자
- 관계연산자 / 논리연산자
- 비트연산자 / 대입연산자
- 조건연산자 / 형변환연산자



연산자의 우선순위와 결합방향

- 연산자의 우선순위
- 연산자의 결합 방향

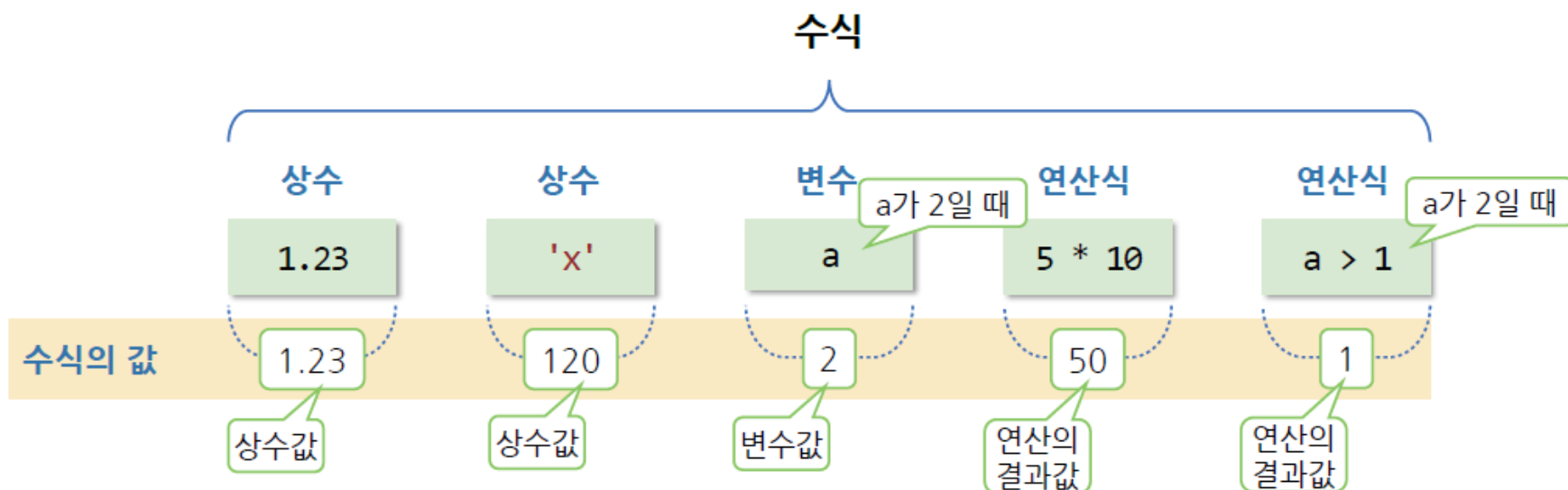
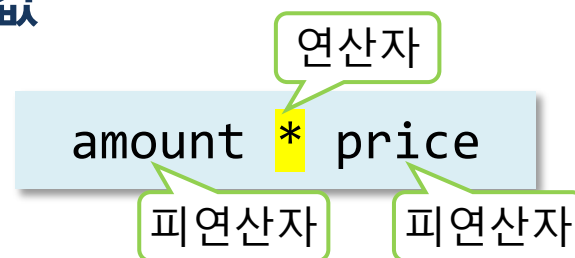
연산자의 기본 개념_수식

❖ 수식(expression) : 연산자와 피연산자의 조합, 값이 있는 요소

- 연산자(operator) : 연산에 사용되는 기호
- 피연산자(operand) : 연산의 대상이 되는 값

❖ 모든 수식에는 반드시 값이 있다.

- 수식의 평가 : 수식의 값을 구하는 것

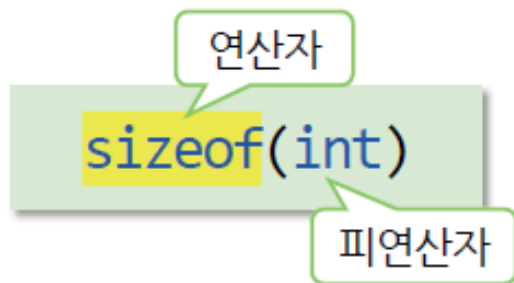


연산자의 기본 개념_연산자와 피연산자

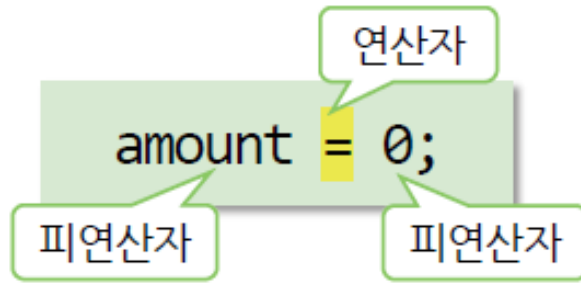
❖ 연산식은 연산자와 하나 이상의 피연산자로 구성

❖ 피연산자의 개수에 따라

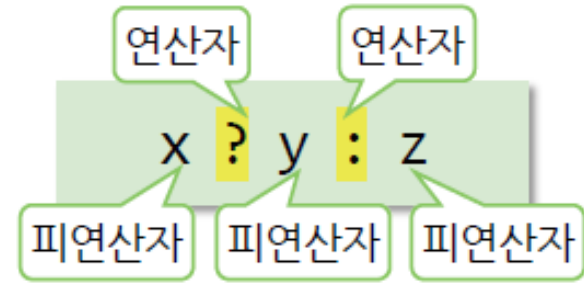
- 단항 연산자(unary operator) : 피연산자가 1개
- 이항 연산자(binary operator) : 피연산자가 2개
- 삼항 연산자(ternary operator) : 피연산자가 3개



단항 연산자



이항 연산자



삼항 연산자

연산자의 기본 개념_연산자와 피연산자

❖ 피연산자의 개수에 따라

종류	연산자 의미	연산자	
단항 연산자	1개의 피연산자	+x -y x++ ++x x-- --y ~x !x &x sizeof(x)	
이항 연산자	2개의 피연산자	산술	x+y x-y x*y x/y x%y
		대입	x=y x+=y x-=y x*=y x/=y x%=y x&=y x =y x^=y x>>=y x<<=y
		관계	x>y x<y x>=y x<=y x==y x!=y
		논리	x&&y x y
		비트	x&y x y x^y x<<y x>>y
삼항 연산자	3개의 피연산자	x?y:z	

연산자의 기본 개념_연산자와 피연산자

❖ 연산자의 종류_연산자의 기능에 의한 분류

종류	연산자
산술 연산자	$x+y$ $x-y$ $x*y$ x/y $x\%y$
증감 연산자	$x++$ $++x$ $x--$ $--y$
관계 연산자	$x>y$ $x<y$ $x>=y$ $x<=y$ $x==y$ $x!=y$
논리 연산자	$x\&\&y$ $x y$ $!x$
비트 연산자	$x\&y$ $x y$ $x\wedge y$ $\sim x$ $x\ll y$ $x\gg y$
대입 연산자	$x=y$ $x+=y$ $x-=y$ $x*=y$ $x/=y$ $x\%=y$ $x\&=y$ $x =y$ $x\wedge=y$ $x\>>=y$ $x\ll=y$
멤버 접근 연산자	$*x$ $\&x$ $x[y]$ $x.y$ $x->y$
그 밖의 연산자	$x?y:z$ x,y $\text{sizeof}(x)$ $(\text{type})x$

연산자의 종류_산술연산자

❖ 산술 연산자(1/2)

- 기본적인 사칙 연산 기능을 제공하는 연산자

단항
연산자

연산자	의미	사용 예	연산의 결과
$+x$	플러스(부호)	$+5$	5
$-x$	마이너스(부호)	-5	-5
$x + y$	더하기	$5 + 7$	12
$x - y$	빼기	$5 - 7$	-2
$x * y$	곱하기	$5 * 7$	35
x / y	나누기	$13 / 5$	2
$x \% y$	나머지 구하기	$13 \% 5$	3

연산자의 종류_산술연산자

❖ 산술 연산자(2/2)

- 부호 연산자 +, - : 단항 연산자

```
short a = 10;  
printf("%d", -a);    // 수식의 값은 -10
```

- 나누기 연산자(/) : 피연산자가 둘 다 정수인 경우, 몫도 정수가 됨

```
int result1 = 10 / 3;    // 수식의 값은 3
```

- 나머지 연산자(%) : 피연산자가 모두 정수인 경우에만 사용

```
int result2 = 10 % 3;    // 수식의 값은 1
```

연산자의 종류_산술연산자

❖ 산술 연산자의 사용 예(1/2)_정수의 산술연산

```
03  int main(void)
04  {
05      int x = 0, y = 0;
06
07      printf("두 개의 정수를 입력하세요 : ");
08      scanf("%d %d", &x, &y);
09
10      printf("+%d = %d\n", x, +x);
11      printf("-%d = %d\n", y, -y);
12      printf("%d + %d = %d\n", x, y, x + y);
13      printf("%d - %d = %d\n", x, y, x - y);
14      printf("%d * %d = %d\n", x, y, x * y);
15      printf("%d / %d = %d\n", x, y, x / y);
16      printf("%d %% %d = %d\n", x, y, x % y);
17
18      return 0;
19  }
```

// 플러스
// 마이너스
// 더하기
// 빼기
// 곱하기
// 나누기
// 나머지

%문자를 출력하려면 %%로 지정

실행결과

두 개의 정수를 입력하세요 : 10 3
+10 = 10
-3 = -3
10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3
10 % 3 = 1

연산자의 종류_산술연산자

❖ 산술 연산자의 사용 예(2/2)_나머지연산자

```
01: /*나머지 연산자*/
02: #include <stdio.h>
03:
04: int main(void)
05: {
06:     int num;
07:     int thousands, tens;
08:
09:     printf("6자리 정수를 입력하세요 : ");
10:     scanf("%d", &num);
11:
12:     thousands = num / 1000;
13:     tens = num % 1000; ← 나머지 연산자 사용
14:     printf("%d,%d\n", thousands, tens);
15:
16:     return 0;
17: }
```

실행 결과

정수를 입력하세요 : 123456
123,456

연산자의 종류_산술연산자

❖ 피연산자의 형변환

- 피연산자의 데이터형이 서로 다르면 피연산자를 같은 형으로 형 변환한 다음 연산을 수행한다.

```
int income = 3000000;  
printf("tax = %f\n", 0.35 * income);
```

income을 double로 변환

- 정수의 승격

- 피연산자가 char, short형일 때는 int형으로 변환해서 연산

```
short w = 2000, h = 4000;  
printf("sizeof(w * h) = %d\n", sizeof(w * h));
```

w와 h는 각각
int로 변환

w * h의 값은 int형

연산자의 종류_증감연산자

❖ 증감 연산자(1/2)

- 변수의 값을 1만큼 증가시키거나 감소시킨다.
- 증감 연산자는 반드시 변수에만 사용해야 한다.
 - 상수나 수식에 사용 불가

`++count;`

count 변수를 1 증가시킨다.



`10++;`

10은 상수이므로 증가할 수 없다.



`(count + 1)--;`

(count+1)은 수식이므로, 수식의 값을 증가할 수 없다.

■ 전위형과 후위형

구분	연산자	수식의 값
전위형	<code>++x</code>	증가된 후 변수 x의 값
	<code>--x</code>	감소된 후 변수 x의 값
후위형	<code>x++</code>	증가되기 전 변수 x의 값
	<code>x--</code>	감소되기 전 변수 x의 값

연산자의 종류_증감연산자

❖ 증감 연산자(2/2)

- 전위형과 후위형

전위형

단일 문장으로
사용되는 경우

```
int index = 0;  
++index;
```

두 문장의 수행
결과가 같다.

후위형

```
int index = 0;  
index++;
```

index가 1만큼 증가된다.

다른 문장의 일부로
사용되는 경우

```
int index = 0;  
int current = ++index;
```

수식의 값은
증가 **후** index의 값

1

```
int index = 0;  
int current = index++;
```

수식의 값은
증가 **전** index의 값

0

연산자의 종류_증감연산자

❖ 증감 연산자의 사용 예

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int stock1 = 10, stock2 = 10;
06      int current;
07
08      current = --stock1;
09      printf("current = %d, stock1 = %d\n", current, stock1);
10
11      current = stock2--;
12      printf("current = %d, stock2 = %d\n", current, stock2);
13  }
```

감소 후의
stock1의
값

실행 결과

current = 9, stock1 = 9
current = 10, stock2 = 9

감소 전의 stock2의 값

전위형

후위형

연산자의 종류_대입연산자(1/5)

❖ 대입 연산자(=)는 연산자의 좌변(변수)에 우변의 값을 저장한다.

형식 변수명 = 값;

사용예

```
width = 100;  
area = width * height;
```

❖ 대입 연산자의 좌변에는 **lvalue**만 사용할 수 있다.

- lvalue는 값을 변경할 수 있는 요소, 즉 변수를 의미

width는 lvalue이므로 대입 연산의 좌변으로 사용

```
int width;  
width = 100;  
100 = width;
```

100은 lvalue가 아니므로 대입 연산의 좌변으로 사용X

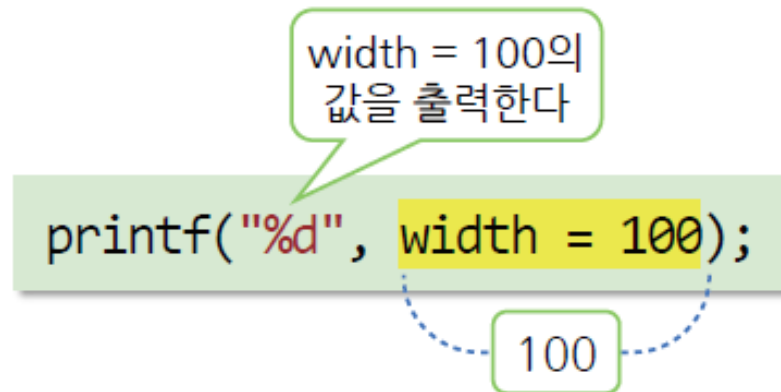
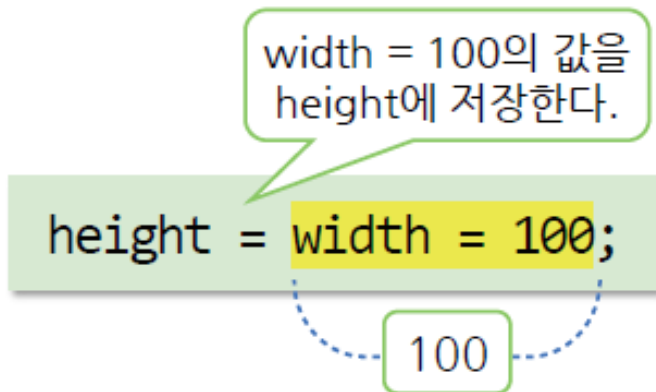
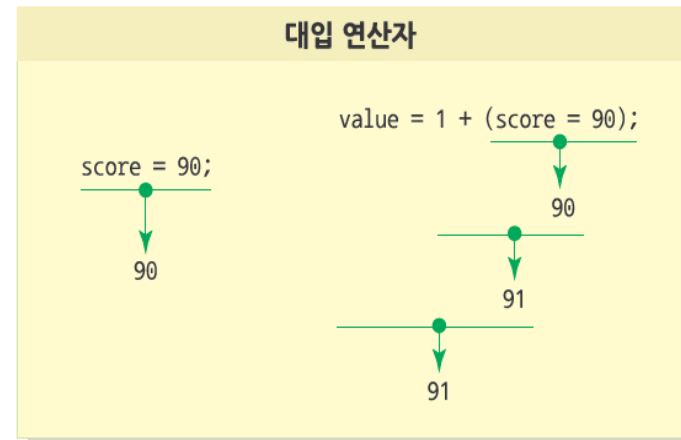
연산자의 종류_대입연산자(2/5)

❖ 매크로 상수나 `const` 변수에는 대입할 수 없다.



```
const double tax_rate = 0.1;  
tax_rate = 0.15;
```

❖ 대입 연산을 수행하면, 대입 연산자의 좌변에 있는 l-value의 값이 연산의 결과가 된다.



연산자의 종류_대입연산자(3/5)

❖ 대입 연산자는 산술 연산자, 비트 연산자와 결합해서 **복합 대입 연산자**로 사용될 수 있다.



복합 대입 연산자	의미	복합 대입 연산자	의미
<code>x += y</code>	<code>x = x + y</code>	<code>x &= y</code>	<code>x = x & y</code>
<code>x -= y</code>	<code>x = x - y</code>	<code>x = y</code>	<code>x = x y</code>
<code>x *= y</code>	<code>x = x * y</code>	<code>x ^= y</code>	<code>x = x ^ y</code>
<code>x /= y</code>	<code>x = x / y</code>	<code>x <<= y</code>	<code>x = x << y</code>
<code>x %= y</code>	<code>x = x % y</code>	<code>x >>= y</code>	<code>x = x >> y</code>

연산자의 종류_대입연산자(4/5)

❖복합대입연산자의 활용

```
left = items % items_per_page;
```

남은 항목의 개수를
별도의 변수에 저장

items는 바뀌지 않는다

```
items %= items_per_page;
```

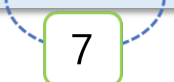
남은 항목의 개수를
items에 다시 저장

items가 바뀐다.

연산자의 종류_대입연산자(5/5)

❖ 복합 대입 연산자를 다른 연산자와 함께 사용할 때는 **복합 대입 연산자는 다른 연산자에 비해서 우선순위가 낮기 때문에 주의해야 한다.**

```
x *= 2 + 5;
```



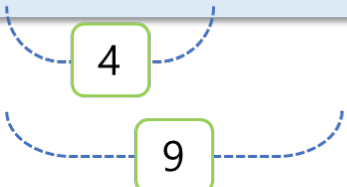
x *= 7로 처리된다.

≠

```
x = x * 2 + 5;
```

x가 2일 때

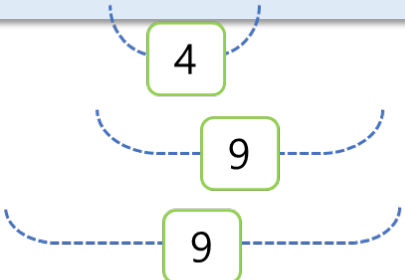
```
(x *= 2) + 5;
```



연산 후 x는 4이다.

≠

```
x = x * 2 + 5;
```



연산 후 x는 9이다.

연산자의 종류_대입연산자

❖ 대입 연산자의 사용 예(1/2)

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int w = 10, x = 20, y = 10, z = 7;
06      int result = 0;
07
08      result += w; ----- result = result + w;
09      printf("result = %d\n", result);
10
11      result *= x; ----- result = result * x;
12      printf("result = %d\n", result);
```

연산자의 종류_대입연산자

❖ 대입 연산자의 사용 예(2/2)

```
13
14     result /= y; ----- result = result / y;
15     printf("result = %d\n", result);
16
17     result %= z; ----- result = result % z;
18     printf("result = %d\n", result);
19
20     result *= 2 + 1; ----- result = result * (2 + 1)
21     printf("result = %d\n", result);
22 }
```

실행 결과

```
result = 10
result = 200
result = 20
result = 6
result = 18
```

연산자의 종류_관계연산자(1/3)

- ❖ 두 수의 값을 비교할 때 사용되는 연산자
- ❖ 관계 연산식의 값은 항상 참 또는 거짓
 - C에서 참(true)은 1이고, 거짓(false)은 0이다.
- ❖ 관계 연산식은 if문, for문, while문 등의 조건식으로 주로 사용

관계 연산자	의미	x = 5, y = 3 일 때 연산의 결과
$x > y$	x가 y보다 큰가?	1
$x \geq y$	x가 y보다 크거나 같은가?	1
$x < y$	x가 y보다 작은가?	0
$x \leq y$	x가 y보다 작거나 같은가?	0
$x == y$	x가 y와 같은가?	0
$x != y$	x가 y와 같지 않은가?	1

연산자의 종류_관계연산자(2/3)

❖ 두 수의 값이 같은지 비교할 때는 = 연산자가 아니라 == 연산자를 사용해야 한다.

x에 0을 대입하고
수식의 값은 0이
되므로 거짓

```
if (x = 0)
    printf("never executed");
```

대입 연산자

x가 0일 때만 참

```
if (x == 0)
    printf("x is zero");
```

같은지 비교하는 관계 연산자

```
if (num = 1) // num에 1을 대입하므로 항상 참
    printf("이 문장은 항상 출력됩니다.");
```

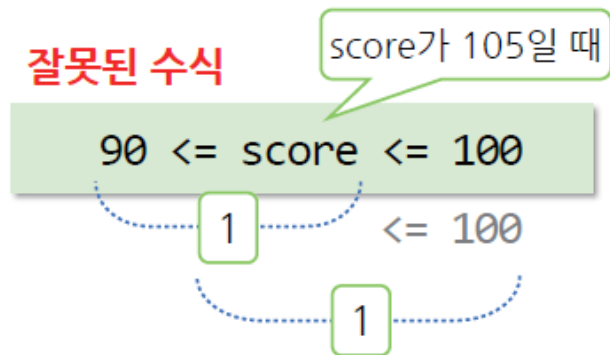
```
if (1 = a) // 1에는 a를 대입할 수 없으므로 1==a에서 =를 빼뜨렸다는 것을 알 수 있다.
    printf("a = %d", a);
```

연산자의 종류_관계연산자(3/3)

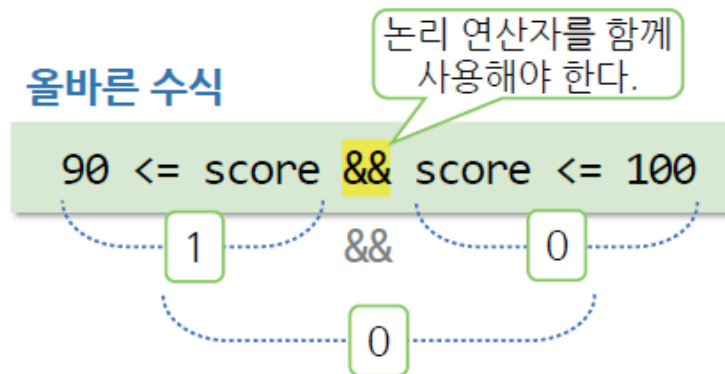
❖ 관계 연산식을 조합할 때는 논리 연산자를 함께 사용한다.

score가 90과 100 사이의 값인지 검사하는 수식

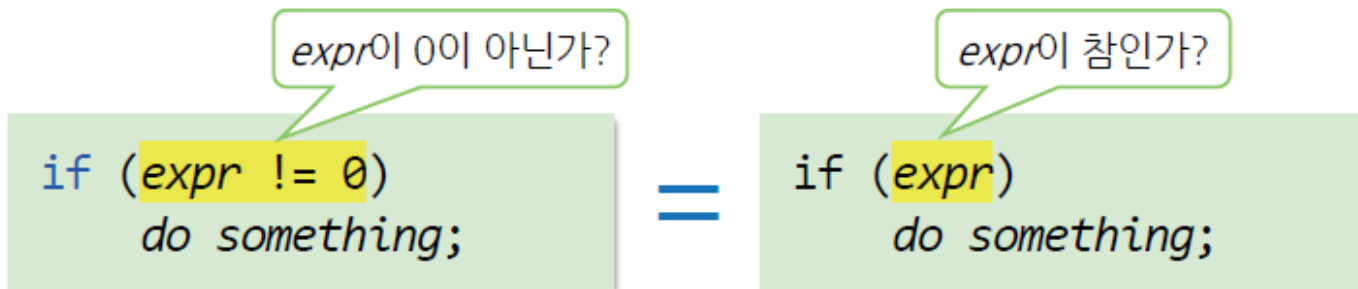
잘못된 수식



올바른 수식



❖ `expr != 0`과 `expr` 는 같은 의미이다.



0이 아닌 값은 참이다.

연산자의 종류_관계연산자 사용 예

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main(void)
05  {
06      int x, y;
07
08      printf("Input two numbers: ");
09      scanf("%d %d", &x, &y);
10
11      printf("%d > %d = %d\n", x, y, x > y);
12      printf("%d < %d = %d\n", x, y, x < y);
13      printf("%d >= %d = %d\n", x, y, x >= y);
14      printf("%d <= %d = %d\n", x, y, x <= y);
15      printf("%d == %d = %d\n", x, y, x == y);
16      printf("%d != %d = %d\n", x, y, x != y);
17  }
```

실행 결과

Input two numbers: 3 3

3 > 3 = 0

3 < 3 = 0

3 >= 3 = 1

3 <= 3 = 1

3 == 3 = 1

3 != 3 = 0

관계 연산의 결과는 참(1)
또는 거짓(0)이다.

연산자의 종류_논리연산자

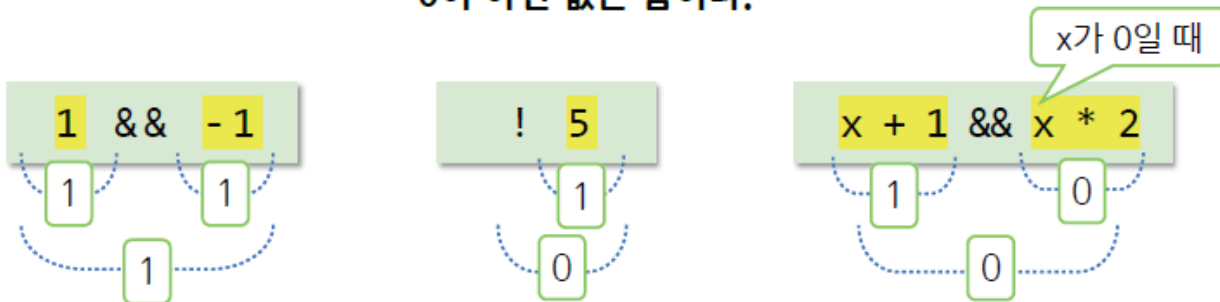
❖ 논리 연산자(1/4)

- 참과 거짓을 이용한 논리 연산 기능을 제공
- AND(&&), OR(||), NOT(!) 연산
- 논리 연산식의 값은 항상 0 또는 1이다.

논리 연산자	부울 대수	의미
$x \ \&\& \ y$	논리 AND	x와 y가 둘 다 0이 아니면 1, 그렇지 않으면 0
$x \ \ y$	논리 OR	x와 y중 하나라도 0이 아니면 1, x와 y가 둘 다 0이면 0
$! \ x$	논리 NOT	x가 0이면 1, x가 0이 아니면 0

- 논리 연산식에서 피연산자가 0이 아니면 참(1)으로 간주

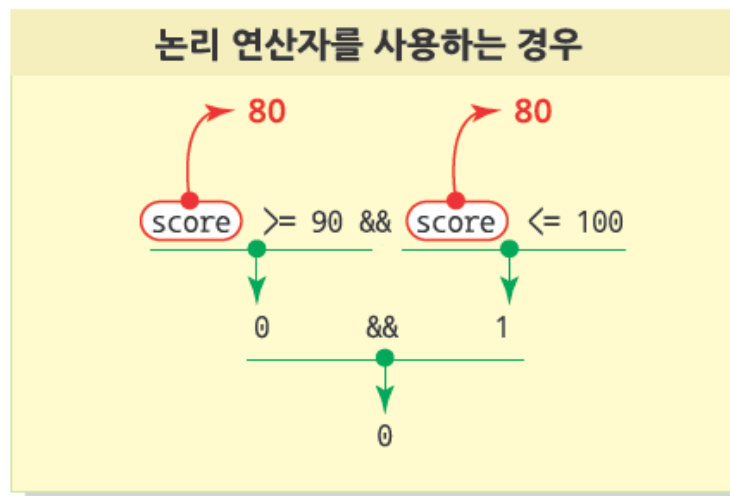
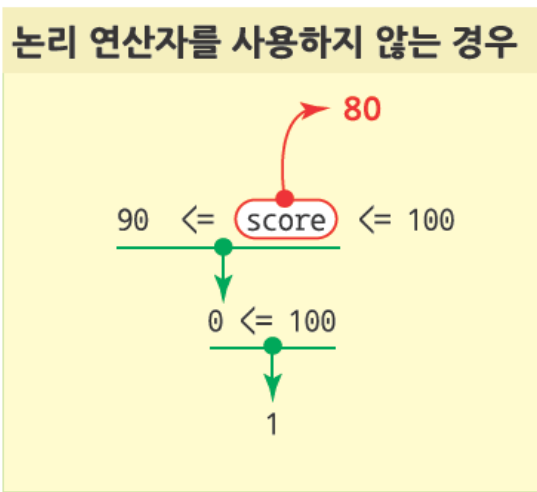
0이 아닌 값은 참이다.



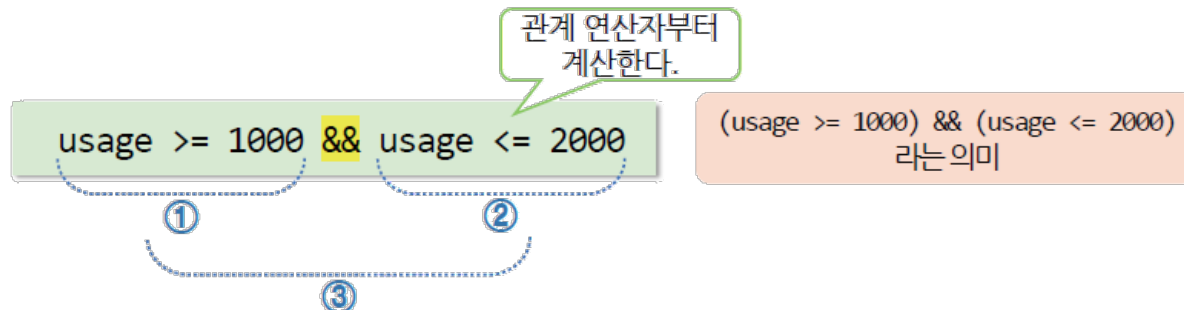
연산자의 종류_논리연산자

❖ 논리 연산자(2/4)

- 논리 연산자를 사용하지 않으면 잘못된 수식을 만들 수 있으므로 주의해야 한다.



- &&, || 연산보다 관계 연산을 먼저 수행



연산자의 종류_논리연산자

❖ 논리 연산자(3/4)

- ! 연산은 관계 연산보다 먼저 수행

단항 연산자의
우선순위가 높다.

`! expr == value`

①

②

`(! expr) == value`
라는 의미

- && 연산이 || 연산보다 먼저 수행

&& 연산을 ||보다
먼저 수행한다.

`expr1 || expr2 && expr3`

①

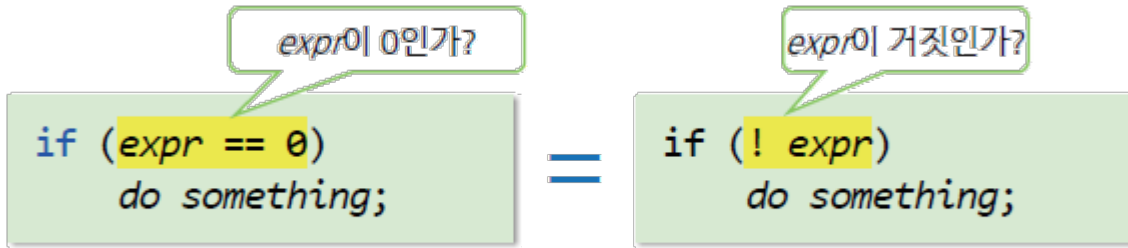
②

`expr1 || (expr2 && expr3)`
라는 의미

연산자의 종류_논리연산자

❖ 논리 연산자(4/4)

- 0인지 비교하는 대신 ! 연산자를 사용할 수 있음



0은 거짓이다.

- 논리 연산자의 연산 결과

x	y	x && y	x y	! x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

연산자의 종류_논리연산자

❖논리 연산자의 사용 예

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main(void)
05  {
06      int usage;
07
08      printf("usage? ");
09      scanf("%d", &usage);
10
11      if (usage >= 1000 && usage <= 2000)
12          printf("usage in range\n");
13
14      if (usage < 1000 || usage > 2000)
15          printf("out of range\n");
16  }
```

실행 결과

usage? 1200

usage in range

usage가 1000~2000 사이의 값인지 검사한다.

usage가 1000~2000 범위 밖의 값인지 검사한다.

연산자의 종류_비트연산자

❖ 비트 논리 연산자

- 각 비트에 대하여 논리 연산을 수행
- `&`, `|`, `^` 연산자는 피연산자의 데이터형이 같지 않으면 형 변환을 수행한 다음 논리 연산을 수행
- `~` 연산자는 피연산자가 `int`형보다 작은 형이면 `int`형으로 승격 후, 피연산자의 각 비트를 반전

```
unsigned short a = 0x1234;  
unsigned short b = 0x5678;  
printf("%d", sizeof(a & b));  
printf("%d", sizeof(~a));
```

a, b는 2바이트 크기

a & b는 4바이트 크기

~a는 4바이트 크기

연산자의 종류_비트연산자

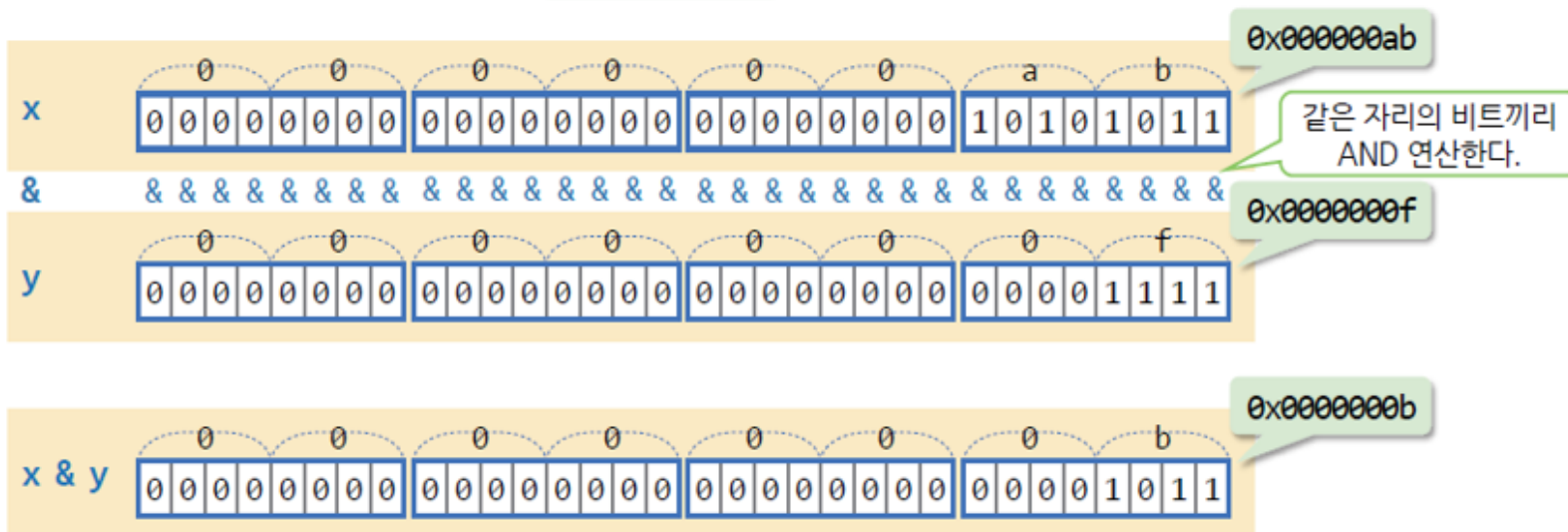
❖ 비트 AND 연산자(&)

- 각 비트 단위로 AND 연산을 수행

```
unsigned int x = 0xab;  
unsigned int y = 0xf;  
x & y
```

수식의 값은
0x0000000b

a의 비트	b의 비트	a의 비트 & b의 비트
0	0	0
0	1	0
1	0	0
1	1	1



연산자의 종류_비트연산자

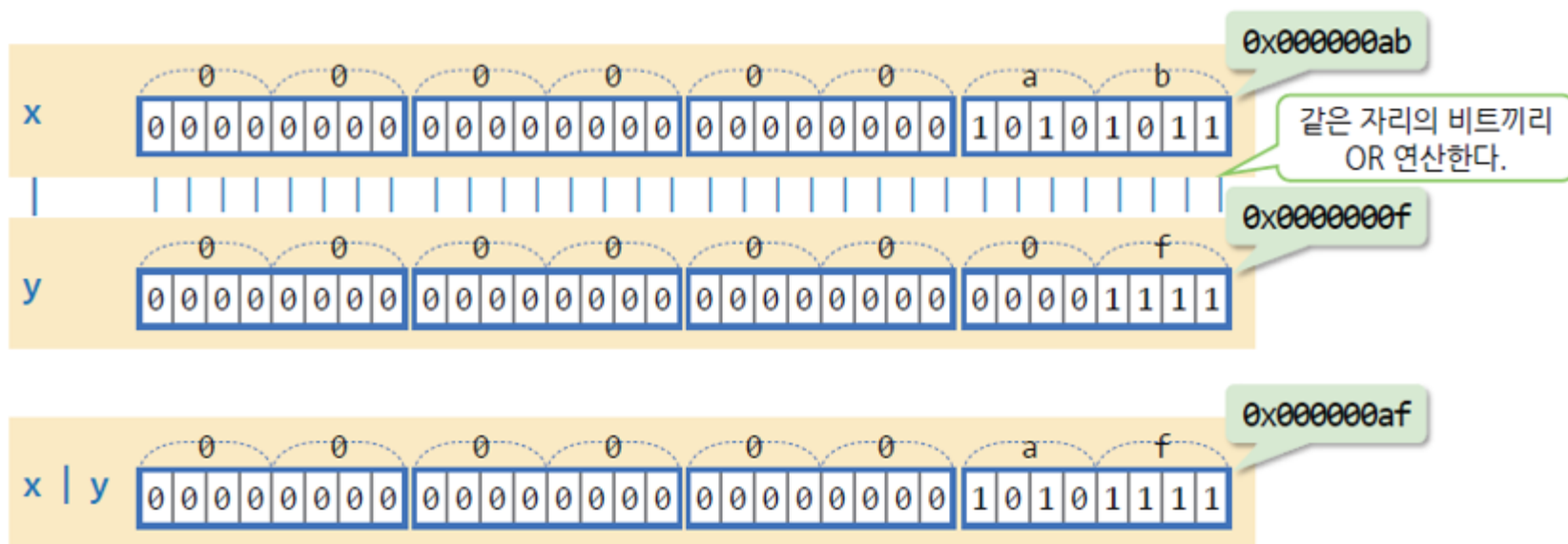
❖ 비트 OR 연산자(|)

- 피연산자의 같은 위치에 있는 비트에 대해서 비트 OR 연산을 수행

```
unsigned int x = 0xab;  
unsigned int y = 0xf;  
x | y
```

수식의 값은
0x000000af

a의 비트	b의 비트	a의 비트 b의 비트
0	0	0
0	1	1
1	0	1
1	1	1



연산자의 종류_비트연산자

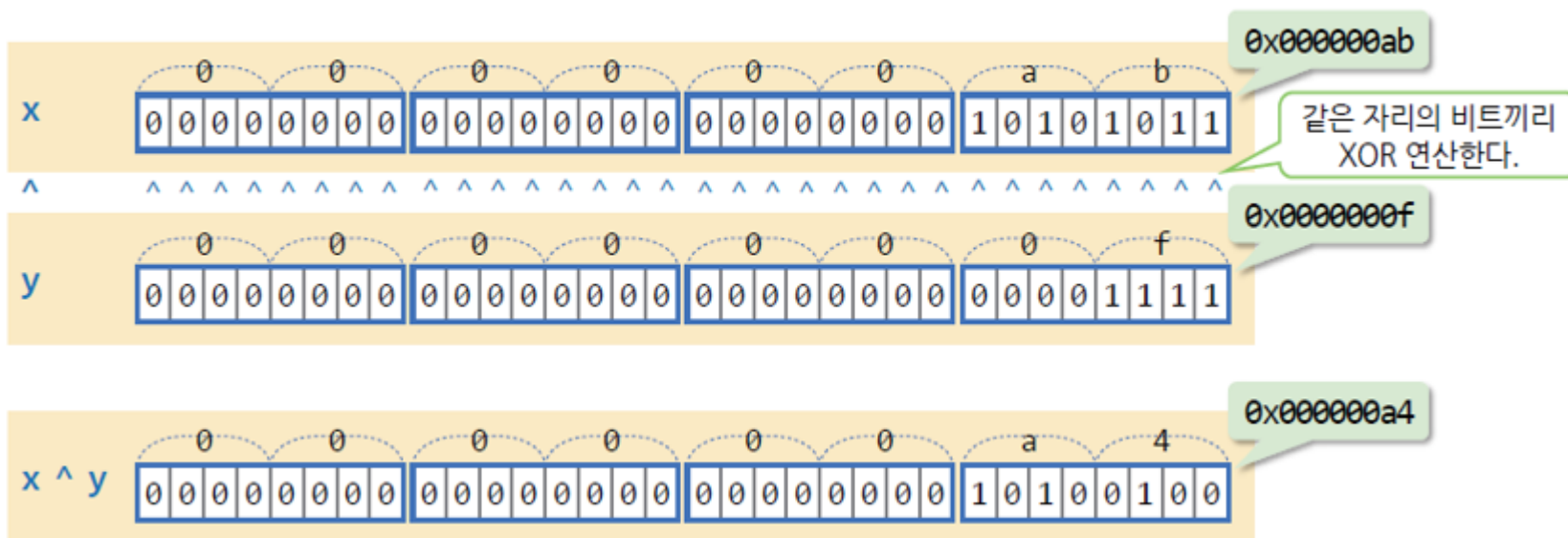
❖ 비트 XOR 연산자(^)

- 피연산자의 같은 위치에 있는 비트에 대해서 비트 XOR 연산을 수행

```
unsigned int x = 0xab;
unsigned int y = 0xf;
x ^ y
```

수식의 값은
0x000000a4

a의 비트	b의 비트	a의 비트^ b의 비트
0	0	0
0	1	1
1	0	1
1	1	0



연산자의 종류_비트연산자

❖ 비트 NOT 연산자(~)

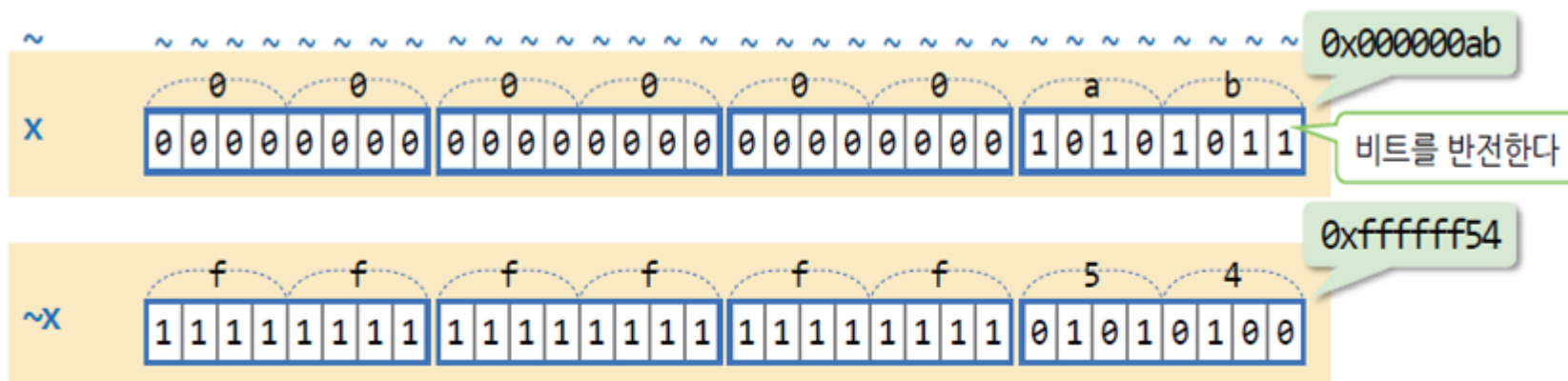
- 피연산자의 각 비트를 반전시킨다. 즉, 0은 1로, 1은 0으로 만든다.

```
unsigned int x = 0xab;
```

~X

수식의 값은
0xffffffff54

a의 비트	~a의 비트
0	1
1	0



연산자의 종류_비트연산자

❖비트 논리 연산자의 사용 예

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      unsigned int x = 0xab;
06      unsigned int y = 0x0f;
07
08      printf("%08x & %08x = %08x\n", x, y, x & y);
09      printf("%08x | %08x = %08x\n", x, y, x | y);
10      printf("%08x ^ %08x = %08x\n", x, y, x ^ y);
11      printf("~%08x = %08x\n", x, ~x);
12  }
```

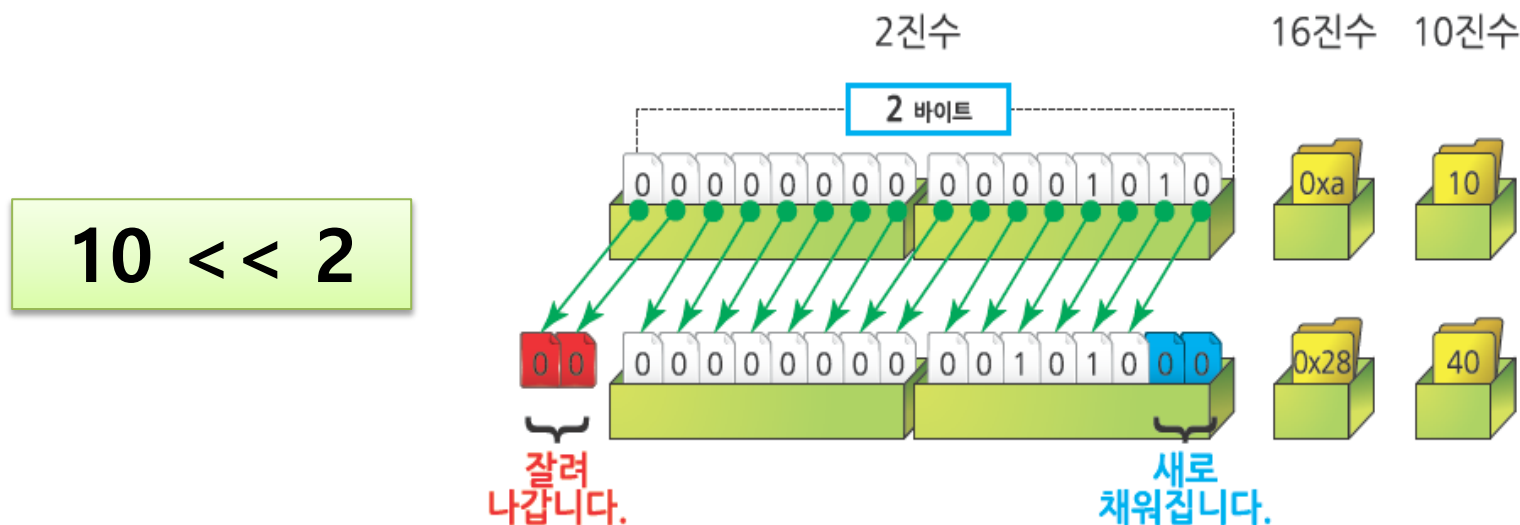
실행 결과

```
000000ab & 0000000f = 0000000b
000000ab | 0000000f = 000000af
000000ab ^ 0000000f = 000000a4
~000000ab = ffffffff54
```

연산자의 종류_비트연산자

❖ 비트 왼쪽 이동 연산자 <<

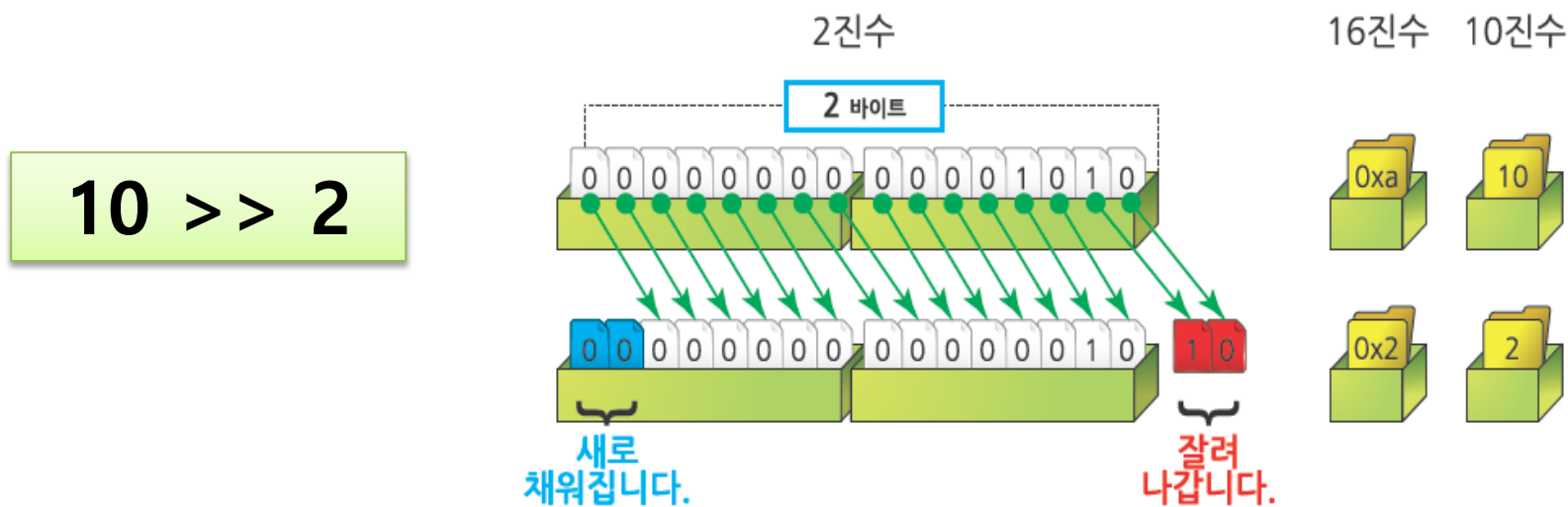
- 비트들을 왼쪽으로 이동(<<)시킴
- 왼쪽으로 밀려난 비트는 사라지고, 오른쪽 빈자리는 0이 채워짐
- N비트 왼쪽 이동은 2^N 을 곱하는 것과 같음



연산자의 종류_비트연산자

❖ 비트 오른쪽 이동 연산자 >>

- 비트들을 오른쪽으로 이동시킴
- 오른쪽으로 밀려난 비트는 사라지고, 왼쪽 빈자리를 부호 비트로 채움 (양수는 0으로, 음수는 1로 채움)
- N비트 오른쪽 이동은 2^N 으로 나누는 것과 같음



연산자의 종류_비트연산자

❖ 비트 이동 연산자의 사용 예

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      unsigned int x = 0xab;
06      unsigned int z;
07
08      printf("x = %#08x, %d\n", x, x);
09
10      z = x >> 2;
11      printf("z = %#08x, %d\n", z, z);
12
13      z = x << 2;
14      printf("z = %#08x, %d\n", z, z);
15  }
```

10진수
171에 해당

$x / 2^2$

$x * 2^2$

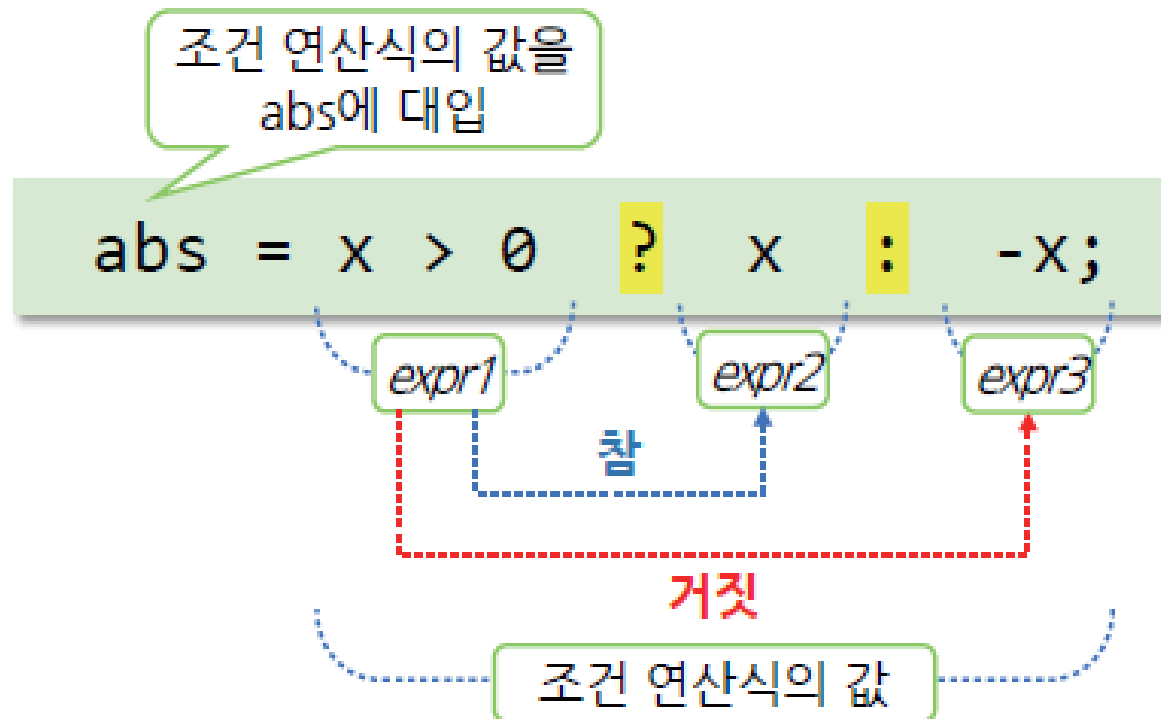
실행 결과

```
x = 0x0000ab, 171
z = 0x00002a, 42
z = 0x0002ac, 684
```

연산자의 종류_조건연산자

❖ 조건 연산자

- 유일한 삼항 연산자
- $expr1 ? expr2 : expr3$ 에서 $expr1$ 이 참(1)이면 $expr2$ 가 연산의 결과가 된다. $expr1$ 이 거짓(0)이면 $expr3$ 이 연산의 결과가 된다



연산자의 종류_조건연산자

❖조건 연산자의 사용 예(1/2)

```
01  #define  _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main(void)
05  {
06      int x, y;
07      int abs, min, max;
08
09      printf("Input two numbers: ");
10      scanf("%d %d", &x, &y);
11
12      abs = x > 0 ? x : -x;
13      printf("absolute value of x  = %d\n", abs);
14
```

x의 절대값을 구한다.

연산자의 종류_조건연산자

❖조건 연산자의 사용 예(2/2)

```
15 abs = y > 0 ? y : -y; ----- y의 절대값을 구한다.
```

```
16 printf("absolute value of y = %d\n", abs);
```

```
17
```

```
18 min = x < y ? x : y; ----- x, y중 최소값을 구한
```

```
19 printf("minimum value of x, y = %d\n", min); ----- 다.
```

```
20
```

```
21 max = x > y ? x : y; ----- x, y중 최대값을 구한다.
```

```
22 printf("maximum value of x, y = %d\n", max);
```

```
23 }
```

실행 결과

```
Input two numbers: 53 -12
absolute value of x = 53
absolute value of y = 12
minimum value of x, y = -12
maximum value of x, y = 53
```


연산자의 종류_형변환연산자

❖ 값의 데이터형을 변경하는 것

❖ 형 변환의 종류

- 자동으로 수행되는 **암시적인 형 변환**
 - 자동 형 변환이라고도 함
- 직접 형 변환을 지정하는 **명시적인 형 변환**
 - 형 변환 연산자를 이용한다.

연산자의 종류_형변환연산자

❖ 암시적인 형 변환

- 컴파일러에 의해서 자동으로 처리되는 형 변환
- 형 변환이 일어나는 경우
 - 서로 다른 형의 값을 혼합 연산하는 경우

```
int a = 10;
double d = 12.34;
printf("%f\n", a + d);
```

○ — a를 double로 변환해서 double + double을 수행한다.

- 피연산자 중 하나가 double이면 나머지를 double로 형 변환한다.
 - float와 정수형을 연산하면 정수형을 float로 형 변환한다.
 - char나 short은 연산 시 int로 형 변환한다. (정수의 승격)
- 변수에 다른 형의 값을 대입할 때

```
int num;
num = 3.14;
```

실수값을 int으로 형 변환하면서 값이 손실되므로
컴파일 경고 발생

연산자의 종류_형변환연산자

❖ 명시적인 형 변환(1/2)

- 프로그래머가 명시적으로 형 변환을 하고 싶을 때, 형 변환 연산자를 이용
- 수식 앞에 () 안에 데이터형을 써준다.

→ 형 변환 연산자

형식	(데이터형) 수식
사용예	<pre>(double) 0 (int) (income * tax_rate) (double) (x + y) / 2</pre>

연산자의 종류_형변환연산자

❖ 명시적인 형 변환(2/2)

- 형 변환 연산자를 사용하면, 연산의 결과가 달라질 수 있으므로 주의해야 한다.

```
int num = 10;
```

```
float f;
```

```
f = 10 / 3; ○———— "int / int" 이므로 연산의 결과도 정수인 3이 된다.
```

```
f = (float) 10 / 3; ○———— "float / int" 이므로 "float / float"로 처리된다.
```

```
f = (float) 10 / (float) 3; ○———— "float / float"로 처리된다.
```

- 연산 전에 형 변환을 하는지, 연산 후에 형 변환을 하는지에 따라 연산의 결과가 달라질 수 있다.

```
num = (int)(12.5 + 10.7); ○———— 12.5+10.7의 연산 결과인 23.2를 int로 변환하므로 23이 된다.
```

```
num = (int)12.5 + (int)10.7; ○———— 먼저 형 변환을 한 다음 12+10을 연산하므로 22가 된다.
```


연산자의 종류_형변환연산자

❖ 명시적인 형 변환이 필요한 경우

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main(void)
05  {
06      int x, y;
07      double ave;
08
09      printf("Input two numbers: ");
10      scanf("%d %d", &x, &y);
11
12      ave = (x + y) / 2;
13      printf("average = %f\n", ave);
14      ave = (double) (x + y) / 2;
15      printf("average = %f\n", ave);
16  }
```

실행 결과

```
Input two numbers: 10 11
average = 10.000000
average = 10.500000
```

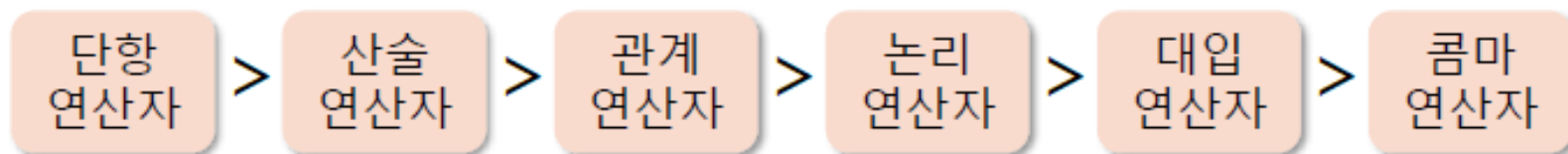
몫을 정수로 구한다.

몫을 실수로 구하려면 (x + y)를 double로 형 변환 해야 한다.

연산자의 우선순위와 결합방향_연산자의 우선순위

❖ 연산자의 우선순위(1/2)

- 수식에서 여러 연산자가 함께 사용될 때, 연산자의 수행 순서를 결정



우선순위	연산자	결합 방향
1	() [] -> .	→
2	++ -- +(부호) -(부호) sizeof ~ ! * & (type)	←
3	* / %	→
4	+ -	→
5	<< >>	→
6	< <= > >=	→
7	== !=	→

우선순위	연산자	결합 방향
8	&	→
9	^	→
10		→
11	&&	→
12		→
13	?:	→
14	= += -= *= /= %= &= = ^= <<= >>=	←
15	,(콤마)	→

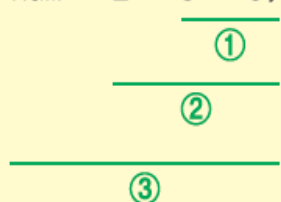
연산자의 우선순위와 결합방향_연산자의 우선순위

❖ 연산자의 우선순위(2/2)

- 기본적인 연산자의 우선순위와는 다른 순서로 연산을 수행하려면 ()를 사용한다.

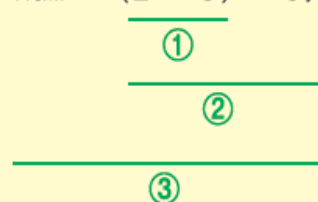
()를 사용하지 않은 경우

```
num = 2 + 3 * 5;
```



()를 사용한 경우

```
num = (2 + 3) * 5;
```



- 단항 연산자는 이항 연산자보다 우선순위가 높다.

```
++x * 2
```

(++x) * 2로 계산

연산자의 우선순위와 결합방향_연산자의 우선순위

❖ 연산자의 우선순위(2/2)

- 산술 연산자는 관계 연산자보다 우선순위가 높다.

```
x + 1 > 0
```

(x + 1) > 0으로 계산

- 대입 연산자는 우선순위가 낮다.

```
result = x << y;
```

x << y를 먼저 계산한
다음 = 연산을 수행

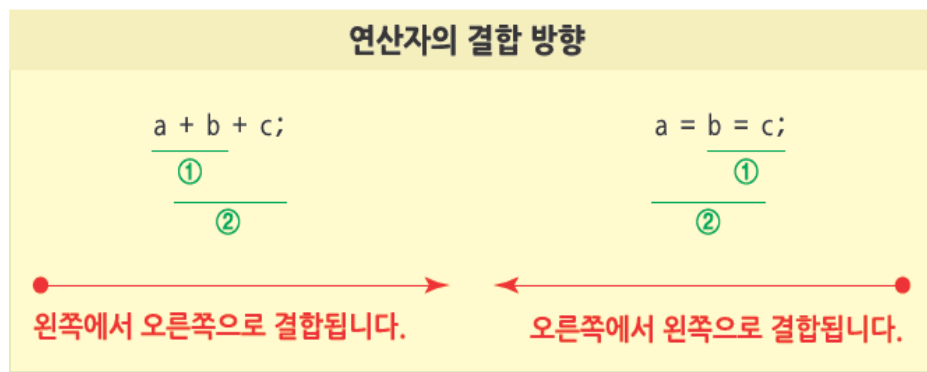
- 우선순위가 혼동될 때는 ()를 사용하는 것이 좋다.

```
result = (x + y) / 2;
```

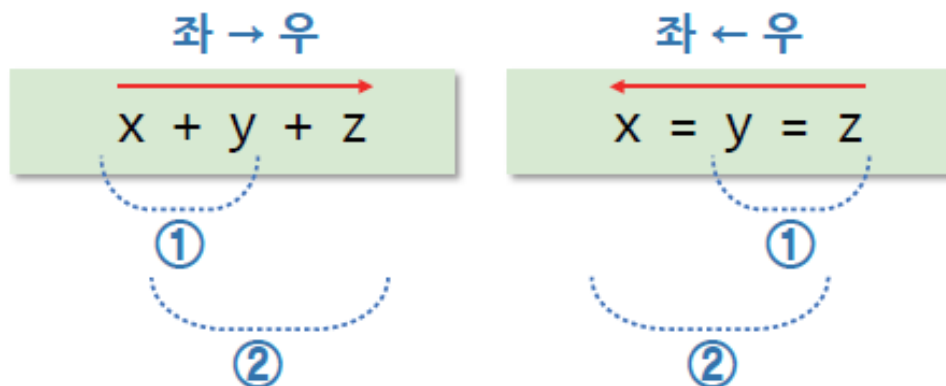

연산자의 우선순위와 결합방향_연산자의 결합방향

❖ 연산자의 결합 방향

- 같은 우선순위의 연산자에 대해서 어느 방향으로 연산을 수행할지 여부



- 대부분의 연산자는 \rightarrow 방향으로 결합된다. 예외적으로 단항 연산자와 대입 연산자는 \leftarrow 방향으로 결합된다.



연산자의 우선순위와 결합방향_연산자의 결합방향

❖ 연산자의 우선순위와 결합 방향 예(1/2)

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int x = 5, y = 1, z = 15;
06      int result;
07
08      result = ++x * 2;
09      printf("result = %d\n", result);
10
11      result = x + 1 > 0;
12      printf("result = %d\n", result);
```

(++x) * 2로 계산
(x는 6이 된다.)

(x + 1) > 0으로 계산

연산자의 우선순위와 결합방향_연산자의 결합방향

❖ 연산자의 우선순위와 결합 방향 예(2/2)

13

14

```
result = x << y;
```

15

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

16

17

```
result = (x + y) / 2;
```

18

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

19

20

```
result = x = y;
```

21

```
printf("result = %x\n", y);
```

22

```
}
```

x << y를 먼저 계산한 다음
= 연산을 수행

우선순위가 혼동되면
()로 묶어준다.

result=(x=y)로 계산

실행 결과

result = 12

result = 1

result = 12

result = 3

result = 1

학습정리

❖ 연산자의 기본 개념

- 수식 : C 프로그램에서 값을 갖는 요소
- 연산자 : +, -, *, /처럼 연산에 사용되는 기호
- 피연산자 : 연산의 대상이 되는 값

연산자의 종류	의미	연산자
단항 연산자	피연산자가 하나인 경우	+, -, ++, --, !, &, ~, sizeof
이항 연산자	피연산자가 두 개인 경우	+, -, *, /, %, =, >, <, >=, <=, !=, &&, , &, , ^, <<, >>, +=, -=, *=, /=, %=, >>=, <<=, &=, =, ^=
삼항 연산자	피연산자가 세 개인 경우	?:

학습정리

❖ 연산자의 종류

연산자의 종류	연산자
산술 연산자	+, -, *, /, %
증감 연산자	++, --
관계 연산자	>, <, >=, <=, ==, !=
논리 연산자	&&, , !
비트 연산자	&, , ^, ~, <<, >>
대입 연산자	=, +=, -=, *=, /=, %=, &=, =, ^=, >>=, <<=
조건 연산자	?:
그 밖의 연산자	, (coma 연산자), sizeof, 형 변환 연산자

❖ 연산자의 우선 순위와 결합 방향

- 우선 순위

단항 > 산술 > 관계 > 논리 > 대입 > 콤마

- 결합 방향 : 같은 우선순위의 연산자에 대해서 어느 방향으로 연산을 수행할지