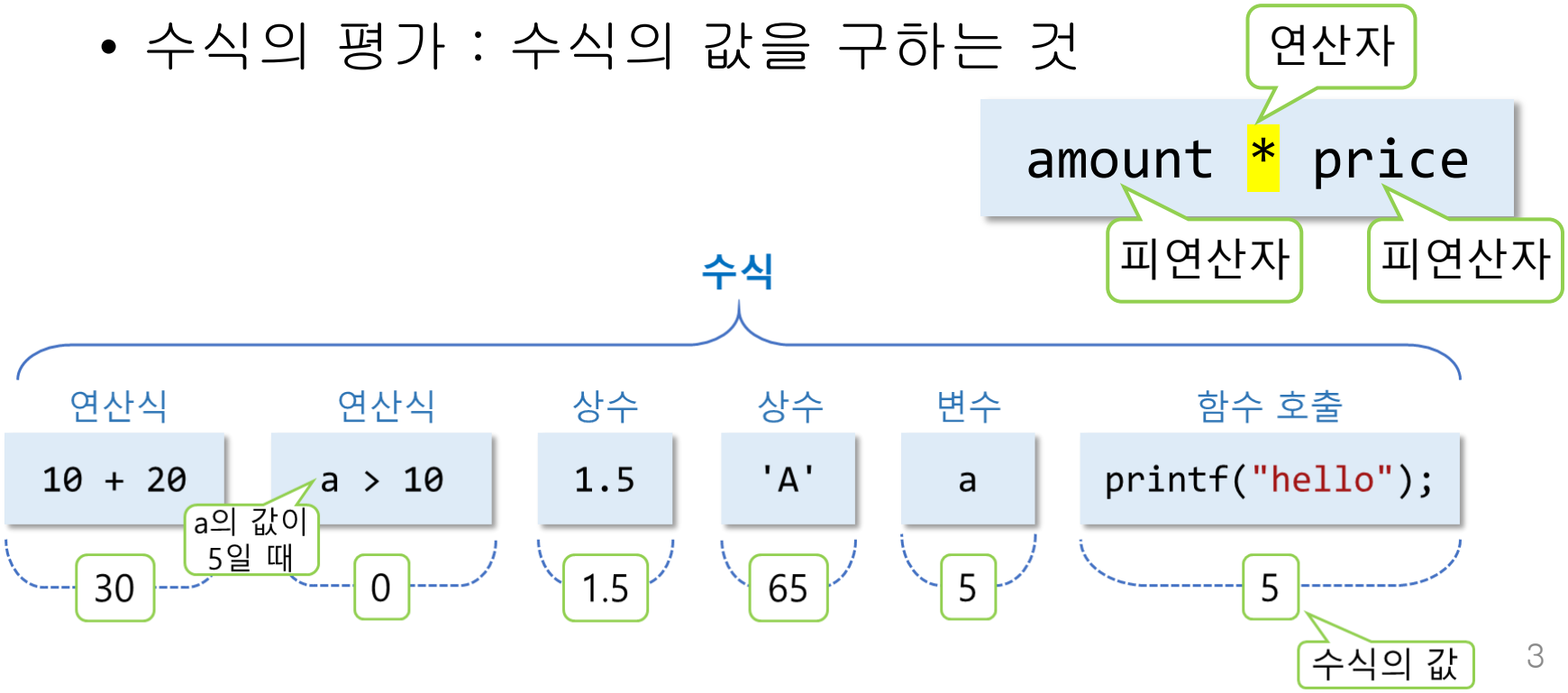
연산자

목차

연산자의 기본 개념

- 수식(expression) : 연산자와 피연산자의 조합
  - 연산자(operator) : 연산에 사용되는 기호
  - 피연산자(operand) : 연산의 대상이 되는 값
- 모든 수식에는 반드시 값이 있다.
  - 수식의 평가 : 수식의 값을 구하는 것



피연산자의 개수에 의한 분류

종류	연산자의 의미	연산자	
단항 연산자	1개의 피연산자	+a -a a++ ++a a-- --a !a &a ~a sizeof a	
이항 연산자	2개의 피연산자	산술	a+b a-b a*b a/b a%b
		대입	a=b a+=b a-=b a*=b a/=b a%=b a>>=b a<<=b a&=b a|=b a^=b
		관계	a>b a<b a>=b a<=b a==b a!=b
		논리	a&&b a||b
		비트	a&b a|b a^b a<<b a>>b
삼항 연산자	3개의 피연산자	a?b:c	

연산자의 기능에 의한 분류

연산자의 종류	연산자
산술 연산자	a+b a-b a*b a/b a%b +a -a
증감 연산자	a++ ++a a-- --a
관계 연산자	a>b a<b a>=b a<=b a==b a!=b
논리 연산자	a&&b a||b !a
비트 연산자	a&b a|b a^b ~a a<<b a>>b
대입 연산자	a=b a+=b a-=b a*=b a/=b a%=b a&=b a|=b a^=b a<<=b a>>=b
멤버 접근 연산자	*a &a a[b] a.b a->b
그 밖의 연산자	a?b:c a,b sizeof a (type)a

산술 연산자 (1/2)

산술 연산자	의미	사용 예	연산의 결과
+a	플러스(부호)	+10	10
-a	마이너스(부호)	-10	-10
a + b	더하기	10 + 3	13
a - b	빼기	10 - 3	7
a * b	곱하기	10 * 3	30
a / b	나누기	10 / 3	3
a % b	나머지 구하기	10 % 3	1

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 된다.

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

- 나머지 연산자(%) : 피연산자가 모두 정수인 경우에만 사용

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

예제 4-1 : 정수의 산술 연산

```
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  }
```

%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
```

예제 4-2 : 실수의 산술 연산

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

double형 입력시 %lf 사용

실수에는 %연산자를 사용할 수 없으므로 컴파일 에러

실행결과

```
두 개의 실수를 입력하세요 : 10 3
+10.000000 = 10.000000
-3.000000 = -3.000000
10.000000 + 3.000000 = 13.000000
10.000000 - 3.000000 = 7.000000
10.000000 * 3.000000 = 30.000000
10.000000 / 3.000000 = 3.333333
```

예제 4-3 : 나머지 연산자의 활용

```
03  int main(void)
04  {
05      int items = 0;                      // 전체 항목 수
06      int pages = 0, left = 0;
07      int items_per_page = 0;             // 한 페이지 당 항목 수
08
09      printf("항목수? ");
10      scanf("%d", &items);
11
12      printf("한 페이지 당 항목수? ");
13      scanf("%d", &items_per_page);
14
15      pages = items / items_per_page;     // 페이지 수
16      left = items % items_per_page;      // 남은 항목 수
17      printf("%d 페이지와 %d 항목\n", pages, left);
18
19      return 0;
20  }
```

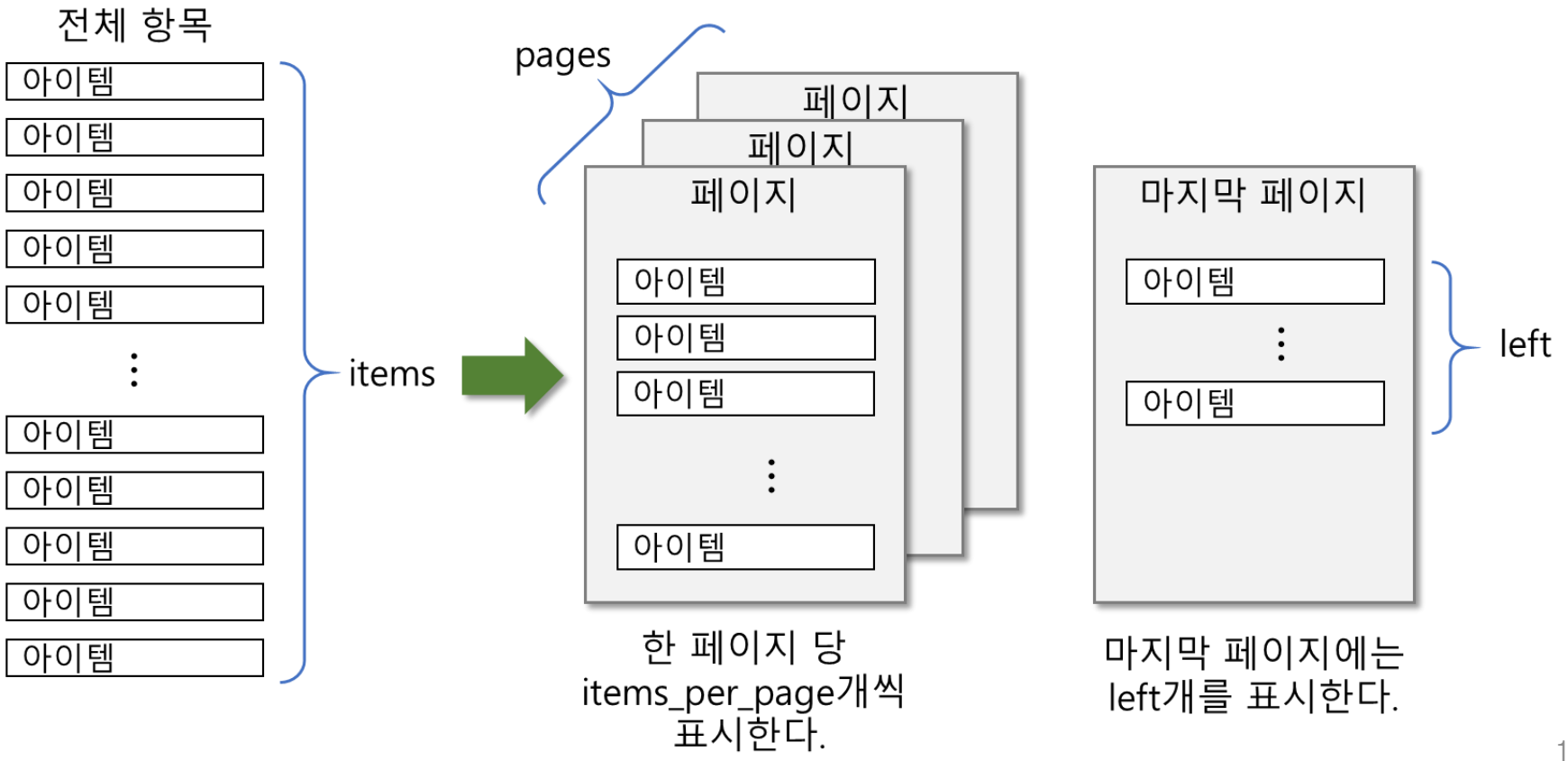
실행결과

```
항목수? 55
한 페이지 당 항목수? 20
2 페이지와 15 항목
```

55개의 항목은 20개씩 2페이지와 15개 항목이 들어 있는 마지막 페이지로 출력할 수 있다.

나머지 연산자의 활용

```
pages = items / items_per_page;
left = items % items_per_page;
```



피연산자의 형 변환 규칙

① 피연산자 중에 double형이 있으면, 나머지 피연산자를 double형으로 변환한다.

```
1.25 * 3            // double * double로 처리
1.25 + 0.5F         // double + double로 처리
```

② 피연산자 중에 float형이 있으면, 나머지 피연산자(정수형)를 float형으로 변환한다.

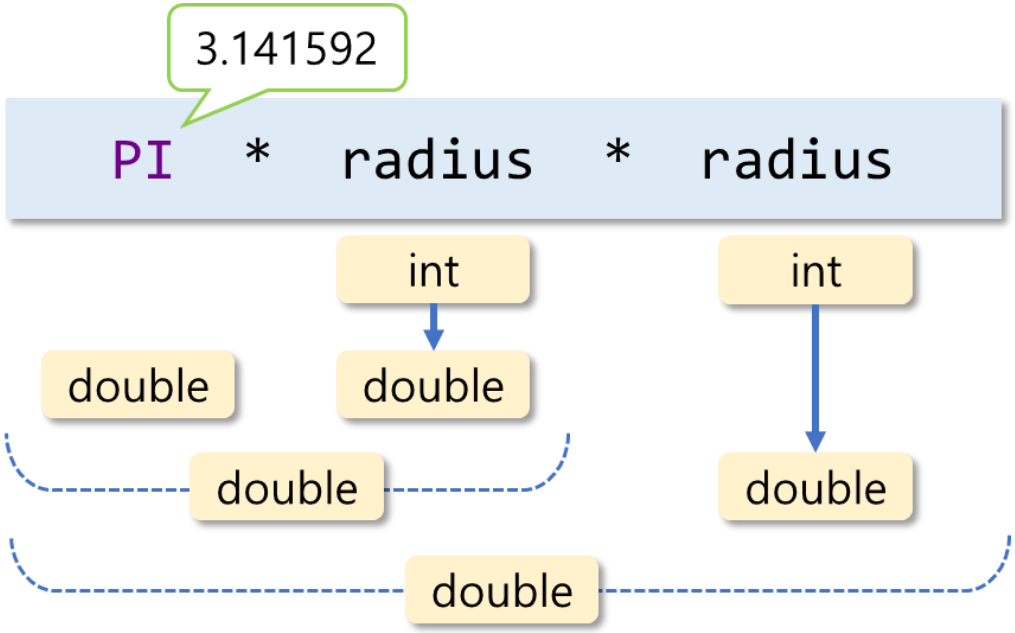
```
1.25F / 2           // float / float로 처리
```

③ 피연산자 중에 float형이 있으면, 나머지 피연산자(정수형)를 float형으로 변환한다.

```
short a = 1000, b = 2000;
a * b     // int * int로 처리
```

자동 형 변환 (1/2)

- 암시적인 형 변환
  - 정수와 실수의 혼합 연산 시 수행되는 형 변환은 자동으로 처리



자동 형 변환 (2/2)

- 단항 연산자에서는 항상 정수의 승격이 일어난다.

```
short a = 10;
printf("%d", sizeof(-a)); // 4
```

int형

- int형보다 크기가 작은 경우에는 항상 int형으로 승격된다

```
short a = 1000, b = 2000;
printf("%d", sizeof(a * b)); // 4
```

int형

예제 4-4 : 정수와 실수의 혼합 연산

```
02  #define PI 3.141592  // 매크로 상수 정의
03
04  int main(void)
05  {
06      int radius = 0;
07      double area, perimeter;
08
09      printf("반지름? ");
10      scanf("%d", &radius);
11
12      area = PI * radius * radius;
13      printf("원의 면적: %.2f\n", area);
14
15      perimeter = 2 * PI * radius;
16      printf("원의 둘레: %.2f\n", perimeter);
17
18      return 0;
19  }
```

PI가 실수이므로 radius를 double형으로 변환해서 연산

PI가 실수이므로 radius를 double형으로 변환해서 연산

실행결과

```
반지름? 5
원의 면적: 78.54
원의 둘레: 31.42
```

증감 연산자 (1/2)

- 변수의 값을 1만큼 증가시키거나 감소시키는 단항 연산자
- 증감 연산자의 의미

구분	증감 연산자	수식의 값
전위형	++a	증가된 변수 a의 값
	--a	감소된 변수 a의 값
후위형	a++	증가되기 전 변수 a의 값
	a--	감소되기 전 변수 a의 값

증감 연산자 (2/2)

- 전위형과 후위형

후위형

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

전위형

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

단일 문장인 경우 두 문장의 결과가 동일하다.

index가 1만큼 증가된다.

후위형

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

수식의 값은 증가 전 index의 값

0

전위형

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

수식의 값은 증가 후 index의 값

1

예제 4-5 : 증감 연산자의 사용 예 (1/2)

```
03  int main(void)
04  {
05      int index1 = 0, index2 = 0;
06      int current1, current2;
07      float x1 = 0.5F, x2 = 0.5F;
08      float y1, y2;
09
10      current1 = index1++;     // 증가 전의 index1
11      printf("index1 = %d, current1 = %d\n", index1, current1);
12
13      current2 = ++index2;     // 증가 후의 index2
14      printf("index2 = %d, current2 = %d\n", index2, current2);
15
16      y1 = x1++;     // 증가 전의 x1
17      printf("x1 = %.2f, y1 = %.2f\n", x1, y1);
18
```

실수형에도 증감 연산자를 사용할 수 있다.

예제 4-5 : 증감 연산자의 사용 예 (2/2)

```
19      y2 = ++x2;      // 증가 후의 x2
20      printf("x2 = %.2f, y2 = %.2f\n", x2, y2);
21
22      return 0;
23  }
```

실행결과

```
index1 = 1, current1 = 0
index2 = 1, current2 = 1
x1 = 1.50, y1 = 0.50
x2 = 1.50, y2 = 1.50
```

대입 연산자

- 연산자의 좌변에 있는 변수(l-value)에 우변의 값(r-value)을 저장

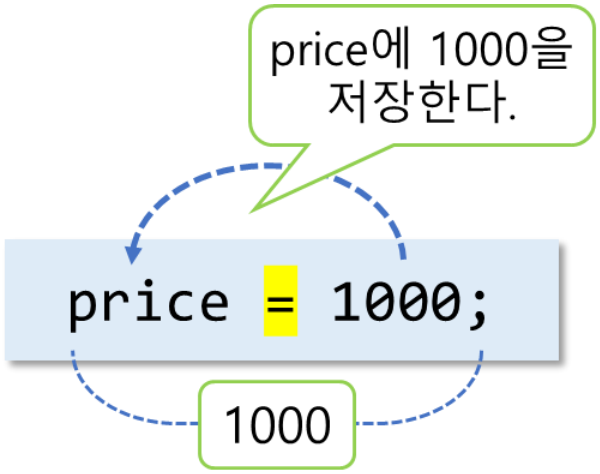
```
int price = 3000, amount = 2;
int total = 0;
int count = 0;
price = 1000;                 // price 변수에 리터럴 상수의 값을 저장한다.
total = amount * price;       // total 변수에 amount * price 연산의 결과 값을 저장한다.
count = printf("hello");      // count 변수에 printf 함수의 리턴 값을 저장한다.
```

- 대입 연산자의 좌변에는 변수만 올 수 있다.

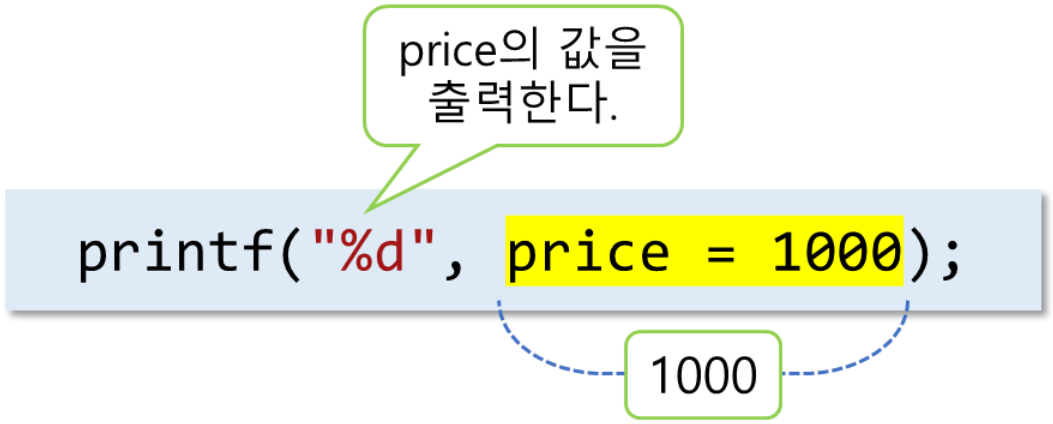
```
10 = x;              // 10은 변수가 아니므로 컴파일 에러
a + 1 = a;           // a + 1은 변수가 아니므로 컴파일 에러
printf("abc") = 10; // printf("abc")는 변수가 아니므로 컴파일 에러

const double PI = 3.141592;
PI = 3.14;           // PI는 const 변수이므로 컴파일 에러
```

대입 연산식의 값

- 대입 연산자의 좌변에 있는 변수의 값이 대입 연산식의 값



대입 연산식의 값



대입 연산식의 값을
다른 수식에 이용할 수 있다.

예제 4-6 : 대입 연산식의 값

```
03  int main(void)
04  {
05      int a = 0;
06      double b = 0;
07      int c = 0;
08
09      a = 123;          // a에 123 저장
10      printf("a = %d\n", a);
11      printf("a = %d\n", a = 456);          // a에 456 저장 후 a의 값이 수식의 값
12      printf("b = %f\n", b = a + 0.5);      // b에 a + 0.5 저장 후 b의 값이 수식의 값
13      printf("c = %d\n", c = printf("ABC\n"));
14          // printf 함수의 리턴 값을 c에 저장하고 c의 값이 수식의 값
15
16      return 0;
17  }
```

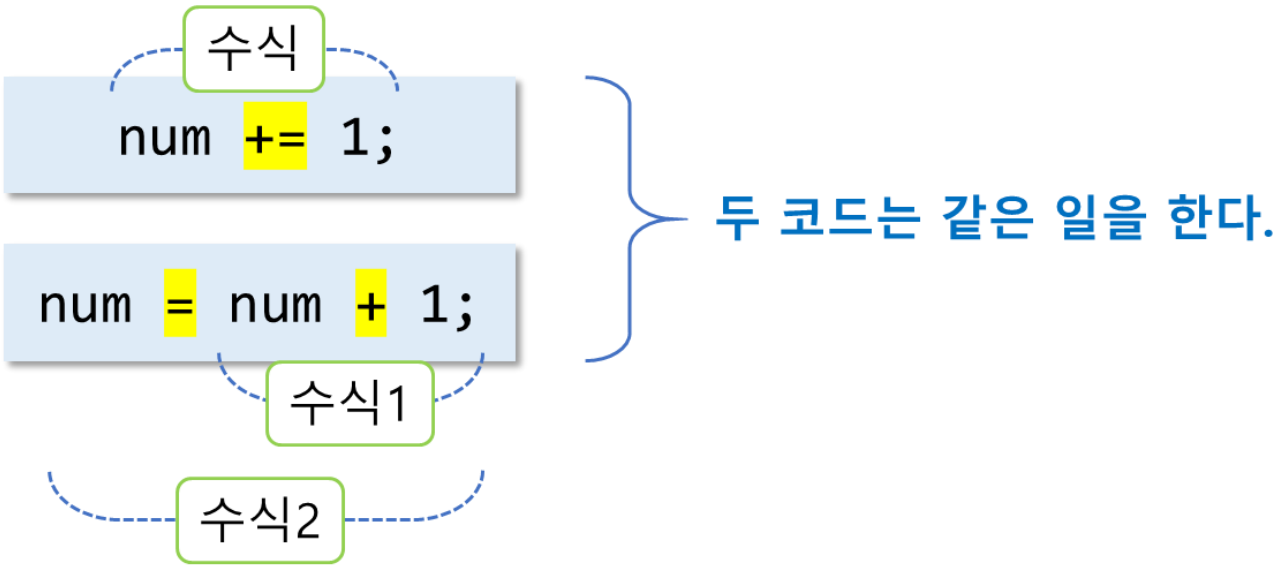
실행결과

```
a = 123
a = 456
b = 456.500000
ABC
c = 4
```

13번째 줄의 print 함수에 의해 출력된다.

복합 대입 연산자 (1/2)

- 좌변의 변수를 피연산자로 이용해서 연산을 수행하고 연산의 결과를 다시 좌변의 변수에 대입한다.



복합 대입 연산자 (2/2)

복합 대입 연산자	의미	복합 대입 연산자	의미
a += b	a = a + b	a &= b	a = a & b
a -= b	a = a - b	a |= b	a = a | b
a *= b	a = a * b	a ^= b	a = a ^ b
a /= b	a = a / b	a <<= b	a = a << b
a %= b	a = a % b	a >>= b	a = a >> b

복합 대입 연산자의 활용

[예제 4-3]

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

남은 항목의 개수를
별도의 변수에 저장

items는 바뀌지 않는다

[예제 4-7]

```
items %= items_per_page;
```

남은 항목의 개수를
items에 다시 저장

items가 바뀐다.

예제 4-7 : 복합 대입 연산자의 활용 (1/2)

```
03  int main(void)
04  {
05      int items = 0;  // 전체 항목 수
06      int pages = 0;
07      int items_per_page = 0;  // 한 페이지 당 항목 수
08
09      printf("항목수? ");
10      scanf("%d", &items);
11
12      printf("한 페이지 당 항목수? ");
13      scanf("%d", &items_per_page);
14
```

예제 4-7 : 복합 대입 연산자의 활용 (2/2)

```
15      pages = items / items_per_page;     // 페이지 수
16      items %= items_per_page;            // 남은 항목 수
17      printf("%d 페이지와 %d 항목\n", pages, items);
18
19      return 0;
20  }
```

나머지 연산의 결과를 다시 items에 저장하므로 left 변수가 필요 없다.

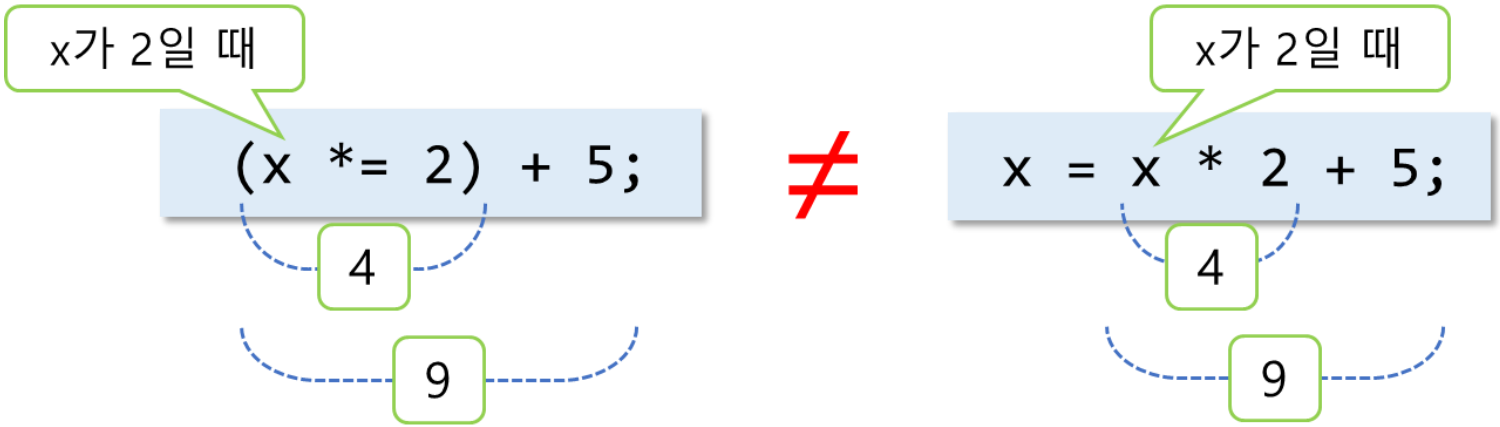
실행결과

```
항목수? 55
한 페이지 당 항목수? 20
2 페이지와 15 항목
```

55개의 항목은 20개씩 2페이지와 15개 항목이 들어 있는 마지막 페이지로 출력할 수 있다.

복합 대입 연산자의 우선순위

x *= 7로 처리된다.



연산 후 x는 4이다.

연산 후 x는 9이다.

관계 연산자

- 두 수의 값을 비교하기 위한 연산자
- 관계 연산식의 값은 항상 참 또는 거짓
  - C에서 참(true)은 1이고, 거짓(false)은 0이다.

관계 연산자	의미	a = 1, b = 2일 때 연산의 결과
a > b	a가 b보다 큰가?	0
a >= b	a가 b보다 크거나 같은가?	0
a < b	a가 b보다 작은가?	1
a <= b	a가 b보다 작거나 같은가?	1
a == b	a가 b와 같은가?	0
a != b	a가 b와 다른가?	1

예제 4-8 : 관계 연산자의 사용 예

```
03  int main(void)
04  {
05      int a = 0, b = 0;
06
07      printf("두 개의 정수? ");
08      scanf("%d %d", &a, &b);
09
10      printf("%d > %d : %d\n", a, b, a > b);
11      printf("%d < %d : %d\n", a, b, a < b);
12      printf("%d >= %d : %d\n", a, b, a >= b);
13      printf("%d <= %d : %d\n", a, b, a <= b);
14      printf("%d == %d : %d\n", a, b, a == b);
15      printf("%d != %d : %d\n", a, b, a != b);
16
17      return 0;
18  }
```

실행결과

```
두 개의 정수? 5 9
5 > 9 : 0
5 < 9 : 1
5 >= 9 : 0
5 <= 9 : 1
5 == 9 : 0
5 != 9 : 1
```

관계 연산의 결과는 0 또는 1이다.

관계 연산자 사용 시 주의 사항 (1/2)

- 두 값이 같은지 비교할 때는 =가 아니라 ==를 이용해야 한다.

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

- 실수를 비교할 때는 오차를 고려해야 한다.

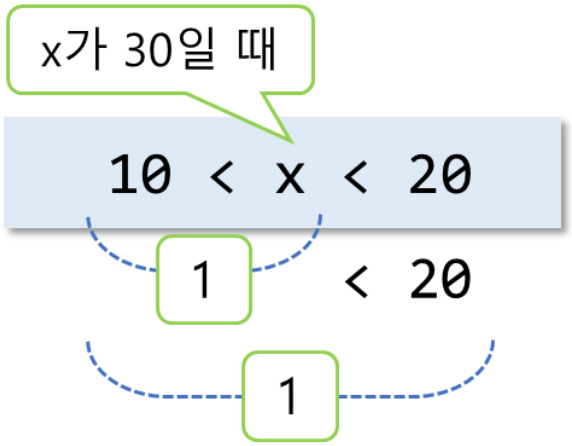
```
if (fabs(result - expected) <= FLT_EPSILON)
    printf("두 수가 같습니다.\n");
```

관계 연산자 사용 시 주의 사항 (2/2)

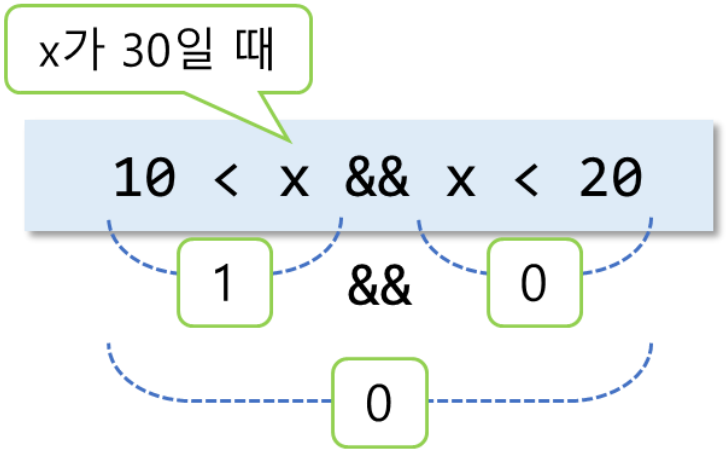
- 10 < x < 20과 같은 수식을 사용해서는 안된다.

x가 10과 20 사이의 값인지 검사한다.

잘못된 수식



올바른 수식



참, 거짓의 판단

- C에서는 수식이 참인지 거짓인지 판단할 때, 0이 아닌 값은 참으로 간주한다.

y가 0이 아닌가?

```
if (y != 0)
    result = x / y;
```

=

y가 참인가?

```
if (y)
    result = x / y;
```

y가 0인가?

```
if (y == 0)
    printf("에러");
```

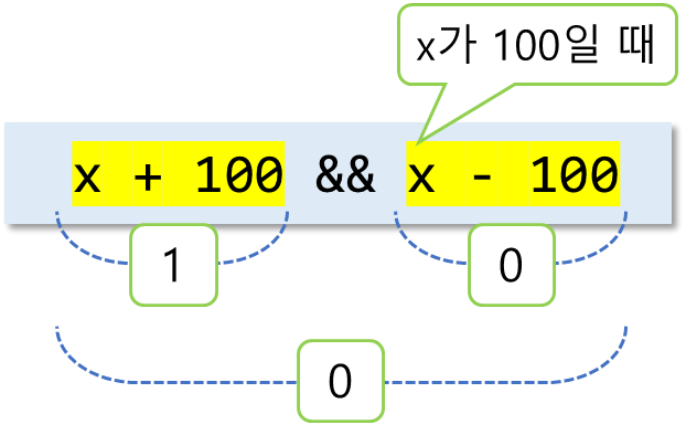
=

y가 거짓인가?

```
if (!y)
    printf("에러");
```

논리 연산자

- 참과 거짓을 이용한 논리 연산 기능
- 연산의 결과가 항상 참(1) 또는 거짓(0)
- 피연산자가 0이 아니면 참으로 간주한다.



논리 연산자	부울 대수	의미
a && b	논리 AND	a와 b가 둘 다 0이 아니면 1 a와 b중 하나만 0이면 0
a || b	논리 OR	a와 b중 하나만 0이 아니면 1 a와 b가 둘 다 0이면 0
! a	논리 NOT	a가 0이면 1, a가 0이 아니면 0

논리 연산의 결과

a	b	a && b	a || b	!a	!b
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

```
month >= 6 && month <= 8
```

month가 성수기(6~8월)에 해당하는지 검사한다.

```
month < 6 || month > 8
```

```
! (month >= 6 && month <= 8)
```

month가 성수기(6~8월)가 아닌지 검사한다.

month < 0이거나 month > 12인 경우는 없다고 가정

예제 4-9 : 논리 연산자의 사용 예

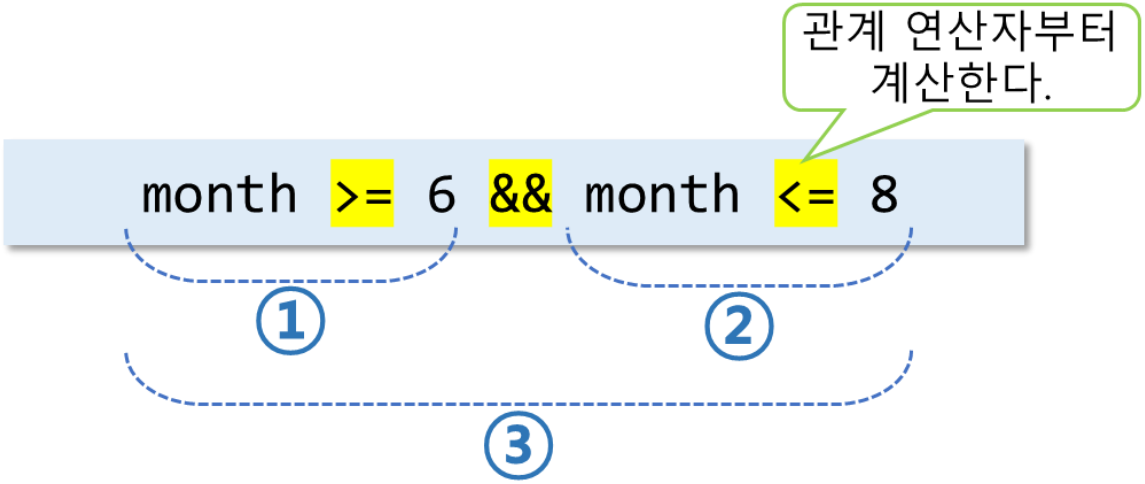
```
03  int main(void)
04  {
05      int month;
06
07      printf("몇 월? ");
08      scanf("%d", &month);
09
10      if (month >= 6 && month <= 8)       // 논리 AND
11          printf("성수기 요금 적용\n");
12
13      if (month < 6 || month > 8)         // 논리OR
14          printf("일반 요금 적용\n");
15
16      //if (!(month >= 6 && month <= 8))  // 논리NOT
17      //    printf("일반 요금 적용\n");
18
19      return 0;
20  }
```

실행결과

```
몇 월? 6
성수기 요금 적용
```

논리 연산자 사용시 주의 사항 (1/2)

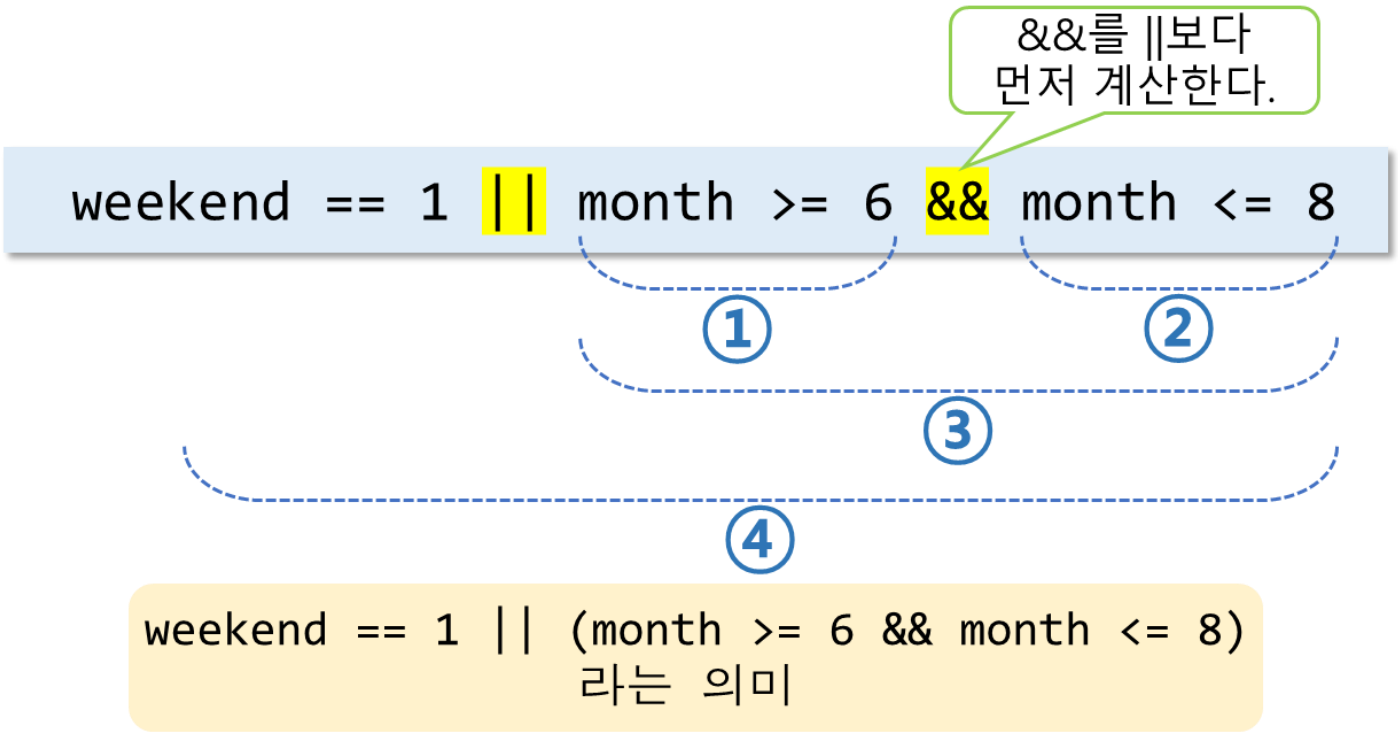
- 관계 연산자과 함께 사용하면 관계 연산자부터 수행된다.




```
(month >= 6) && (month <= 8)
```
라는 의미

논리 연산자 사용시 주의 사항 (2/2)

- && 연산자가 || 연산자보다 우선순위가 높다.



논리 연산자의 단축 계산

- 'expr1 && expr2'에서 expr1이 거짓이면 expr2는 평가되지 않는다. 즉, expr2는 expr1이 참일 때만 평가된다.

```
a > 0 && printf("abc") == 3
```

a <= 0이면 printf("abc") == 3은 평가되지 않는다.

- 'expr1 || expr2'에서 expr1이 참이면 expr2는 평가되지 않는다. 즉, expr2는 expr1이 거짓일 때만 평가된다.

```
a > 0 || --b < 0
```

a > 0이면 –b < 0은 평가되지 않는다.

비트 연산자

- 피연산자의 각 비트 단위로 수행되는 연산자

구분	비트 연산자	의미
비트 논리 연산자	a & b	a와 b의 각 비트 단위로 논리 AND 연산
	a | b	a와 b의 각 비트 단위로 논리 OR 연산
	a ^ b	a와 b의 각 비트 단위로 논리 XOR 연산
	~a	a의 각 비트 단위로 논리 NOT 연산
비트 이동 연산자	a << b	a의 각 비트를 b개만큼 왼쪽으로 이동
	a >> b	a의 각 비트를 b개만큼 오른쪽으로 이동

비트 논리 연산자

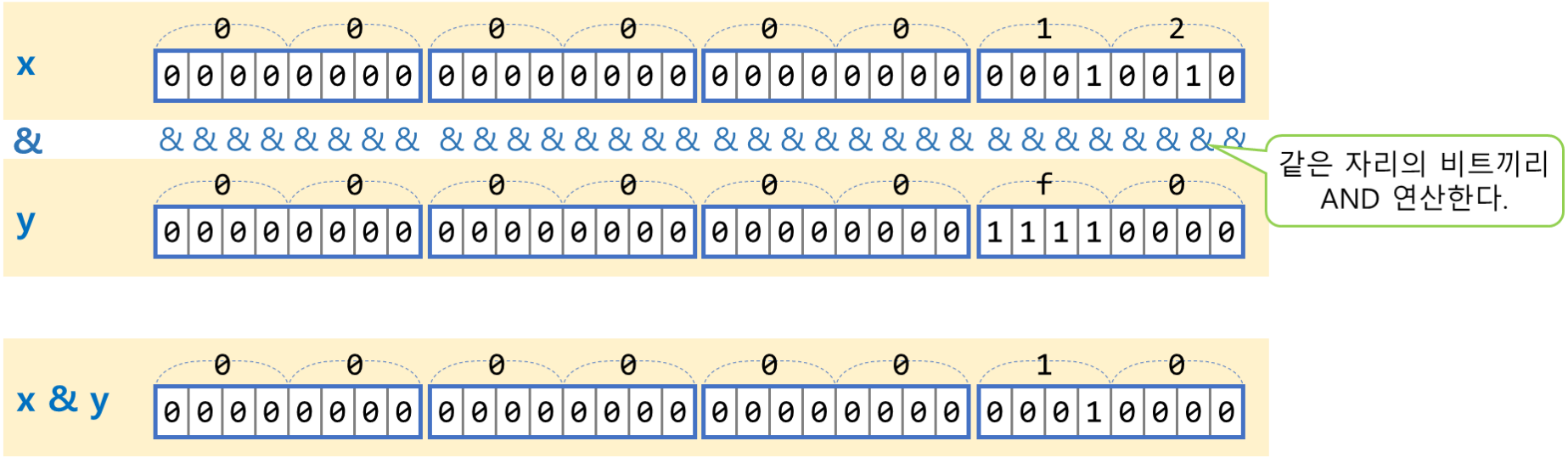
- 논리 연산자가 피연산자의 전체 값으로 연산을 수행하는 반면에, 비트 논리 연산자는 각 비트에 대하여 연산을 수행한다.
- &, |, ^ 연산자는 피연산자의 데이터형이 일치하지 않으면 형 변환을 수행하여 같은 형으로 만든 후, 피연산자의 각 비트에 대하여 비트 논리 연산을 수행한다.

a의 비트	b의 비트	a & b의 비트	a | b의 비트	a ^ b의 비트	~a의 비트
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

비트 AND 연산 (1/2)

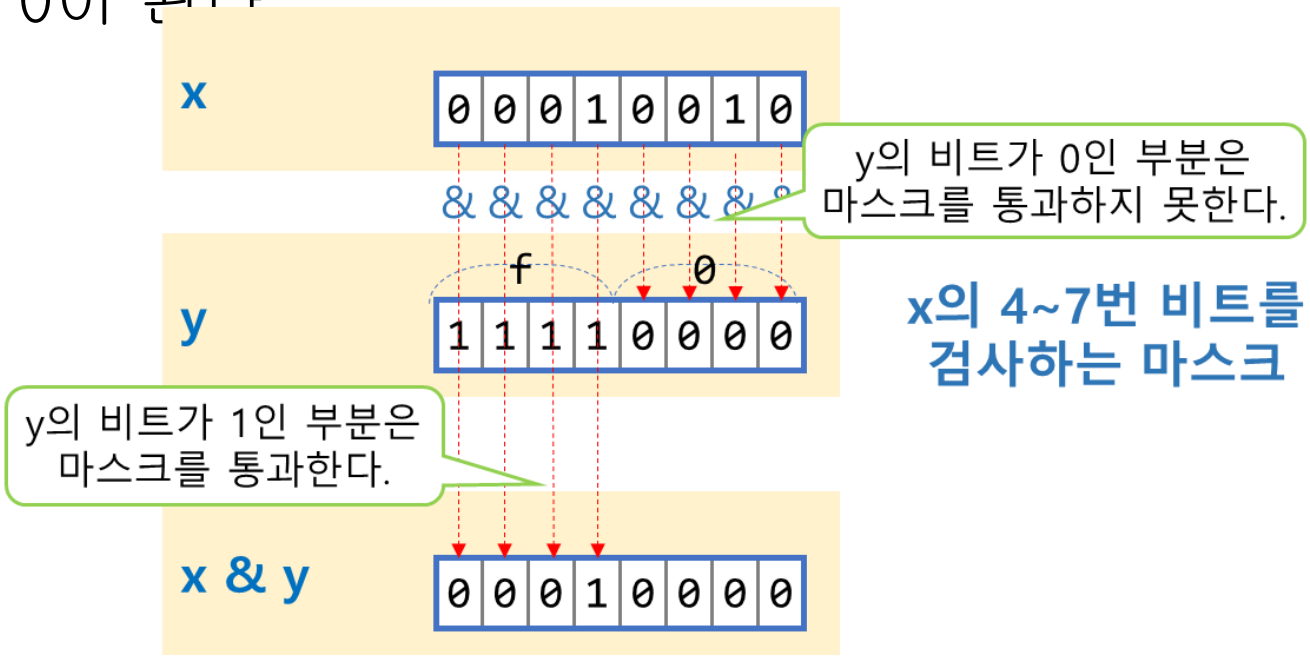
```
unsigned short x = 0x12;
unsigned short y = 0xf0;
x & y
```

수식의 값은 0x00000010



비트 AND 연산 (2/2)

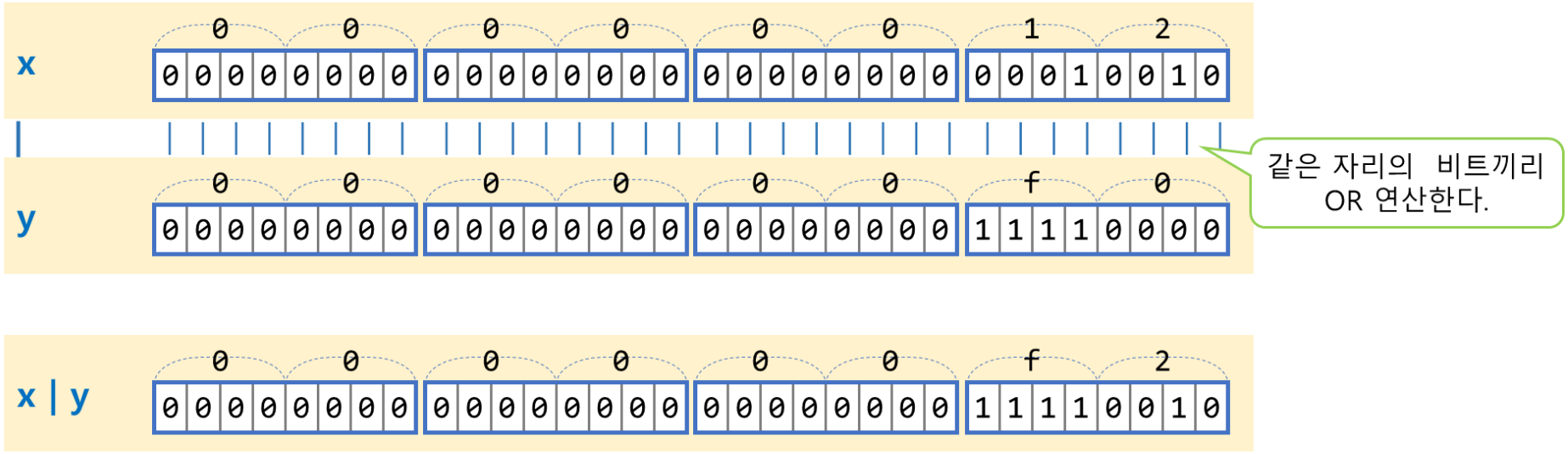
- 비트마스크(bitmask) 또는 마스크(mask)
  - 비트 논리 연산에서 이용되어 특정 비트 값을 조작하기 위한 목적의 데이터
  - x와 y를 비트 AND 연산하면 x의 값 중에 y의 비트가 1인 부분의 값만 유지되고, 나머지 비트는 모두 0이 된다



비트 OR 연산 (1/2)

```
unsigned short x = 0x12;
unsigned short y = 0xf0;
x | y
```

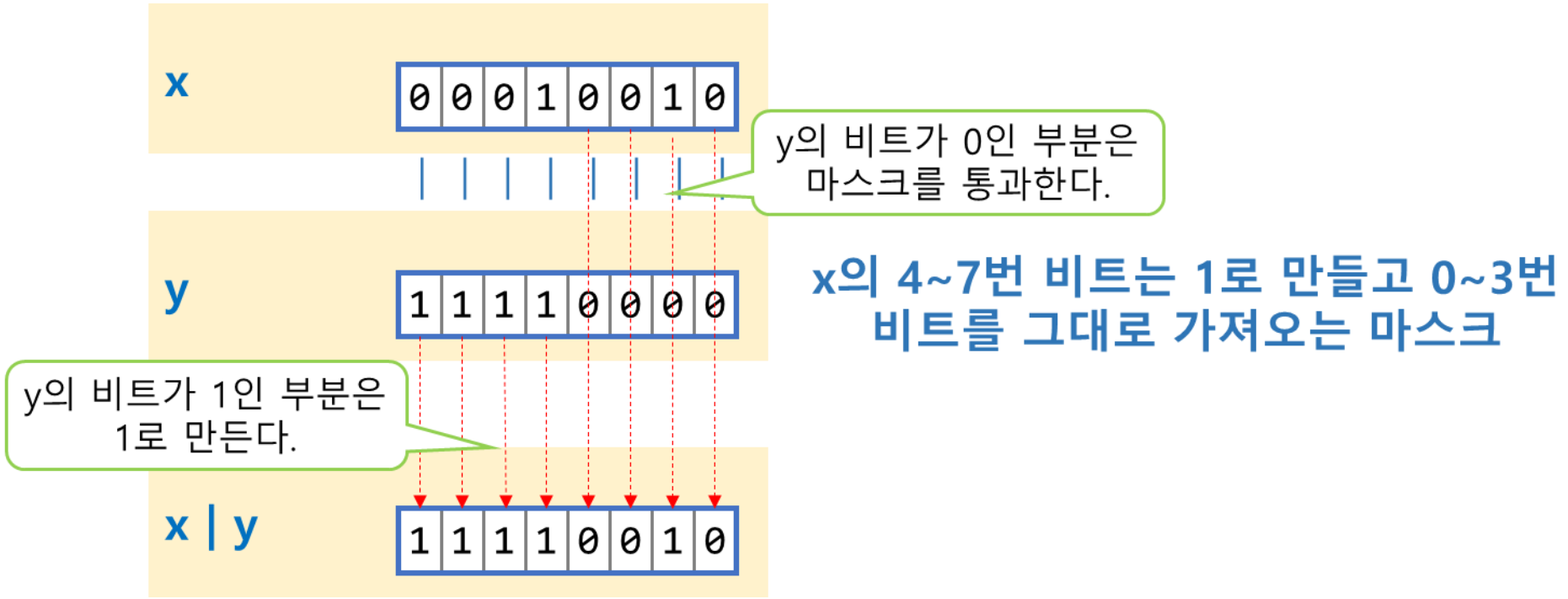
수식의 값은
0x000000f2



같은 자리의 비트끼리
OR 연산한다.

비트 OR 연산 (2/2)

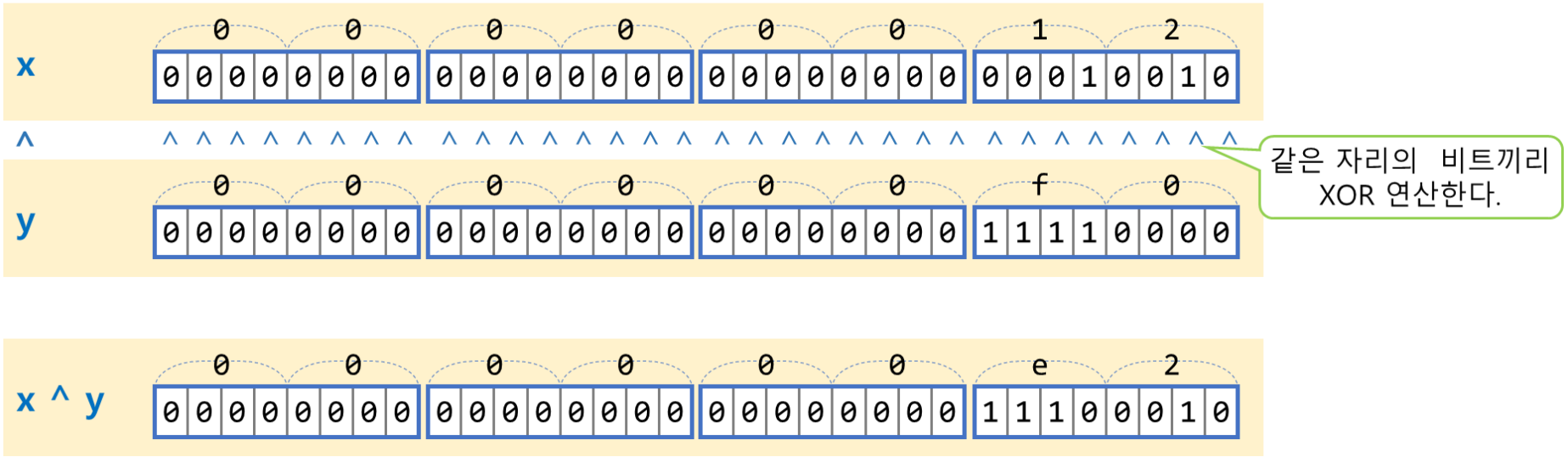
- x와 y를 비트 OR 연산하면 x의 값 중에 y의 비트가 0인 부분의 값만 유지되고, 나머지 비트는 모두 1이 된다.



비트 XOR 연산 (1/2)

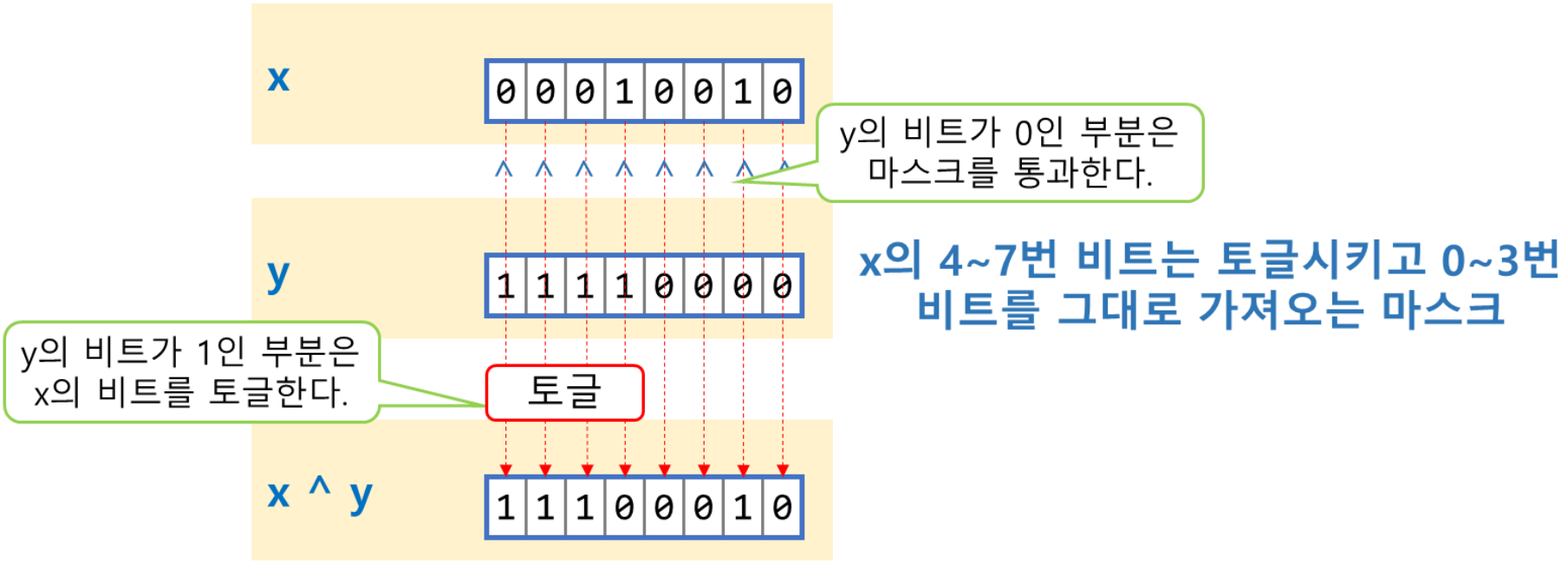
```
unsigned short x = 0x12;
unsigned short y = 0xf0;
x ^ y
```

수식의 값은 0x000000e2



비트 XOR 연산 (2/2)

- x와 y를 비트 XOR 연산하면 x의 값 중에 y의 비트가 1인 부분은 토글되고, y의 비트가 0인 부분의 값은 유지된다.

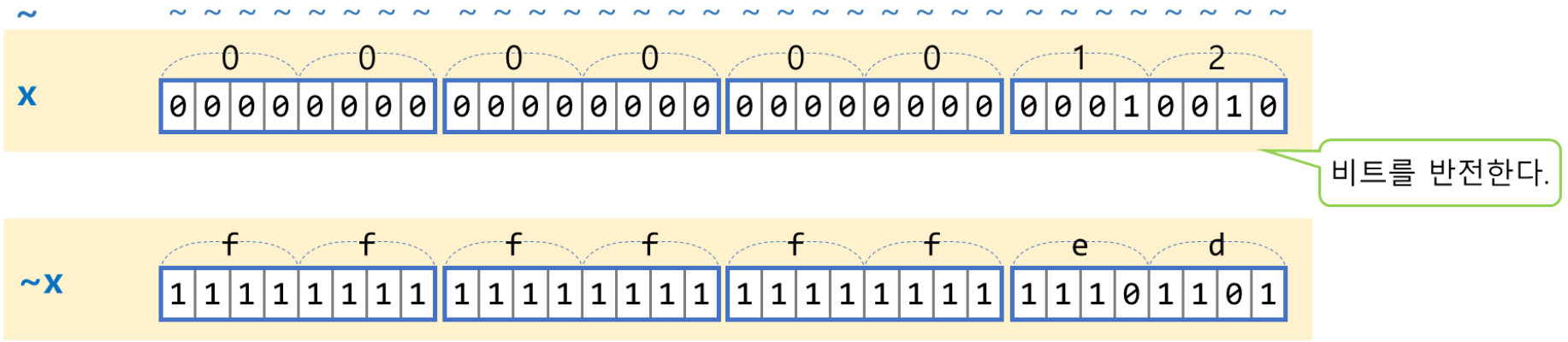


비트 NOT 연산

- 0은 1로, 1은 0으로 반전한다.

```
unsigned short x = 0x12;

~x
```

수식의 값은 0xffffffed



예제 4-10 : 비트 논리 연산자의 사용 예

```
03  int main(void)
04  {
05
06      unsigned short x = 0x12;
07      unsigned short y = 0xF0;
08
09      printf("%08x & %08x = %08x\n", x, y, x & y);    // 비트 AND
10      printf("%08x | %08x = %08x\n", x, y, x | y);    // 비트 OR
11      printf("%08x ^ %08x = %08x\n", x, y, x ^ y);    // 비트 XOR
12      printf("~%08x = %08x\n", x, ~x);                // 비트 NOT
13
14      return 0;
15  }
```

실행결과

```
00000012 & 000000f0 = 00000010
00000012 | 000000f0 = 000000f2
00000012 ^ 000000f0 = 000000e2
~00000012 = ffffffed
```

비트 이동 연산자 (1/2)

- 좌변에 있는 피연산자의 비트들을 우변의 피연산자가 지정하는 만큼 왼쪽으로 또는 오른쪽으로 이동(shift)한다.

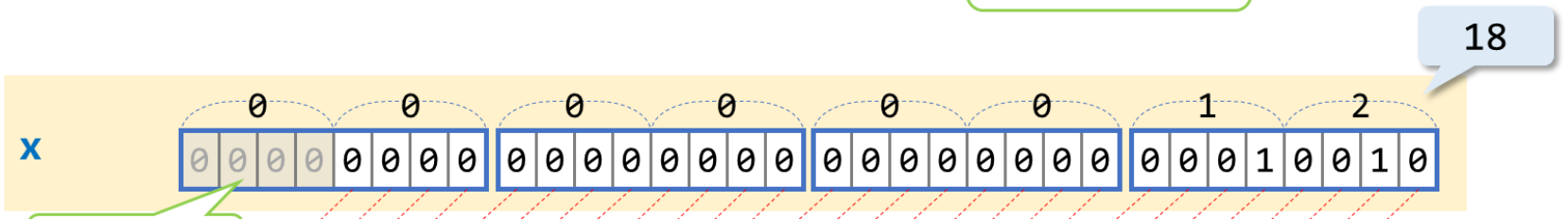
비트 이동 연산자 (2/2)

- 비트 왼쪽 이동(<<) 연산자
  - 비트들을 왼쪽으로 이동한다.
  - 왼쪽으로 밀려난 비트는 사라지고 오른쪽 빈 자리에 0을 채운다.
  - n 비트 왼쪽 이동은 2^n을 곱하는 것과 같다.
- 비트 오른쪽 이동(>>) 연산자
  - 비트들을 오른쪽으로 이동
  - 오른쪽으로 밀려난 비트는 사라지고 왼쪽 빈 자리에 부호 비트를 채운다.
  - 연산자의 좌변이 부호 없는 정수형이면 왼쪽 빈 자리를 0으로 채운다.
  - n 비트 오른쪽 이동은 2^n 으로 나누는 것과 같다.

비트 왼쪽 이동

```
int x = 0x12;
int y = x << 4;
```

수식의 값은
0x00000120



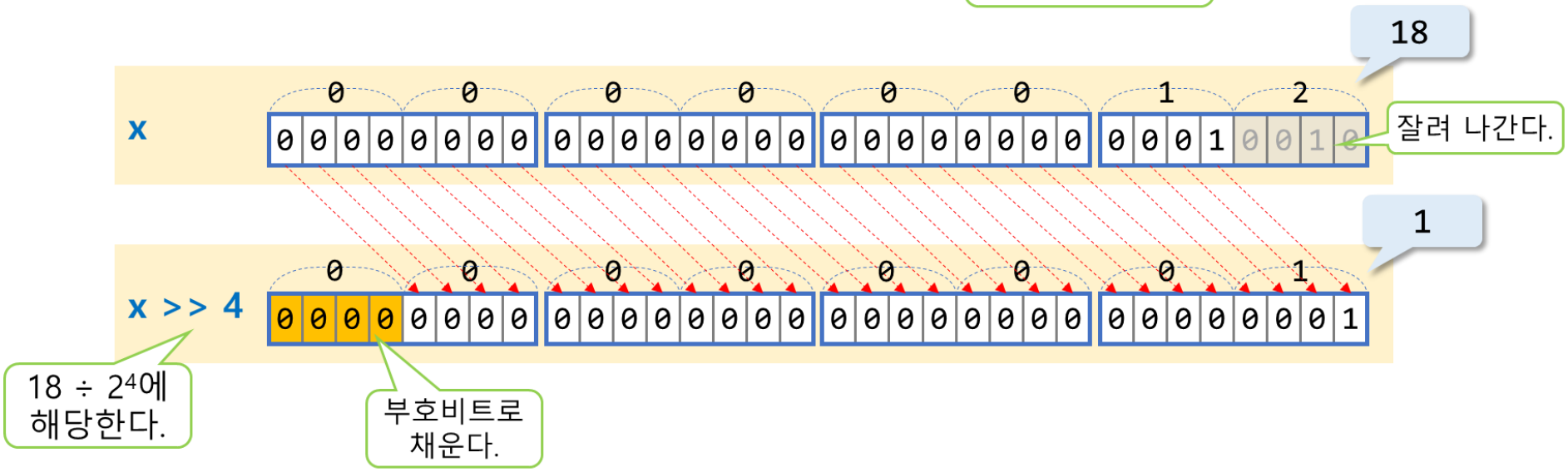
18 × 2^4에
해당한다.

0으로 채운다.

비트 오른쪽 이동

```
int x = 0x12;
int z = x >> 4;
```

수식의 값은 0x00000001



예제 4-11 : 비트 이동 연산자의 사용 예

```
03  int main(void)
04  {
05      int x = 0x00000012;
06      int y = x << 4;
07      int z = x >> 4;
08
09      printf("x = %#08x, %d\n", x, x);     // 0x00000012, 18
10      printf("y = %#08x, %d\n", y, y);     // 0x00000120, 288 (18 * 16)
11      printf("z = %#08x, %d\n", z, z);     // 0x00000001, 1 (18 / 16)
12
13      return 0;
14  }
```

실행결과

```
x = 0x000012, 18
y = 0x000120, 288
z = 0x000001, 1
```

18×2^4

$18 \div 2^4$

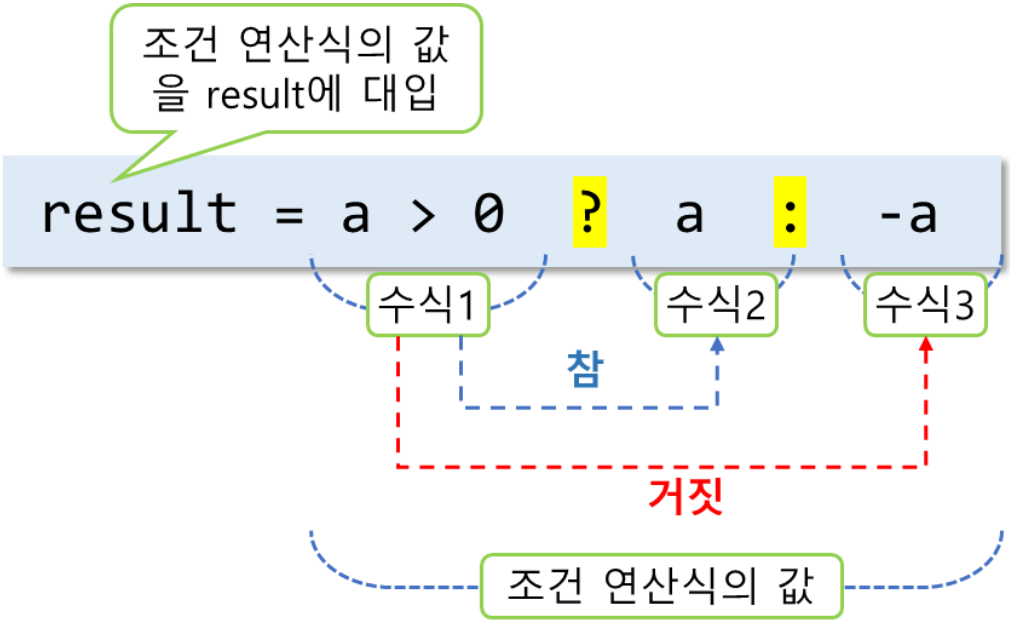
조건 연산자

- 피연산자가 3개인 삼항 연산자

형식 수식1 ? 수식2 : 수식3

사용예

```
a > 0 ? a : -a
a > b ? a : b
num % 2 ? printf("odd"):printf("even")
```



예제 4-12 : 조건 연산자의 사용 예 (1/2)

```
03  int main(void)
04  {
05      int a, b;
06      int result, max;
07
08      printf("2개의 정수? ");
09      scanf("%d %d", &a, &b);
10
11      printf("%d는 ", a);
12      a % 2 != 0 ? printf("홀수") : printf("짝수");
13      printf("입니다.\n");
14
15      result = a > 0 ? a : -a;
16      printf("a의 절대값: %d\n", result);
17
```

예제 4-12 : 조건 연산자의 사용 예 (2/2)

```
18      result = b > 0 ? b : -b;
19      printf("b의 절대값: %d\n", result);
20
21      max = a > b ? a : b;
22      printf("a, b 중 큰 값: %d\n", max);
23
24      return 0;
25  }
```

실행결과

```
2개의 정수? 3 -5
3는 홀수입니다
a의 절대값: 3
b의 절대값: 5
a, b 중 큰 값: 3
```

예제 4-13 : 게시판 프로그램의 페이지 수 구하기 (1/2)

```
03  int main(void)
04  {
05      int items = 0;  // 전체 항목 수
06      int pages = 0, left = 0;
07      int items_per_page = 0;  // 한 페이지 당 항목 수
08
09      printf("항목수? ");
10     scanf("%d", &items);
11
12      printf("한 페이지 당 항목수? ");
13      scanf("%d", &items_per_page);
14
15      pages = items / items_per_page;     // 페이지 수
16      left = items % items_per_page;      // 남은 항목 수
```

예제 4-13 : 게시판 프로그램의 페이지 수 구하기 (2/2)

```
18      printf("필요한 총 페이지수: %d\n", pages);
19      printf("마지막 페이지의 항목수: %d\n",
20          left > 0 ? left : items_per_page);
21
22      return 0;
23  }
```

left 〉0이면 left를 출력하고,
그렇지 않으면 items_per_page를 출력한다.

실행결과

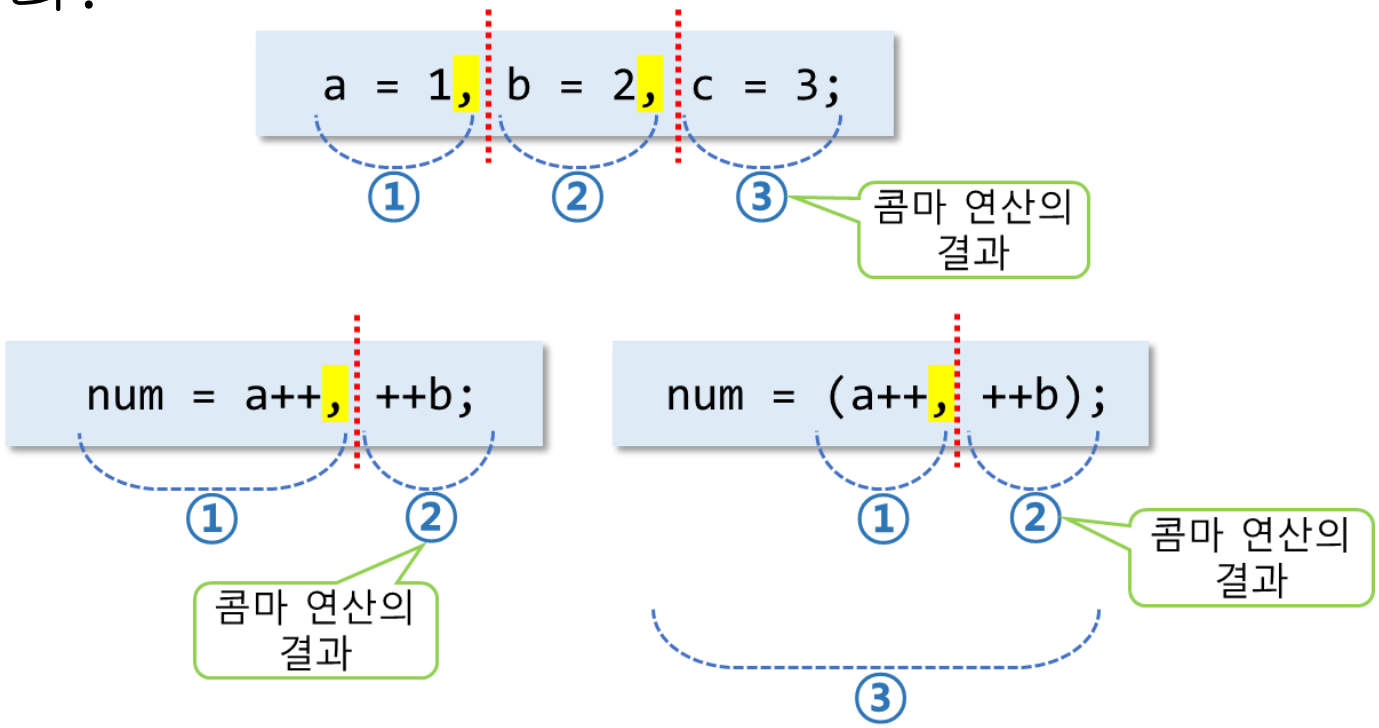
```
항목수? 55
한 페이지 당 항목수? 20
필요한 총 페이지수: 3
마지막 페이지의 항목수: 15
```

실행결과

```
항목수? 60
한 페이지 당 항목수? 20
필요한 총 페이지수: 3
마지막 페이지의 항목수: 20
```

콤마 연산자

- 수식의 값을 왼쪽부터 계산해서 마지막으로 계산한 오른쪽 수식의 값이 연산의 결과가 된다.
- 여러 수식을 한 문장으로 연결할 때 주로 사용된다.



암시적인 형 변환 (1/2)

- 서로 다른 데이터형의 값을 혼합 연산할 때 암시적인 형 변환이 일어난다.

수식	설명	형 변환	예
double op *type* *type* op double	*type*은 실수형 또는 정수형	double op double	`1.23 * 4` `100 > 99.0`
float op *type* *type* op float	*type*은 정수형	float op float	`3.0F + 'A'` `12 < 10.0F`

type op *type*	*type*은 char 또는 short	int op int	`short a = 100, b = 200;`
			`a * b`
			`a & b`

type1 op *type2*
type2 op *type1*
	type1, *type2*가 정수형이고,
	*type1*의 크기가 *type2*보다 클 때
		type1 op *type1*	`int a = 100;`
			`long b = 200L;`
			`a - b`
			`a == b`

암시적인 형 변환 (2/2)

- 정수의 승격
  - char형이나 short형의 값이 사용될 때마다 자동으로 int형으로 형 변환된다.
  - 정수의 승격도 암시적인 형 변환이다.
- 대입 연산이나 변수의 초기화에서도 암시적인 형 변환이 수행된다.
  - 대입 연산자의 좌변과 대입 연산자의 우변의 데이터형이 다르면, 좌변의 데이터형에 맞추도록 형 변환이 일어난다.
  - 이 때 값이 손실되면, 컴파일 경고가 발생한다.

```
int data = 1234.5;              // r-value 값이 손실되므로 컴파일 경고
```

예제 4-14 : 암시적인 형 변환

```
03  int main(void)
04  {
05      short a, b, c;
06
07      printf("정수 2개? ");
08      scanf("%hd %hd", &a, &b); // short형 변수 입력시 %hd 사용
09
10      printf("%d * %d = %d\n", a, b, a * b);       // a * b는 int * int로 처리
11      printf("sizeof(a * b) = %d\n", sizeof(a * b));// 4
12
13      c = a * b;      // int형인 (a * b)를 short형으로 변환해서 대입
14      printf("c = %d\n", c);
15      printf("sizeof(c) = %d\n", sizeof(c));       // 2
16
17      return 0;
18  }
```

실행결과

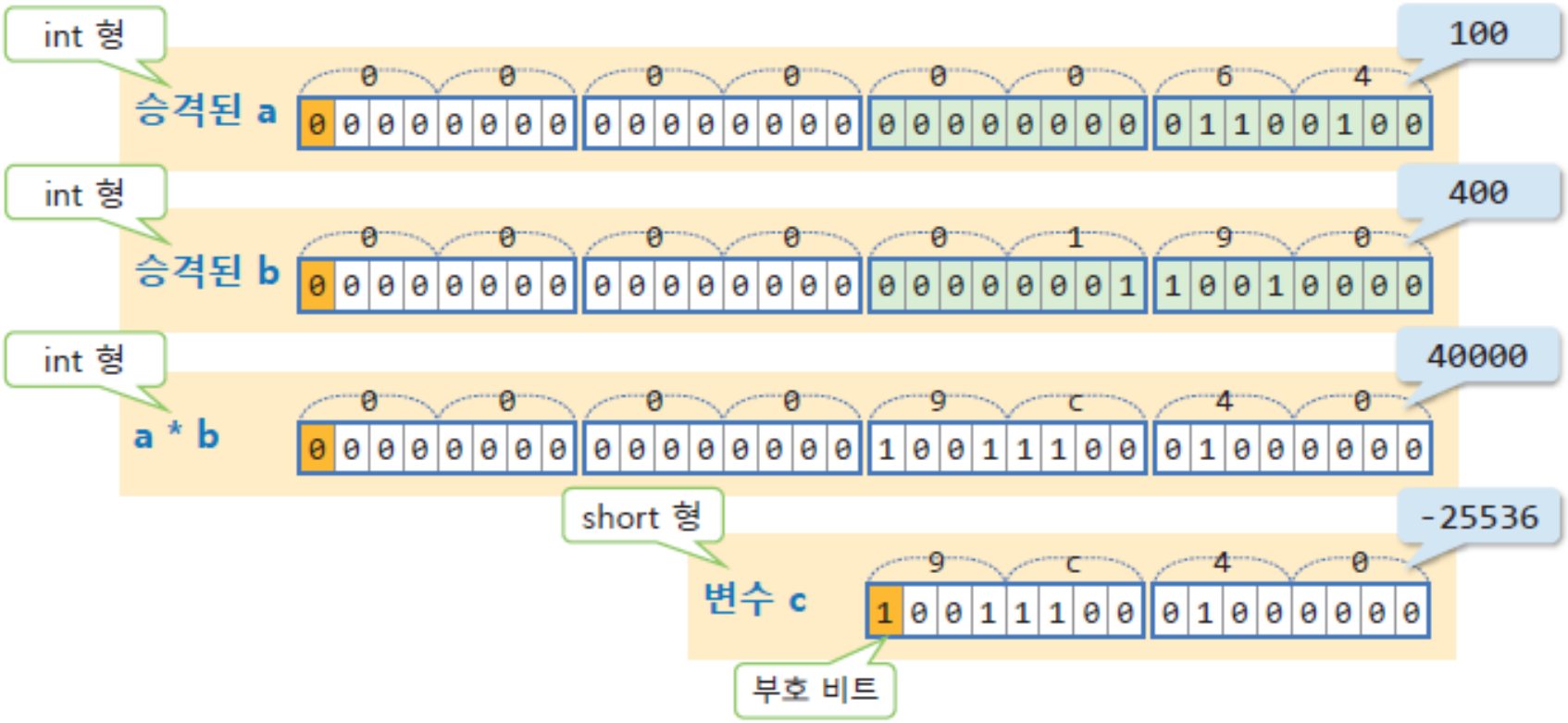
```
정수 2개? 100 400
100 * 400 = 40000
sizeof(a * b) = 4
c = -25536
sizeof(c) = 2
```

오버플로우

정수의 승격과 오버플로우

```
short a = 100;
short b = 400;
short c = a * b;
```

a * b를 c에 저장할 때 오버플로우 발생



형 변환 연산자 (1/2)

- 명시적인 형 변환

형식	(데이터형) 수식

사용예	`(double) 0`
	`(int) price * rate`
	`(double) 10 / (double) 20`

- / 연산의 결과를 실수로 구하려면 명시적인 형 변환이 필요하다.

```
average = (double)(a + b) / 2; // double / int이므로 double / double로 처리
```

예제 4-15 : 세 수의 평균 구하기

```
03  int main(void)
04  {
05      int a, b, c;
06      double average;
07
08      printf("정수 3개? ");
09      scanf("%d %d %d", &a, &b, &c);
10
11      average = (double)(a + b + c) / 3;    // 명시적인 형 변환 필요
12      printf("평균: %f\n", average);
13
14      return 0;
15  }
```

/ 연산자의 피연산자 중 하나를 실수형으로 형 변환 한다.

실행결과

```
정수 3개? 14 54 33
평균: 33.666667
```

몫을 소수점 이하까지 구할 수 있다.

형 변환 연산자 (2/2)

- 형 변환 연산자를 사용할 때는 형 변환이 언제 수행되는지에 따라 연산의 결과가 달라진다.

```
int result1, result2;
result1 = (int)(1.5 + 3.8);
result2 = (int)1.5 + (int)3.8;
```

(1.5 + 3.8)을 먼저 계산한 다음에 (int) 5.3을 수행하므로 5가 된다.

(int)1.5와 (int)3.8을 먼저 수행한 다음 1 + 3을 계산하므로 4가 된다.

연산자의 우선순위

단항 연산자 > 산술 연산자 > 관계 연산자 > 논리 연산자 > 대입 연산자 > 콤마 연산자

- '변수 = 수식'의 형태인 경우 항상 수식의 값을 먼저 계산한다.

```
result = a + b * c;          // =의 우변에 있는 a + b * c를 먼저 계산한다.
```

- 관계 연산자는 논리 연산자보다 우선순위가 높

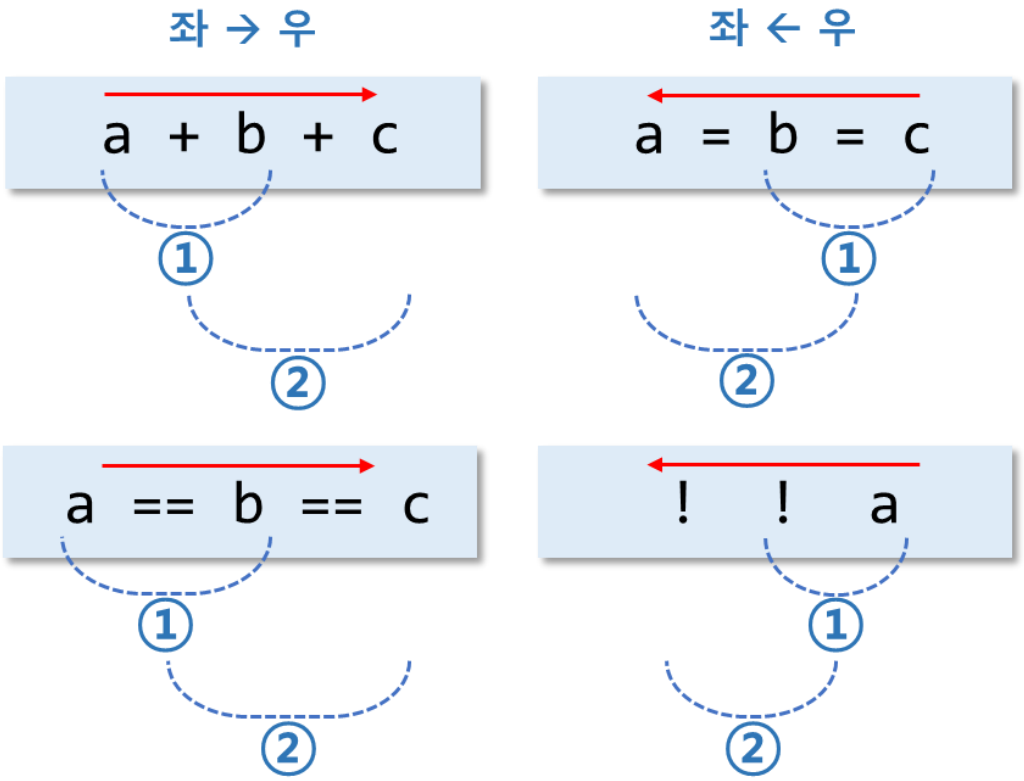
```
a > b && a < c               // (a > b) && (a < c)의 의미
```

- 산술 연산자는 관계 연산자나 논리 연산자보다

```
a + 100 == b * 3             // (a + 100) == (b * 3)의 의미
```

연산자의 결합 규칙

- 대부분의 연산자는 좌 → 우 방향으로 결합하고 단항 연산자와 대입 연산자는 우 → 좌 방향으로 결합한다.



예제 4-16 : 연산자의 우선순위와 결합 규칙

```
03  int main(void)
04  {
05      int a = 10, b = 20, c = 30;
06      int result;
07
08      result = a + b * c;                 // a + (b * c)
09      printf("result = %d\n", result);
10
11      result = (a + b) * c;               // (a + b) * c
12      printf("result = %d\n", result);
13
14      result = a < b && c < 0;            // (a < b) && (c < 0)
15      printf("result = %d\n", result);
16
17      return 0;
18  }
```

실행결과

```
result = 610
result = 900
result = 0
```