



제 5 장

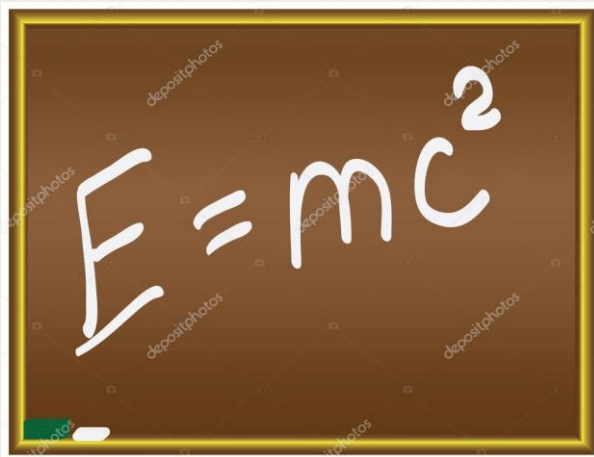
수식과 연산자





수식-연산자-피연산자

- **수식**(expression)
 - ▣ **연산자**(operator)와 **피연산자**(operand)로 이루어진 표현
 - ▣ 상수, 변수 모두 피연산자이며 연산자는 연산 유형에 따라 산술연산(arithmetic expression)과 논리연산(logical expression)으로 구분된다


$$E = mc^2$$

$$E = m * c * c$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

기능에 따른 연산자의 분류

연산자의 분류	연산자	의미
대입	=	오른쪽을 왼쪽에 대입
산술	+ - * / %	사칙연산과 나머지 연산
부호	+ -	
증감	++ --	증가, 감소 연산
관계	> < == != >= <=	오른쪽과 왼쪽을 비교
논리	&& !	논리적인 AND, OR
조건	?	조건에 따라 선택
coma	,	피연산자들을 순차적으로 실행
비트 연산자	& ^ ~ << >>	비트별 AND, OR, XOR, 이동, 반전
sizeof 연산자	sizeof	자료형이나 변수의 크기를 바이트 단위로 반환
형변환	(type)	변수나 상수의 자료형을 변환
포인터 연산자	* & []	주소계산, 포인터가 가리키는 곳의 내용 추출
구조체 연산자	. ->	구조체의 멤버 참조



피연산자 수에 따른 연산자 분류

- 단항 연산자(unary operator): 피연산자의 수가 1개

```
++x;  
--y;
```

- ❖ 이항 연산자(binary operator) : 피연산자의 수가 2개

```
x + y  
x - y
```

- ❖ 삼항 연산자(ternary operator) : 연산자의 수가 3개

```
x ? y : z
```



대입(치환) 연산자

- 왼쪽에 있는 변수에 오른쪽의 수식의 결과 값(반환값)을 계산하여 대입

변수(variable) = 수식(expression);

```
x = 10;      // 상수 10을 변수 x에 대입한다.  
y = x;       // 변수 x의 값을 변수 y에 대입한다.  
z = 2 * x + y; // 수식 2 * x + y를 계산하여 변수 z에 대입한다.
```

대입 덧셈의 결과인 3을 리턴

$a = 1 + 2;$

대입된 값 3을 리턴한다

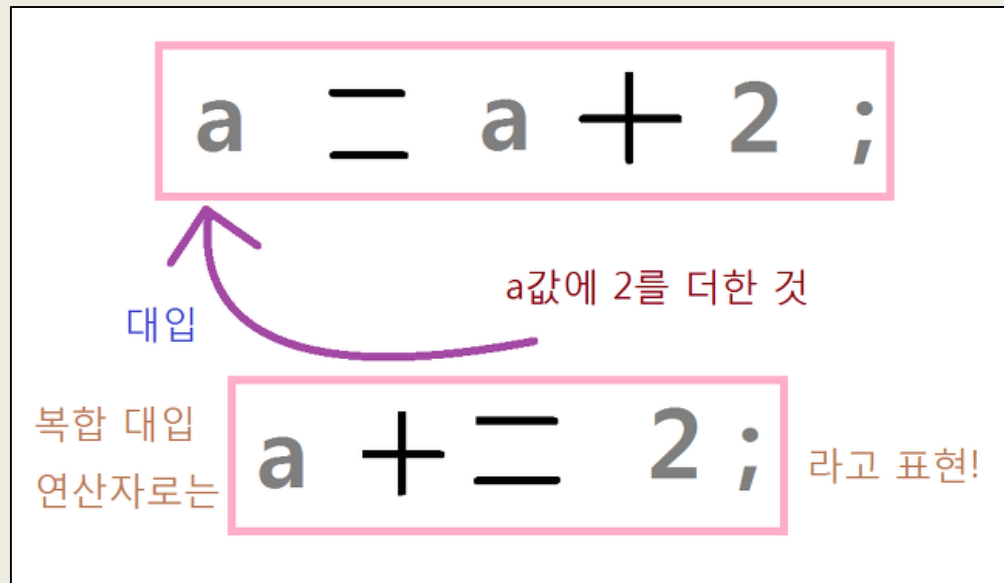


❑ 좌변은 항상 변수

```
x + 2 = 0;           // 왼편이 변수이름이 아니기 때문에 잘못된 수식!!  
2 = x;               // 왼편이 변수이름이 아니기 때문에 잘못된 수식!!
```

- 다음의 문장은 수학적으로는 올바르지 않지만 C에서는 가능.

```
x = x + 1;           // x의 값이 하나 증가 된다.
```





산술 연산자

- 덧셈, 뺄셈, 곱셈, 나눗셈 등 계산 연산을 수행하는 연산자

연산자	기호	의미	예
덧셈	+	x와 y를 더한 결과 반환	x+y
뺄셈	-	x에서 y를 뺀 결과 반환.	x-y
곱셈	*	x와 y를 곱한 결과 반환	x*y
나눗셈	/	x를 y로 나눈 결과 반환	x/y
나머지	%	x를 y로 나눈 나머지 반환	x%y

$$y = mx + b$$

$$y = m * x + b$$

$$y = ax^2 + bx + c$$

$$y = a * x * x + b * x + c$$

$$m = \frac{x + y + z}{3}$$

$$m = (x + y + z) / 3$$

(참고) 거듭 제곱 연산자는?

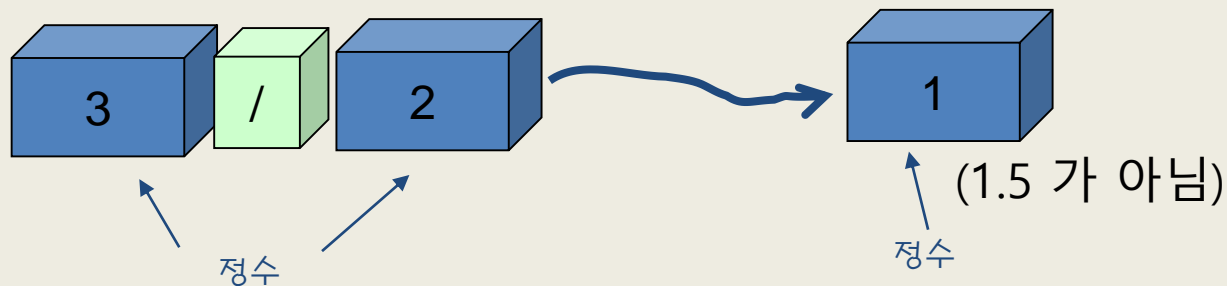
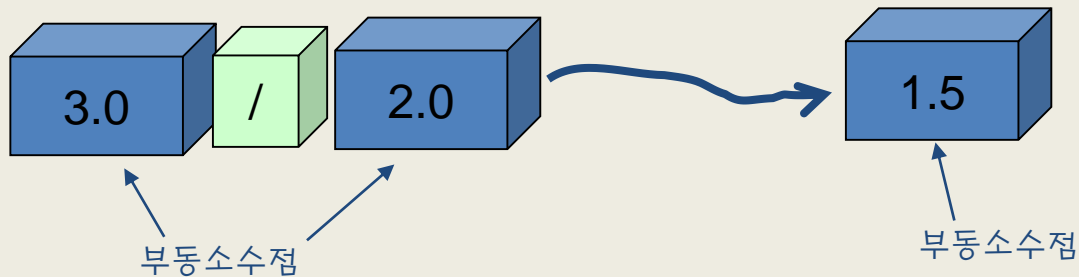
C에는 거듭 제곱을 나타내는 연산자는 없다.

x * x와 같이 단순히 변수를 두 번 곱한다.



나눗셈 연산자

- ❑ 정수형끼리의 나눗셈 → 결과도 정수형
 - ❑ 정수형끼리의 나눗셈에서는 소수점 이하는 버려진다
 - ❑ 0 으로 나누는 경우가 발생하지 않도록 주의(오류 발생)
- ❑ 실수(부동소수점)형이 섞인 나눗셈 → 실수(부동소수점) 값




```
// 나눗셈연산자프로그램
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    printf("3/2 = %d \n", 3/2);
```

```
// 정수 / 정수
```

```
    printf("4/2 = %d \n", 4/2);
```

```
    printf("5/2 = %d \n", 5/2);
```

```
    printf("3.0/2.0 = %f \n", 3.0/2.0);
```

```
// 부동 소수점 / 부동 소수점
```

```
    printf("4.0/2.0 = %f \n", 4.0/2.0);
```

```
    printf("5.0/2.0 = %f \n", 5.0/2.0);
```

```
    printf("3.0/2 = %f \n", 3.0/2);
```

```
// 부동 소수점 / 정수
```

```
    return 0;
```

```
}
```

```
3/2 = 1
```

```
4/2 = 2
```

```
5/2 = 2
```

```
3.0/2.0 = 1.500000
```

```
4.0/2.0 = 2.000000
```

```
5.0/2.0 = 2.500000
```

```
3.0/2 = 1.500000
```



나머지(modulus) 연산자

- ❑ 나머지 연산자(modulus operator)는 첫 번째 피연산자를 두 번째 피연산자로 나누었을 경우의 나머지를 반환
 - ❑ $10 \% 2$ 는 0이다.
 - ❑ $5 \% 7$ 는 5이다.
- ❑ 나머지 연산자를 이용한 짝수와 홀수를 구분
 - ❑ $x \% 2$ 가 0이면 짝수
- ❑ 나머지 연산자를 이용한 배수 판단
 - ❑ $x \% 5$ 가 0이면 5의 배수

```

// 나머지 연산자 프로그램
#include <stdio.h>
#define SEC_PER_MINUTE 60 // 1분은 60초

int main(void)
{
    int input, minute, second;

    printf("초단위의 시간을 입력하시요:(32억초이하) ");
    scanf("%d", &input); // 초단위의 시간을 읽는다.

    minute = input / SEC_PER_MINUTE; // 몇 분
    second = input % SEC_PER_MINUTE; // 몇 초

    printf("%d초는 %d분 %d초입니다. \n", input, minute, second);
    return 0;
}

```

초단위의 시간을 입력하시요:(32억초이하) 70
70초는 1분 10초입니다.

복합 대입 연산자

- ❑ 복합 대입 연산자란 +=처럼 대입연산자 =와 산술연산자 +를 합쳐 놓은 연산자
- ❑ 소스를 간결한게 만들 수 있음

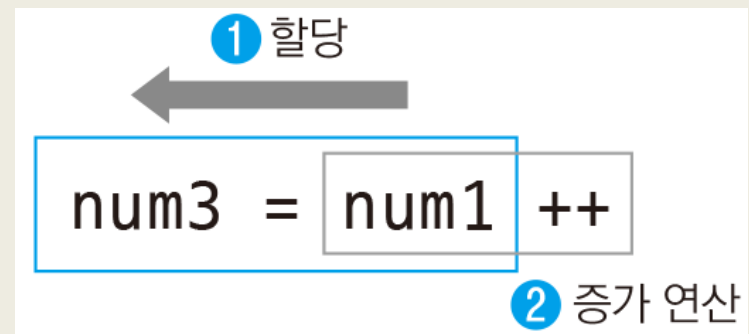
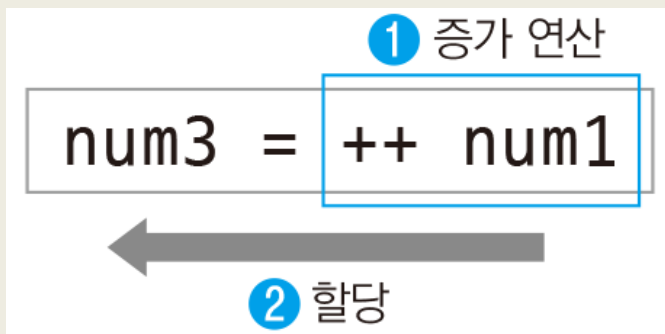
<code>a = a + b</code>	⇐ 동일 연산 ⇒	<code>a += b</code>
<code>a = a - b</code>	⇐ 동일 연산 ⇒	<code>a -= b</code>
<code>a = a * b</code>	⇐ 동일 연산 ⇒	<code>a *= b</code>
<code>a = a / b</code>	⇐ 동일 연산 ⇒	<code>a /= b</code>
<code>a = a % b</code>	⇐ 동일 연산 ⇒	<code>a %= b</code>

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>



증감 연산자

증감 연산자	의미
<code>++x</code> (pre-increment)	x값을 먼저 증가시킨 후에 연산에 사용한다. x의 결과 값은 증가된 x값이다.
<code>x++</code> (post-increment)	x값을 먼저 연산에 사용한 후에, 증가시킨다. x의 결과 값은 증가된 x값이다.
<code>--x</code> (pre-decrement)	x값을 먼저 감소시킨 후에 연산에 사용한다. x의 결과 값은 감소된 x값이다.
<code>x--</code> (post-decrement)	x값을 먼저 사용한 후에, 감소시킨다. x의 결과 값은 감소된 x값이다.



증감연산자 예

```
// 증감연산자를 이용한 프로그램
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x = 10;
```

```
    printf("수식 x++ 의 값: %d \n", x++);
```

```
    printf("현재 x의 값: %d \n", x);
```

```
    printf("수식 ++x 의 값: %d \n", ++x);
```

```
    printf("현재 x의 값: %d \n", x);
```

```
    printf("수식 x-- 의 값: %d \n", x--);
```

```
    printf("현재 x의 값: %d \n", x);
```

```
    printf("수식 --x 의 값: %d \n", --x);
```

```
    printf("현재 x의 값: %d \n", x);
```

```
}
```

수식 $x++$ 의 값: 10

현재 x 의 값: 11

수식 $++x$ 의 값: 12

현재 x 의 값: 12

수식 $x--$ 의 값: 12

현재 x 의 값: 11

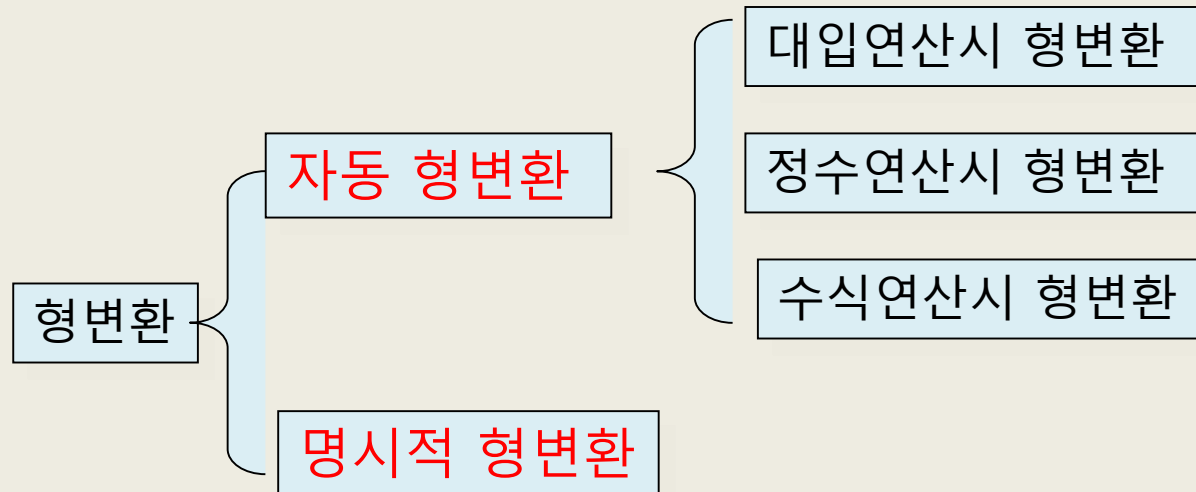
수식 $--x$ 의 값: 10

현재 x 의 값: 10



형 변환

- ❑ Type cast 혹은 type conversion
- ❑ 연산 도중 데이터의 유형을 변경하고자 할 경우 사용





자동 형변환

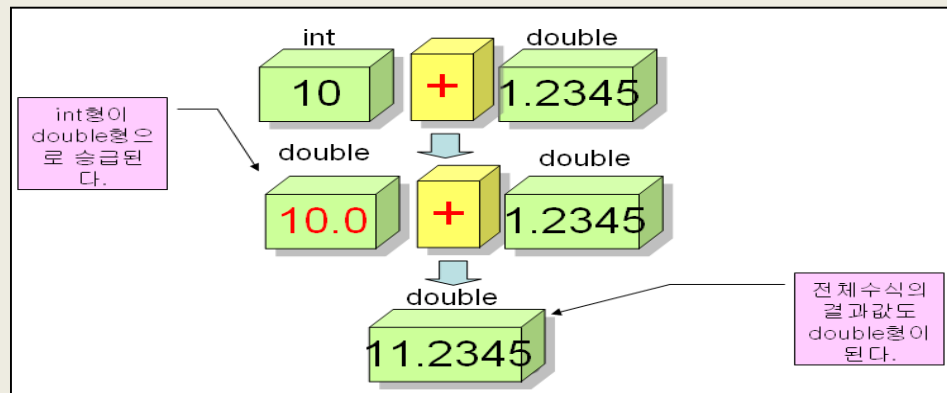
- 올림 변환 : 길이가 작은 유형 → 길이가 큰 유형

```
double f;  
f = 10 + 20;           // f에는 30.0이 저장된다.
```

- 내림 변환 : 길이가 큰 유형 → 길이가 작은 유형

```
int i;  
i = 3.141592;          // i에는 3이 저장된다.
```

- 수식에서 서로 다른 자료형이 혼합하여 사용되는 경우, 더 큰 자료형으로 통일된다.




```

#include <stdio.h>
int main(void)
{
    char c;
    int i;
    float f;

    c = 10000;           // 내림 변환 --- 16진수로 0x2710
    i = 1.23456 + 10;    // 내림 변환
    f = 10 + 20;         // 올림 변환
    printf("c = %d, i = %d, f = %f \n", c, i, f);
    return 0;
}

```

C:\CPROGRAM\convert1\convert1.c(10) : warning C4305: '=' : truncation from 'const int' to 'char'

C:\CPROGRAM\convert1\convert1.c(11) : warning C4244: '=' : conversion from 'const double' to 'int', possible loss of data

c = 16, i = 11, f = 30.000000



명시적인 형변환

- 사용자가 데이터의 타입을 명시적(혹은 강제적)으로 변경

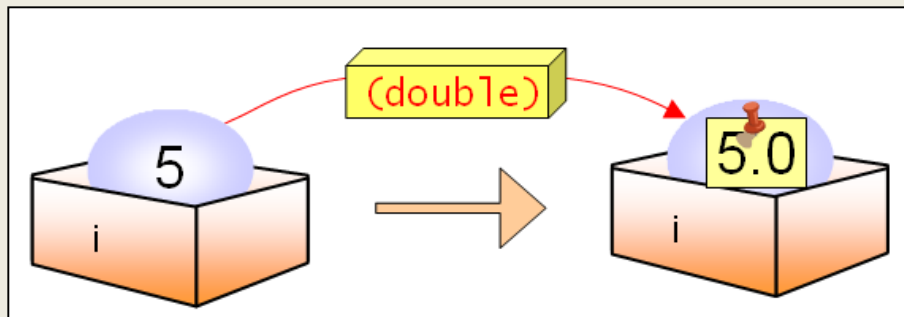
(자료형)상수 또는 변수

`(double) (5)`

`f = (double)i + (double)j;`

`f = (double)((int)y + 3);`

`f = (float)(x = 5);` // 수식 `x = 5`의 결과값인 5가 float형으로 변환



예제

1. <code>int i;</code> 2. <code>double f;</code> 3. <code>f = 5 / 4;</code> 4. <code>f = (double)5 / 4;</code> 5. <code>f = 5 / (double)4;</code> 6. <code>f = (double)5 / (double)4;</code> 7. <code>i = 1.3 + 1.8;</code> 8. <code>i = (int)1.3 + (int)1.8;</code>	<code>// f는 1</code> <code>// f는 1.25</code> <code>// f는 1.25</code> <code>// f는 1.25</code> <code>// i는 3</code> <code>// i는 2</code>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------

1. 정수형 변수 `i` 선언
2. 부동 소수점형 변수 `f` 선언
3. (정수/ 정수)는 정수
4. 5를 부동소수점으로 변환하여 계산, 전체는 부동소수점형이 됨
5. 4를 부동소수점으로 변환하여 계산, 전체는 부동소수점형이 됨
6. 5와 4를 모두 부동소수점으로 변환하여 계산
7. `1.3+1.8`은 `3.1`로 계산되고 정수형 변수에 대입되므로 `i`는 `3`
8. `(int)1.3 + (int)1.8`은 `1+1`로 되어서 `i`는 `2`



관계 연산자

- ❑ 두 개의 피 연산자 간의 **관계**를 나타내는(비교) 연산자
- ❑ 결과값은 참(1) 아니면 거짓(0)

연산자 기호	의미	사용예
==	x와 y가 같은가?	<code>x == y</code>
!=	x와 y가 다른가?	<code>x != y</code>
>	x가 y보다 큰가?	<code>x > y</code>
<	x가 y보다 작은가?	<code>x < y</code>
>=	x가 y보다 크거나 같은가?	<code>x >= y</code>
<=	x가 y보다 작거나 같은가?	<code>x <= y</code>



사용 예

```
1. #include <stdio.h>
2. int main(void)
3. {
4.     int x=10, y=20;
5.     int r1, r2, r3, r4;
6.
7.     r1 = (x == y);           // 같으면 1
8.     r2 = (x != y);           // 다르면 1
9.     r3 = (x >= y);           // 크거나 같으면 1
10.    r4 = (x <= y);           // 작거나 같으면 1
11.
12.    printf("r1=%d \n", r1);
13.    printf("r2=%d \n", r2);
14.    printf("r3=%d \n", r3);
15.    printf("r4=%d \n", r4);
16.    return 0;
17. }
```

```
r1=0
r2=1
r3=0
r4=1
```



논리 연산자

- ❑ 피연산자들 간의 논리적 참과 거짓을 판단하는 연산자
- ❑ 결과값은 참(1) 아니면 거짓(0)

연산자 기호	사용예	의미
&&	x && y	AND 연산, x와 y가 모두 참이면 참, 그렇지 않으면 거짓
	x y	OR 연산, x나 y중에서 하나만 참이면 참, 모두 거짓이면 거짓
!	!x	NOT 연산, x가 참이면 거짓, x가 거짓이면 참



진리표(Truth Table)

수학적인 논리 연산

x	y	x AND y	x OR y	NOT x
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

C에서의 논리 연산

x	y	x && y	x y	!x
0	0	0	0	1
0	0이 아닌값	0	1	1
0이 아닌값	0	0	1	0
0이 아닌값	0이 아닌값	1	1	0



참과 거짓의 표현

- ❑ 관계식이나 논리식이 참이면 1, 거짓이면 0
- ❑ 피연산자의 참, 거짓을 가릴 때에는
 - ▣ 0이 아니면 참이고
 - ▣ 0이면 거짓으로 판단한다
- ❑ 음수는 거짓으로 판단한다.
- ❑ (예) NOT 연산자를 적용하는 경우

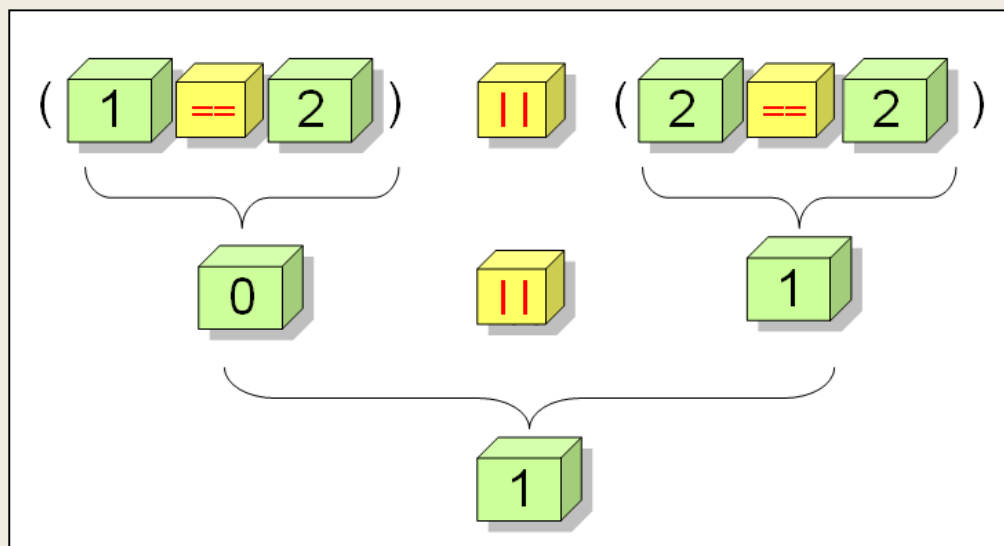
```
!0           // 0이면 1로 만든다.  
!3           // 0이 아니면 0로 만든다.  
!100        // 0이 아니면 0로 만든다.  
!-3         //  
!x           // 변수 x의 값이 0이면 1, 0이 아니면 0  
!(x + 1)     // 수식 (x+1)의 값이 0이면 1, 0이 아니면 0  
!(x > y && x < z) // 관계수식 x > y && x < z의 값이 0이면 1, 0이 아니면 0
```




AND와 OR 연산자

1 && 2 // 피연산자 모두 0이 아니므로 전체 수식은 참
(1==2) && (2==2) // 하나의 피연산자만 참이므로 전체 수식은 거짓
(1==2) || (2==2) // 하나의 피연산자가 참이므로 전체 수식은 참
(x>10) && (x<20) // x가 10보다 크고 20보다 작으면 참이다.
(x>10) || (x<20) // x가 10보다 크거나 20보다 작으면 참이다(항상 참).

- 논리 연산의 결과값은 항상 1 (참) 또는 0 (거짓)이다.
- (예)



예제

```
// 논리 연산자 프로그램
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x=10, y=20;
```

```
    int r1, r2, r3, r4;
```

```
    r1 = (x == 10 && y == 20);
```

```
    r2 = (x == 10 && y == 30);
```

```
    r3 = (x >= 10 || y >= 30);
```

```
    r4 = !(x == 5);
```

```
    printf("r1=%d \n", r1);
```

```
    printf("r2=%d \n", r2);
```

```
    printf("r3=%d \n", r3);
```

```
    printf("r4=%d \n", r4);
```

```
    return 0;
```

```
}
```

r1=1

r2=0

r3=1

r4=1



논리 연산자의 우선 순위

- ❑ !연산자의 우선 순위는 증가 연산자 ++나 감소 연산자 --와 동일
- ❑ &&와 || 연산자의 우선 순위는 모든 산술 연산자나 관계 연산자보다 낮다.
- ❑ &&가 || 연산자보다는 우선 순위가 높다

```
x < 0 || x > 10
x > 5 || x < 10 && x > 0           // x > 5 || (x < 10 && x > 0) 와 동일
(x > 5 || x < 10) && x > 0
```

- ❑ && 연산자의 경우, 첫번째 피연산자가 거짓이면 다른 피연산자들을 계산하지 않는다

```
( 2 > 3 ) && ( ++x < 5 )
```

- ❑ || 연산자의 경우, 첫번째 피연산자가 참이면 다른 피연산자들을 계산하지 않는다

```
( 3 > 2 ) || ( --x < 5 )
```



예제

□ 윤년을 판단하는 문제

- ① 연도가 4로 나누어 떨어진다.
- ② 100으로 나누어 떨어지는 연도는 제외한다.
- ③ 400으로 나누어 떨어지는 연도는 윤년이다.

```
// 윤년 프로그램
#include <stdio.h>

int main(void)
{
    int year, result;

    printf("연도를 입력하시오: ");
    scanf("%d", &year);

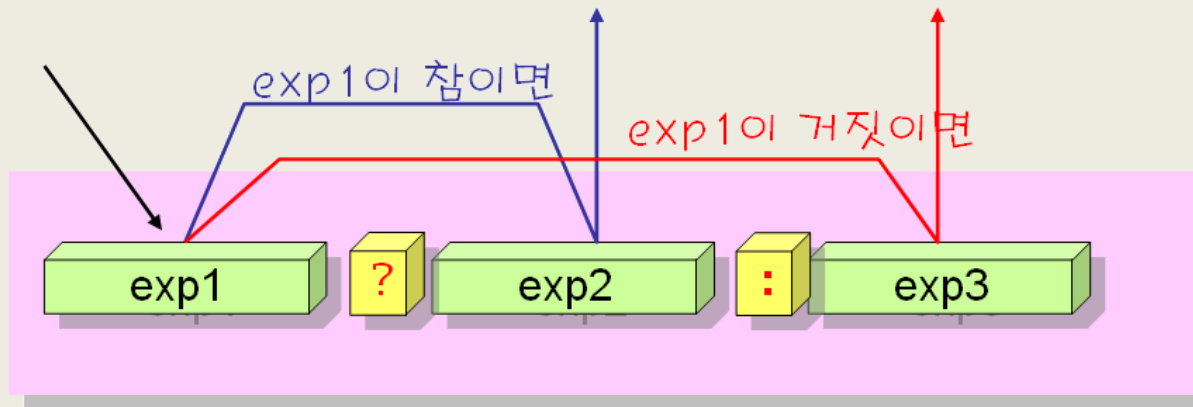
    result = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    printf("result=%d \n", result);
    return 0;
}
```



조건 연산자

- exp1가 참이면 exp2를 반환, 그렇지 않으면 exp3를 반환

exp1 ? exp2 : exp3



`(5 > 2) ? 5 : 2` // 5가 2보다 크므로 5
`(1.2 > 1.1) ? 1 : 0` // 1.2가 1.1보다 크므로 1
`(x == 0) ? 100 : 200` // x가 0과 같으면 100 그렇지 않으면 200

예제

```
#include <stdio.h>
int main(void)
{
    int x,y;

    printf("첫번째 수=");
    scanf("%d", &x);
    printf("두번째 수=");
    scanf("%d", &y);

    printf("큰수=%d \n", (x > y) ? x : y);
    printf("작은수=%d \n", (x < y) ? x : y);
}
```

첫번째 수=2

두번째 수=3

큰수=3

작은수=2



콤마 연산자

- ❑ 콤마로 연결된 수식은 순차적으로 계산된다.

```
x=1, y=2;  
x = ( 2+5, 5-3 );  
x = 2+3, 5-3;  
x++, y++;  
printf("Thank"), printf(" you!\n");
```

```
x=1; y=2;와 동일  
x=2가 된다  
x=5가 된다  
x와 y는 1 증가된다.  
Thank you!
```



비트연산자

- ❑ 비트 연산자는 피연산자 값에 대하여 비트(bit) 단위로, 주어진 연산을 수행
 - ❑ **&** : bit 단위로 AND 연산 수행
 - ❑ **|** : bit 단위로 OR 연산 수행
 - ❑ **^** : bit 단위로 XOR 연산수행
 - ❑ **<<** : 왼쪽으로 지정한 수만큼 bit를 이동 (shift left)
 - ❑ **>>** : 오른쪽으로 지정한 수만큼 bit를 이동 (shift right)
 - ❑ **~** : bit 단위로 NOT 연산 수행
- ❑ 비트 단위 연산은 **char** 형을 포함하여 **int** 형 자료에만 적용



□ Examples

- $0x43 \& 0x38$ 의 결과는? vs. $0x43 \&\& 0x38$ 의 결과는?
- $0x43 | 0x38$ 의 결과는? vs. $0x43 || 0x38$ 의 결과는?
- $0x43 \wedge 0x38$ 의 결과는?
- $\sim 0x43$ 의 결과는? vs. $!0x43$ 의 결과는?
- $0x43 << 2$
 - $0100\ 0011$ 을 왼쪽으로 2번 bit 이동. 그 결과는? $\rightarrow 0000\ 11\mathbf{00}$
- $0x0043 << 2$?
 - $0000\ 0000\ 0100\ 0011$ 을 왼쪽으로 2번 bit 이동.
 - 그 결과는 $\rightarrow 0000\ 0001\ 0000\ 11\mathbf{00}$
- $0x43 >> 2$
 - $0100\ 0011$ 을 오른쪽으로 2번 bit 이동. 그 결과는 ? $\rightarrow \mathbf{00}01\ 0000$
 - 오른쪽으로 이동할 경우, **비워지는 왼쪽 MSB 부분은 이동 전의 부호 비트로 채워진다**



비트연산자 사용 예

□ 비트 마스크 (bit mask)

- 주어진 비트 패턴에 대해 원하지 않는 부분을 가려서 제거하는 방법
- 모니터에서의 컬러 표현 : pixel들의 모임
 - 하나의 pixel은 (투명도+Red+Green+Blue) 의 32비트로 표현 (true color)
- 모니터의 해상도 : M x N
 - 예를 들면, 어떤 모니터가 해상도 1024 x 768 을 가지고 있다면,
 - 이 모니터 화면(frame)은 768개의 수평 line으로 구성되는데, 각 수평 line은 1024개의 pixel로 구성되어 있다는 것을 의미한다.
 - 그러므로 이 모니터의 하나 화면은 총 1024 x 768 (786K)개의 pixel들로 구성된다는 것이고, 바이트로 계산하면 1024 x 768 x 4 bytes 로 구성되는 것이다
- 순수한 빨강색 pixel은 00000000 11111111 00000000 00000000 의 값을 가지게 된다
- 그렇다면, 어떤 pixel에서 red 부분이 얼마의 값을 가지고 있는가 알고 싶다면 어떻게 할 것인가?



sizeof 연산자

- ❑ 피 연산자를 구성하는 바이트의 수를 결과 값으로 반환

```
size_t n = sizeof( int );
```

```
#include <stdio.h>
int main(void) {
    int i;
    double f;
    size_t n;

    n = sizeof(int);
    printf("int형의 크기=%u \Wn", n);

    n = sizeof(i);
    printf("변수 i의 크기=%u \Wn", n);

    n = sizeof(f);
    printf("변수 f의 크기=%u \Wn", n);
}
```

```
int형의 크기=4
변수 i의 크기=4
변수 f의 크기=8
```



연산자 우선 순위(Priority)

- 수식에서 어떤 연산자를 먼저 계산할 것인지를 결정

우선 순위	연산자	결합 규칙
1	() [] -> . ++(후위) --(후위)	-> (좌에서 우)
2	sizeof &(주소) ++(전위) --(전위) ~ ! *(역참조) +(부호) -(부호), 형변환	<- (우에서 좌)
3	*(곱셈) / %	-> (좌에서 우)
4	+(덧셈) -(뺄셈)	-> (좌에서 우)
5	<< >>	-> (좌에서 우)
6	< <= >= >	-> (좌에서 우)
7	== !=	-> (좌에서 우)
8	&(비트연산)	-> (좌에서 우)
9	^	-> (좌에서 우)
10		-> (좌에서 우)
11	&&	-> (좌에서 우)
12		-> (좌에서 우)
13	?(삼항)	-> (우에서 좌)
14	= += *= /= %= &= ^= = <<= >>=	-> (우에서 좌)
15	,(콤마)	-> (좌에서 우)



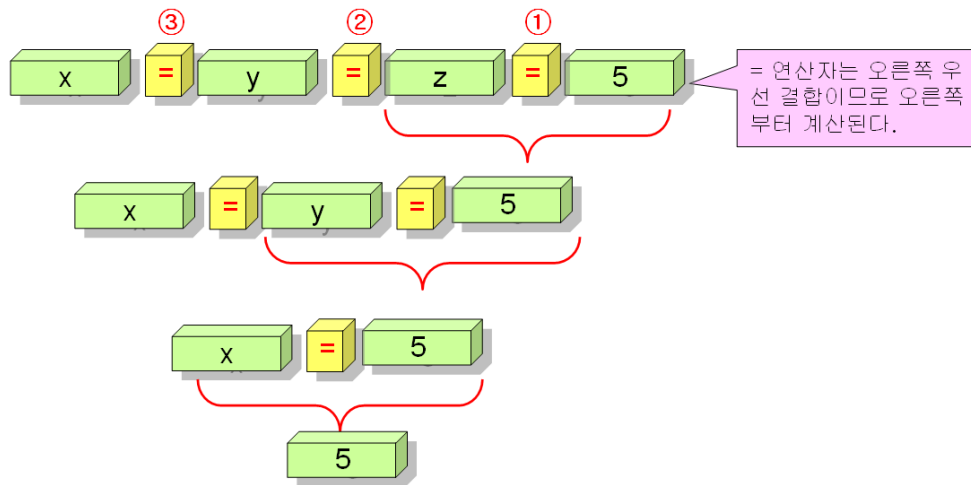
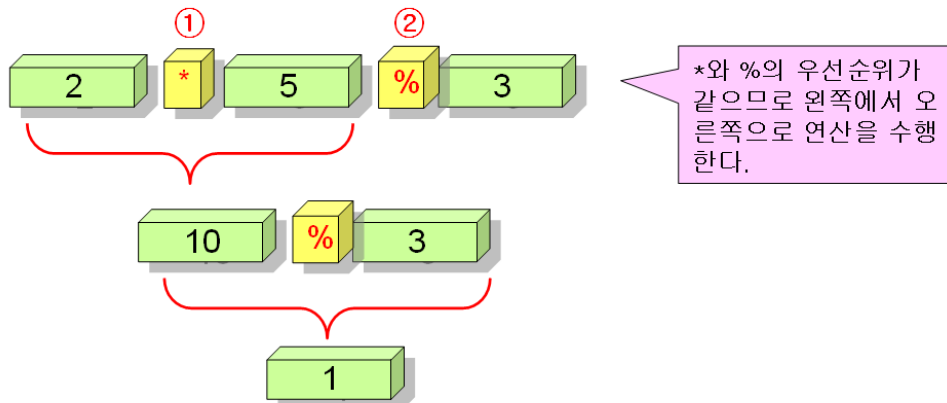
우선 순위의 일반적인 지침

- ❑ 콤마 < 대입 < 논리 < 관계 < 산술 < 단항
- ❑ 괄호 연산자는 가장 우선순위가 높다.
- ❑ 모든 단항 연산자들은 이항 연산자들보다 우선순위가 높다.
- ❑ 콤마 연산자를 제외하고는 대입 연산자가 가장 우선순위가 낮다.
- ❑ 연산자들의 우선 순위가 생각나지 않으면 괄호를 이용
 - $(x \leq 10) \ \&\& \ (y \geq 20)$
- ❑ 관계 연산자나 논리 연산자는 산술 연산자보다 우선순위가 낮다.
 - $x + 2 == y + 3$



결합 규칙

- 만약 같은 우선순위를 가지는 연산자들이 여러 개가 있으면 어떤 것을 먼저 수행하여야 하는가의 규칙



예제

```
#include <stdio.h>
int main(void)
{
    int x = 2, y = 3, z = 4;

    printf("%d \n", 2 + 3 >= 3 + !2);
    printf("%d \n", 2 > 3 || 6 > 7);
    printf("%d \n", 2 || 3 && 3 > 2);
    printf("%d \n", - ++x + y--);
    printf("%d \n", x = y = z = 6 );
    printf("%d \n", (x = 2 + 3, 2 - 3));
    printf("%d \n", x /= x = x * y );
}
```

```
1
0
1
0
6
-1
1
```