

개정3판

Visual  
Studio  
2017

쉽게 풀어쓴

C언어  
EXPRESS



천인국 지음

## 제5장 수식과 연산자

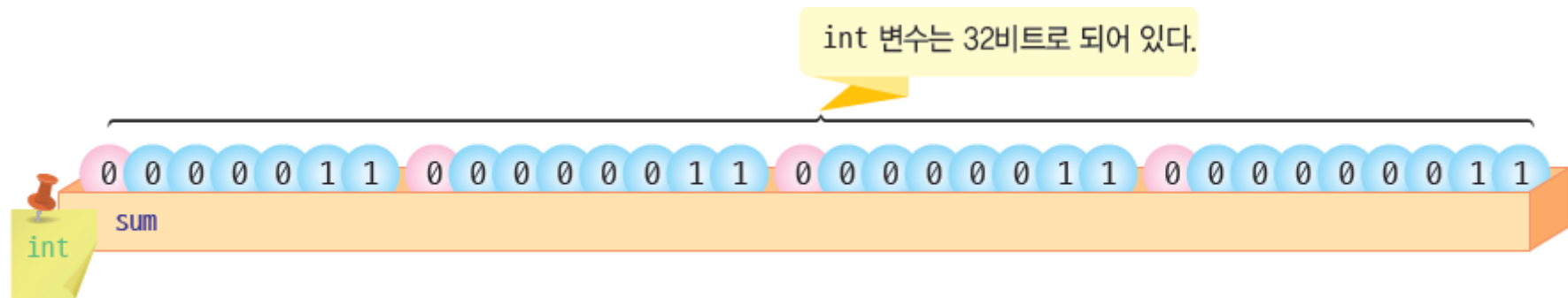


# 비트 연산자

연산자	연산자의 의미	예
&	비트 AND	두개의 피연산자의 해당 비트가 모두 1이면 1, 아니면 0
	비트 OR	두개의 피연산자의 해당 비트중 하나만 1이면 1, 아니면 0
^	비트 XOR	두개의 피연산자의 해당 비트의 값이 같으면 0, 아니면 1
<<	왼쪽으로 이동	지정된 개수만큼 모든 비트를 왼쪽으로 이동한다.
>>	오른쪽으로 이동	지정된 개수만큼 모든 비트를 오른쪽으로 이동한다.
~	비트 NOT	0은 1로 만들고 1은 0로 만든다.



# 모든 데이터는 비트로 이루어진다.





# 비트 AND 연산자

0 AND 0 = 0
1 AND 0 = 0
0 AND 1 = 0
1 AND 1 = 1

```
변수1 00000000 00000000 00000000 00001001 (9)
변수2 00000000 00000000 00000000 00001010 (10)
-----
(변수1 AND 변수2) 00000000 00000000 00000000 00001000 (8)
```



# 비트 OR 연산자

$0 \text{ OR } 0 = 0$
$1 \text{ OR } 0 = 1$
$0 \text{ OR } 1 = 1$
$1 \text{ OR } 1 = 1$

변수1 00000000 00000000 00000000 00001001 (9)  
변수2 00000000 00000000 00000000 00001010 (10)

---

(변수1 OR 변수2) 00000000 00000000 00000000 00001011 (11)



# 비트 XOR 연산자

$0 \text{ XOR } 0 = 0$
$1 \text{ XOR } 0 = 1$
$0 \text{ XOR } 1 = 1$
$1 \text{ XOR } 1 = 0$

변수1 00000000 00000000 00000000 00001001 (9)  
변수2 00000000 00000000 00000000 00001010 (10)

---

(변수1 XOR 변수2) 00000000 00000000 00000000 00000011 (3)



# 비트 NOT 연산자

NOT 0 = 1
NOT 1 = 0

부호비트가 반전되었기 때문에 음수가 된다.

변수1 00000000 00000000 00000000 00001001 (9)

(NOT 변수1) 11111111 11111111 11111111 11110110 (-10)



# 비트 이동 연산자

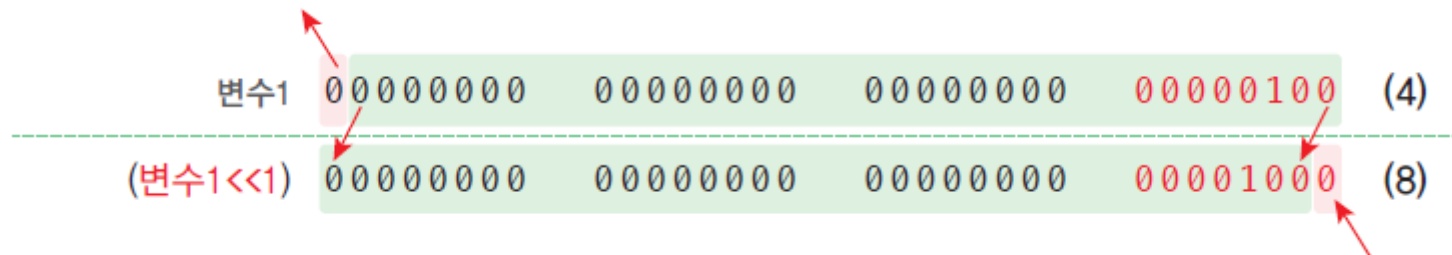
연산자	기호	설명
왼쪽 비트 이동	<code>&lt;&lt;</code>	$x \ll y$ $x$ 의 비트들을 $y$ 칸만큼 왼쪽으로 이동
오른쪽 비트 이동	<code>&gt;&gt;</code>	$x \gg y$ $x$ 의 비트들을 $y$ 칸만큼 오른쪽으로 이동





## << 연산자

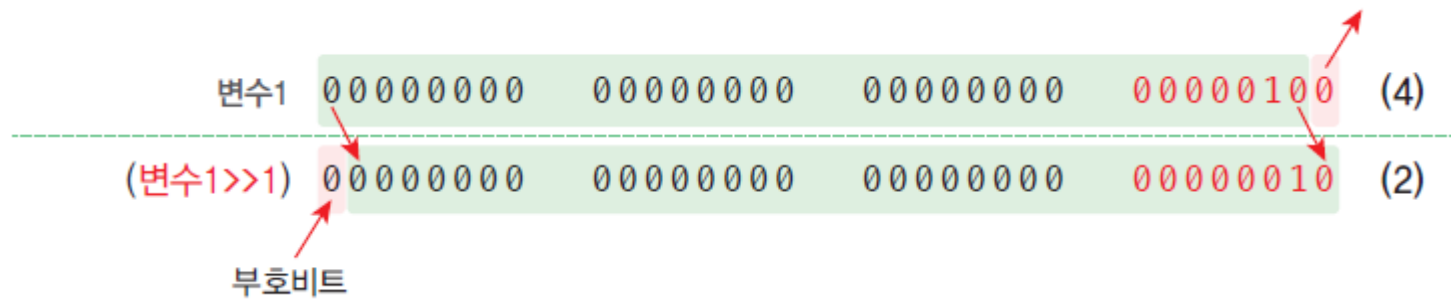
- 비트를 왼쪽으로 이동
- 값은 2배가 된다.





## >> 연산자

- 비트를 오른쪽으로 이동
- 값은 1/2배가 된다.





## 예제: 비트 연산자

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

```
int main(void)
```

```
{
```

```
    printf("AND : %08X\n", 0x9 & 0xA);
```

```
    printf("OR : %08X\n", 0x9 | 0xA);
```

```
    printf("XOR : %08X\n", 0x9 ^ 0xA);
```

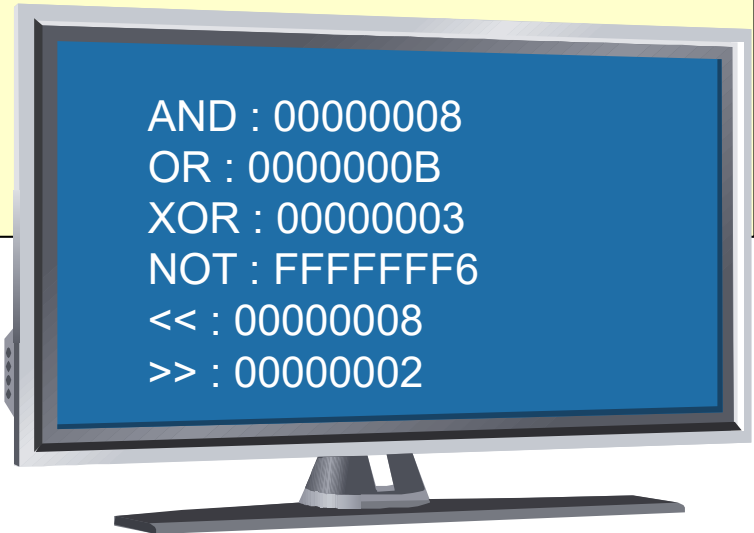
```
    printf("NOT : %08X\n", ~0x9);
```

```
    printf("<< : %08X\n", 0x4 << 1);
```

```
    printf(">> : %08X\n", 0x4 >> 1);
```

```
    return 0;
```

```
}
```

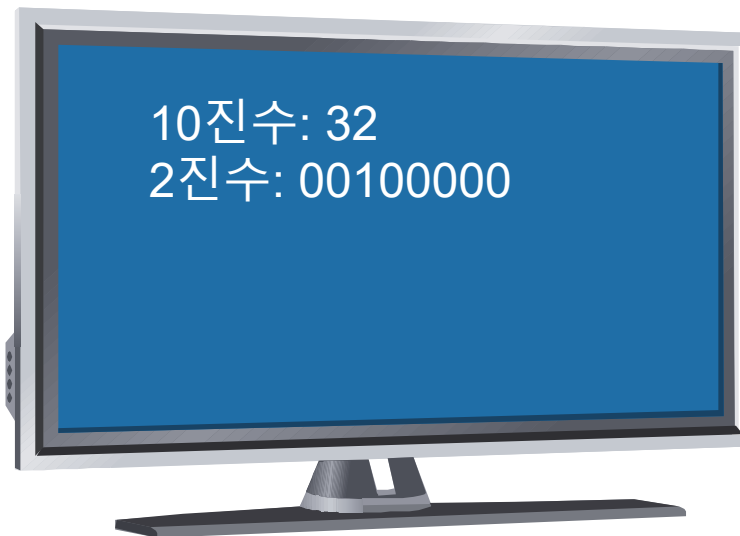


```
AND : 00000008  
OR : 0000000B  
XOR : 00000003  
NOT : FFFFFFFF  
<< : 00000008  
>> : 00000002
```



## Lab: 10진수를 2진수로 출력하기

- 비트 연산자를 이용하여 128보다 작은 10진수를 2진수 형식으로 화면에 출력해보자.





## Lab: 10진수를 2진수로 출력하기

```
#include<stdio.h>

int main(void)
{
    unsigned int num;
    printf("십진수: ");
    scanf("%u", &num);

    unsigned int mask = 1 << 7;  // mask = 10000000
    printf("이진수: ");

    ((num & mask) == 0) ? printf("0") : printf("1");
    mask = mask >> 1;          // 오른쪽으로 1비트 이동한다.
    ((num & mask) == 0) ? printf("0") : printf("1");
    mask = mask >> 1;          // 오른쪽으로 1비트 이동한다.
    ((num & mask) == 0) ? printf("0") : printf("1");
    mask = mask >> 1;          // 오른쪽으로 1비트 이동한다.
```



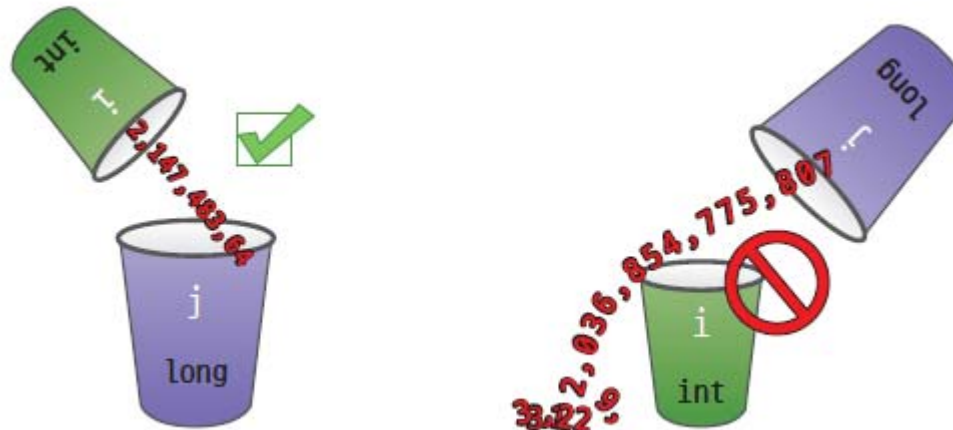
## Lab: 10진수를 2진수로 출력하기

```
((num & mask) == 0) ? printf("0") : printf("1");  
mask = mask >> 1;  
((num & mask) == 0) ? printf("0") : printf("1");  
mask = mask >> 1;  
((num & mask) == 0) ? printf("0") : printf("1");  
mask = mask >> 1;  
((num & mask) == 0) ? printf("0") : printf("1");  
mask = mask >> 1;  
((num & mask) == 0) ? printf("0") : printf("1");  
printf("\n");  
  
return 0;  
}
```



# 형 변환

- 형 변환(type conversion)이란 실행 중에 데이터의 타입을 변경하는 것이다





# 형 변환

- 연산시에 데이터의 유형이 변환되는 것



자동으로  
변환되기도  
하고 사용자가  
바꾸어 주기도  
하죠



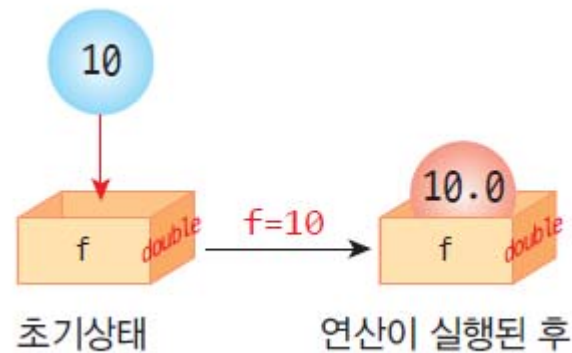




# 대입 연산시의 자동적인 형변환

## □ 올림 변환

```
double f;  
f = 10; // f에는 10.0이 저장된다.
```

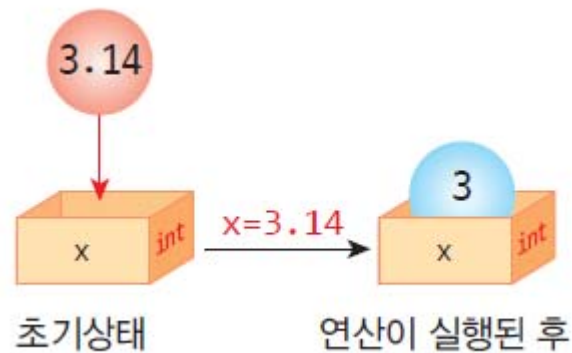




# 대입 연산시의 자동적인 형변환

## □ 내림변환

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





# 올림 변환과 내림 변환

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

    c = 10000;           // 내림 변환
    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:\...\convert1.c(10) : warning C4305: '=' : 'int'에서 'char'(으)로  
잘립니다.  
c:\...\convert1.c(11) : warning C4244: '=' : 'double'에서 'int'(으)로 변  
환하면서 데이터가 손실될 수 있습니다.

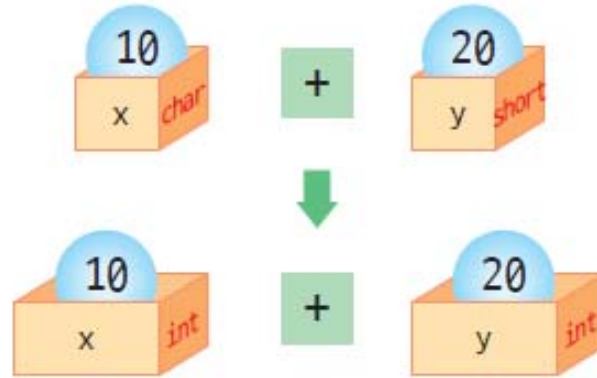
*c=16, i=11, f=30.000000*



# 정수 연산시의 자동적인 형변환

- 정수 연산시 char형이나 short형의 경우, 자동적으로 int형으로 변환하여 계산한다.

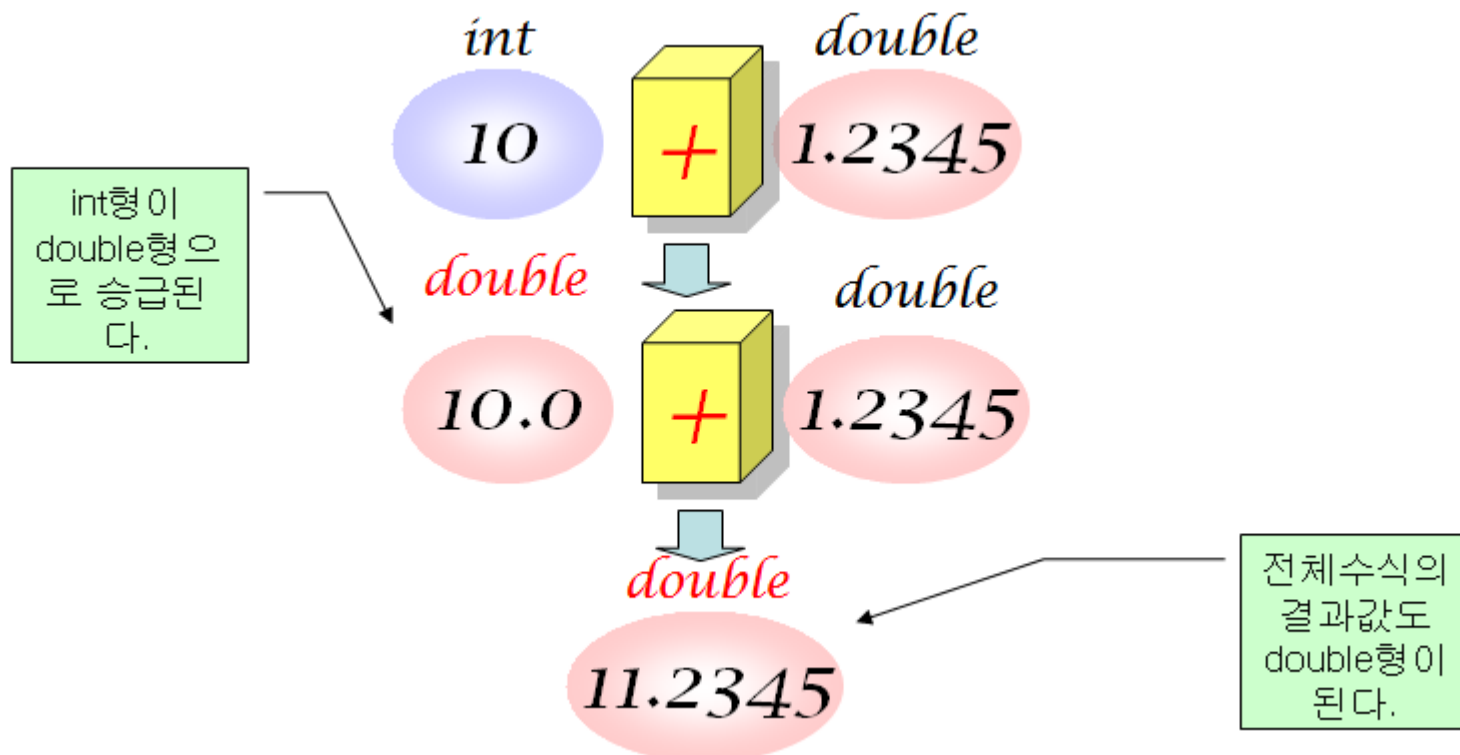
char나 short형은 int형으로 통일하여서 처리합니다.





# 수식에서의 자동적인 형변환

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





# 명시적인 형변환

Syntax: 형변환

자료형

수식

예

(int)1.23456

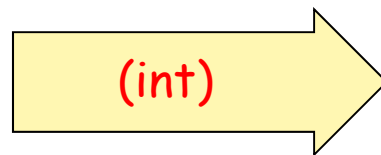
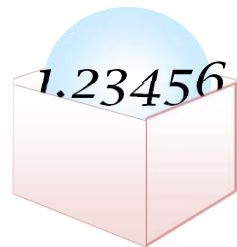
// int형으로 변환

(double) x

// double형으로 변환

(long) (x+y)

// long형으로 변환





# 예제

```
#include <stdio.h>

int main(void)
{
    int i;
    double f;

    f = 5 / 4;

    printf("%f\n", f);

    f = (double)5 / 4;
    printf("%f\n", f);

    f = 5.0 / 4;
    printf("%f\n", f);
}
```



# 예제

```
f = (double)5/ (double)4;  
printf("%f\n", f);
```

```
i = 1.3 + 1.8;  
printf("%d\n", i);
```

```
i = (int)1.3+ (int)1.8;
```

```
printf("%d\n", i);  
return 0;
```

```
}
```







# 우선 순위

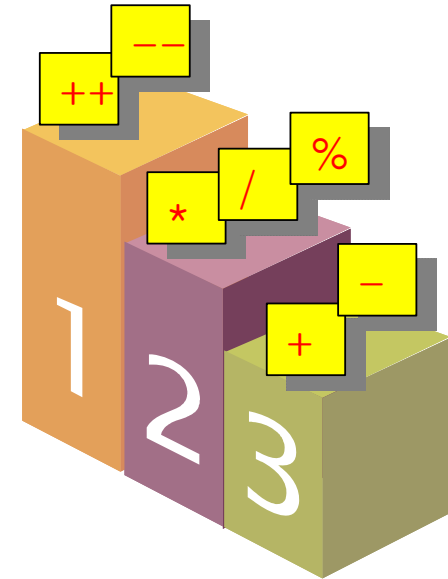
- 어떤 연산자를 먼저 계산할 것인지에 대한 규칙

$$x + y * z$$

Diagram illustrating operator precedence for the expression  $x + y * z$ . A bracket labeled ① groups  $y * z$ , indicating that multiplication is performed first. A second bracket labeled ② groups the entire expression  $x + (y * z)$ , indicating that addition is performed second.

$$(x + y) * z$$

Diagram illustrating operator precedence for the expression  $(x + y) * z$ . A bracket labeled ① groups  $x + y$ , indicating that addition is performed first. A second bracket labeled ② groups the entire expression  $(x + y) * z$ , indicating that multiplication is performed second.





# 우선 순위

우선순위	연산자	설명	결합성
1	++ --	후위 증감 연산자	→ (좌에서 우)
	()	함수 호출	
	[]	배열 인덱스 연산자	
	.	구조체 멤버 접근	
	→	구조체 포인터 접근	
	(type){list}	복합 리터럴(C99 규격)	
2	++ --	전위 증감 연산자	← (우에서 좌)
	+ -	양수, 음수 부호	
	! ~	논리적인 부정, 비트 NOT	
	(type)	형변환	
	*	간접 참조 연산자	
	&	주소 추출 연산자	
	sizeof	크기 계산 연산자	
	_Alignof	정렬 요구 연산자 (C11 규격)	



3	* / %	곱셈, 나눗셈, 나머지	→ (좌에서 우)
4	+ -	덧셈, 뺄셈	
5	<< >>	비트 이동 연산자	
6	< <=	관계 연산자	
	> >=	관계 연산자	
7	== !=	관계 연산자	
8	&	비트 AND	
9	^	비트 XOR	
10		비트 OR	
11	&&	논리 AND 연산자	
12		논리 OR 연산자	
13	?:	삼항 조건 연산자	← (우에서 좌)
14	=	대입 연산자	
	+= -=	복합 대입 연산자	
	*= /= %=	복합 대입 연산자	
	<<= >>=	복합 대입 연산자	
	&= ^=  =	복합 대입 연산자	
15	,	coma 연산자	→ (좌에서 우)



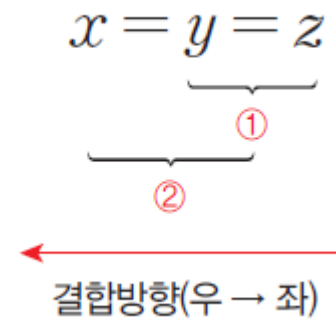
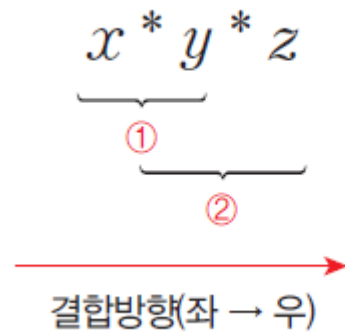
# 우선 순위의 일반적인 지침

- 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.
  - ▣  $x + 2 == y + 3$



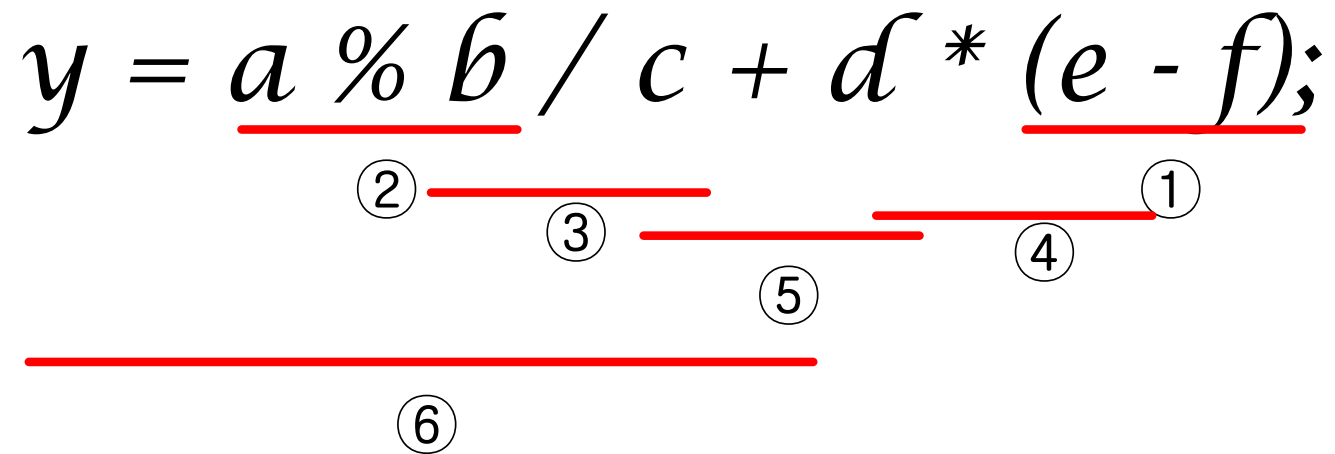
# 결합 규칙

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





## 결합 규칙의 예





# 예제

```
#include <stdio.h>
int main(void)
{
    int x=0, y=0;
    int result;

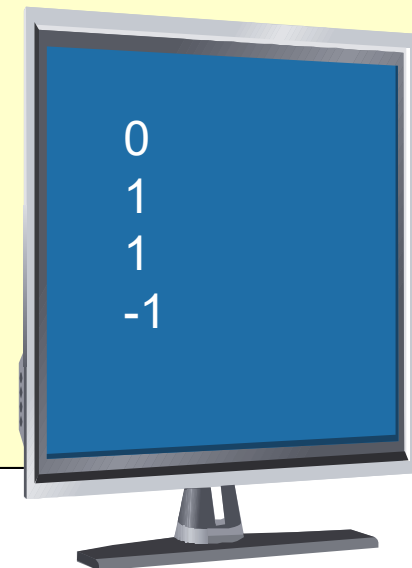
    result = 2 > 3 || 6 > 7;
    printf("%d", result);

    result = 2 || 3 && 3 > 2;
    printf("%d", result);

    result = x = y = 1;
    printf("%d", result);

    result = - ++x + y--;
    printf("%d", result);

    return 0;
}
```





## 중간 점검

1. 연산자 중에서 가장 우선 순위가 낮은 연산자는 무엇인가?
2. 논리 연산자인 &&과 || 중에서 우선 순위가 더 높은 연산자는 무엇인가?
3. 단항 연산자와 이항 연산자 중에서 어떤 연산자가 더 우선 순위가 높은가?
4. 관계 연산자와 산술 연산자 중에서 어떤 연산자가 더 우선 순위가 높은가?



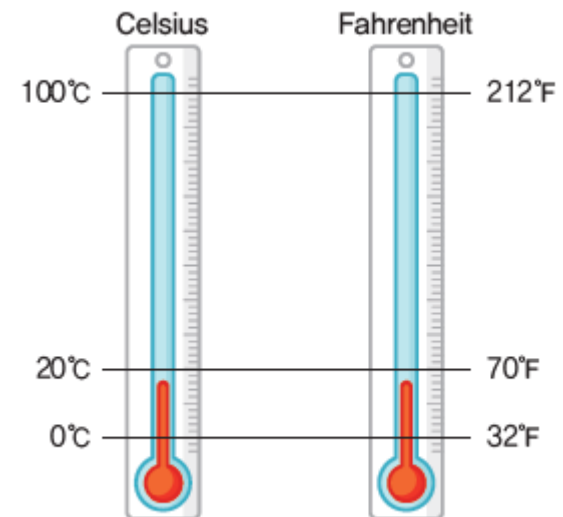




# mini project: 화씨 온도를 섭씨로 바꾸기

- 화씨 온도를 섭씨 온도로 바꾸는 프로그램을 작성하여 보자.

$$\text{섭씨온도} = \frac{5}{9}(\text{화씨온도} - 32)$$



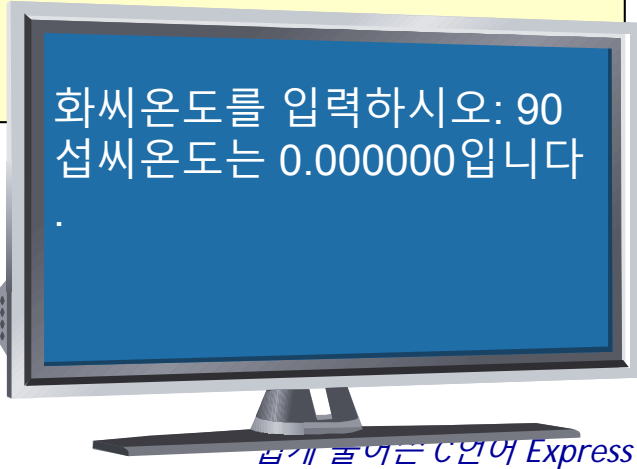


# 잘못된 부분은 어디에?

```
#include <stdio.h>
int main(void)
{
    double f_temp;
    double c_temp;

    printf("화씨온도를 입력하시오");
    scanf("%lf", &f_temp);
    c_temp = 5 / 9 * (f_temp - 32);
    printf("섭씨온도는 %f입니다, c_temp);

    return 0;
}
```



화씨온도를 입력하시오: 90  
섭씨온도는 0.000000입니다  
.



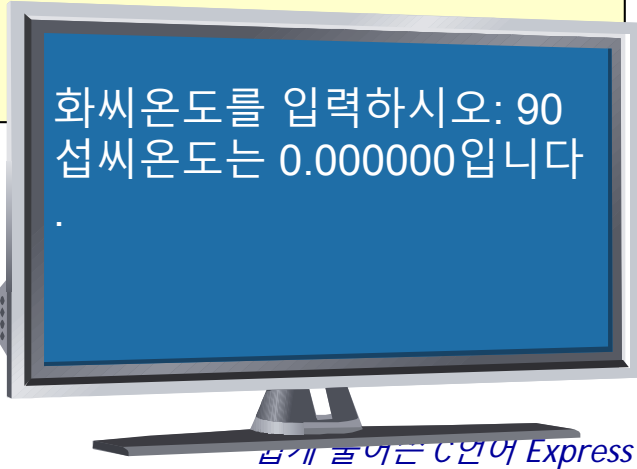
# 잘못된 부분은 어디에?

```
#include <stdio.h>
int main(void)
{
    double f_temp;
    double c_temp;

    printf("화씨온도를 입력하시오");
    scanf("%lf", &f_temp);
    c_temp = 5 / 9 * (f_temp - 32);
    printf("섭씨온도는 %f입니다, c_temp);

    return 0;
}
```

c\_temp = 5.0 / 9.0 \* (f\_temp - 32);



화씨온도를 입력하시오: 90  
섭씨온도는 0.000000입니다  
.



# 도전문제

- 위에서 제시한 방법 외에 다른 방법은 없을까?
- $((\text{double})5 / (\text{double})9) * (f\_temp - 32);$  가 되는지 확인하여 보자.
- $((\text{double})5 / 9) * (f\_temp - 32);$  가 되는지 확인하여 보자.





## Q & A

