



누구나 즐기는 C언어 콘서트

제4장 수식과 연산자





이번 장에서 학습할 내용

- 수식과 연산자란?
- 산술 연산
- 대입 연산
- 논리 연산
- 관계 연산
- 우선 순위와 결합 법칙

이번 장에서는
수식과
연산자를
살펴봅니다.





수식의 예

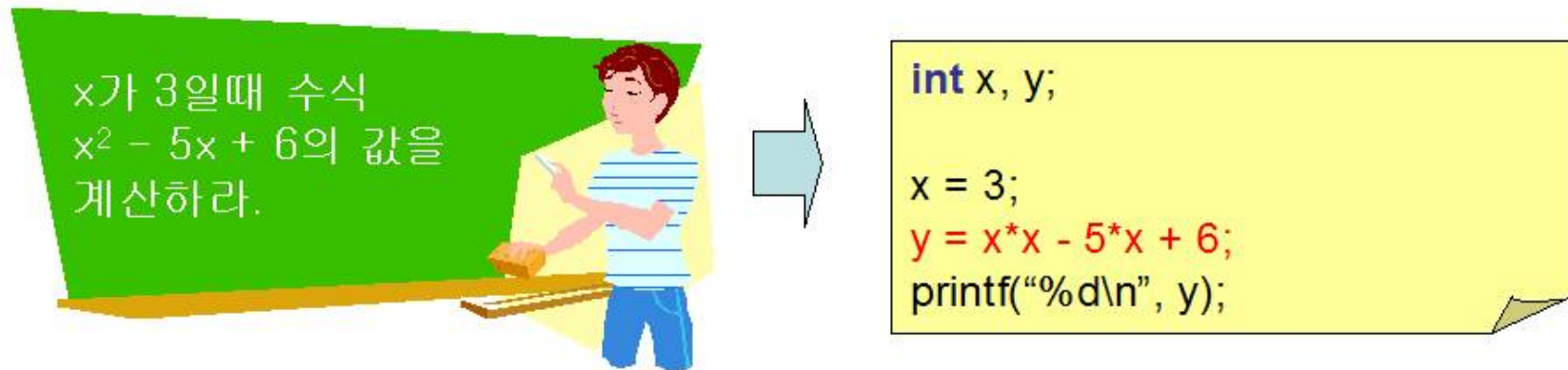
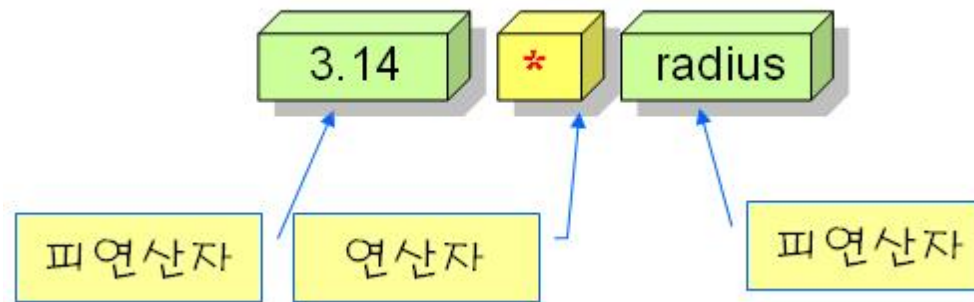


그림 4.2 수식의 예



수식의 정의

- 수식의 정의
 - 상수, 변수, 연산자의 조합
 - 연산자와 피연산자로 나누어진다.
 - 결과값을 갖는다.





간단한 수식 예제

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

```
int main()
```

```
{
```

```
    int x=1;
```

```
    printf("3+2의 값은 %d\n", 3+2);
```

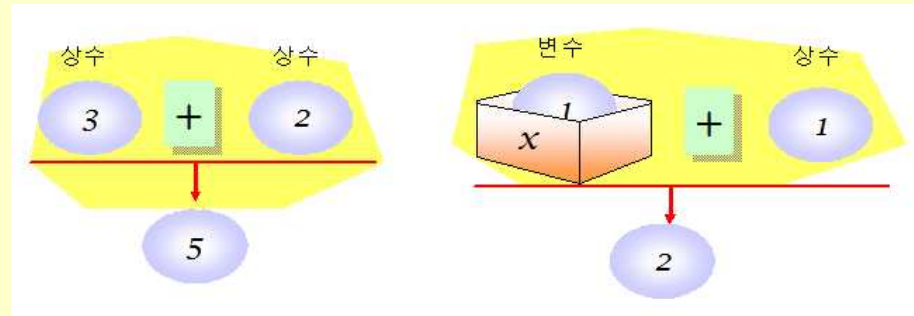
```
    printf("3-2의 값은 %d\n", 3-2);
```

```
    printf("x의 값은 %d\n", x);
```

```
    printf("x+1의 값은 %d\n", x+1);
```

```
    return 0;
```

```
}
```



3+2의 값은 5

3-2의 값은 1

x의 값은 1

x+1의 값은 2



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 단위 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



피연산자수에 따른 연산자 분류

- 단항 연산자: 피연산자의 수가 1개

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

- 이항 연산자: 피연산자의 수가 2개

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

- 삼항 연산자: 연산자의 수가 3개

```
x ? y : z
```



중간 점검

1. 수식(**expression**)이란 어떻게 정의되는가?
2. 상수 **10**도 수식이라고 할 수 있는가?
3. 아래의 수식에서 피연산자와 연산자를 구분하여 보라.
 $y = 10 + 20;$
4. 연산자를 단항 연산자, 이항 연산자, 삼항 연산자로 나누는 기준은 무엇인가?





이번 장에서 학습할 내용

- 수식과 연산자란?
- 산술 연산
- 대입 연산
- 논리 연산
- 관계 연산
- 우선 순위와 결합 법칙

이번 장에서는
수식과
연산자를
살펴봅니다.





산술 연산자

- 덧셈, 뺄셈, 곱셈, 나눗셈 등의 사칙 연산을 수행하는 연산자

연산자	기호	의미	예
덧셈	+	x와 y를 더한다	x+y
뺄셈	-	x에서 y를 뺀다.	x-y
곱셈	*	x와 y를 곱한다.	x*y
나눗셈	/	x를 y로 나눈다.	x/y
나머지	%	x를 y로 나눌 때의 나머지값	x%y



$$y = mx + b$$

$$y = m * x + b$$

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

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

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

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

(참고) 거듭 제곱 연산자는?

C에는 거듭 제곱을 나타내는 연산자는 없다.

x * x와 같이 단순히 변수를 두번 곱한다,



예제

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

```
int main()
```

```
{
```

```
    int x, y, result;
```

```
    printf("두개의 정수를 입력하시오: ");
```

```
    scanf("%d %d", &x, &y);
```

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

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

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

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

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

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

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

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

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

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

```
    return 0;
```

```
    }
```



두개의 정수를 입력하시오: 7 4

$7 + 4 = 11$

$7 - 4 = 3$

$7 * 4 = 28$

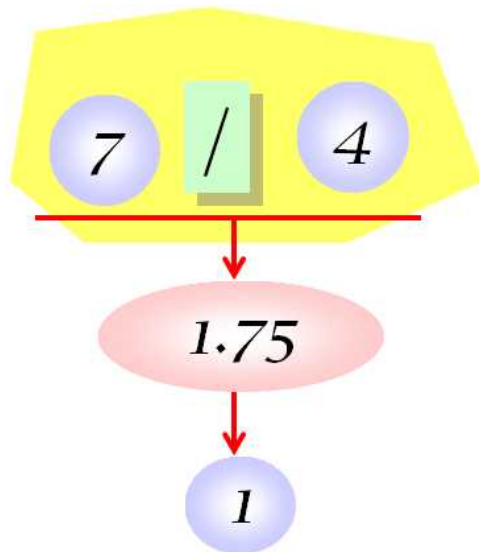
$7 / 4 = 1$

$7 \% 4 = 3$

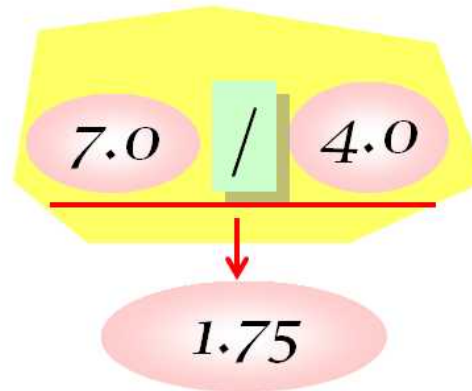


나눗셈 연산자

- 정수형끼리의 나눗셈에서는 결과가 정수형으로 생성하고 부동소수점형끼리는 부동소수점 값을 생성된다.
- 정수형끼리의 나눗셈에서는 소수점 이하는 버려진다.



정수와 정수 끼리의 나눗셈.



실수와 실수 끼리의 나눗셈.





나눗셈 연산자

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

    printf("두개의 실수를 입력하시오: ");
    scanf("%lf %lf", &x, &y);

    result = x + y; // 덧셈 연산을 하여서 결과를 result에 대입
    printf("%f / %f = %f\n", x, y, result);
    ...
    result = x / y; // 나눗셈 연산을 하여서 결과를 result에 대입
    printf("%f / %f = %f\n", x, y, result);
    return 0;
}
```



두개의 실수를 입력하시오: 7 4
7.000000 + 4.000000 = 11.000000
7.000000 - 4.000000 = 3.000000
7.000000 * 4.000000 = 28.000000
7.000000 / 4.000000 = 1.750000



나머지 연산자

- 나머지 연산자(modulus operator)는 첫 번째 피연산자를 두 번째 피연산자로 나누었을 경우의 나머지를 계산
 - $10 \% 2$ 는 0이다.
 - $5 \% 7$ 는 5이다.
 - $30 \% 9$ 는 3이다.
- (예) 나머지 연산자를 이용한 짝수와 홀수를 구분
 - $x \% 2$ 가 0이면 짝수
- (예) 나머지 연산자를 이용한 5의 배수를 판단
 - $x \% 5$ 가 0이면 5의 배수

아주
유용한
연산자입니
다.





나머지 연산자

```
// 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;
}
```

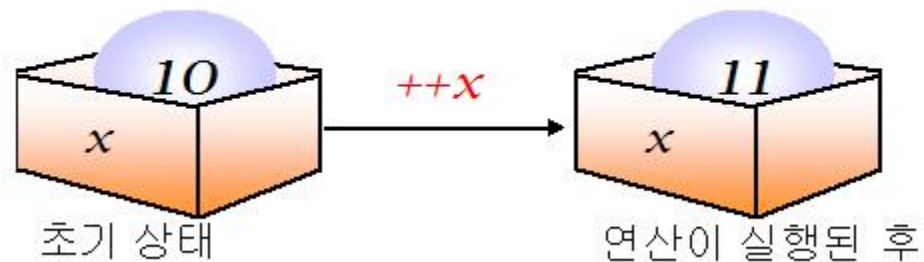


초단위의 시간을 입력하시요:(32억초이하) 70
70초는 1분 10초입니다.



증감 연산자

증감 연산자	의미
$++x$	x값을 먼저 증가한 후에 다른 연산에 사용한다. 이 수식의 값은 증가된 x값이다.
$x++$	x값을 먼저 사용한 후에, 증가한다. 이 수식의 값은 증가되지 않은 원래의 x값이다.
$--x$	x값을 먼저 감소한 후에 다른 연산에 사용한다. 이 수식의 값은 감소된 x값이다.
$x--$	x값을 먼저 사용한 후에, 감소한다. 이 수식의 값은 감소되지 않은 원래의 x값이다.





주의할 점

- `x = 1;`
- `y = 1;`
- `nextx = ++x;` // `x`의 값이 증가된 후에 사용된다. `nextx`는 2가 된다.
- `nexty = y++;` // `y`의 값이 사용된 후에 증가된다. `nexty`는 1이 된다.



증감 연산자

```
#include <stdio.h>
int main(void)
{
    int x=1, nextx=0;
    x = 0;
    nextx = ++x;
    printf("nextx=%d, x=%d\n", nextx, x);

    x = 0;
    nextx = x++;
    printf("nextx=%d, x=%d\n", nextx, x);

    x = 0;
    nextx = --x;
    printf("nextx=%d, x=%d\n", nextx, x);

    x = 0;
    nextx = x--;
    printf("nextx=%d, x=%d\n", nextx, x);
    return 0;
}
```



nextx=1, x=1
nextx=0, x=1
nextx=-1, x=-1
nextx=0, x=-1



이번 장에서 학습할 내용

- 수식과 연산자란?
- 산술 연산
- **대입 연산**
- 논리 연산
- 관계 연산
- 우선 순위와 결합 법칙

이번 장에서는
수식과
연산자를
살펴봅니다.



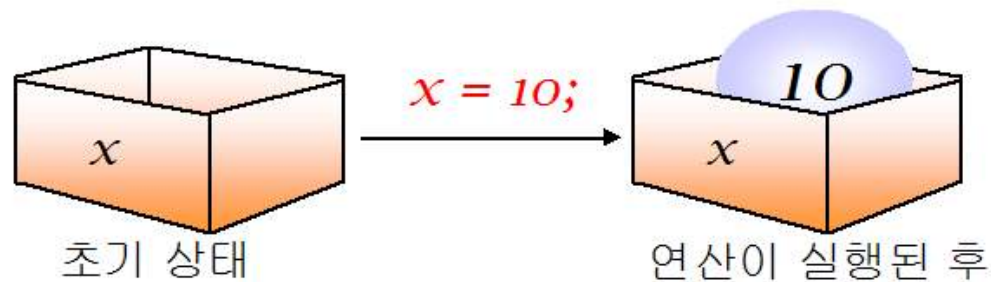


대입(배정, 할당) 연산자

- 왼쪽에 있는 변수에 오른쪽의 수식의 값을 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.
```





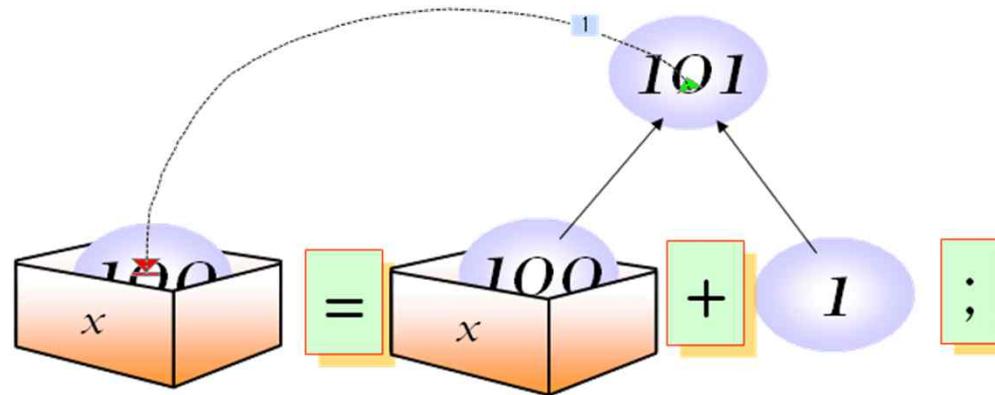
대입 연산자 주의점

- 왼쪽에는 항상 변수가 와야 한다.

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

- 다음의 문장은 수학적으로는 올바르지 않지만 **C**에서는 가능.

$x = x + 1;$ // x의 값이 하나 증가 된다.





대입 연산의 결과값

덧셈연산의 결과값은 9

$$x = 2 + 7;$$

대입연산의 결과값은 3(현재는 사용되지 않음)

대입 연산의 결과값은 1

$$y = x = 1;$$

대입 연산의 결과값은 1(현재는 사용되지 않음)

모든 연산에는
결과값이 있고
대입 연산도
결과값이
있습니다.





예제

```
/* 대입 연산자 프로그램 */
```

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

```
int main(void)
```

```
{
```

```
    int x, y;
```

```
    x = 1;
```

```
    printf("수식 x+1의 값은 %d\n", x+1);
```

```
    printf("수식 y=x+1의 값은 %d\n", y=x+1);
```

```
    printf("수식 y=10+(x=2+7)의 값은 %d\n", y=10+(x=2+7));
```

```
    printf("수식 y=x=3의 값은 %d\n", y=x=3);
```

```
    return 0;
```

```
}
```

수식의 결과값을
출력하여 보는
예제입니다.



수식 x+1의 값은 2

수식 y=x+1의 값은 2

수식 y=10+(x=2+7)의 값은 19

수식 y=x=3의 값은 3



복합 대입 연산자

- 복합 대입 연산자란 +=처럼 대입연산자 =와 산술연산자를 합쳐 놓은 연산자
- 소스를 간결하게 만들 수 있음

복합 대입 연산자	의미
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% = y$	$x = x \% y$
$x \& = y$	$x = x \& y$
$x = y$	$x = x y$
$x \wedge = y$	$x = x \wedge y$
$x \gg = y$	$x = x \gg y$
$x \ll = y$	$x = x \ll y$

```
x += 1      // x = x + 1
x *= y + 1   // x = x * (y + 1)
x %= x + y   // x = x % (x + y)
```




복합 대입 연산자



// 복합 대입 연산자 프로그램

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

```
int main(void)
```

```
{
```

```
    int x = 10, y = 10, z = 33;
```

```
    x += 1;        // x = x + 1;
```

```
    y *= 2;        // y = y * 2;
```

```
    z %= x + y;    // z = z % (x + y); 주의!!
```

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

```
    return 0;
```

```
}
```



x = 11 y = 20 z = 1



이번 장에서 학습할 내용

- 수식과 연산자란?
- 산술 연산
- 대입 연산
- **관계 연산**
- 논리 연산
- 우선 순위와 결합 법칙

이번 장에서는
수식과
연산자를
살펴봅니다.

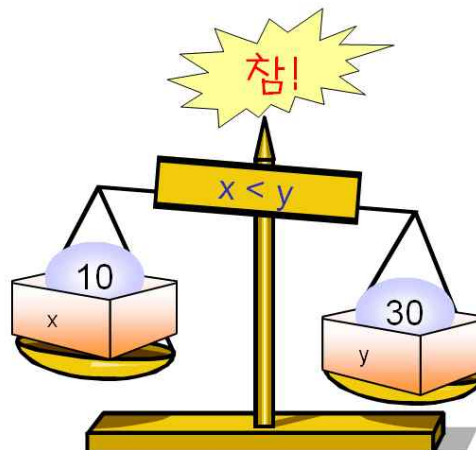




관계 연산자

- 두개의 피연산자를 비교하는 연산자
- 결과값은 참(1) 아니면 거짓(0)

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





사용예

- $1 == 2$ // 1과 2가 같으므로 참(1)
- $1 != 2$ // 1와 2가 다르므로 참(1)
- $2 < 1$ // 2가 1보다 작지 않으므로 거짓(0)
- $x \geq y$ // x가 y보다 크거나 같으면 참(1) 그렇지 않으면 거짓(0)



예제

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

    printf("두개의 정수를 입력하시오: ");
    scanf("%d %d", &x, &y);

    printf("x == y의 결과값: %d\n", x == y);
    printf("x != y의 결과값: %d\n", x != y);
    printf("x > y의 결과값: %d\n", x > y);
    printf("x < y의 결과값: %d\n", x < y);
    printf("x >= y의 결과값: %d\n", x >= y);
    printf("x <= y의 결과값: %d\n", x <= y);

    return 0;
}
```



두개의 정수를 입력하시오: 3 4
x == y의 결과값: 0
x != y의 결과값: 1
x > y의 결과값: 0
x < y의 결과값: 1
x >= y의 결과값: 0
x <= y의 결과값: 1



주의할 점!

- $(x = y)$
 - x 의 값을 y 에 대입한다. 이 수식의 값은 x 의 값이다.
- $(x == y)$
 - x 와 y 가 같으면 **1**, 다르면 **0**이 수식의 값이 된다.
- `if(x==y)`를 `if(x=y)`로 잘못 쓰지 않도록 주의!



중간 점검

1. 관계 수식의 결과로 생성될 수 있는 값은 무엇인가?
2. $(3 \geq 2) + 5$ 의 값은?





이번 장에서 학습할 내용

- 수식과 연산자란?
- 산술 연산
- 대입 연산
- 관계 연산
- 논리 연산
- 우선 순위와 결합 법칙

이번 장에서는
수식과
연산자를
살펴봅니다.

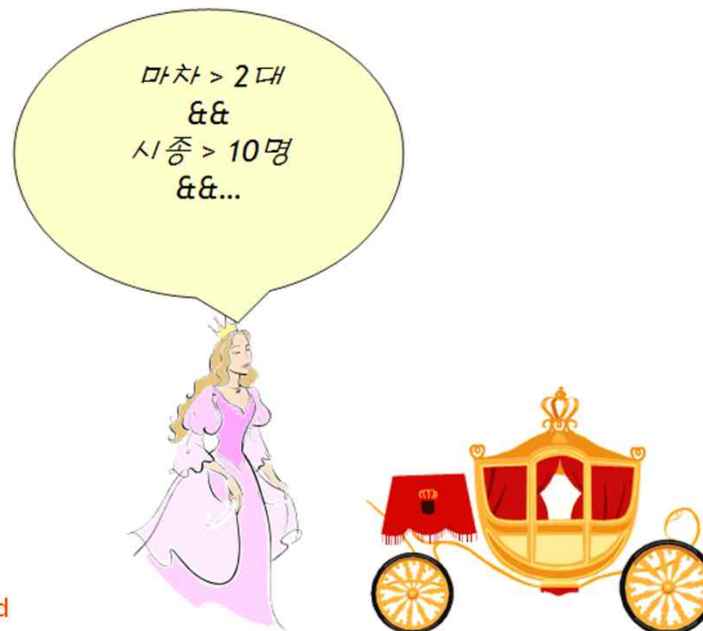




논리 연산자

- 여러 개의 조건을 조합하여 참과 거짓을 따지는 연산자
- 결과값은 참(1) 아니면 거짓(0)

사용예	의미
$x \ \&\& \ y$	AND 연산, x 와 y 가 모두 참이면 참, 그렇지 않으면 거짓
$x \ \ y$	OR 연산, x 나 y 중에서 하나만 참이면 참, 모두 거짓이면 거짓
$!x$	NOT 연산, x 가 참이면 거짓, x 가 거짓이면 참





논리 연산의 결과값

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



참과 거짓의 표현 방법

- 기본적으로 참(**true**)은 1로, 거짓(**false**)은 0로 나타낸다.
- 관계 수식이나 논리 수식이 거짓으로 계산되면 0을 생성하고 참으로 계산되면 1을 생성한다.
- 하지만 피연산자의 참, 거짓을 가릴 때에는 0이 아니면 참이고 0이면 거짓으로 판단한다.-> 주의!!
- (예) -1도 참으로 간주.



AND 연산자

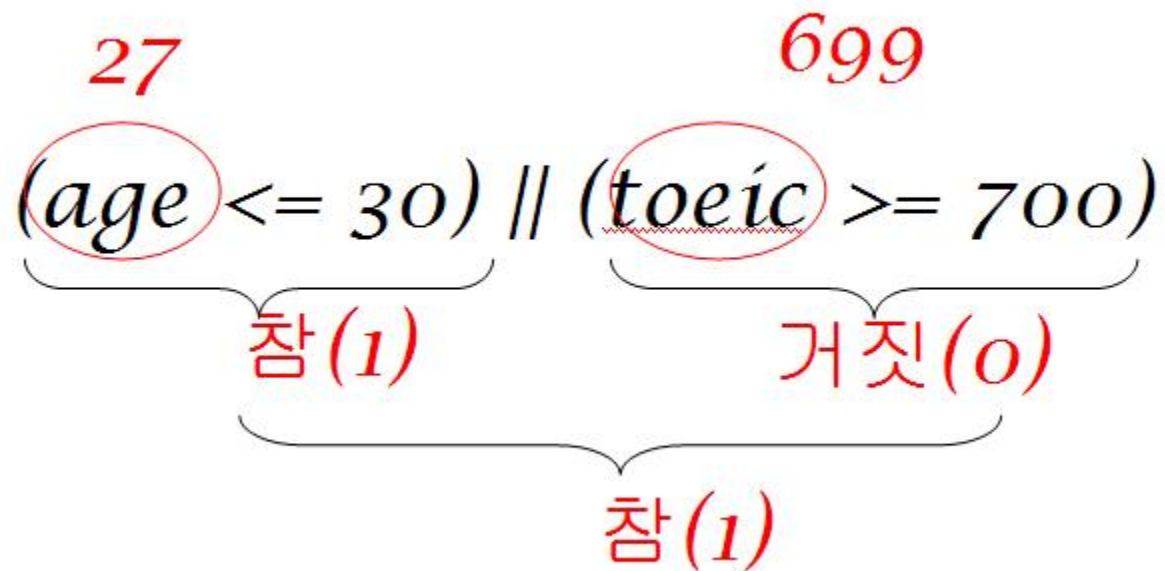
- 두 개의 피연산자가 모두 참일 때만 연산 결과가 참이 된다

$$\begin{array}{c} 27 \qquad \qquad \qquad 800 \\ \underbrace{(age \leq 30)}_{\text{참}(1)} \ \&\& \ \underbrace{(toeic \geq 700)}_{\text{참}(1)} \\ \underbrace{\hspace{10em}}_{\text{참}(1)} \end{array}$$



OR 연산자

- 하나의 피연산자만 참이면 연산 결과가 참이 된다





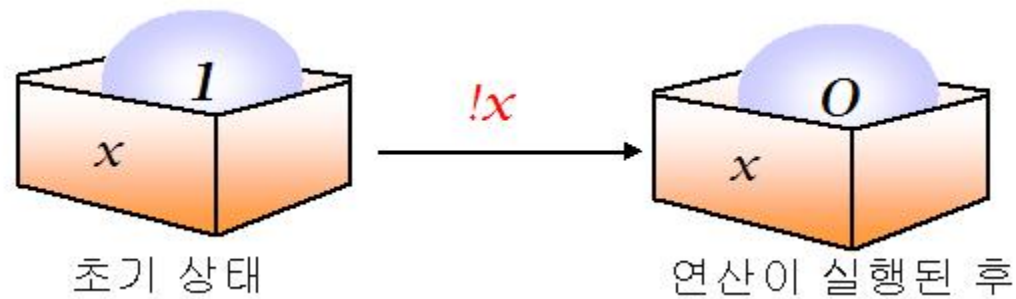
주의할 점

- $(2 < x < 5)$
 - 가능하지만 논리적으로 잘못된 수식
 - $((2 < x) < 5)$ 으로 계산된다.
- $(2 < x) \&\& (x < 5)$
 - 올바른 수식



NOT 연산자

- 피연산자의 값이 참이면 연산의 결과값을 거짓으로 만들고, 피연산자의 값이 거짓이면 연산의 결과값을 참으로 만든다.

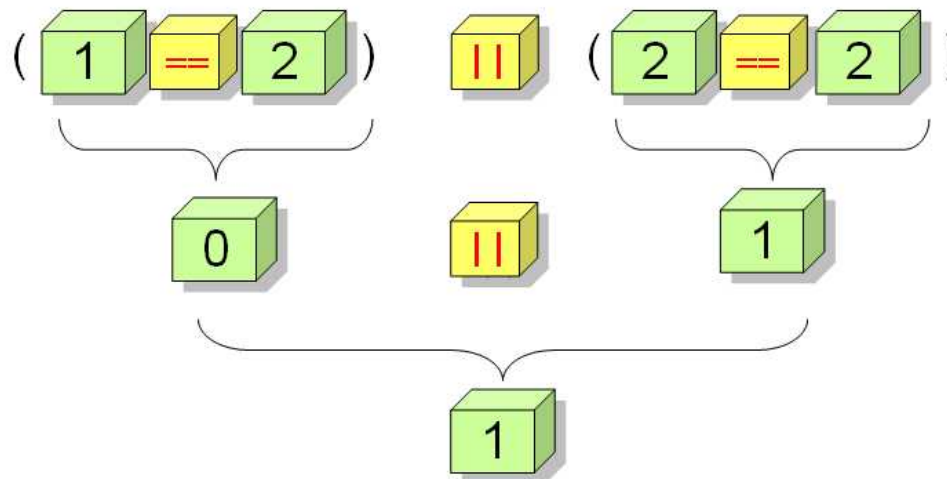


- `result = !1;` // result에는 0가 대입된다.
- `result = !(2==3);` // result에는 1이 대입된다.



논리 연산자의 계산 과정

- 논리 연산의 결과값은 항상 1 또는 0이다.
- (예)



0이 아닌 값을
참으로
취급하지만 논리
연산의 결과값은
항상 1 또는
0입니다.

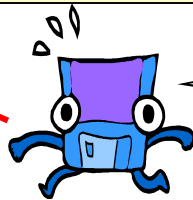




단축 계산

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

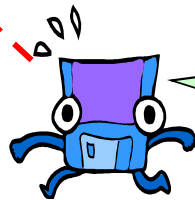
`(2 > 3) && (++x < 5)`



첫번째 연산자가
거짓이면 두번째
연산자는 계산할
필요가 없겠군!!.

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

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



첫번째 연산자가
참이면 다른
연산자는 계산할
필요가 없겠군!!.



예제 #1

```
#include <stdio.h>
int main(void)
{

    int x, y;

    printf("두개의 정수를 입력하시오: ");
    scanf("%d%d", &x, &y);

    printf("%d && %d의 결과값: %d\n", x, y, x && y);
    printf("%d || %d의 결과값: %d\n", x, y, x || y);
    printf("!%d의 결과값: %d\n", x, !x);

    return 0;
}
```



두개의 정수를 입력하시오: 1 0
1 && 0의 결과값: 0
1 || 0의 결과값: 1
!1의 결과값: 0



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.

$x < 0 \parallel x > 10$

$x > 5 \parallel x < 10 \ \&\& \ x > 0$

$(x > 5 \parallel x < 10) \ \&\& \ x > 0$

// $x > 5 \parallel (x < 10 \ \&\& \ x > 0)$ 와 동일

정신적인
안정을
위하여
괄호를
사용하여도
됩니다.





예제 #2

- 윤년을 판단하는 문제
 - ① 4로 나누어 떨어지는 연도 중에서 100으로 나누어 떨어지는 연도는 제외한다.
 - ② 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;
}
```



중간 점검

1. 다음의 조건에 해당하는 논리 연산식을 만들어 보시오. 변수는 적절하게 선언되어 있다고 가정한다.
“무주택 기간 **3**년 이상, 가구주의 연령이 **40**세 이상, 가족의 수가 **3**명 이상”
2. 상수 **10**은 참인가 거짓인가?
3. 수식 **!3**의 값은?
4. 단축 계산의 예를 들어보라.

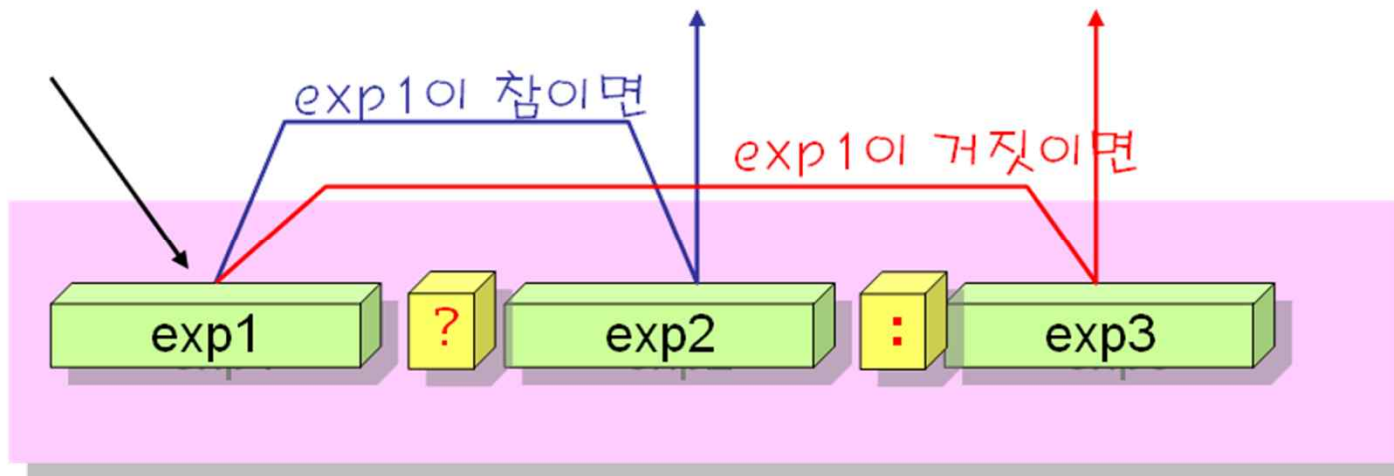




조건 연산자

- `exp1`가 참이면 `exp2`를 반환, 그렇지 않으면 `exp3`를 반환

`exp1 ? exp2 : exp3`



`absolute_value = (x > 0) x: -x; // 절대값 계산`

`max_value = (x > y) x: y; // 최대값 계산`

`min_value = (x < y) x: y; // 최소값 계산`



예제

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

    printf("첫 번째 수=");
    scanf("%d", &x);
    printf("두 번째 수=");
    scanf("%d", &y);

    printf("큰 수=%d \n", (x > y) ? x : y);
    printf("작은 수=%d \n", (x < y) ? x : y);
}
```



첫 번째 수=2
두 번째 수=3
큰 수=3
작은 수=2



이번 장에서 학습할 내용

- 수식과 연산자란?
- 산술 연산
- 대입 연산
- 관계 연산
- 논리 연산
- 우선 순위와 결합 법칙

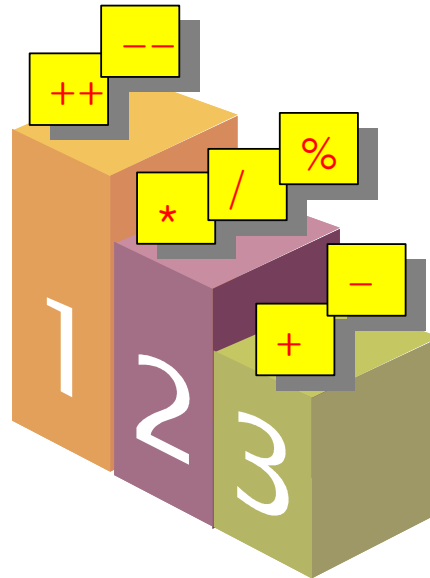
여기에서는
우선순위와
결합법칙을
살펴봅니다.





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 $y * z$, and a larger bracket labeled ② groups the entire expression $x + y * z$.

$$(x + y) * z$$

Diagram illustrating operator precedence for the expression $(x + y) * z$. A bracket labeled ① groups $x + y$, and a larger bracket labeled ② groups the entire expression $(x + y) * z$.



우선 순위

- 수식에서 어떤 연산자를 먼저 계산할 것인지의 문제

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)



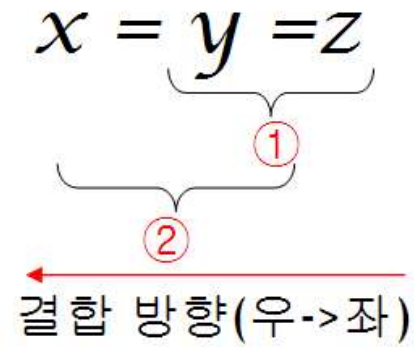
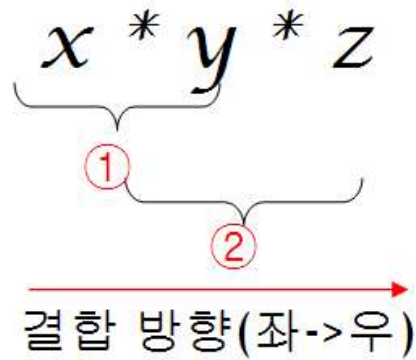
우선 순위의 일반적인 지침

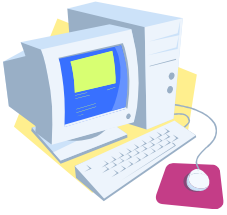
- 콤마 < 대입 < 논리 < 관계 < 산술 < 단항
- 괄호 연산자는 가장 우선순위가 높다.
- 모든 단항 연산자들은 이항 연산자들보다 우선순위가 높다.
- 콤마 연산자를 제외하고는 대입 연산자가 가장 우선순위가 낮다.
- 연산자들의 우선 순위가 생각나지 않으면 괄호를 이용
 - $(x \leq 10) \&\& (y \geq 20)$
- 관계 연산자나 논리 연산자는 산술 연산자보다 우선순위가 낮다.
 - $x + 2 == y + 3$



결합 규칙

- 동일한 우선 순위를 가지는 연산들이 여러 개가 있으면 어떤 것을 먼저 수행하여야 하는가에 대한 규칙





예제

$$y = \underline{a \% b} / c + d * \underline{(e - f)};$$

Diagram illustrating the evaluation order of the expression $y = a \% b / c + d * (e - f);$ using numbered annotations and red underlines:

- ①: Underline for the sub-expression $(e - f)$.
- ②: Underline for the sub-expression $a \% b$.
- ③: Underline for the sub-expression $a \% b / c$.
- ④: Underline for the sub-expression $d * (e - f)$.
- ⑤: Underline for the sub-expression $a \% b / c + d * (e - f)$.
- ⑥: Underline for the entire expression $y = a \% b / c + d * (e - f);$.



예제

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

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

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

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

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

    return 0;
}
```



```
0
1
1
-1
```



중간 점검

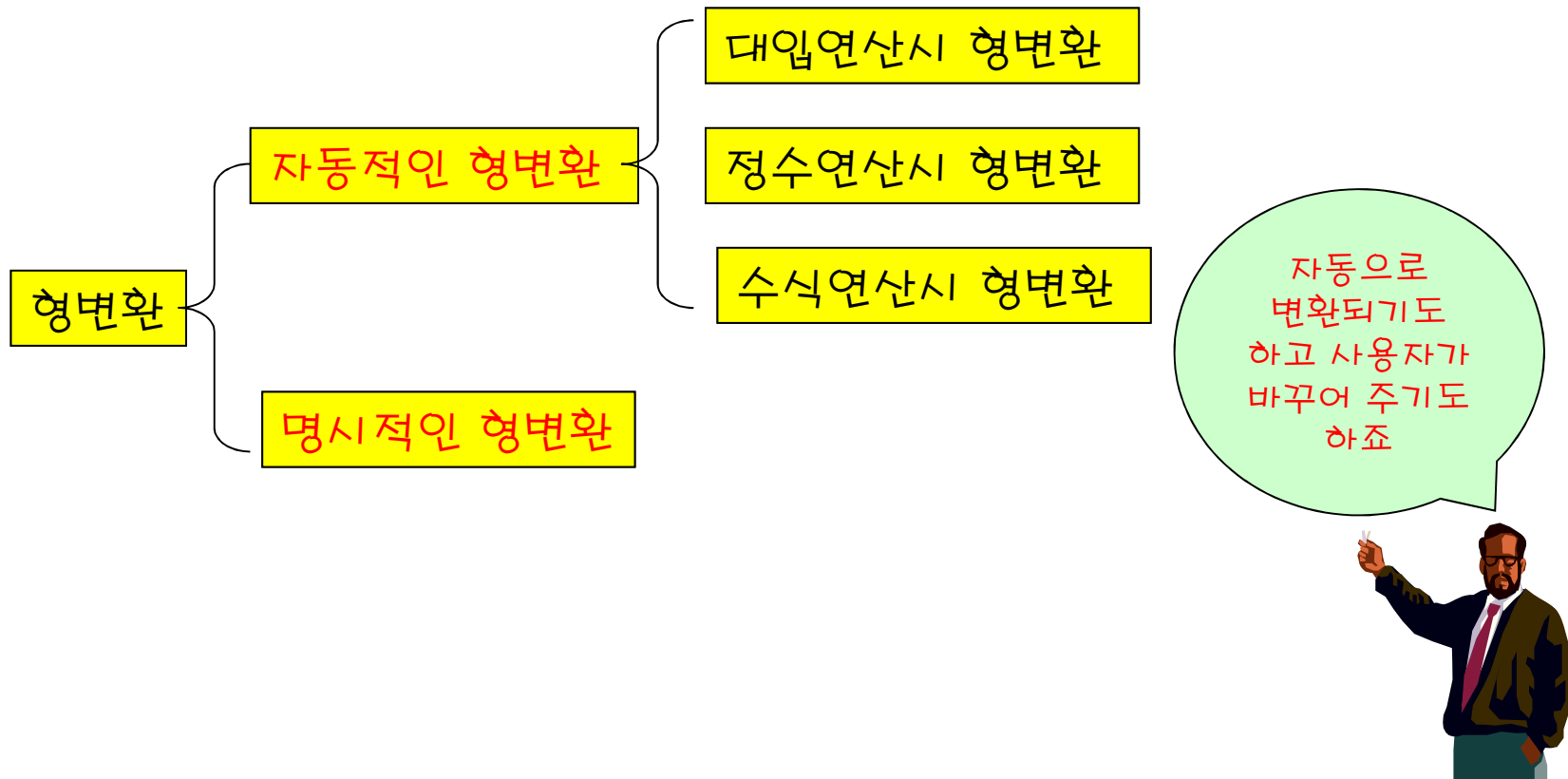
1. 연산자 중에서 가장 우선 순위가 낮은 연산자는 무엇인가?
2. 논리 연산자인 `&&`과 `||` 중에서 우선 순위가 더 높은 연산자는 무엇인가?
3. 단항 연산자와 이항 연산자 중에서 어떤 연산자가 더 우선 순위가 높은가?
4. 관계 연산자와 산술 연산자 중에서 어떤 연산자가 더 우선 순위가 높은가?





형 변환

- 연산시에 데이터의 유형이 변환되는 것

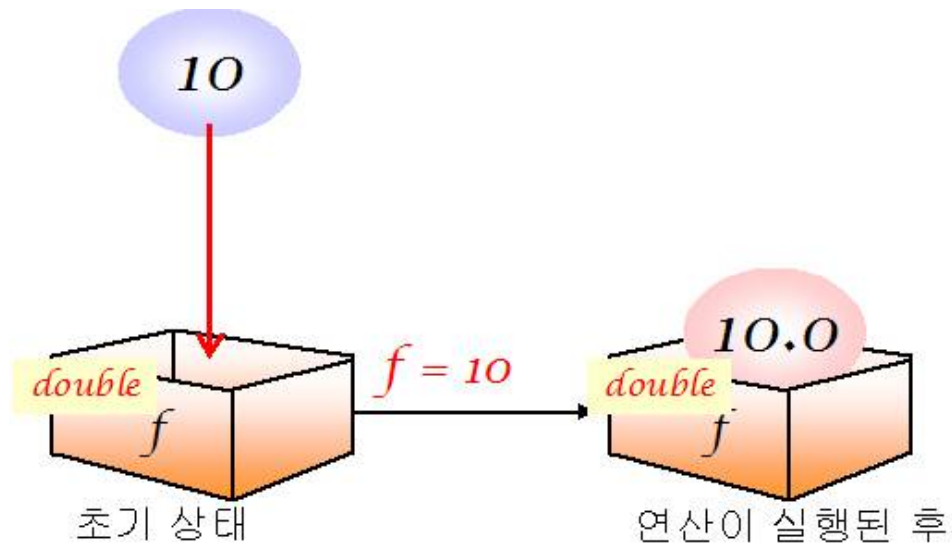




대입 연산시의 자동적인 형변환

- 올림 변환

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

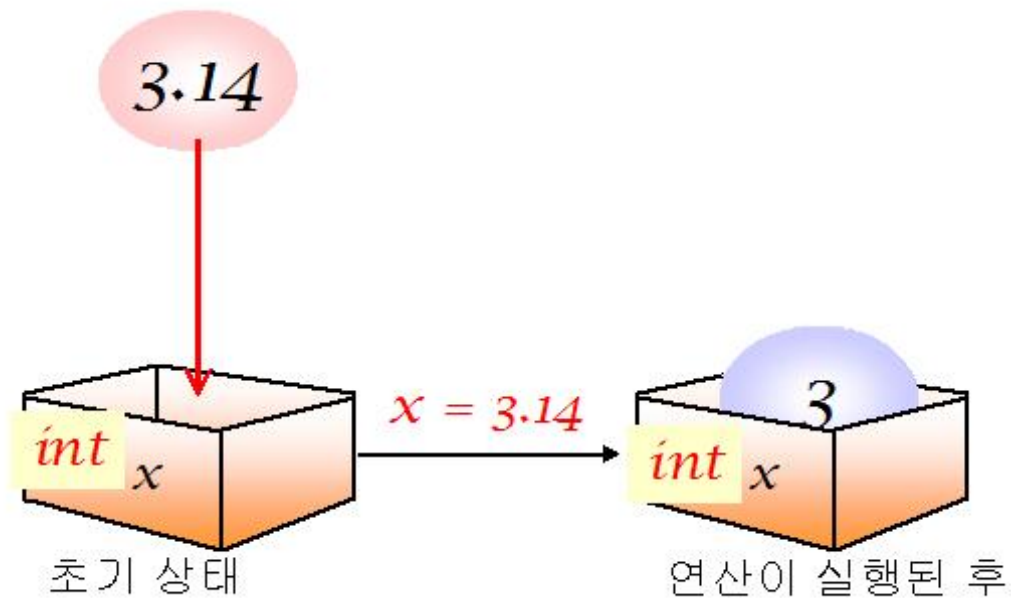




대입 연산시의 자동적인 형변환

- 내림 변환

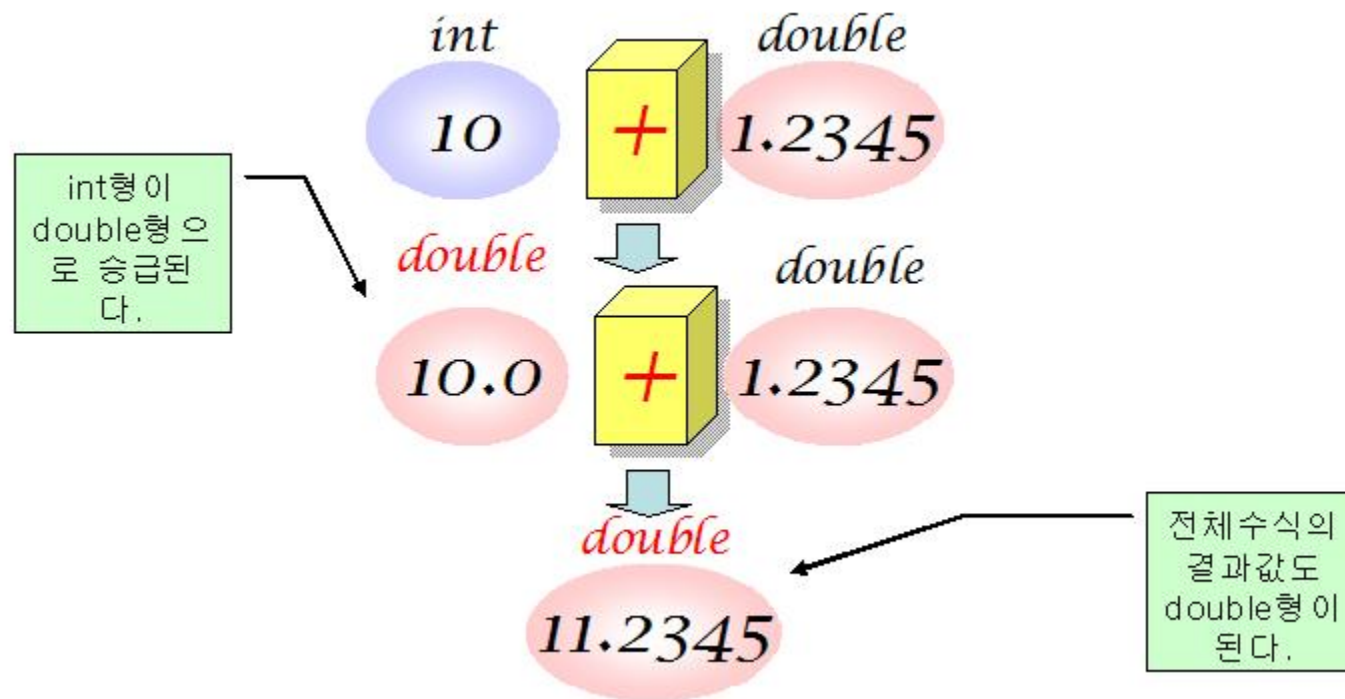
```
int i;  
i = 3.141592;           // i에는 3이 저장된다.
```





수식에서의 자동적인 형변환

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





명시적인 형변환

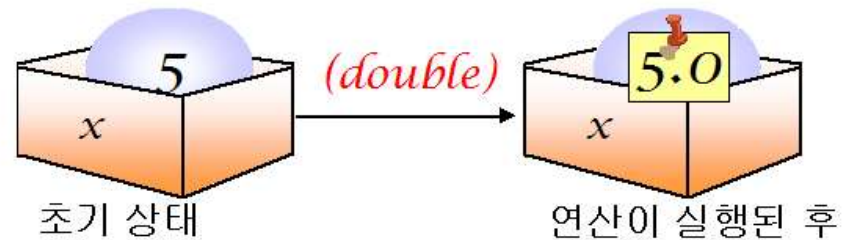
- 형변환(**type cast**): 사용자가 데이터의 타입을 변경하는 것

(자료형) 상수 또는 변수

(int)1.23456 // 상수

(double) x // 변수

(long) (x+1) // 수식





예제

```
f = 5 / 4;           // f는1  
printf("%f", f);
```

```
f = (double)5/ 4;    //f는1.25  
printf("%f", f);
```

```
f = 5 / (double)4;    // f는1.25  
printf("%f", f);
```

```
f = (double)5/ (double)4; // f는1.25  
printf("%f", f);
```

```
i = 1.3 + 1.8;        // i는3  
printf("%d", i);
```

```
i = (int)1.3+ (int)1.8; // i는2  
printf("%d", i);
```



```
1.000000  
1.250000  
1.250000  
1.250000  
3  
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!

어떤
문장이던지
순차적으로
실행됩니다.





중간 점검

1. $10\%6$ 의 값은?
2. $10/6$ 의 값은 얼마인가?
3. 복합 대입 연산자 $x *= y$ 의 의미를 설명하라.
4. 증감 연산자 $x--$ 를 풀어쓰면 어떻게 되는가?
5. 내림 변환과 올림 변환을 설명하라.
6. `int`형 변수 x 를 `double`형으로 형변환하는 문장을 써보라.
7. 하나의 수식에 정수와 부동소수점수가 섞여 있으면 어떻게 되는가?





Q & A

