



CH03 수식과 연산자



이번 장에서 학습할 내용

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

이번 장에서는 수식과
연산자를 살펴봅니다.





수식의 예



```
int x, y;
```

```
x = 3;
```

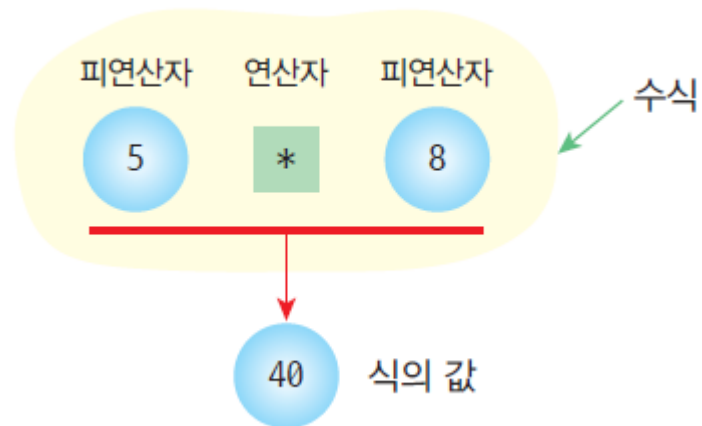
```
y = x*x - 5*x + 6;
```

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



수식

- 수식(expression)
 - 상수, 변수, 연산자의 조합
 - 연산자와 피연산자로 나누어진다.





기능에 따른 연산자의 분류

연산자의 분류	연산자	의미
대입	=	오른쪽을 왼쪽에 대입
산술	+ - * / %	사칙연산과 나머지 연산
부호	+ -	양수와 음수 표시
증감	++ --	증가, 감소 연산
관계	> < == != >= <=	오른쪽과 왼쪽을 비교
논리	&& !	논리적인 AND, OR
조건	?	조건에 따라 선택
콤마	,	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



산술 연산자

- 산술 연산: 컴퓨터의 가장 기본적인 연산
- 덧셈, 뺄셈, 곱셈, 나눗셈 등의 사칙 연산을 수행하는 연산자

연산자	기호	사용예	결과값
덧셈	+	$7 + 4$	11
뺄셈	-	$7 - 4$	3
곱셈	*	$7 * 4$	28
나눗셈	/	$7 / 4$	1
나머지	%	$7 \% 4$	3



산술 연산자의 예

$$y = mx + b \quad \rightarrow \quad y = m * x + b;$$

$$y = ax^2 + bx + c \quad \rightarrow \quad y = a * x * x + b * x + c;$$

$$m = \frac{x + y + z}{3} \quad \rightarrow \quad m = (x + y + z) / 3;$$



(참고) 거듭 제곱 연산자는?

C에는 거듭 제곱을 나타내는 연산자는 없다.
 $x * x$ 와 같이 단순히 변수를 두 번 곱한다.



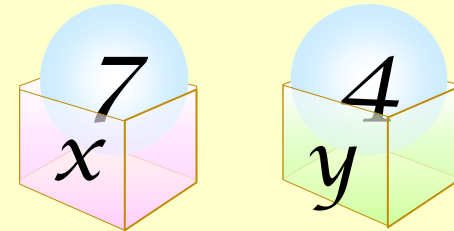
정수 사칙 연산

ch03_ex1.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void)
{
    int x, y, result;
    printf("두개의 정수를 입력하시오: ");
    scanf("%d %d", &x, &y);

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

    result = x - y;           // 뺄셈
    printf("%d - %d = %d", x, y, result);
    result = x * y;           // 곱셈
    printf("%d * %d = %d", x, y, result);
    result = x / y;           // 나눗셈
    printf("%d / %d = %d", x, y, result);
    result = x % y;           // 나머지
    printf("%d %% %d = %d", x, y, result);
    return 0;
}
```

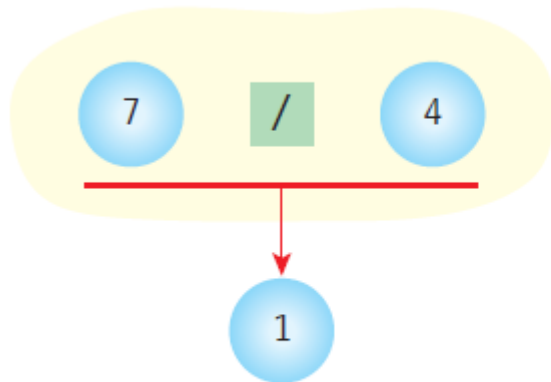


두개의 정수를 입력하시오: 7 4
7 + 4 = 11
7 - 4 = 3
7 * 4 = 28
7 / 4 = 1
7 % 4 = 3

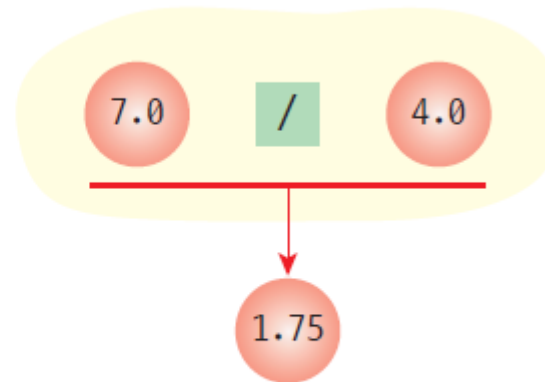


나눗셈 연산자

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



정수와 정수끼리의 나눗셈



실수와 실수끼리의 나눗셈

형변환에서
자세히
학습합니다





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

int main()
{
    double x, y, result;
    printf("두개의 실수를 입력하세요: ");
    scanf("%lf %lf", &x, &y);

    result = x / y;
    printf("%f / %f = %f", x, y, result);

    return 0;
}
```

두개의 실수를 입력하시오: 7 4
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 3의 배수 판단
 - $x \% 3$ 이 0이면 3의 배수

아주
유용한
연산자
입니다.





나머지 연산자

ch03_ex3.c

```
#define _CRT_SECURE_NO_WARNINGS
```

```
// 나머지 연산자 프로그램
```

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

```
int main(void)
```

```
{
```

```
    int input, minute, second;
```

```
    printf( " 초를 입력하시요: ");
```

```
    scanf("%d", &input);          // 초단위의 시간을 읽는다.
```

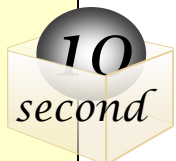
```
    minute = input / 60; // 몇 분
```

```
    second = input % 60; // 몇 초
```

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

```
    return 0;
```

```
}
```



초를 입력하시요: 1000
1000초는 16분 40초 입니다

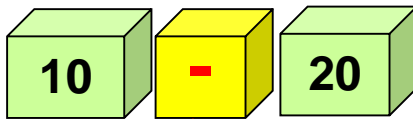


부호 연산자

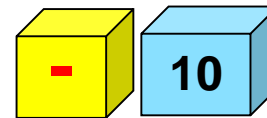
- 변수나 상수의 부호를 변경

```
x = -10;
```

```
y = -x; // 변수 y의 값은 10이 된다.
```



이항연산자



단항연산자

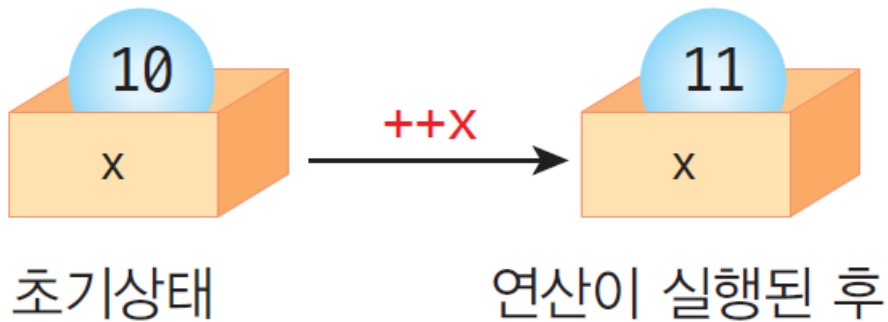
-는 이항
연산자이기도
하고 단항
연산자이기도
하죠





증감 연산자

- 증감 연산자: $++$, $--$
- 변수의 값을 하나 증가시키거나 감소시키는 연산자
- (예) $++x$, $--x$;





증감 연산자 정리

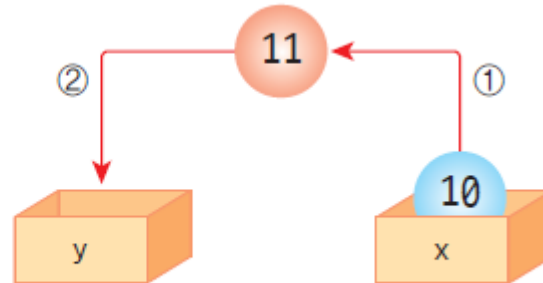
증감 연산자	차이점
$++x$	수식의 값은 증가된 x 값이다.
$x++$	수식의 값은 증가되지 않은 원래의 x 값이다.
$--x$	수식의 값은 감소된 x 값이다.
$x--$	수식의 값은 감소되지 않은 원래의 x 값이다.



++x와 x++의 차이

`y=++x;`

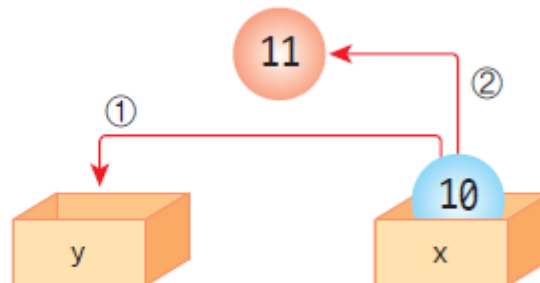
`x=x+1;`
`y=x;`



증가된 x의 값이 y에 대입된다.

`y=x++;`

`y=x;`
`x=x+1;`



먼저 대입하고 나중에 증가한다.



예제: 증감 연산자

ch03_ex4.c

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

```
int main(void)
```

```
{
```

```
    int x = 10, y = 10, res= 0;
```

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

```
    res = ++x;
```

```
    printf("res 의 값=%d\n", res);
```

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

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

```
    res = y++;
```

```
    printf("res 의 값=%d\n", res);
```

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

```
    return 0;
```

```
}
```

먼저 증가하고 증가된 값이
수식에 사용된다.

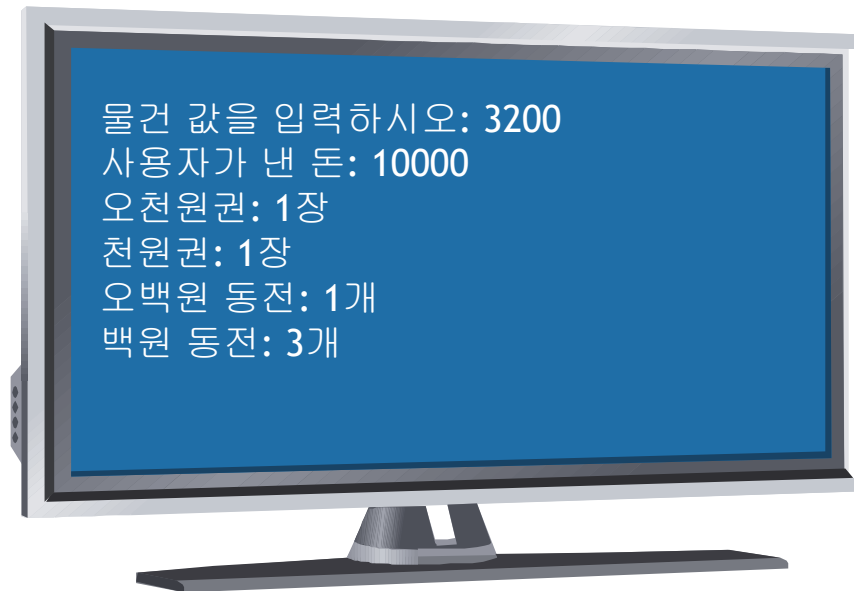
현재 값을 먼저 수식에 사용
하고 나중에 증가된다.

```
x=10  
res 의 값=11  
x=11  
y=10  
res 의 값=10  
y=11
```



Lab1: 거스름돈 계산하기

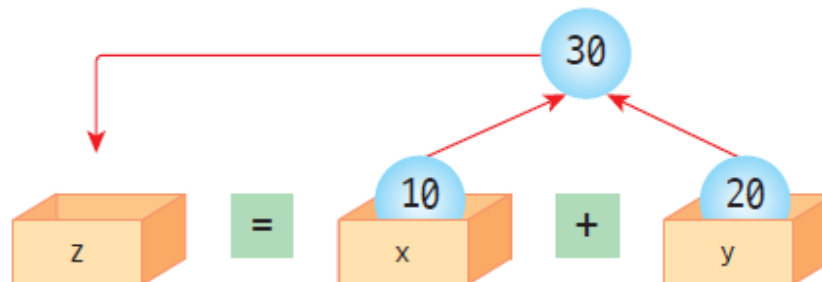
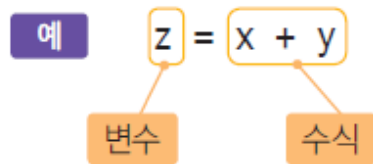
- 편의점에서 물건을 구입하고 만 원을 냈을 때, 거스름돈의 액수와 점원이 지급해야 할 거스름돈을 화폐와 동전수를 계산하는 프로그램을 작성해보자.





대입(배정, 할당) 연산자

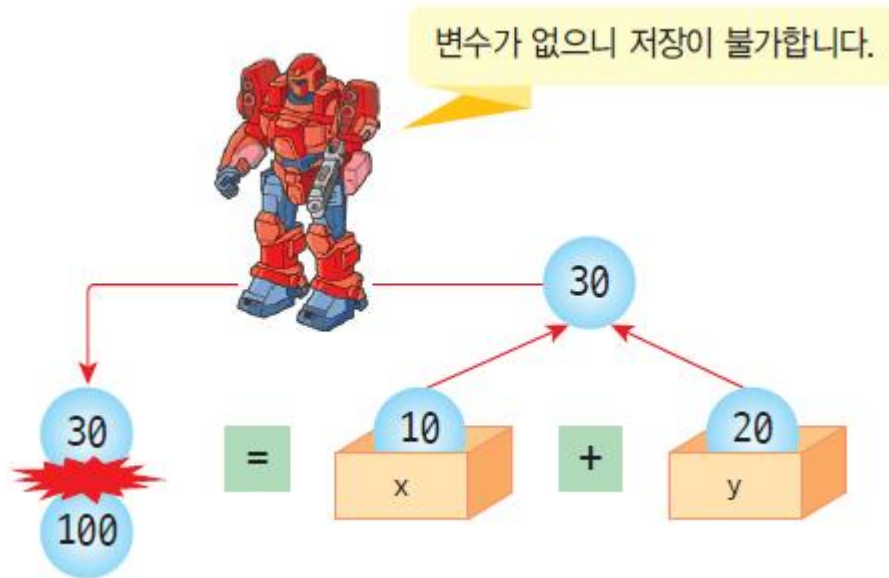
Syntax: 대입 연산자





대입 연산자 주의점

- $100 = x + y;$ // 컴파일 오류!

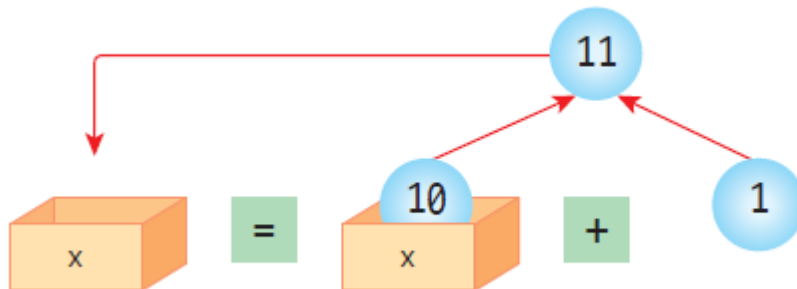




대입 연산자 주의점

수학적으로는 올바르지 않지만
C에서는 올바른 문장임

$x = x + 1;$





다음과 같은 문장도 가능하다.

$y = x = 3;$

여러 변수에다가 같은 값을 대입하는 문장을 다음과 같이 작성할 수 있다. 여기서는 먼저 $x = 3$ 이 수행되고 그 결과값인 3이 다시 y 에 대입된다



복합 대입 연산자

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

$x = x + y$ 와 의미가 같음!

$x += y$



복합 대입 연산자

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

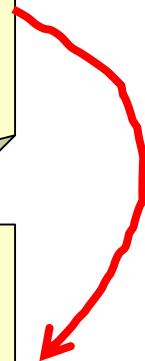


Quiz

- 다음 수식을 풀어서 다시 작성하면?

$x *= y + 1$
 $x \% = x + y$

$x = x * (y + 1)$
 $x = x \% (x + y)$





복합 대입 연산자

ch03_ex7.c

```
#include <stdio.h>

int main(void)
{
    int x = 10, y = 10;
    int i = 0, j = 0;

    i = i + 1;
    printf("수식 i의 값은 %d\n", i);
    i = j = 3;
    printf("수식 i=j=3 -> i=%d, j=%d\n", i , j);

    x += 1;
    y *= 2;
    printf("x = %d    y = %d\n", x, y);
    return 0;
}
```

수식 i 의 값은 1
수식 $i=j=3 \rightarrow i=3, j=3$
 $x = 11$ $y = 20$



오류 주의

오류 주의

다음과 같은 수식은 오류이다. 왜 그럴까?

`++x = 10;` // 등호의 왼쪽은 항상 변수이어야 한다.

`x + 1 = 20;` // 등호의 왼쪽은 항상 변수이어야 한다.

`x =* y;` // `=*` 이 아니라 `*=` 이다.

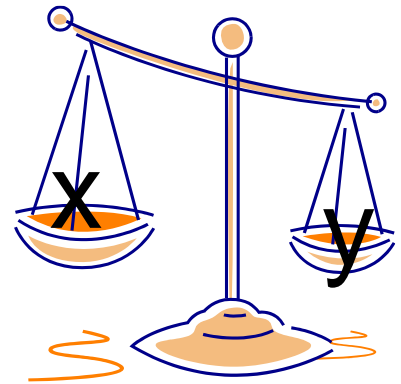


관계 연산자

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

$x == y$

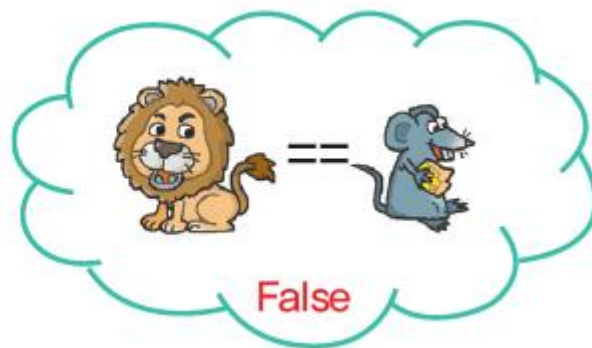
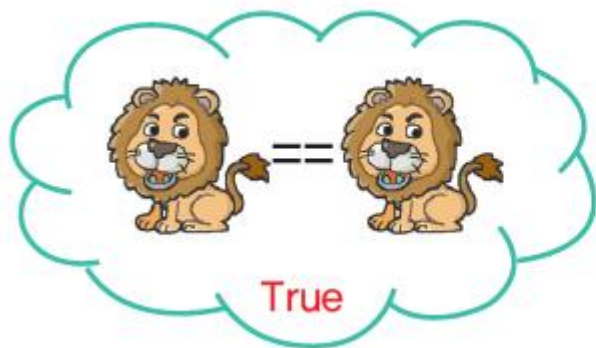
x 와 y의
값이 같은지
비교한다.





관계 연산자

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





예제

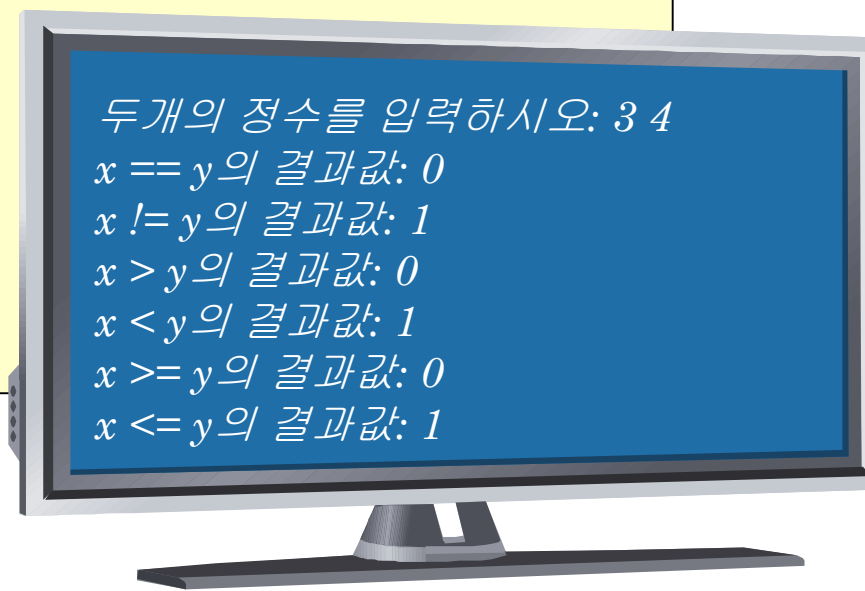
ch03_ex8.c

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

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

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



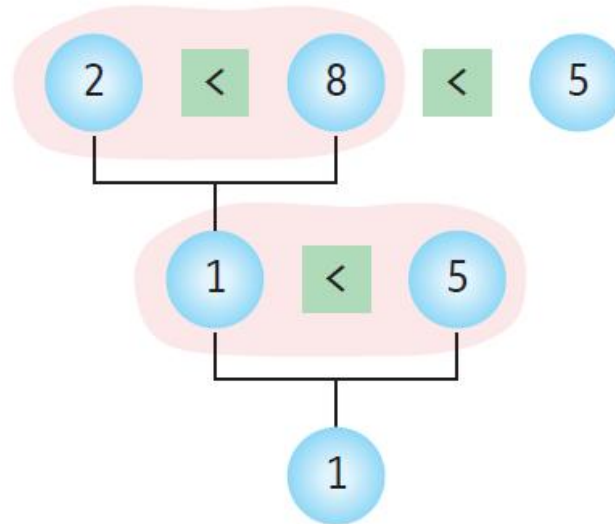
주의할 점!

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



관계 연산자 사용시 주의점

- 수학에서처럼 $2 < x < 5$ 와 같이 작성하면 잘못된 결과가 나온다.



- 올바른 방법: $(2 < x) \&\& (x < 5)$



조건 평가

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



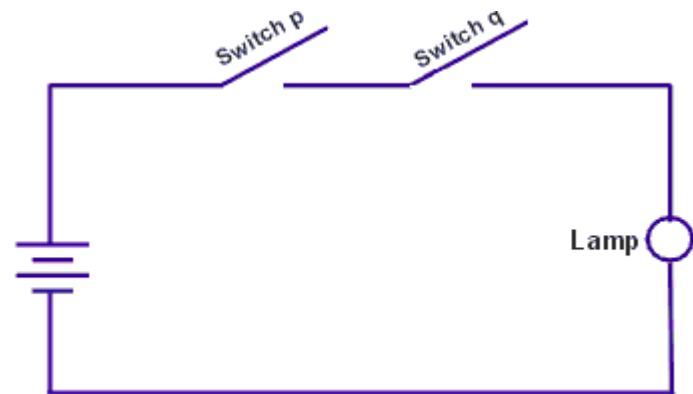


논리 연산자

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

$x \ \&\& \ y$

x와 y가 모두 참인 경우에만 참이 된다.





논리 연산자

연산	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

&&

입력1	입력2	출력
True	True	True
True	False	False
False	True	False
False	False	False

||

입력1	입력2	출력
True	True	True
True	False	True
False	True	True
False	False	False

!

입력	출력
True	False
False	True



AND 연산자

- 어떤 회사에서 신입 사원을 채용하는데 나이가 30살 이하이고 토익 성적이 700점 이상 이라는 조건을 걸었다고 가정하자.

²⁷
(age <= 30) && (⁸⁰⁰toeic >= 700)

참(1) 참(1)

참(1)



OR 연산자

- 신입 사원을 채용하는 조건이 변경되어서 나이가 30살 이하이거나 토익 성적이 700점 이상이면 된다고 하자.

$$\begin{array}{c} 27 \\ \text{((age)} \leq 30) \quad || \quad \text{((toeic)} \geq 700) \\ \underbrace{\hspace{10em}} \\ \text{참(1)} \quad \text{거짓(0)} \\ \underbrace{\hspace{10em}} \\ \text{참(1)} \end{array}$$



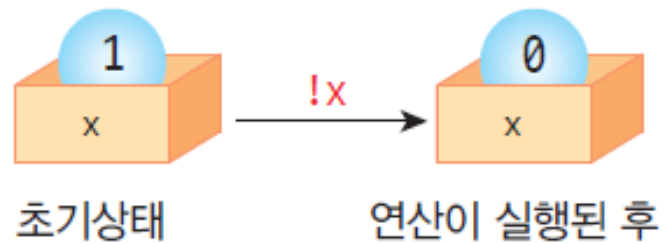
논리 연산자의 예

- “x는 1, 2, 3중의 하나인가”
 - `(x == 1) || (x == 2) || (x == 3)`
- “x가 60이상 100미만이다.”
 - `(x >= 60) && (x < 100)`
- “x가 0도 아니고 1도 아니다.”
 - `(x != 0) && (x != 1)` // x≠0 이고 x≠1이다.



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.
```



참과 거짓의 표현 방법

- 관계 수식이나 논리 수식이 만약 참이면 1이 생성되고 거짓이면 0이 생성된다.
- 피연산자의 참, 거짓을 가릴 때에는 0이 아니면 참이고 0이면 거짓으로 판단한다.
- 음수는 거짓으로 판단한다.
- (예) NOT 연산자를 적용하는 경우

!0	// 식의 값은 1
!3	// 식의 값은 0
!-3	// 식의 값은 0



예제

ch03_ex9.c

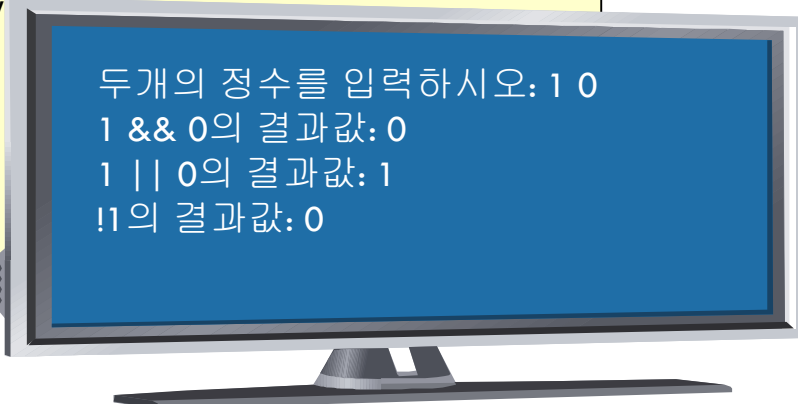
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int x, y;

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

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

    return 0;
}
```

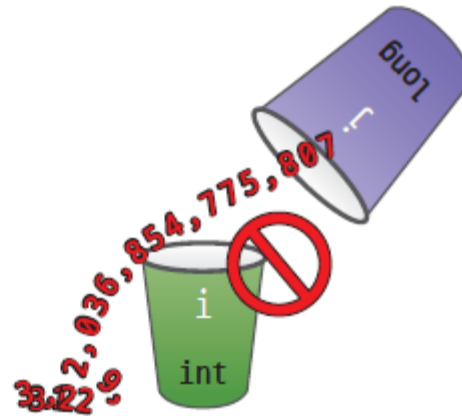
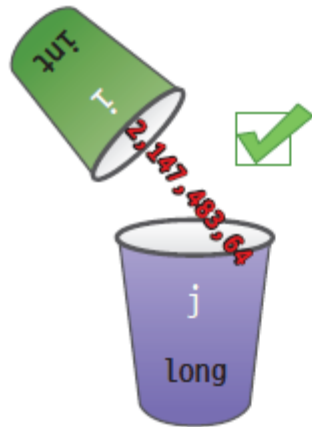


```
두개의 정수를 입력하시오: 1 0
1 && 0의 결과값: 0
1 || 0의 결과값: 1
!!1의 결과값: 1
```



형변환

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



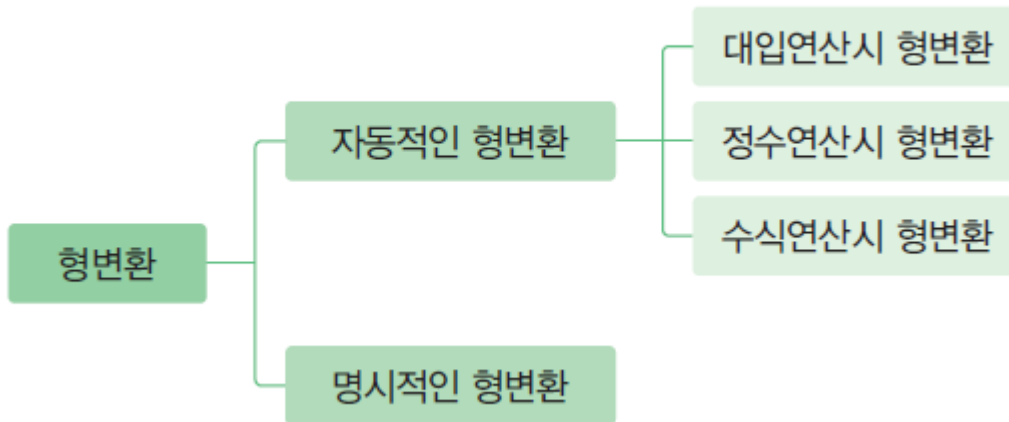
형변환을
잘못하면
데이터의 일부가
사라질 수도
있기 때문에
주의하여야
한다.





형변화

- 연산시에 데이터의 유형이 변환되는 것



변수의 타입이
변경되는 것이
아니고 변수에
저장되는
데이터의 타입이
변경됩니다.

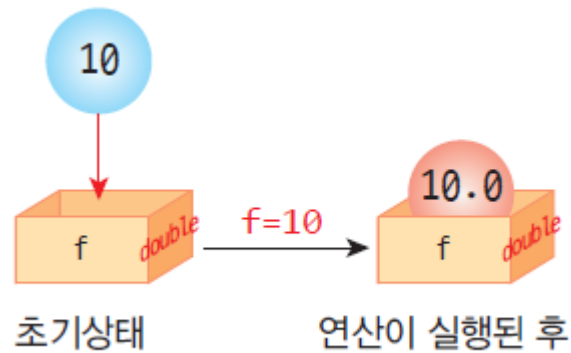




대입 연산자의 자동적인 형변환

- 올림 변환

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

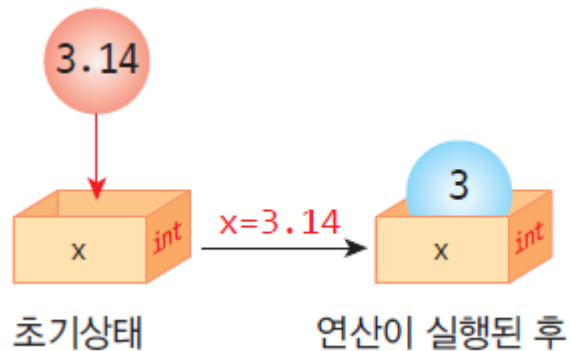




대입 연산시의 자동적인 형변환

- 내림변환

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





올림 변환과 내림 변환

ch03_ex10.c

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#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;
```

```
}
```

기본적으로 소수점이 있는 실수 상수는 double 로 간주됨

c:\...\convert1.c(10) : warning C4305: '=' : 'int'에서 'char'(' ')로 잘립니다

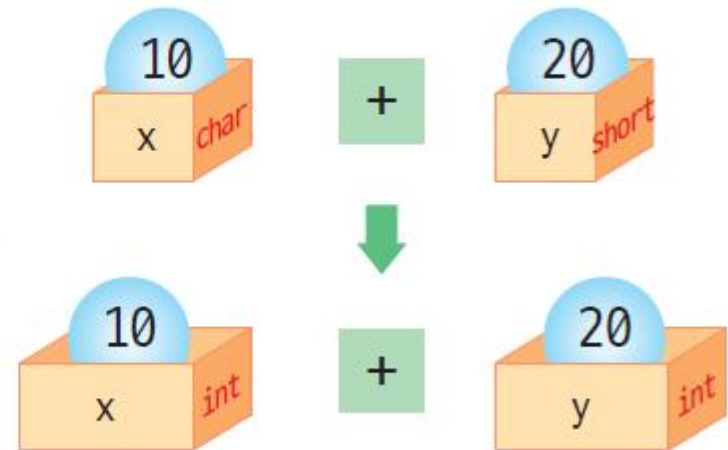
c:\...\convert1.c(11) : warning C4244: '=' : 'double'에서 'int'(' ')로 변환하면서
데이터가 손실될 수 있습니다.



정수 연산시의 자동적인 형변환

- 정수 연산시 **char**형이나 **short**형의 경우, 자동적으로 **int**형으로 변환하여 계산한다.

char나 short형은 int형으로 통일하여서 처리합니다.

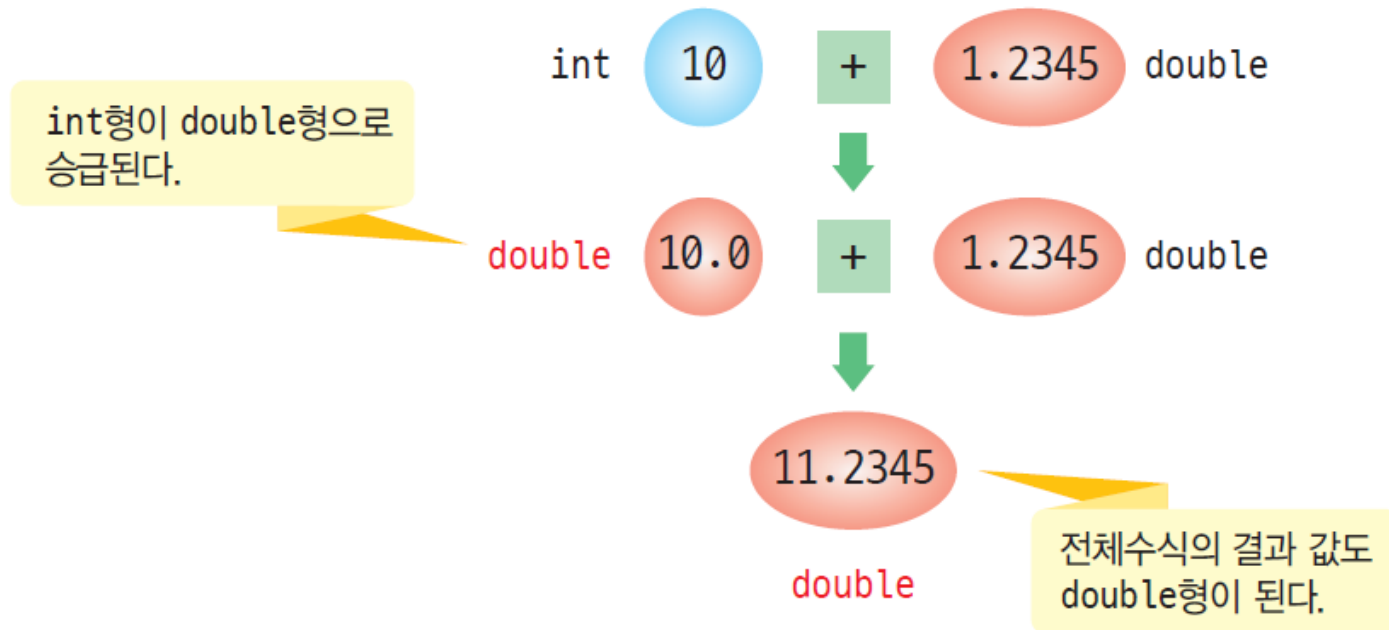


- float**와 **double** 을 사용하는 실수 연산시, 자동적으로 **double**형으로 변환하여 계산한다



수식에서의 자동적인 형변환

- 서로 다른 자료형이 혼합하여 사용되는 경우, 더 큰 자료형으로 통일된다.
 - double > float > int > short > byte





명시적인 형변환

Syntax: 형변환

자료형

수식

예

(int)1.23456

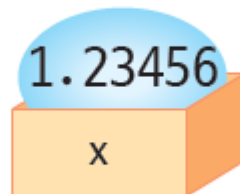
// int형으로 변환

(double) x

// double형으로 변환

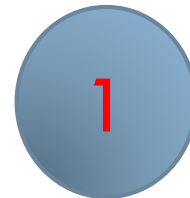
(long) (x+y)

// long형으로 변환



초기상태

(int)



연산이 실행된 후



예제

ch03_ex11.c

```
#define _CRT_SECURE_NO_WARNINGS  
#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);
```

5/4는 1이 되고
이것이 1.0이
된다.

5가 5.0으로 되어서
전체 결과가
1.25가 된다.



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;  
}
```

1.3이 1이 되고
1.8도 1이 되어서
최종 결과는 2가
된다.





우선 순위

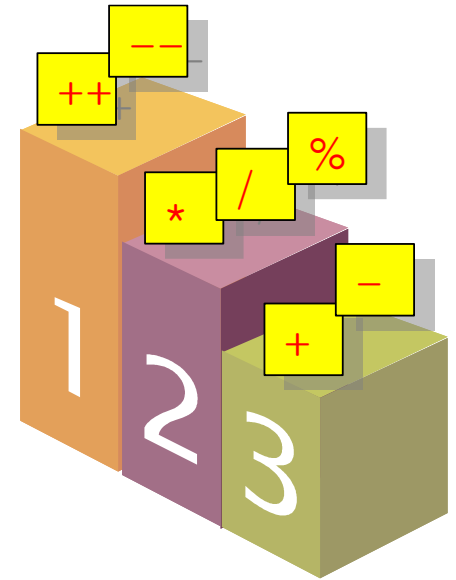
- 어떤 연산자를 먼저 계산할 것인지에 대한 규칙

$$x + y * z$$

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

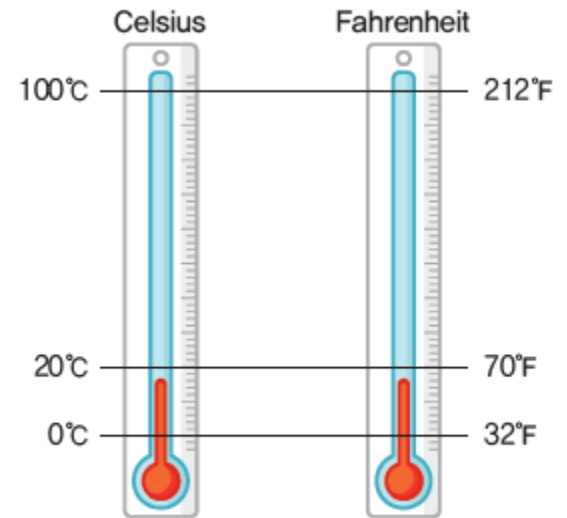
- 콤마 < 대입 < 논리 < 관계 < 산술 < 단항
- 괄호 연산자는 가장 우선순위가 높다.
- 모든 단항 연산자들은 이항 연산자들보다 우선순위가 높다.
- 콤마 연산자를 제외하고는 대입 연산자가 가장 우선순위가 낮다.
- 연산자들의 우선 순위가 생각나지 않으면 괄호를 이용
 - $(x \leq 10) \&\& (y \geq 20)$
- 관계 연산자나 논리 연산자는 산술 연산자보다 우선순위가 낮다.
 - $x + 2 == y + 3$
- 관계 연산자는 논리 연산자보다 우선 순위가 높다. 따라서 다음과 같은 문장은 안심하고 사용하라.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



lab: 화씨 온도를 섭씨로 바꾸기

- 화씨 온도를 섭씨 온도로 바꾸는 프로그램을 작성하여 보자.

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





lab 1

ch03_lab1.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void)
{
    int user, change = 0;
    int price, c5000, c1000, c500, c100;

    printf("물건 값을 입력하시오: ");
    scanf("%d", &price); // 물건 값을 입력받는다.
    printf("사용자가 낸 돈: ");
    scanf("%d", &user);
    change = user - price; // 거스름돈을 change에 저장
```



c5000 = change / 5000; // 몫 연산자를 사용하여 5000원권의 개수를 계산한다.
change = change % 5000; // 나머지 연산자를 사용하여 남은 잔돈을 계산한다.

c1000 = change / 1000; // 남은 잔돈에서 1000원권의 개수를 계산한다.
change = change % 1000; //나머지 연산자를 사용하여 남은 잔돈을 계산한다.

c500 = change / 500; // 남은 잔돈에서 500원 동전의 개수를 계산한다.
change = change % 500; //나머지 연산자를 사용하여 남은 잔돈을 계산한다.

c100 = change / 100; // 남은 잔돈에서 100원 동전의 개수를 계산한다.
change = change % 100; //나머지 연산자를 사용하여 남은 잔돈을 계산한다.

printf("오천원권: %d장\n", c5000);
printf("천원권: %d장\n", c1000);
printf("오백원 동전: %d개\n", c500);
printf("백원 동전: %d개\n", c100);
return 0;

}



lab2

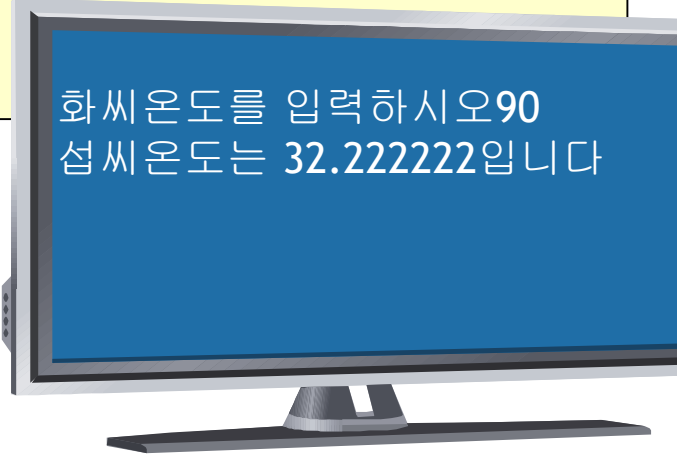
ch03_lab2.c

```
#define _CRT_SECURE_NO_WARNINGS

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

    printf("화씨온도를 입력하시오 : ");
    scanf("%lf", &f_temp);
    c_temp = 5.0 / 9.0 * (f_temp - 32);
    printf("섭씨온도는 %f입니다", c_temp);

    return 0;
}
```



화씨온도를 입력하시오 90
섭씨온도는 32.222222입니다