
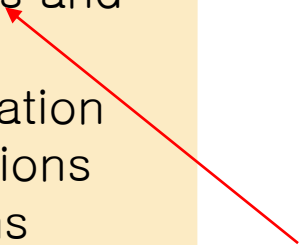


Ch.5 Formulas and operator

What you will learn in this chapter

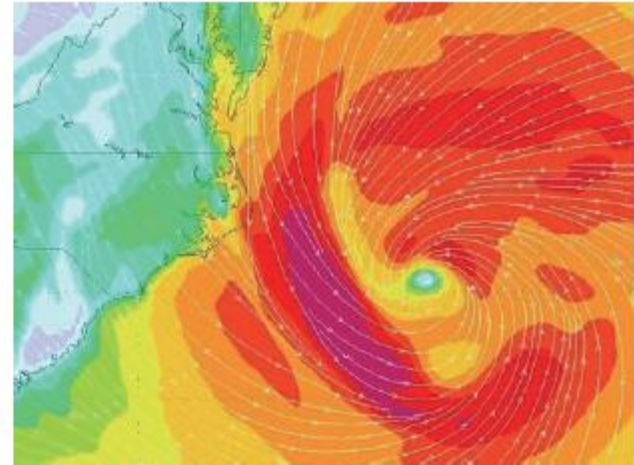
- 
- * What are formulas and operators ?
 - * Assignment operation
 - * Arithmetic operations
 - * Logical operations
 - * Relational operations
 - * Priority and associativity rules
- 

In this chapter,
we will look at
formulas and
operators .



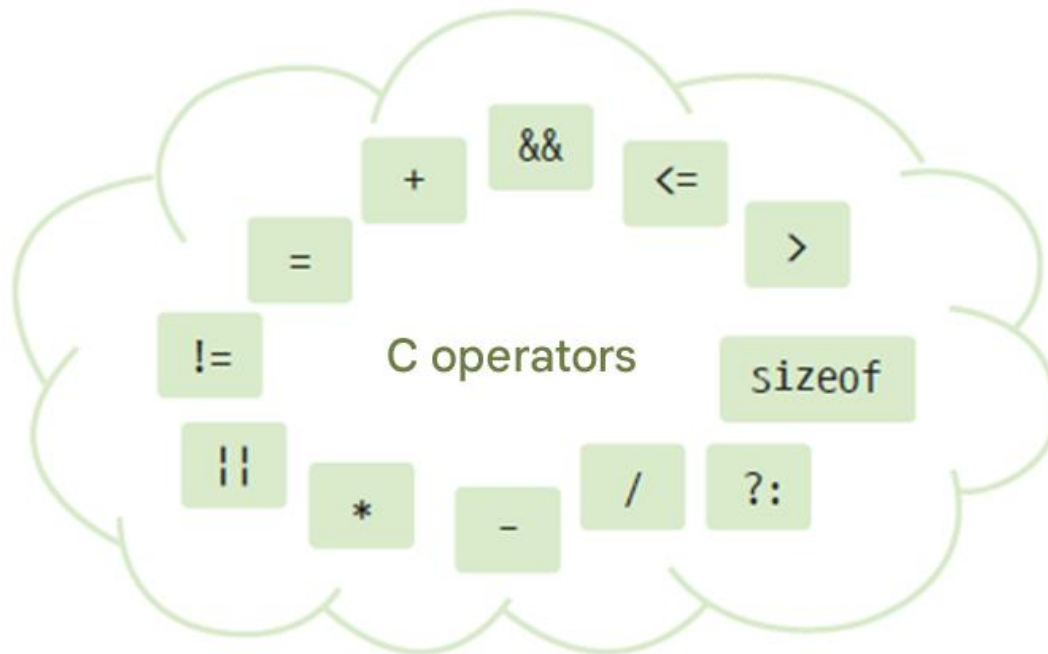
A computer is fundamentally a calculating machine.

The Korea Meteorological Administration uses supercomputers to calculate the weather .

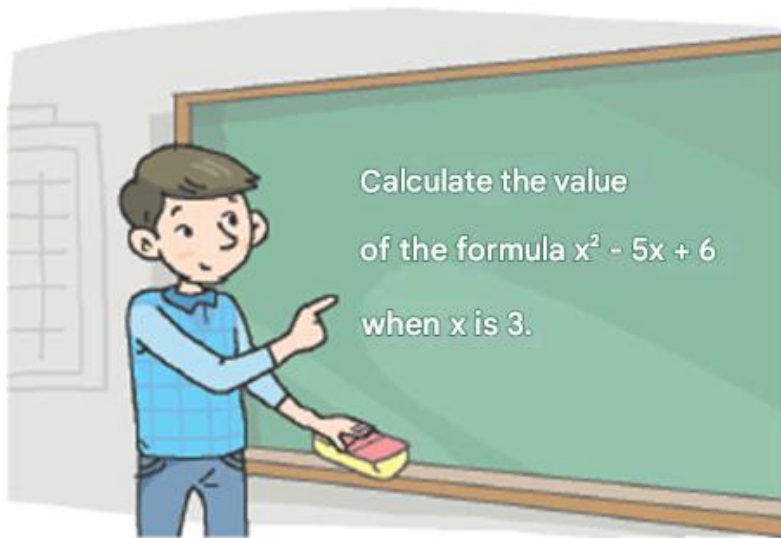


Operators in the C language

- The de facto industry standard
- Modern languages such as Java , C++ , Python , and JavaScript use C language operators almost as they are .



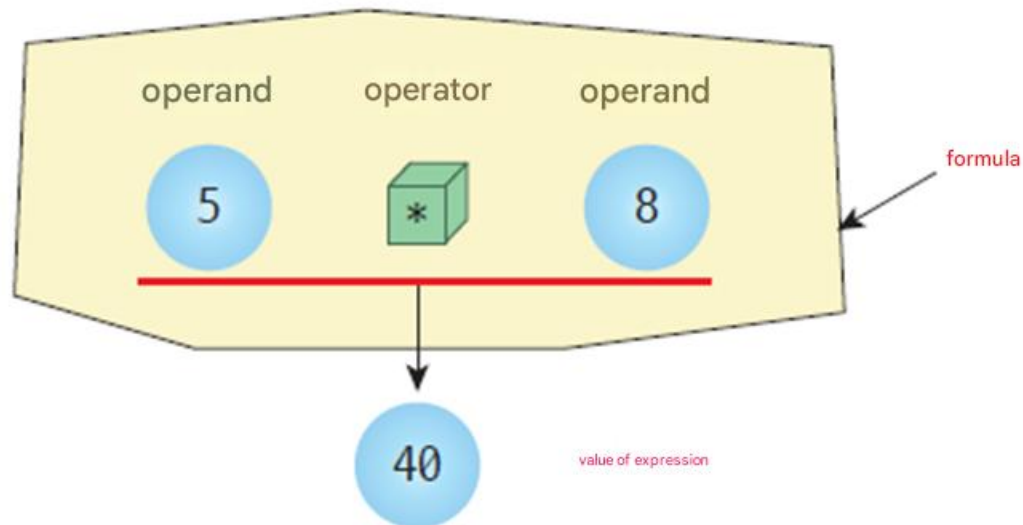
Example of a formula



```
int x, y;  
  
x = 3;  
y = x*x - 5*x + 6;  
printf("%d\n", y);
```

formula

- expression
 - constants , variables , and operators
 - It is divided into operators and operands .



Classification of operators by function

	Operators	Type
Unary Operator →	++, --	Unary Operator
Binary Operator {	+, -, *, /, %	Arithmetic Operator
	<, <=, >, >=, ==, !=	Relational Operator
	&&, , !	Logical Operator
	&, , <<, >>, ~, ^	Bitwise Operator
	=, +=, -=, *=, /=, %=	Assignment Operator
Ternary Operator →	?:	Ternary or Conditional Operator

Arithmetic Operators

- Arithmetic Operations : The Most Basic Operations on a Computer
- Operators that perform basic arithmetic operations such as addition , subtraction , multiplication , and division.

operator	sign	Example of use	result
addition	+	$7 + 4$	11
subtraction	-	$7 - 4$	3
multiplication	*	$7 * 4$	28
division	/	$7 / 4$	1
remain	%	$7 \% 4$	3

Examples of arithmetic operators

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

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

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



(Note) What is the exponentiation operator ?

C does not have an operator for exponentiation .
Simply multiply the variable twice, like $x * x$.

Integer arithmetic operations

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

```
int main(void)
```

```
{
```

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

```
    printf ( " Enter two integers : ");
```

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

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

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

```
    result = x - y; // subtraction
```

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

```
    result = x * y; // multiplication
```

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

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

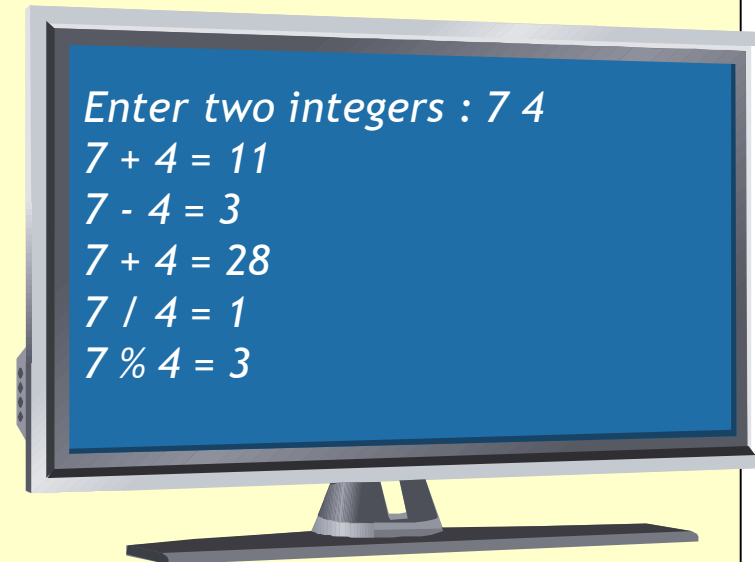
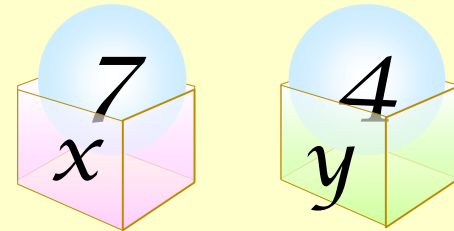
```
    printf( "%d / %d = %d" , x, y, result);
```

```
    result = x % y; // remainder
```

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

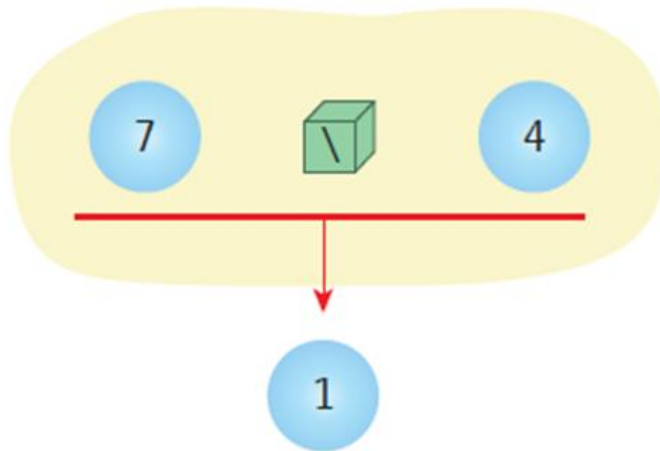
```
    return 0;
```

```
}
```

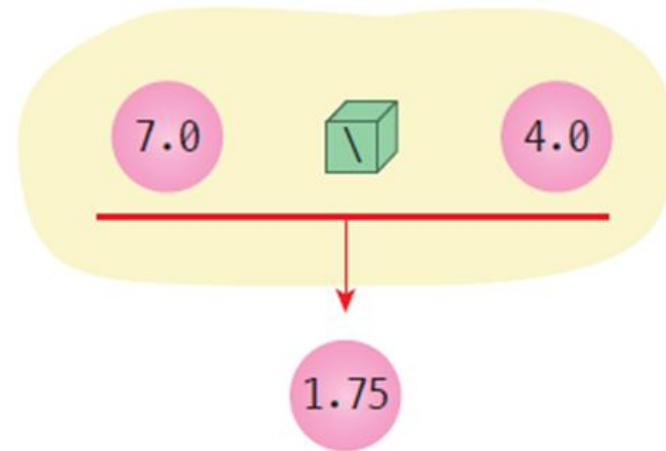


Division operator

- Division between integers produces an integer result, and division between floating-point numbers produces a floating-point value .
- In division between integers, the decimal places are discarded .



Division of integers



Division of real numbers and real numbers

Real number Arithmetic operations

```
#include <stdio.h>

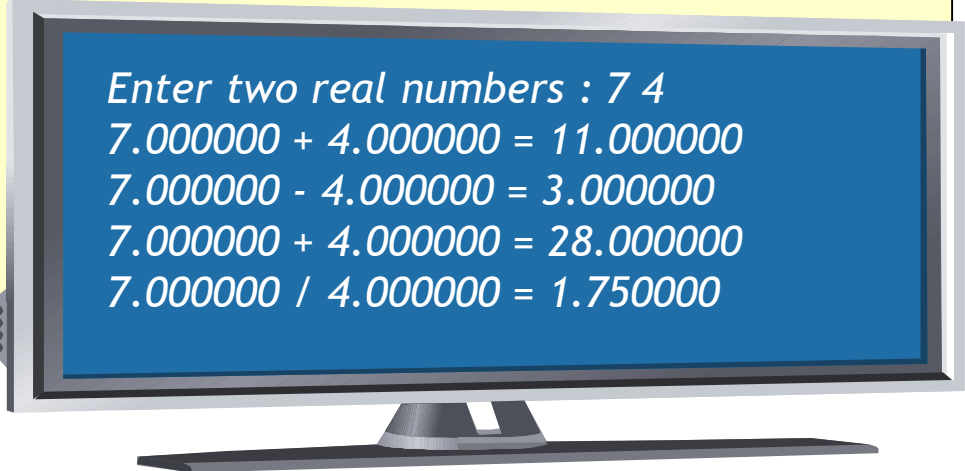
int main()
{
    double x, y, result;

    printf ( " Enter two real numbers : ");
    scanf( "%lf %lf" , &x, &y);

    result = x + y; // Perform addition operation and assign the result to result
    printf( "%f / %f = %f" , x, y, result);

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

    return 0;
}
```



Enter two real numbers : 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

Remainder operator

- The modulus operator calculates the remainder when the first operand is divided by the second operand.
 - $10 \% 2$ is 0 .
 - $5 \% 7$ is 5 .
 - $30 \% 9$ is 3 .
- (Example) Distinguishing between even and odd numbers using the remainder operator
 - Even if $x \% 2$ is 0
- (Example) Determining multiples of 3 using the remainder operator
 - $x \% 3$ is 0, then it is a multiple of 3.

Remainder operator

```
// Remainder operator program
#include <stdio.h>
#define SEC_PER_MINUTE 60 // 1 minute is 60 seconds
```

```
int main( void )
{
```

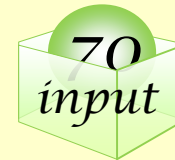
```
    int input, minute, second;
```

```
    printf ( " Please enter seconds : " );
    scanf ( "%d" , &input); // Read the time in seconds .
```

```
    minute = input / SEC_PER_MINUTE; // how many minutes
    second = input % SEC_PER_MINUTE; // how many seconds
```

```
    printf ( "%d seconds are %d minutes %d seconds . \n" , input, minute, second);
    return 0;
```

```
}
```

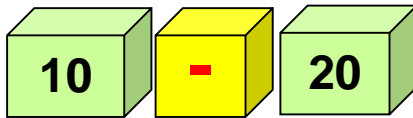
A computer monitor displaying the program's output.

```
Enter seconds : 1000
1000 seconds is 16 minutes and 40 seconds .
```

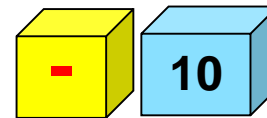
Sign operator

- Change the sign of a variable or constant

```
x = -10;  
y = -x; // The value of variable y becomes 10 .
```



Binary
operator



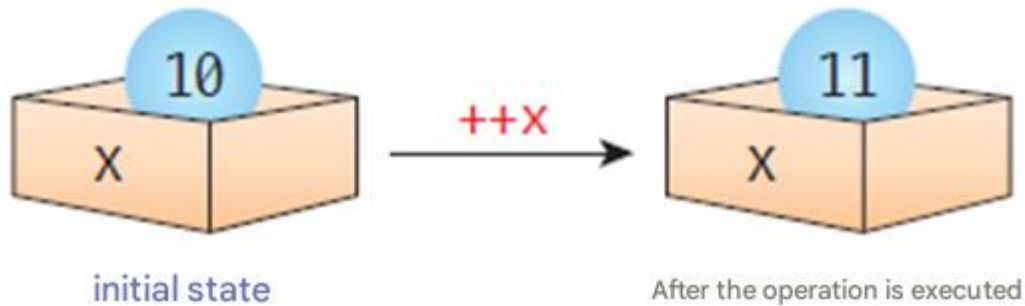
Unary
operator

- is both a
binary
operator and
a unary
operator.



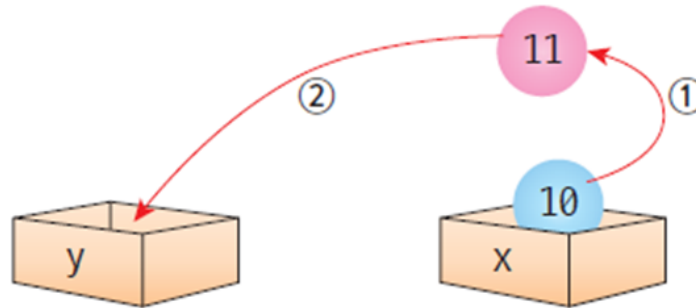
Increment/decrement operator

- Increment/decrement operators : ++, --
- Operator that increases or decreases the value of a variable by one.
- (Example) ++x, --x;



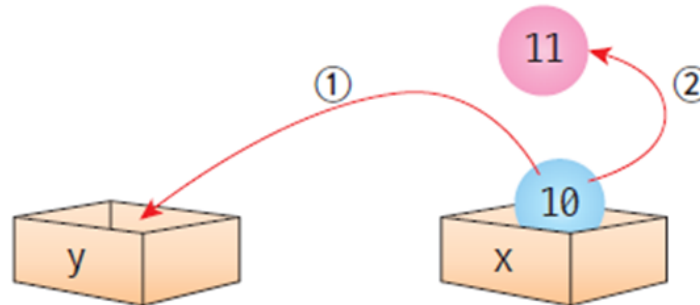
Difference between $++x$ and $x++$

$y = ++x;$



The increased value of x is assigned to y .

$y = x++;$



Substitute first, then increase later.

Increment/decrement operator summary

increment operator	difference
<code>++X</code>	The value of the formula is the incremented value.
<code>X++</code>	The value of the formula is the original x value that has not been increased.
<code>--X</code>	The value of the formula is the reduced value.
<code>X--</code>	The value of the formula is the original, undecreased x-value.

Examples of increment and decrement operators

`y = (1 + x++) + 10;` // Even if there are parentheses, the increase in the value of x is executed last .

`x = 10++;` // Cannot be applied to constants .
`y = (x+1)++;` // Cannot be applied to formulas .

Example : Increment/decrement operator

```
#include <stdio.h>
int main( void )
{
    int x=10, y=10;

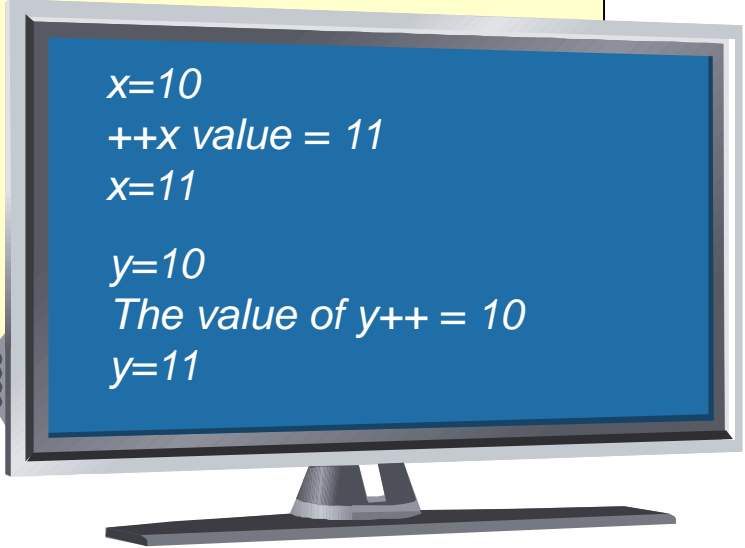
    printf ( "x=%d\n" , x);
    printf ( "++x value =%d\n" , ++x);
    printf ( "x=%d\n\n" , x);

    printf ( "y=%d\n" , y);
    printf ( " value of y++ =%d\n" , y++);
    printf ( "y=%d\n" , y);

    return 0;
}
```

First, the value is increased and the increased value is used in the formula .

Use the current value in the formula first and increases later .



x=10
++x value = 11
x=11

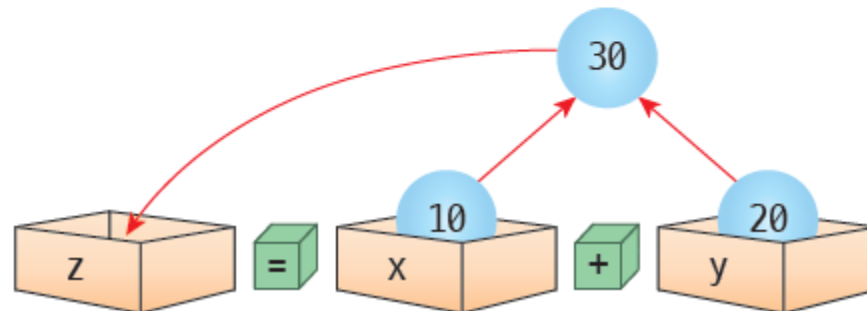
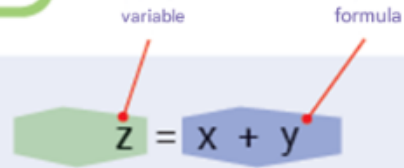
y=10
The value of y++ = 10
y=11

Assignment operator

Syntax

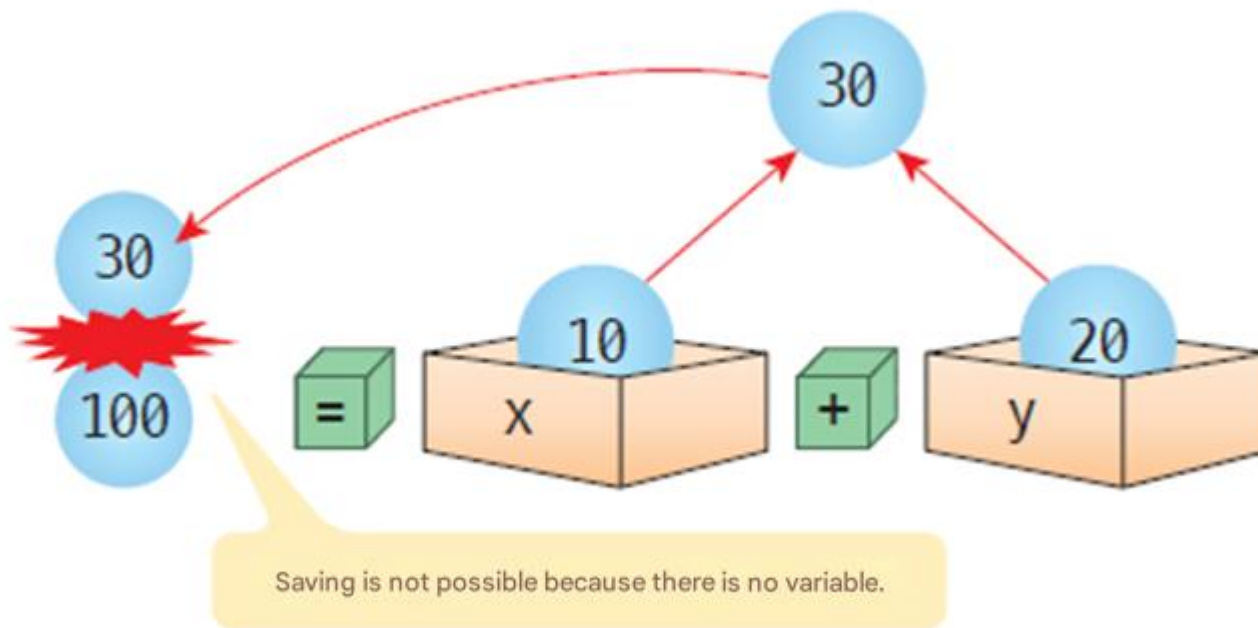
assignment operator

yes



Caution: assignment operators

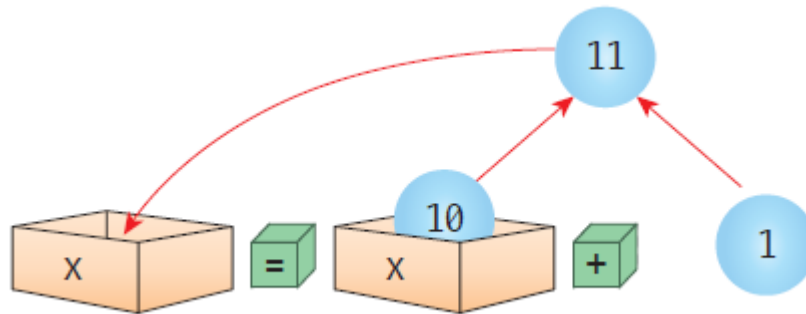
- `100 = x + y; // Compile error !`



Caution: assignment operators

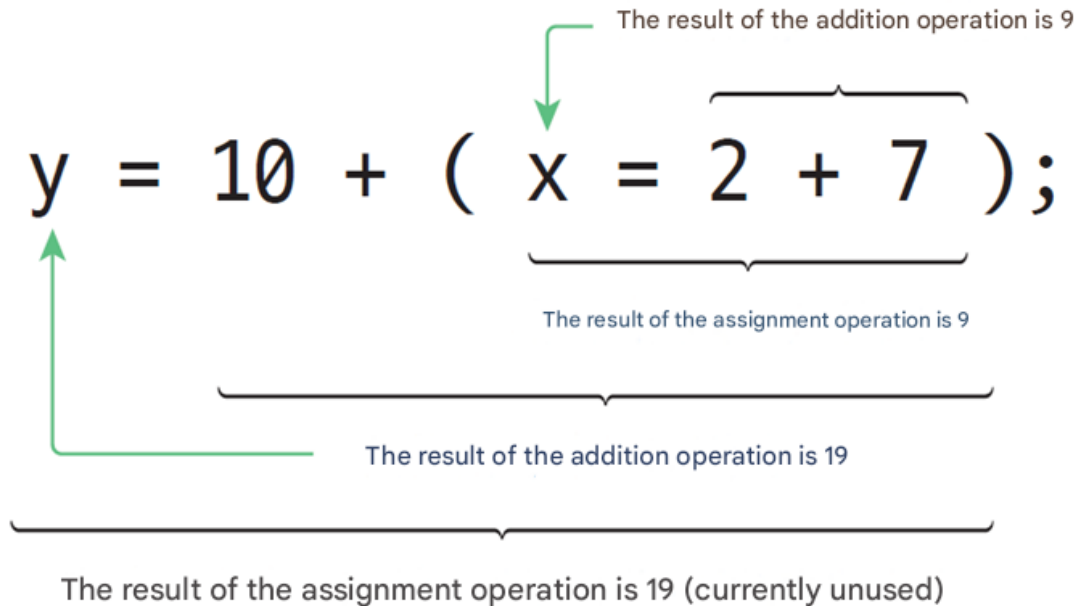
It is a correct statement in C , but mathematically incorrect.

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



The result of the assignment operation

All operations involve
There is a result value
Substitution operation
There is a result value .



Next sentence is also possible .

The diagram shows a yellow rectangular box containing the code `y = x = 3;`. Two red curved arrows originate from the number '3' and point to the variables 'x' and 'y' respectively, indicating the assignment of the value 3 to both. A blue line extends from the bottom left corner of the yellow box towards the explanatory text box.

```
y = x = 3;
```

A statement that assigns the same value to multiple variables can be written as follows . Here, `x = 3` is first performed, and then the resulting value, 3 , is assigned to



Example

```
/* Assignment operator program * /  
#include < stdio.h >
```

```
int main( void )  
{
```

```
    int x, y;
```

```
    x = 1;
```

```
    printf ( " The value of formula x+1 is %d\n" , x+1);
```

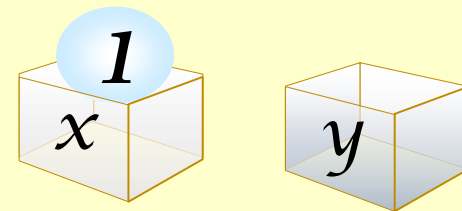
```
    printf ( " The value of the formula y=x+1 is %d\n" , y=x+1);
```

```
    printf ( " The value of the formula y=10+(x=2+7) is %d\n" , y=10+(x=2+7));
```

```
    printf ( " The value of the formula y=x=3 is %d\n" , y=x=3);
```

```
    return 0;
```

```
}
```



The value of the formula x+1 is 2

The value of the formula y=x+1 is 2

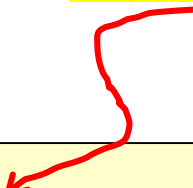
The value of the formula y=10+(x=2+7) is 19

The value of the formula y=x=3 is 3

Compound assignment operator

- A compound assignment operator is an operator that combines
- You can make the source simpler

It has the same meaning as $x = x + y$!



```
x += y
```

Compound assignment operator

compound assignment operator	meaning	compound assignment operator	meaning
<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>

Quiz

- If we solve the following equation and rewrite it, what would it be?

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

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



Compound assignment operator

```
// Compound assignment operator program
```

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

```
int main( void )
```

```
{
```

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

```
    x += 1;
```

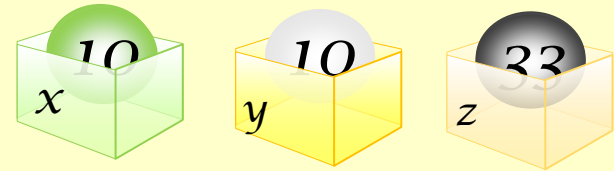
```
    y *= 2;
```

```
    z %= 10 + 20;
```

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

```
    return 0;
```

```
}
```



x = 11 y = 20 z = 3

Error Alert

beware of errors

The following formula is incorrect. Why is that?

$++x = 10;$ // The left side of the equal sign must always be a variable.

$x + 1 = 20;$ // The left side of the equal sign must always be a variable.

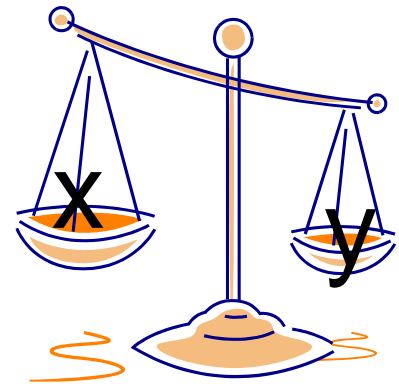
$x = *y;$ // It's **$*=$** , not **$=*$** .

Relational Operators

- Operator that compares two operands
- The result is true (1) or false (0).

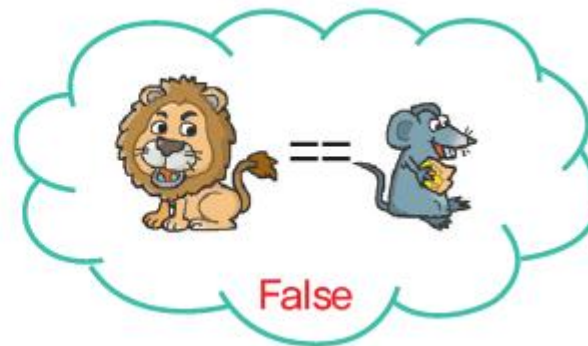
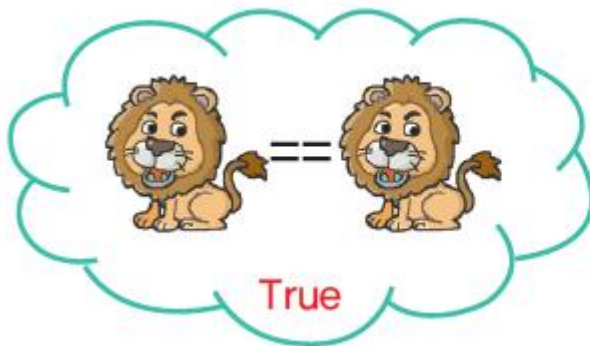
$x == y$

Compares whether the values of x and y are equal .



Relational Operators

calculation	meaning	calculation	meaning
$x == y$	Are x and y equal?	$x < y$	Is x less than y?
$x != y$	Are x and y different?	$x >= y$	Is x greater than or equal to y?
$x > y$	Is x greater than y?	$x <= y$	Is x less than or equal to y?



Examples of relational operators

```
1 == 1 // true (1)
```

```
1 != 2 // true (1)
```

```
2 > 1 // true (1)
```

```
x >= y // true if x is greater than or equal to y (1) , otherwise false (0)
```

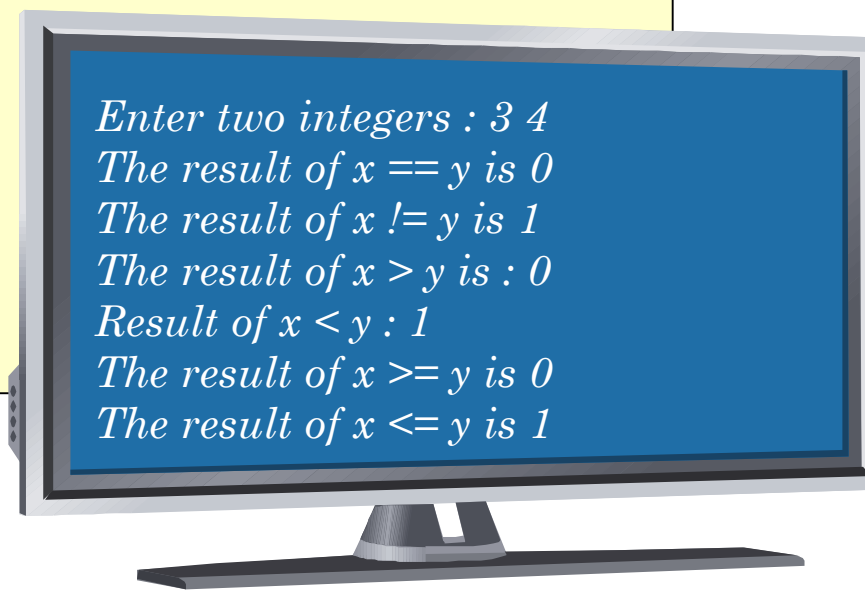
Example

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

    printf ( " Enter two integers : ");
    scanf ( "% d%d " , &x, &y);

    printf ( " The result of x == y : %d", x == y);
    printf ( " The result of x != y : %d", x != y);
    printf ( " Result of x > y : %d", x > y);
    printf ( " The result of x < y : %d", x < y);
    printf ( " The result of x >= y : %d", x >= y);
    printf ( " The result of x <= y : %d", x <= y);

    return 0;
}
```



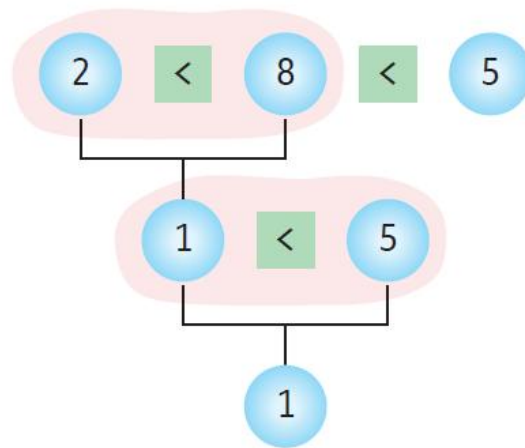
*Enter two integers : 3 4
The result of x == y is 0
The result of x != y is 1
The result of x > y is : 0
Result of x < y : 1
The result of x >= y is 0
The result of x <= y is 1*

Caution!

- $(x = y)$
 - Substitute the value of y into x . The value of this formula is the value of x .
- $(x == y)$
 - x and y are equal, the value of the formula is 1, otherwise it is 0.
 - $(x == y)$ to $(x = y)$ Be careful not to use it incorrectly !

Caution: when using relational operators

- As in mathematics, $2 < x < 5$ and If you write them together, you will get wrong results .



- The right way : $(2 < x) \ \&\& \ (x < 5)$

When comparing real numbers

- $(1e32 + 0.01) > 1e32$
 - -> False because the values on both sides are considered equal

- $(\text{fabs}(xy)) < 0.0001$
 - Correct formula

Mistakes may
have some
errors !

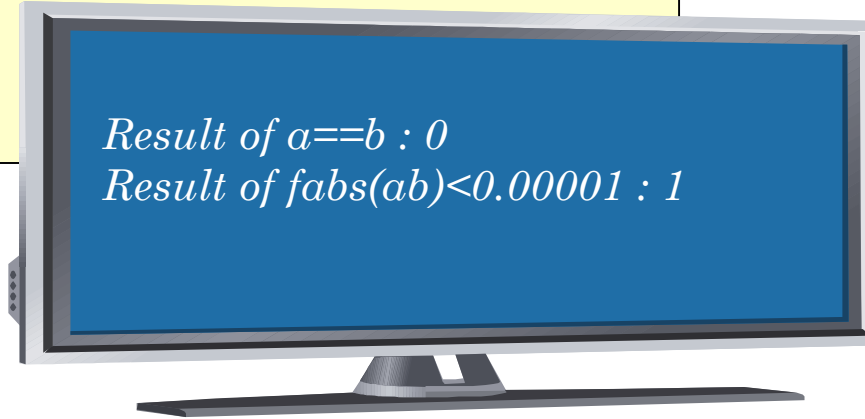


Example

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

int main( void )
{
    double a, b;
    a = (0.3 * 3) + 0.1;
    b = 1;
    printf ( " Result of a==b : %d \n" , a == b);

    printf ( " Result of fabs(ab)<0.00001 : %d \n" , fabs(a - b) < 0.0001);
    return 0;
}
```



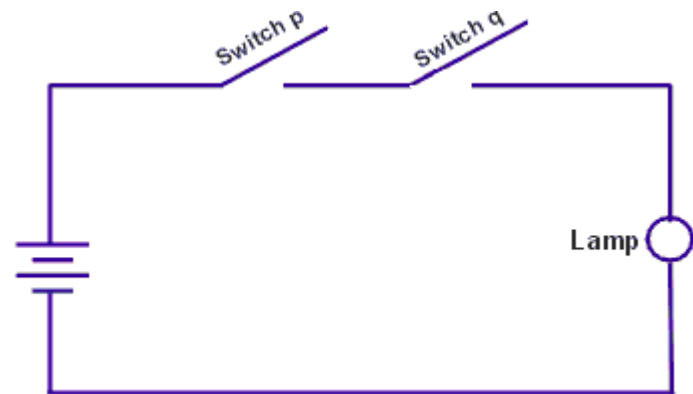
Result of a==b : 0
Result of fabs(ab)<0.00001 : 1

Logical Operators

- An operator that combines multiple conditions to determine true or false.
- The result is true (1) or false (0).

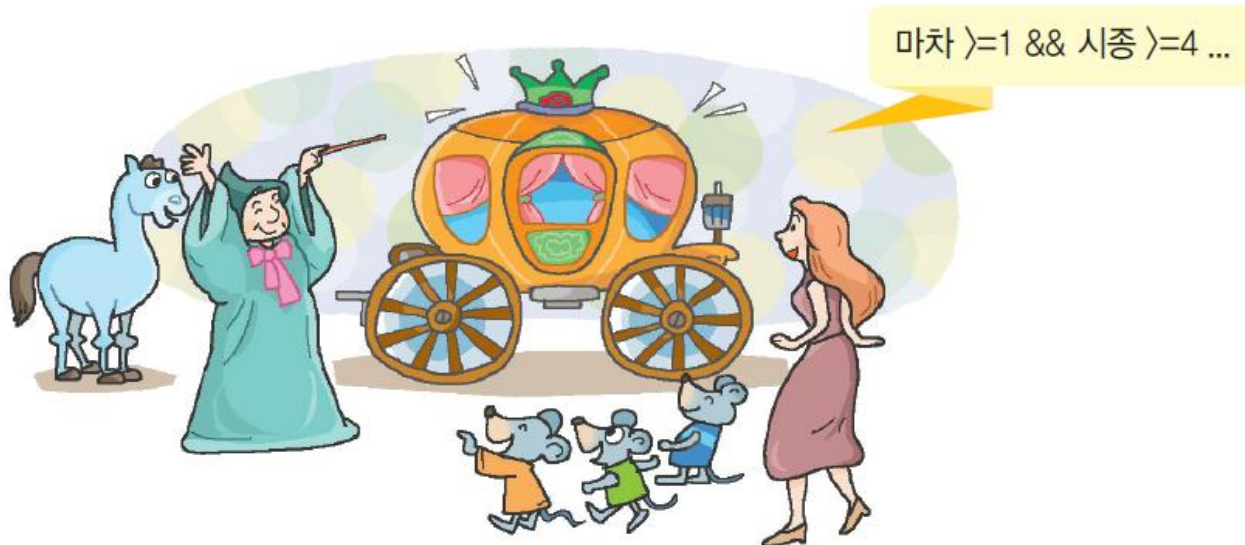
$x \ \&\& \ y$

Both x and y are true
It is true only if .



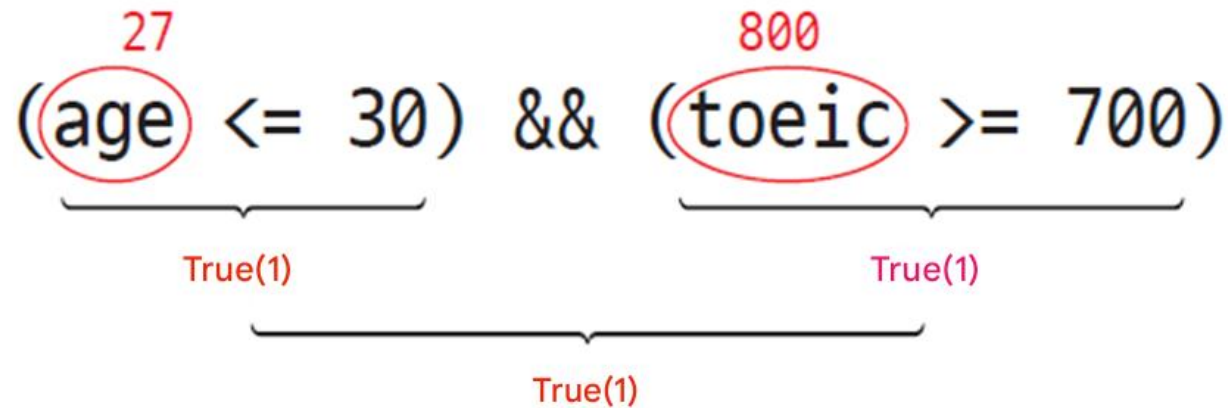
Logical Operators

calculation	meaning
$x \ \&\& \ y$	AND operation, true if both x and y are true, otherwise false
$x \ \ y$	OR operation, true if only one of x or y is true, false if both are false
$!x$	NOT operation, false if x is true, true if x is false



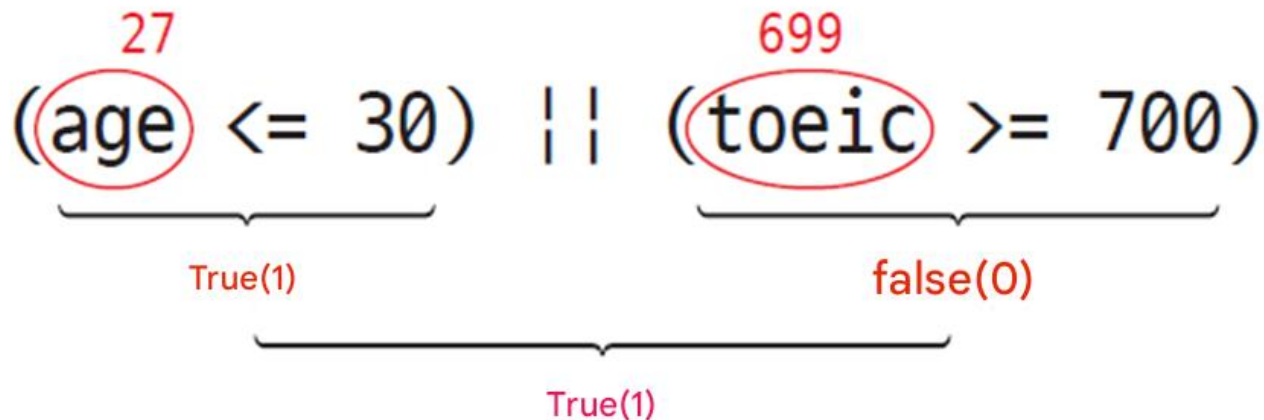
AND operator

- a company is hiring new employees and they set a requirement that the applicants be under 30 years old and have a TOEIC score of 700 or higher .



OR operator

- the conditions for hiring new employees have changed so that they must be under 30 years old or have a TOEIC score of 700 or higher .

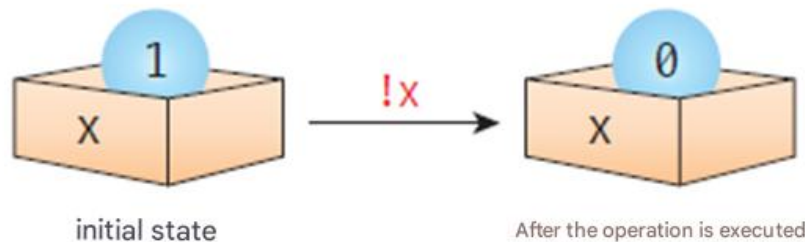


Examples of logical operators

- " Is x one of 1, 2, or 3 ?"
 - `(x == 1) || (x == 2) || (x == 3)`
- " x is greater than or equal to 60 and less than 100 ."
 - `(x >= 60) && (x < 100)`
- " x is neither 0 nor 1 ."
 - `(x != 0) && (x != 1) //`

NOT operator

- If the value of the operand is true, the result of the operation is made false , and if the value of the operand is false, the result of the operation is made true .



```
result = !1; // 0 is assigned to result .  
result = !(2==3); // 1 is assigned to result .
```

How to express truth and false

- If a relational formula or logical formula is true, 1 is generated , and if it is false, 0 is generated .
- the truth or falsity of an operand, it is considered true if it is not 0 , and false if it is 0 .
- Negative numbers are considered false .
- (Example) When applying the NOT operator

```
!0 // The value of the expression is 1
```

```
!3 // The value of the expression is 0
```

```
!-3 // The value of the expression is 0
```

Example

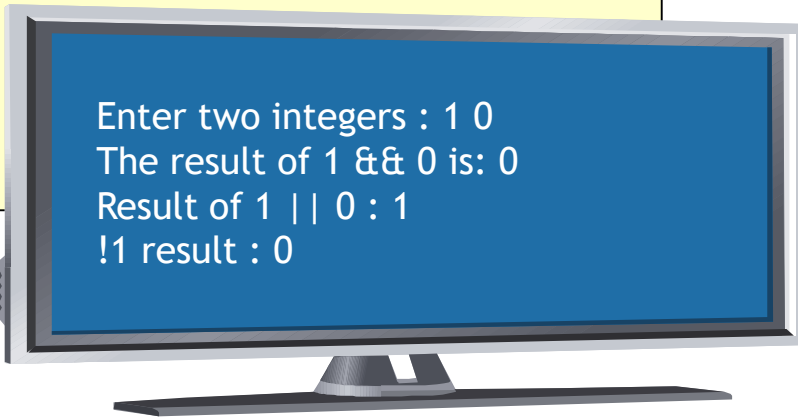
```
#include <stdio.h>

int main( void )
{
    int x, y;

    printf ( " Enter two integers : ");
    scanf ( "%d %d" , &x, &y) ;

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

    return 0;
}
```



Enter two integers : 1 0
The result of 1 && 0 is: 0
Result of 1 || 0 : 1
!1 result : 0

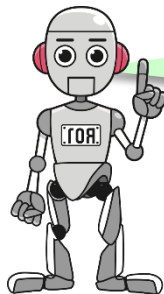
Shortcut calculation

- For the && operator , if the first operand is false, the other operands are not evaluated .

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

- For the || operator , if the first operand is true, the other operands are not evaluated .

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



The first operator is
If it's false, then don't need
To calculate the rest

Please be
careful that
++ and --
may not run .



Lab: Leap year

- Conditions for a leap year
 - The year is divisible by 4 .
 - Years divisible by 100 are excluded .
 - A year that is divisible by 400 is a leap year .



Lab: Leap year

- Expressing the conditions for a leap year in a formula
 - `((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)`

Are parentheses
really necessary ?



Parentheses are
optional, but
they make
reading easier .



Lab: Leap year

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

    printf ( " Enter the year : " );
    scanf ( "%d" , &year);

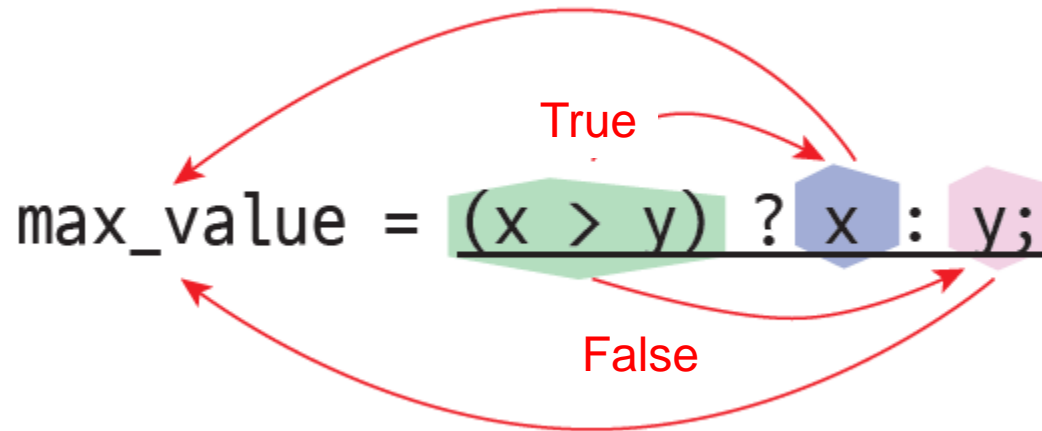
    result = ( (year % 4 == 0) && (year % 100 != 0) || (year % 400 == 0);
    printf ( "result=%d \n" , result);

    return 0;
}
```



Enter the year : 2012
result=1

Conditional Operator



```
absolute_value = (x > 0) ? x : -x; // Calculate absolute value  
max_value = (x > y) ? x : y; // Calculate maximum value  
min_value = (x < y) ? x : y; // Calculate minimum value  
(age > 20) ? printf ( " Adult \n" ) : printf ( " Teenager \n" );
```

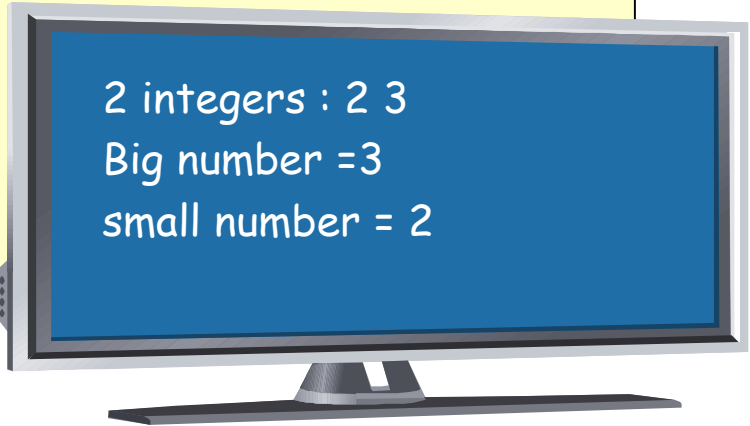
Example

```
// Conditional operator program
#include <stdio.h>

int main( void )
{
    int x,y ;

    printf ( " Two integers : " );
    scanf( "%d %d" , &x, &y);

    printf ( " large number =%d \n" , (x > y) ? x : y );
    printf ( " small number =%d \n" , (x < y) ? x : y );
}
```



2 integers : 2 3
Big number =3
small number = 2

comma operator

- Formulas connected by commas are calculated sequentially .

It is calculated first.

$x++$,

It is calculated later.

$y++$;

Any
sentences are
executed
sequentially .



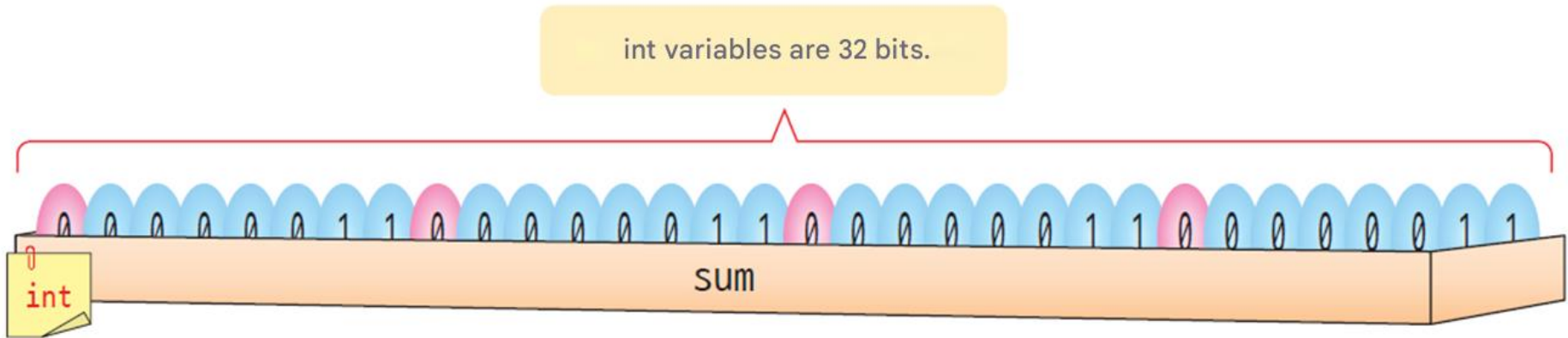
Examples of comma operators

```
x = 2+3, 5-3; // x=2+3 is executed first .
```

```
printf ( "Thank" ), printf ( "you!\n" );
```

```
x = 2, y = 3, z = 4;
```

All data is made up of bits .



Bitwise Operator

operator	meaning of operator	yes
&	bitwise AND	1 if both corresponding bits of the two operands are 1, otherwise 0
	bit OR	1 if only one of the corresponding bits of the two operands is 1, otherwise 0
^	bitwise XOR	1 if the corresponding bits of the two operands have different values, 0 otherwise
<<	move left	Shifts all bits to the left by a specified number.
>>	move right	Shifts all bits to the right by the specified number.
~	bit NOT	0 becomes 1 and 1 becomes 0.

Bitwise AND operator

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

Variable 1 00000000 00000000 00000000 00001001 (9)
Variable 2 00000000 00000000 00000000 00001010 (10)

(variable 1 AND variable 2) 00000000 00000000 00000000 00001000 (8)

Bit OR operator

0 OR 0 = 0
1 OR 0 = 1
0 OR 1 = 1
1 OR 1 = 1

Variable 1 00000000 00000000 00000000 00001001 (9)

Variable 2 00000000 00000000 00000000 00001010 (10)

(variable1 OR variable2) 00000000 00000000 00000000 00001011 (11)

Bitwise XOR operator

0 XOR 0 = 0
1 XOR 0 = 1
0 XOR 1 = 1
1 XOR 1 = 0

Variable1 00000000 00000000 00000000000001001 (9)

Variable2 00000000 00000000 00000000 00001010 (10)

(variable1 XOR variable2) 00000000 00000000 0000000000000011 (3)

Bitwise NOT operator

NOT 0 = 1
NOT 1 = 0

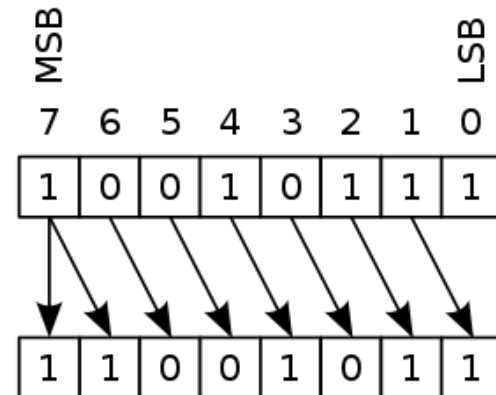
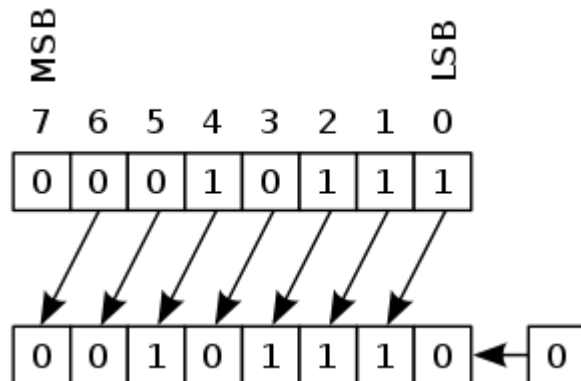
It becomes a negative number because the sign bit is inverted.

Variable1 00000000 00000000 00000000 00001001 (9)

(NOT variable 1) 11111111 11111111 11111111 11110110 (-10)

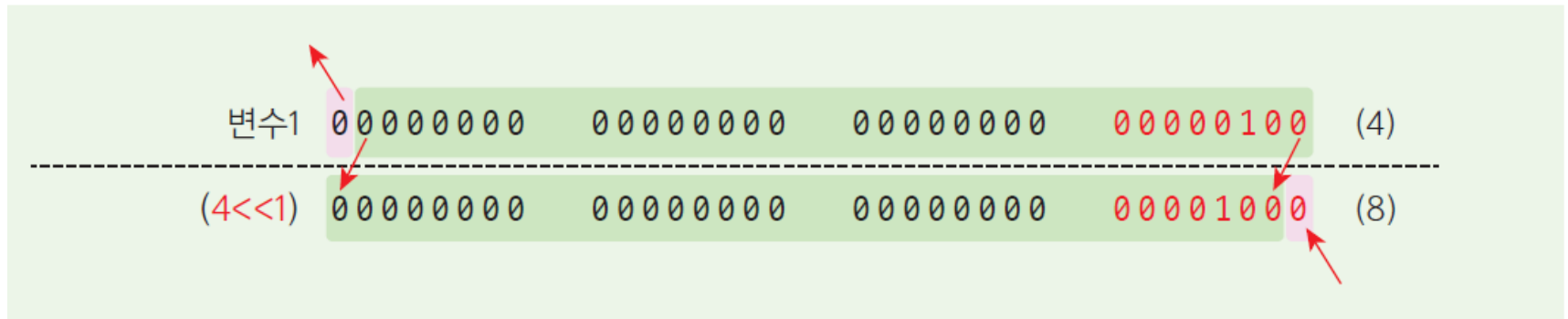
Bit shift operator

operator	sign	explanation
shift left bit	<<	$x \ll y$ shift the bits of x y spaces to the left
shift right bit	>>	$x \gg y$ Shift the bits of x y places to the right



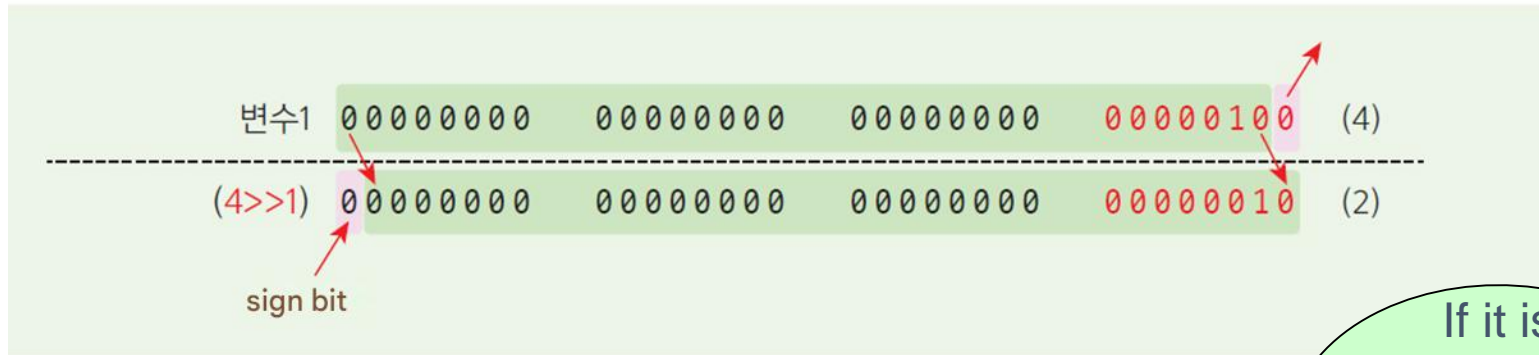
<< operator

- bit left
- The value is doubled .



>> operator

- move
- The value is multiplied by $1/2$.



If it is positive, 0 comes in from the left

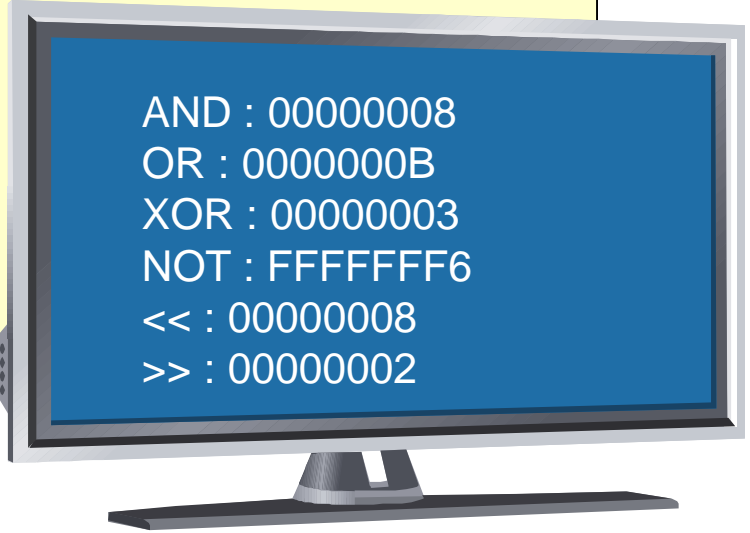


Example : Bitwise Operators

```
#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 : FFFFFFFF6
<< : 00000008
>> : 00000002

Example : Creating 2 's Complement with Bitwise Operators

```
#include <stdio.h>

int main( void )
{
    int a = 32;
    a = ~a; // Make it 1 's complement using NOT operator .
    a = a + 0x01; // add 1 .
    printf( "a= %d \n" , a);

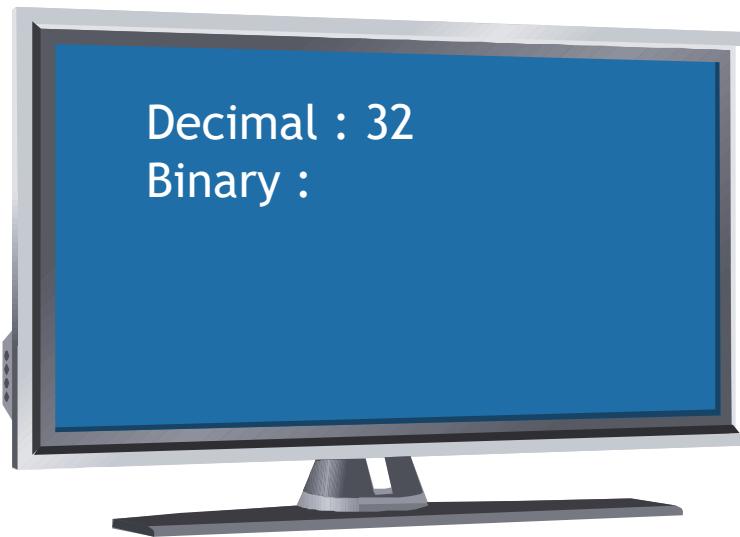
    return 0;
}
```



a= -32

Lab: Outputting decimal to binary

- use bitwise operators to display decimal numbers less than 128 in binary format on the screen .



Lab: Outputting decimal to binary

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

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

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

    ((num & mask) == 0) ? printf( "0" ) : printf( "1" );
    mask = mask >> 1; // Shift 1 bit to the right .
    ((num & mask) == 0) ? printf( "0" ) : printf( "1" );
    mask = mask >> 1; // Shift 1 bit to the right .
    ((num & mask) == 0) ? printf( "0" ) : printf( "1" );
```

Lab: Outputting decimal to binary

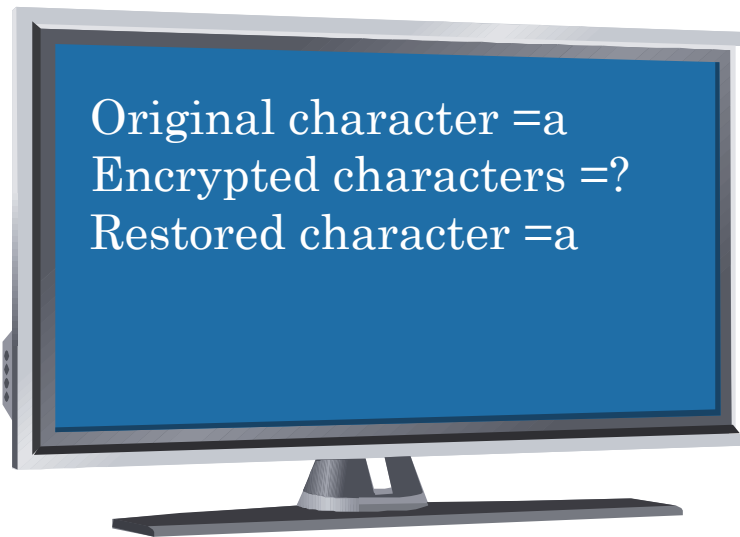
```
mask = mask >> 1; // Shift 1 bit to the right .
((num & mask) == 0) ? printf( "0" ) : printf( "1" );
mask = mask >> 1;
((num & mask) == 0) ? printf( "0" ) : printf( "1" );
mask = mask >> 1;
((num & mask) == 0) ? printf( "0" ) : printf( "1" );
mask = mask >> 1;
((num & mask) == 0) ? printf( "0" ) : printf( "1" );
mask = mask >> 1;
((num & mask) == 0) ? printf( "0" ) : printf( "1" );
printf ( "\n" );

return 0;

}
```

Lab: Encryption using XOR

- To encrypt a single character, just do $x = x \oplus \text{key}$; . Decryption is also possible . $x = x \oplus \text{key}$; That's it .



Lab: Encryption using XOR

```
#include <stdio.h>
int main(void)
{
    char data = 'a' ;
    char key = 0xff;
    char encrypted_data , orig_data ;

    printf ( " Original character =%c\n" , data );

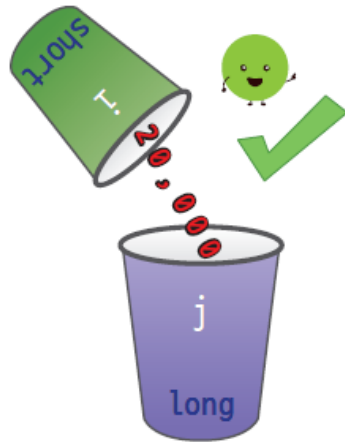
    encrypted_data = data ^ key;
    printf ( " Encrypted character =%c \n" , encrypted_data );

    orig_data = encrypted_data ^ key;
    printf ( " Restored character =%c\n" , orig_data );

    return 0;
}
```

Type conversion

- Type conversion is changing the type of data during execution.

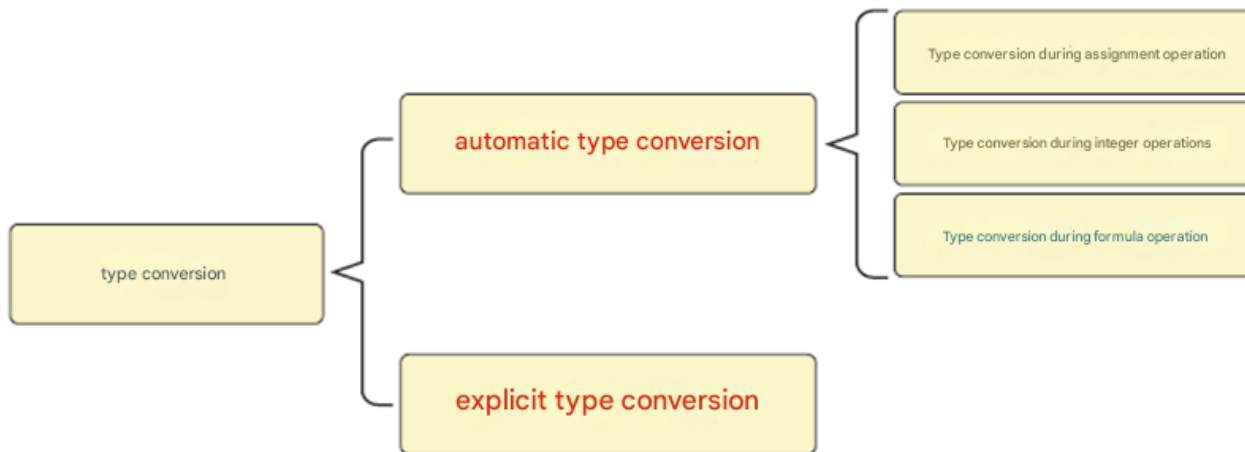


Be careful
because if you
do the
conversion
incorrectly,
some of the
data may be
lost.
Do it .



Type conversion

- The type of data is converted during operation.



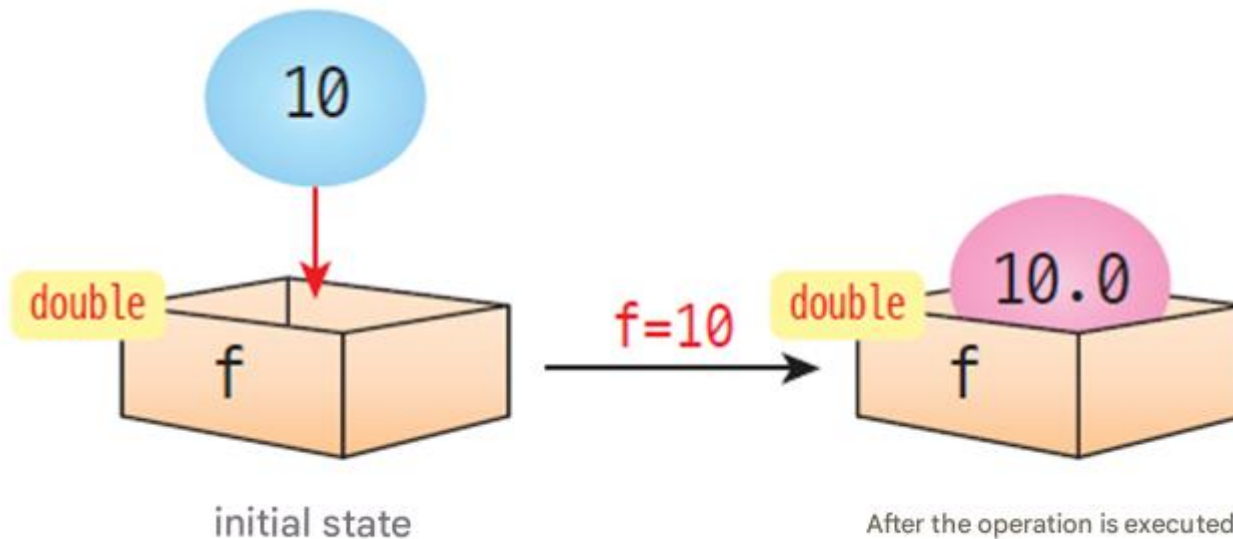
The type of the variable does not change, but the type of data stored in the variable changes .



Automatic type conversion during assignment operations

- Upward conversion

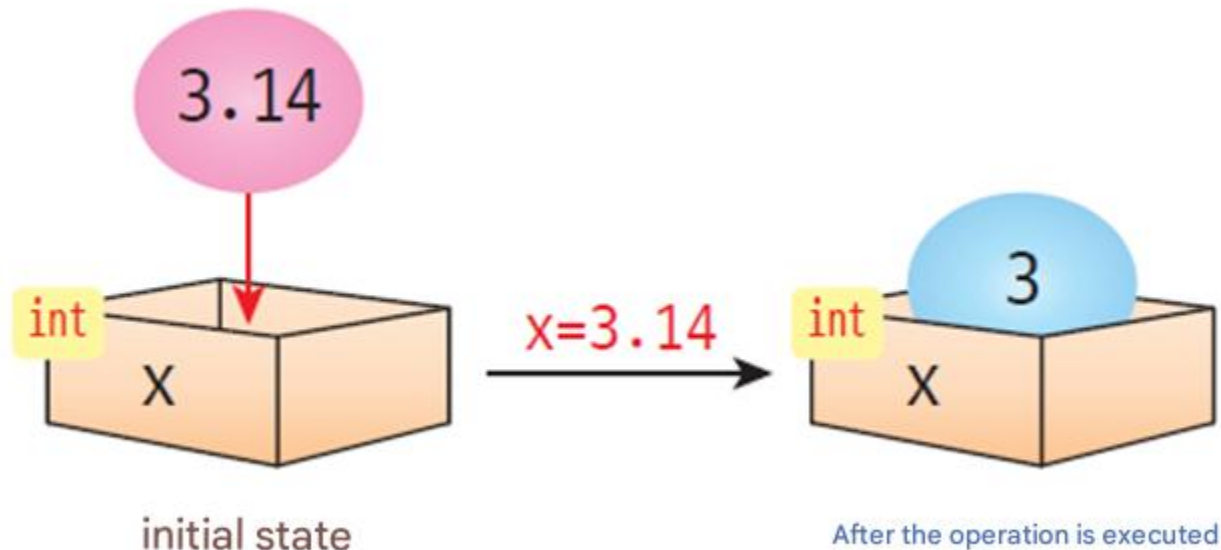
```
double f;  
f = 10 ; // 10.0 is stored in f .
```



Automatic type conversion during assignment operations

- Downward conversion

```
int i;  
i = 3.141592; // 3 is stored in i .
```

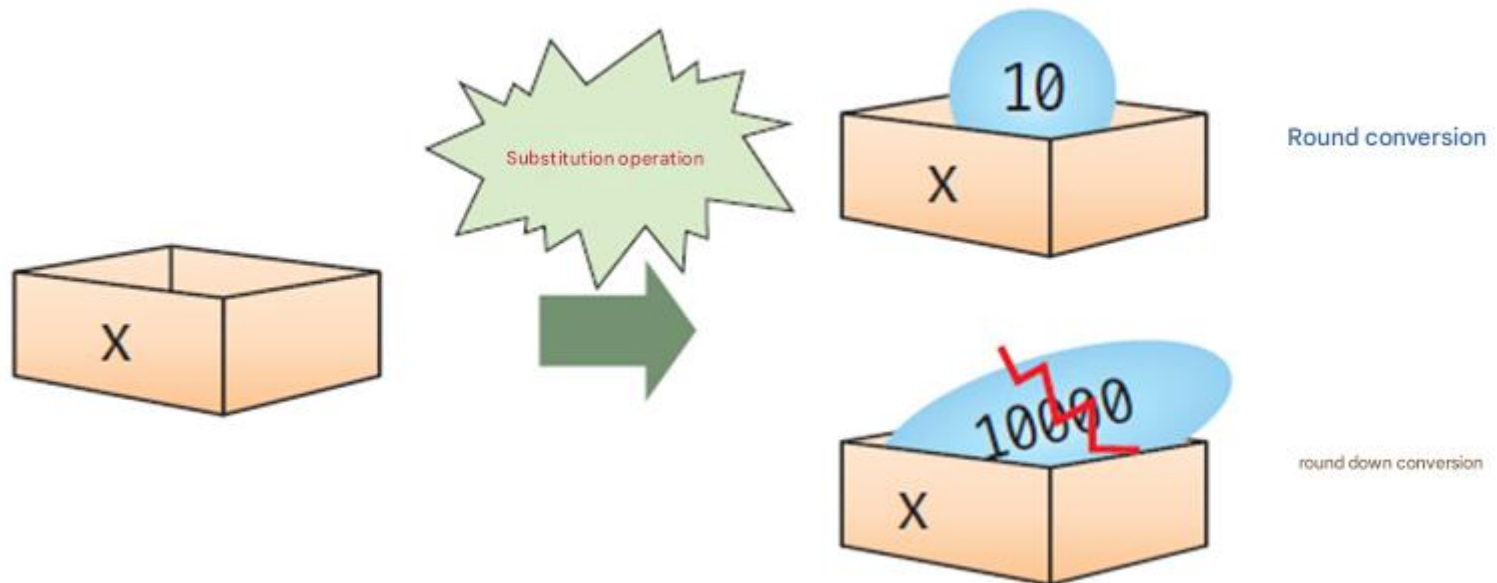


Integer type conversion

```
char x;
```

```
x = 10; // OK
```

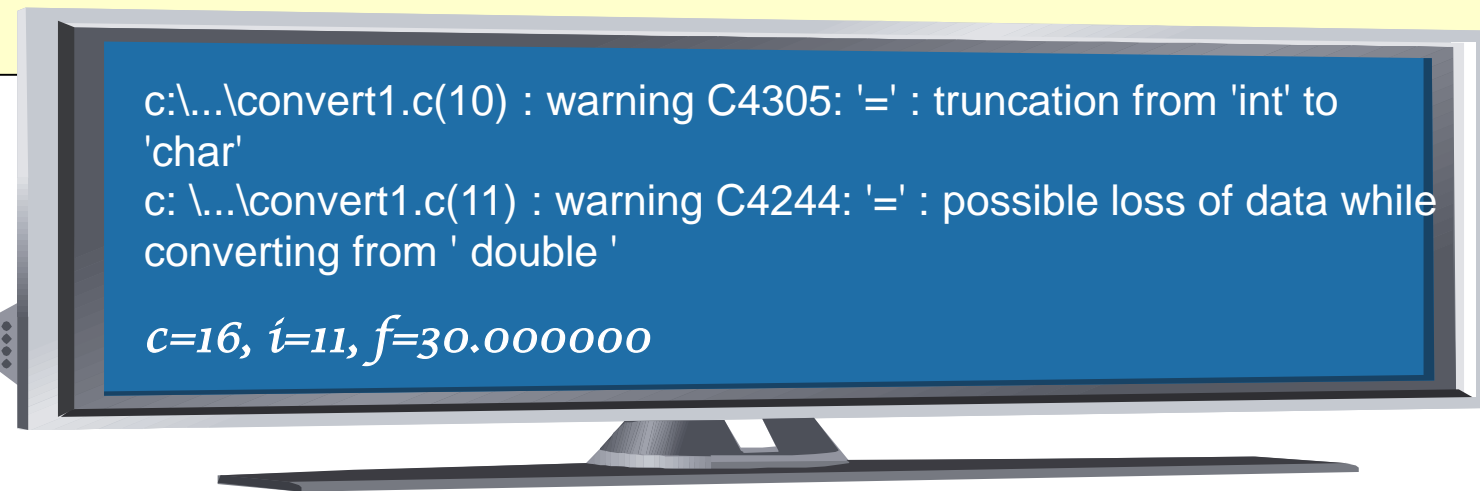
```
x = 10000; // upper The bytes are gone .
```



Up and down conversions

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

    c = 10000; // Round down
    i = 1.23456 + 10; // Round down
    f = 10 + 20; // round up
    printf( "c = %d, i = %d, f = %f \n" , c, i, f);
    return 0;
}
```

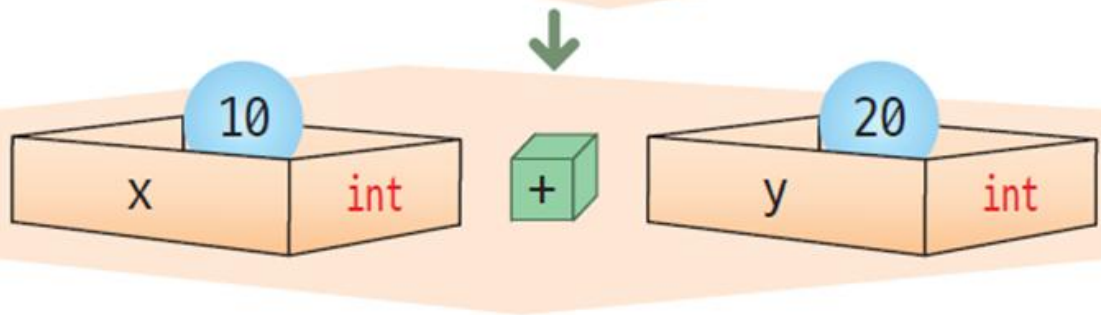
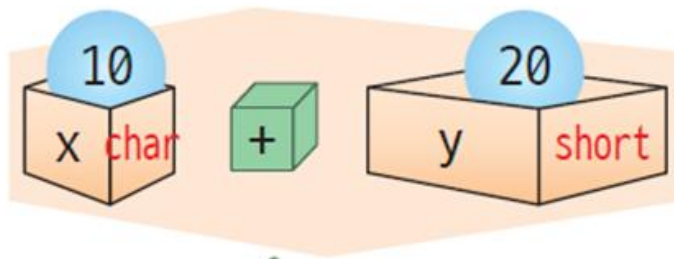


c:\...\convert1.c(10) : warning C4305: '=' : truncation from 'int' to 'char'
c:\...\convert1.c(11) : warning C4244: '=' : possible loss of data while converting from 'double'

c=16, i=11, f=30.000000

Automatic type conversion when performing integer operations

- When performing integer operations , if the type is char or short , it is automatically converted to int type and then calculated .

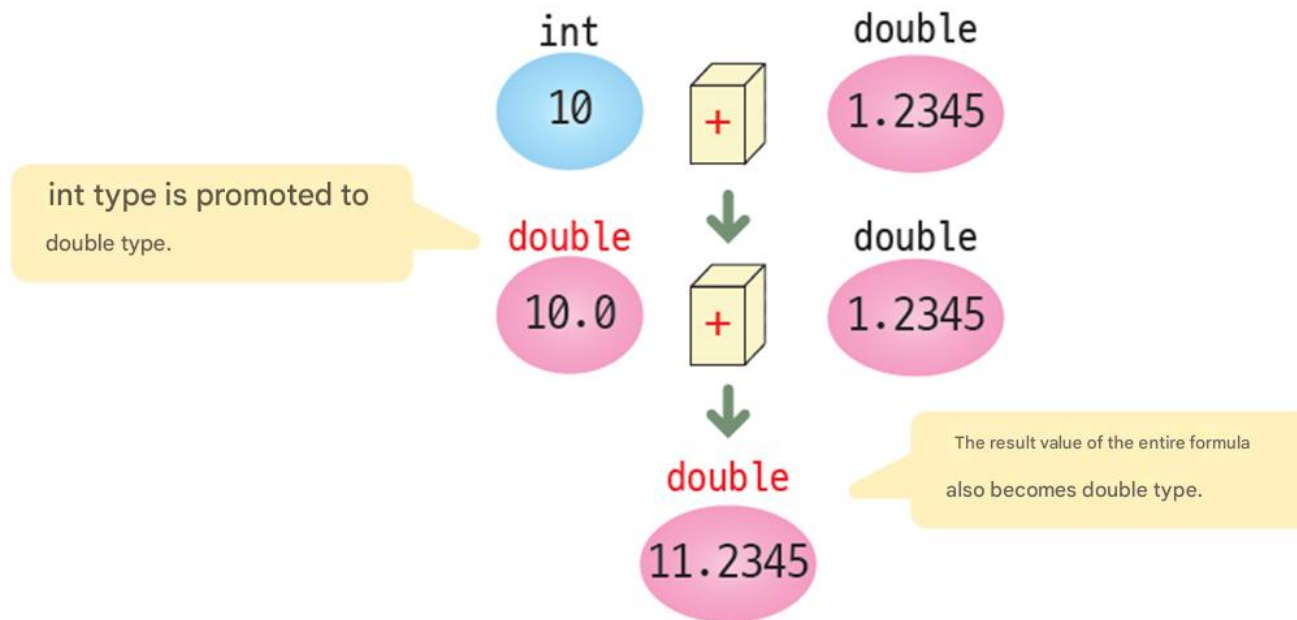


Char and short types are processed as int types.



Automatic type conversion in formulas

- When different data types are used together , they are unified into a larger data type .



Explicit type conversion

Syntax

type conversion

yes

data type

formula

(int)1.23456

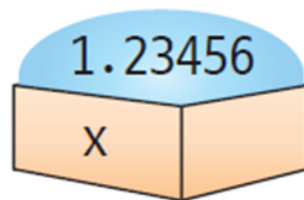
(double) x

(long) (x+y)

// Convert to int type

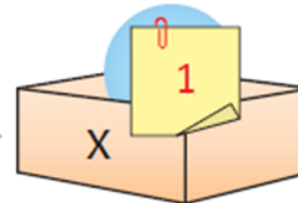
// Convert to double type

// Convert to long type



initial state

(int)



After the operation is executed

Example

```
#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);
```


5/4 becomes 1 ,
which becomes 1.0

5 becomes 5.0 , so the
overall result is 1.25

Example

```
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;  
}
```

1.3 becomes 1 and
1.8 also becomes 1 ,
so the final result is 2



1.000000
1.250000
1.250000
1.250000
3
2

Priority

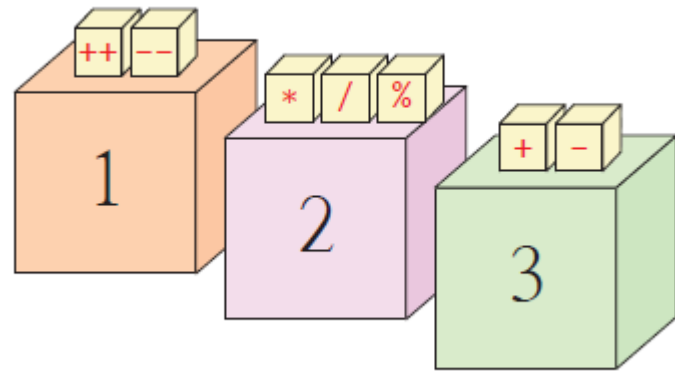
- Rules for which operator to evaluate first

$$x + y * z$$

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.



Priority

priority	operator	explanation	Combinability
1	<code>++ --</code>	postfix increment/decrement operator	→ (from left to right)
	<code>()</code>	function call	
	<code>[]</code>	array index operator	
	<code>.</code>	Accessing structure members	
	<code>-></code>	Structure pointer access	
	<code>(type){list}</code>	Compound literals (C99 standard)	
2	<code>++ --</code>	Potential increment/decrement operator	← (right to left)
	<code>+ -</code>	positive and negative signs	
	<code>! ~</code>	Logical negation, bitwise NOT	
	<code>(type)</code>	type conversion	
	<code>*</code>	indirect reference operator	
	<code>&</code>	address extraction operator	
	<code>sizeof</code>	size calculation operator	
	<code>_Alignof</code>	Alignment Requirement Operator (C11 Specification)	

3	* / %	Multiplication, Division, Remainder	→(from left to right)
4	+ -	Addition, subtraction	
5	<< >>	bit shift operator	
6	< <=	relational operators	
	> >=	relational operators	
7	== !=	relational operators	
8	&	bitwise AND	
9	^	bitwise	
10		bit OR	
11	&&	Logical AND operator	
12		Logical OR operator	
13	?:	Ternary Conditional Operator	← (right to left)
14	=	assignment operator	
	+= -=	compound assignment operator	
	*= /= %=	compound assignment operator	
	<<= >>=	compound assignment operator	
	&= ^= =	compound assignment operator	
15	,	comma operator	→(from left to right)

General guidelines for priorities

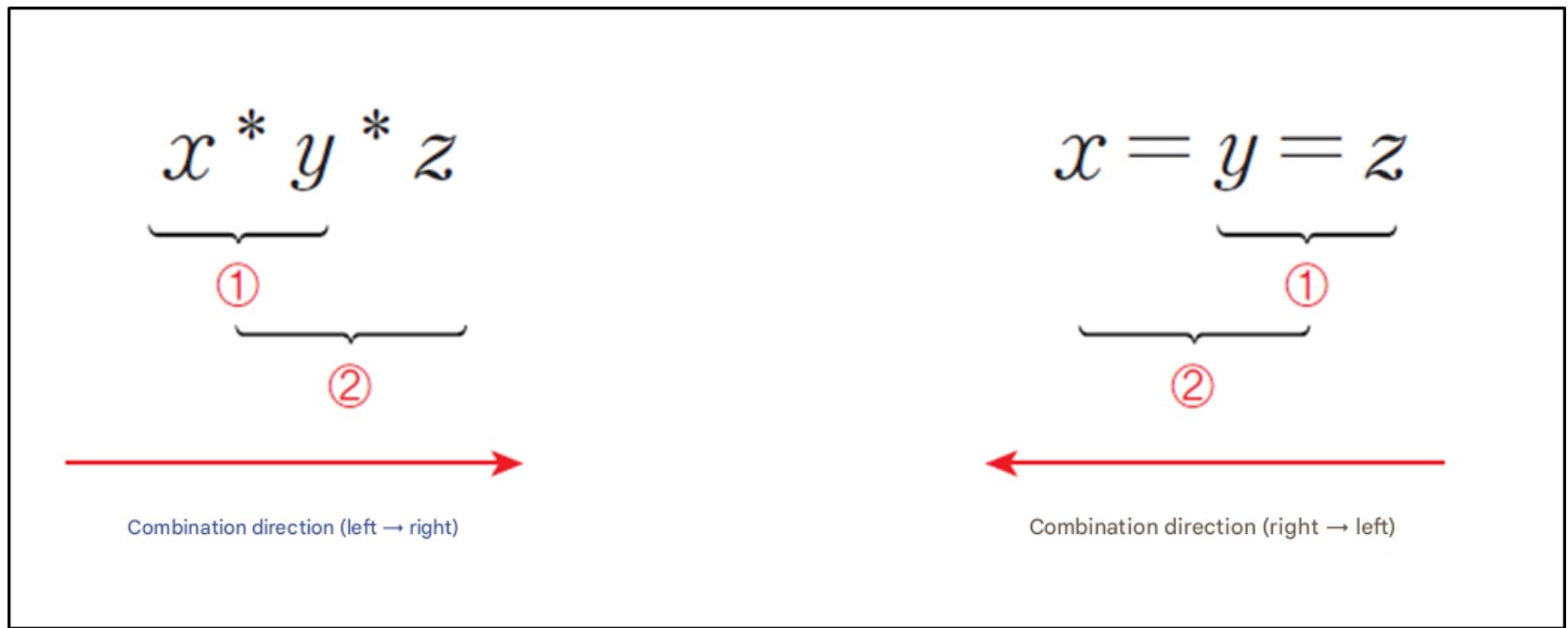
- Comma < Assignment < Logic < Relation < Arithmetic < Unary
- Parentheses operators have the highest precedence .
- All unary operators have higher precedence than binary operators .
- Assignment operators have the lowest precedence, except for the comma operator .
- If you can't remember the precedence of operators, use parentheses.
 - `(x <= 10) && (y >= 20)`
- Relational and logical operators have lower precedence than arithmetic operators .
 - `x + 2 == y + 3`
- Relational operators have higher precedence than logical operators . Therefore, you can use sentences like the following with confidence .
 - `x > y && z > y // Same as (x > y) && (z > y) .`

General guidelines for priorities

- among logical operators, the && operator has higher precedence than the || operator .
 - `x < 5 || x > 10 && x > 0` // Same as
- Sometimes the order of evaluation of operators can be quite confusing . In `x * y + w * y` It is unclear which of `x * y` and `w * y` will be computed first .

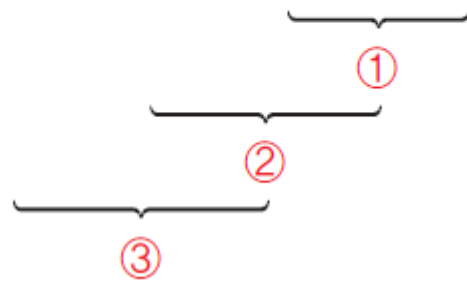
Combination Rules

- If there are multiple operators with the same priority, the rule for which one should be performed first

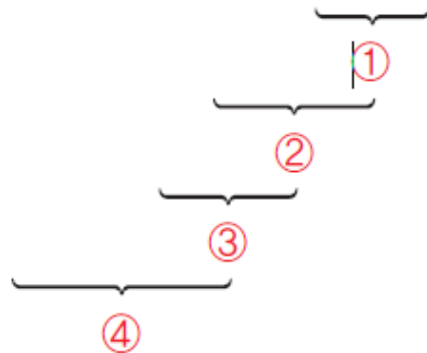


Example of a combination rule

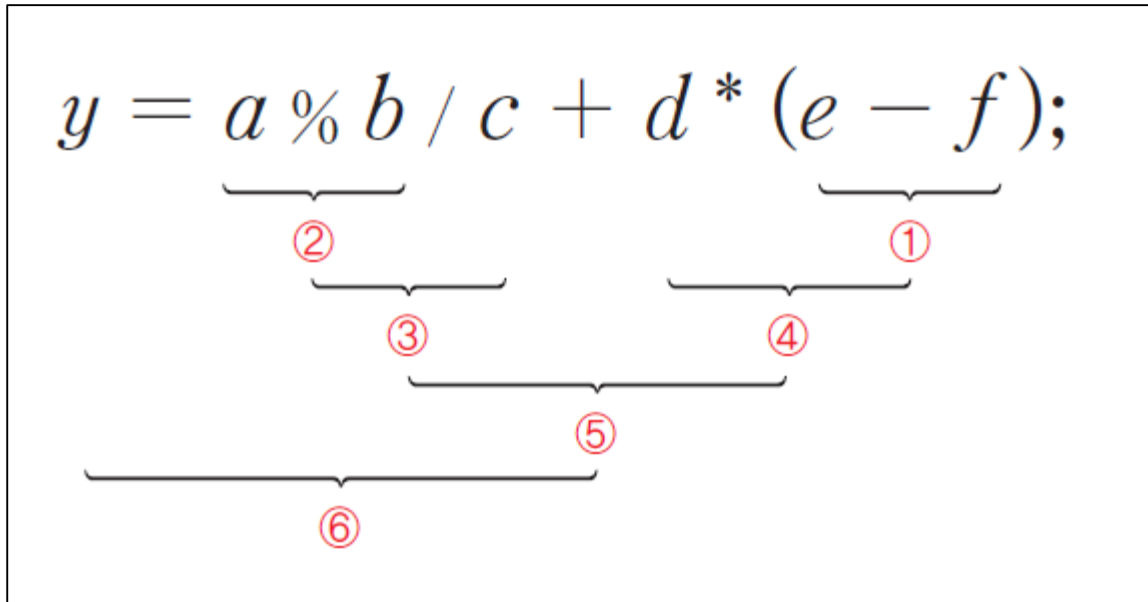
`x = y = z = 5;`



`y = - ++ --x;`



Example of a combination rule



Example

```
#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;
}
```



Q & A

