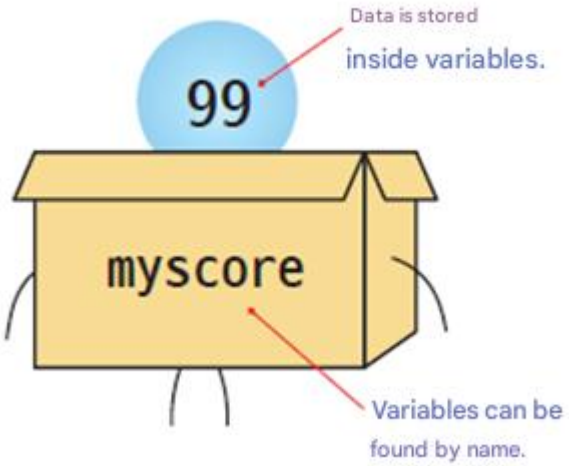
Ch.4 Variables and Data Types

What you will learn in this chapter

* Understanding the concept of variables and constants
* Data type
* Integer
* Real number
* Text type
* Use symbolic constants
* Overflow and Understanding Underflow

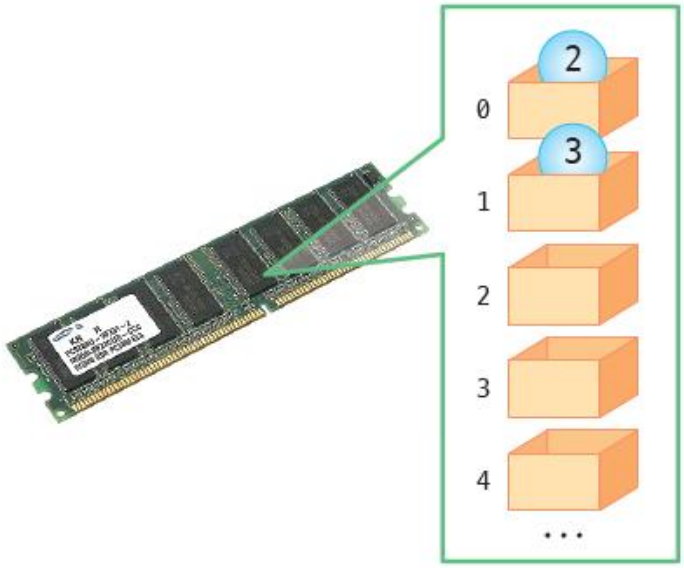
Variable

- Computer programs use variables to store values.
- Variables can be used to store scores in a game, or to store the prices of items we purchased at a supermarket.



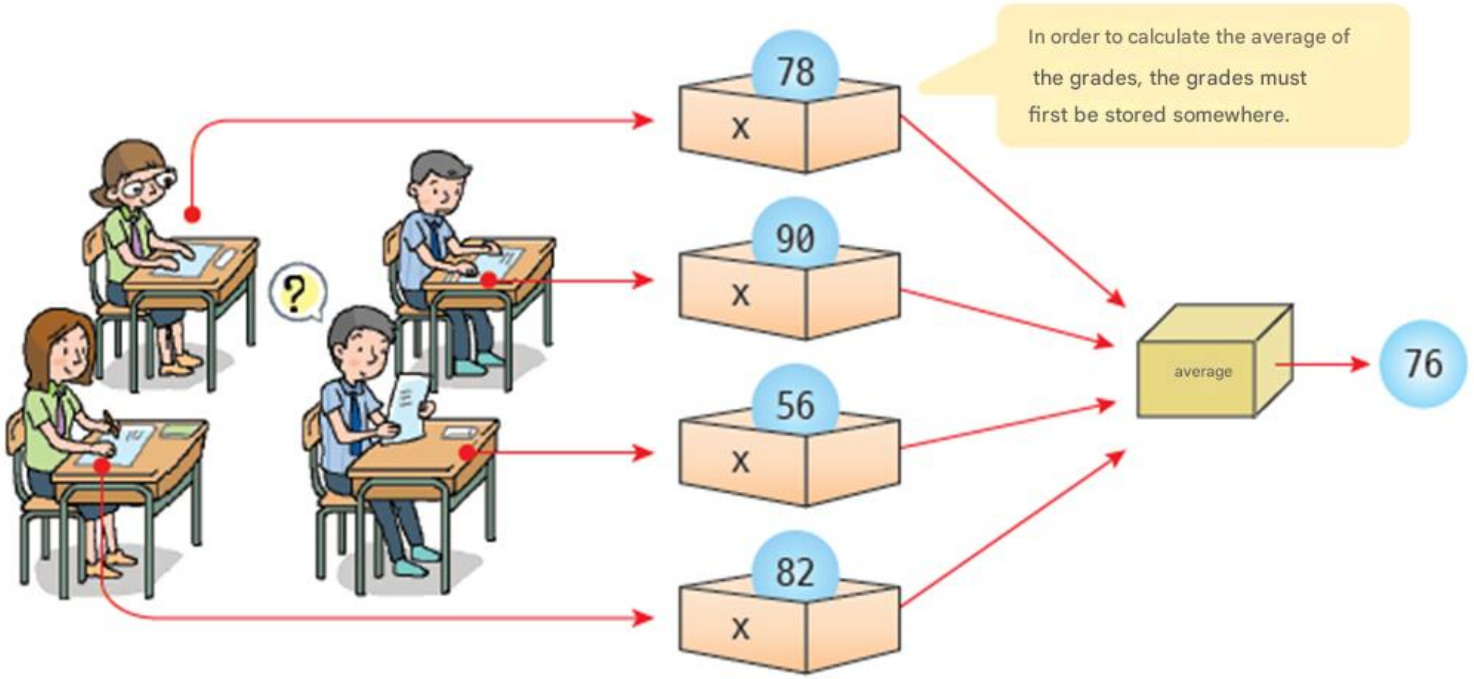
The variable is Where is it made ?

- Variables are created directly in main memory.
- We use memory space by using variable names.



Why do we need variables ? #1

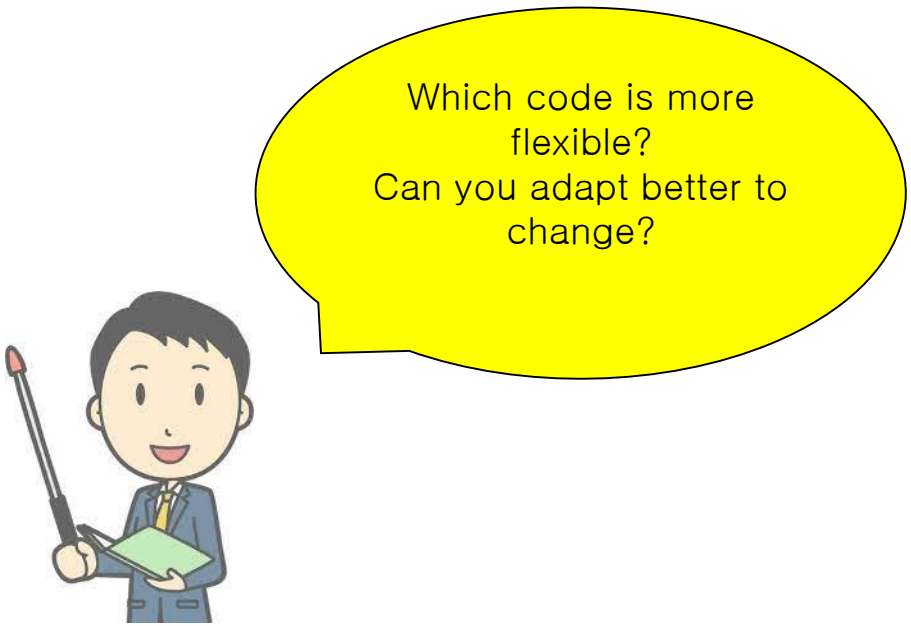
- A place to store data received from users. – If there are no variables, where will the data received from users be stored?



Why do we need variables ? #2

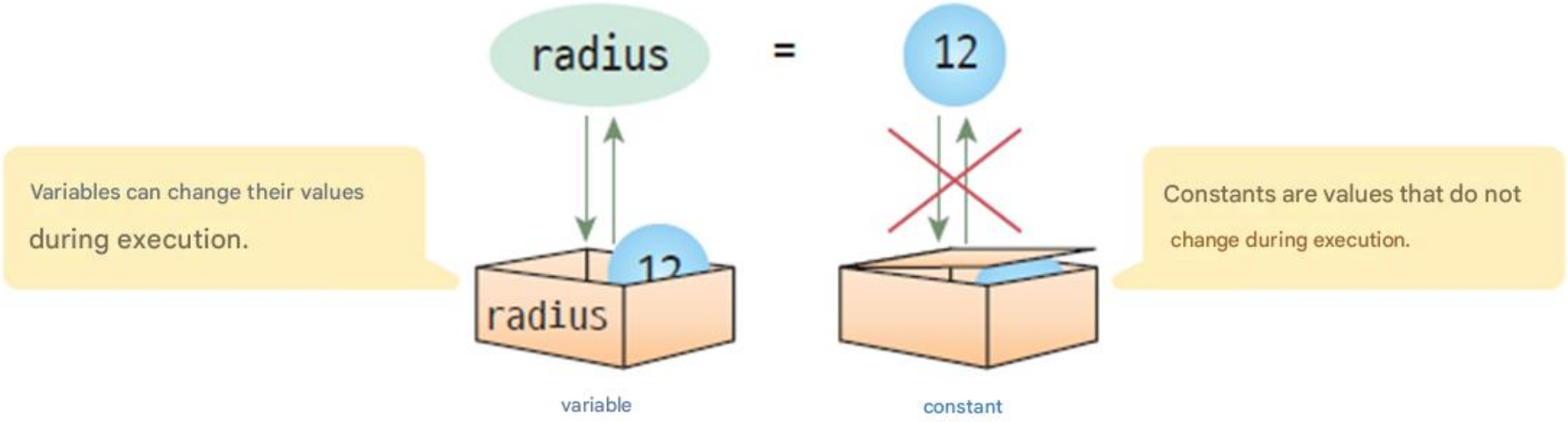
Code that does not use variables	Code that uses variables

// Area of a square with size 100×200
`area = 100 * 200;`
	// Area of a rectangle whose size is width×height
	`width = 100;`
	`height = 200;`
	`area = width * height;`



Variables and Constants

- Variable : A space where the stored value can be changed.
- Constant : A space where the stored value cannot be changed
  (Example) 3.14, 100, 'A', “Hello World!”



Example : Variables and Constants

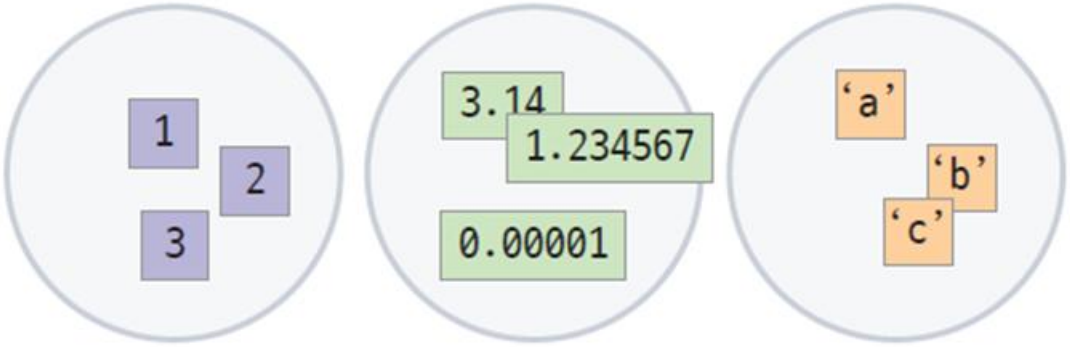
```
/* Program to calculate the area of a circle * /
#include < stdio.h >
int main( void )
{
        float radius; // Radius of the circle
        float area;  // Area of a circle

        printf ( " Enter the area of the circle :" );
        scanf ( "%f" , &radius);

        area = 3.141592 * radius * radius;
        printf ( " Area of the circle : %f ₩n" , area);

 return 0;
}
```

Variable

constant

Data type

- type : type (kind) of data
  - short, int, long: Integer data (100)
  - double, float: floating point data (3.141592)
  - char: character data ('A', 'a', ' han ')



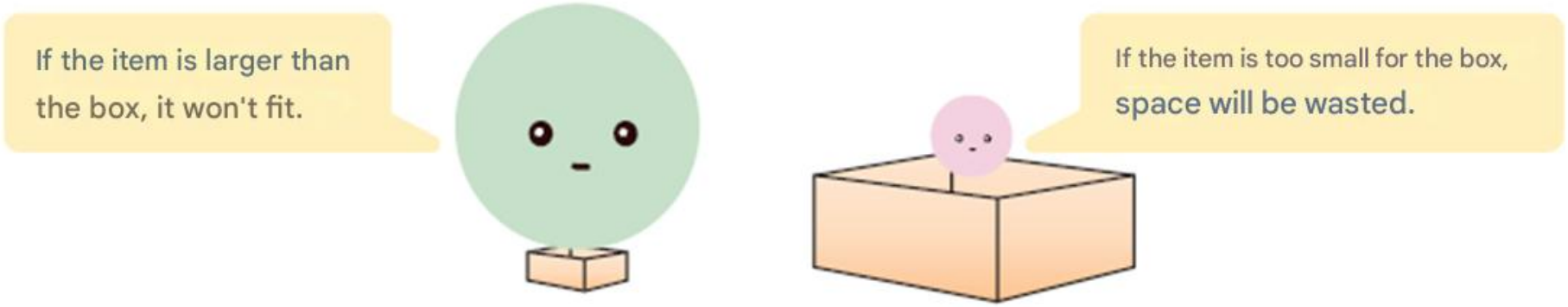
integer type　　float type　　character type

Terminology (Data type)

- 42 (decimal integer), 0x2A (hexadecimal integer), 052 (octal integer)
  - Decimal (base 10): 42
  - Hexadecimal (base 16): 0x2A
  - Octal (base 8): 052
  - All represent the same value → 42 in decimal

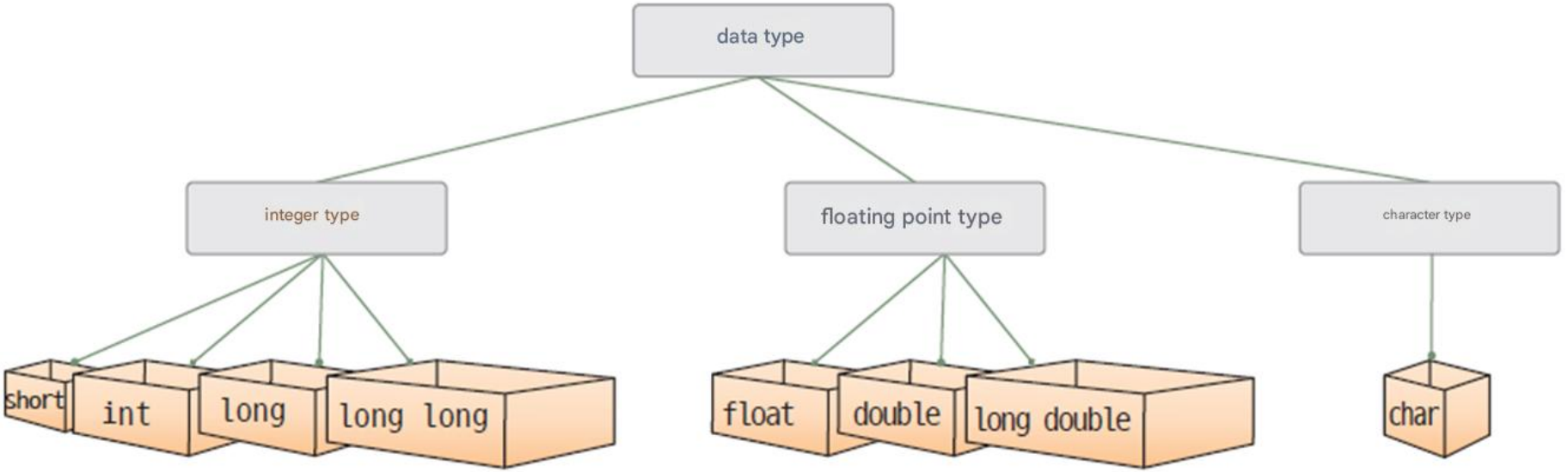
Why we need different data types

- It's like storing things in boxes.



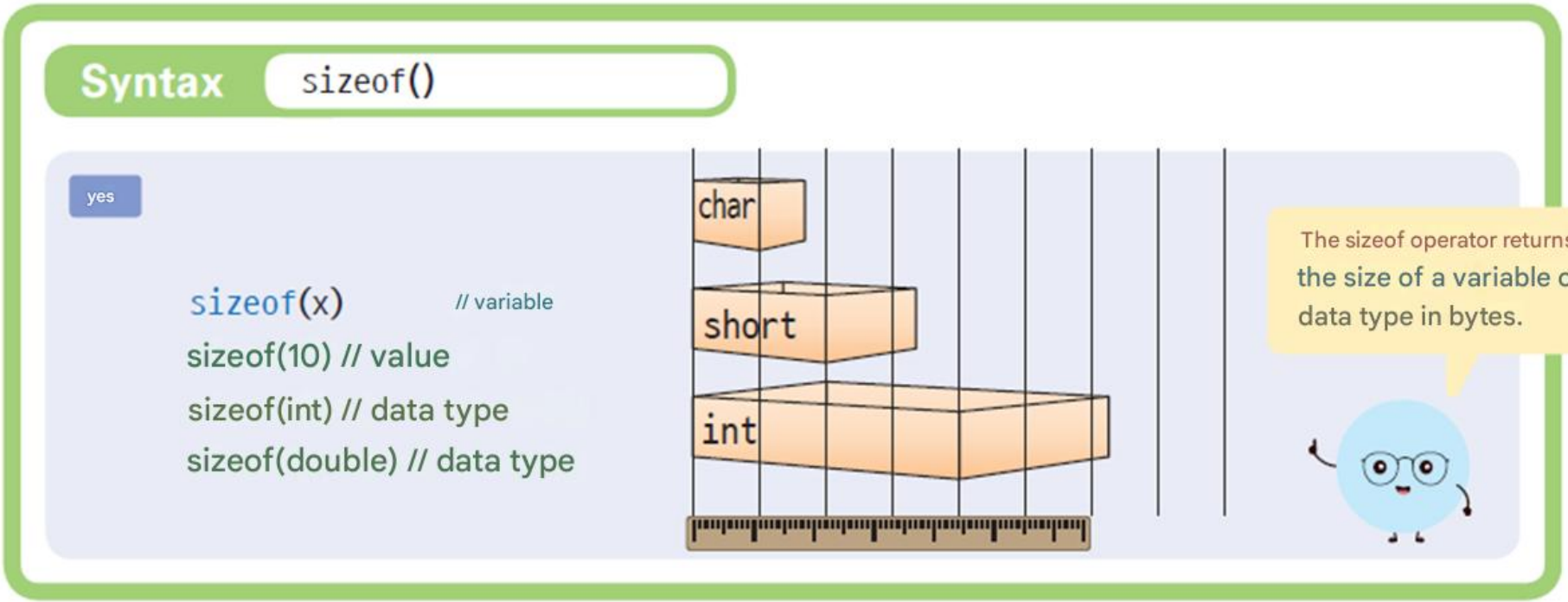
Classification of data types

- Data types can be broadly divided into integer types, floating-point types, and character types.



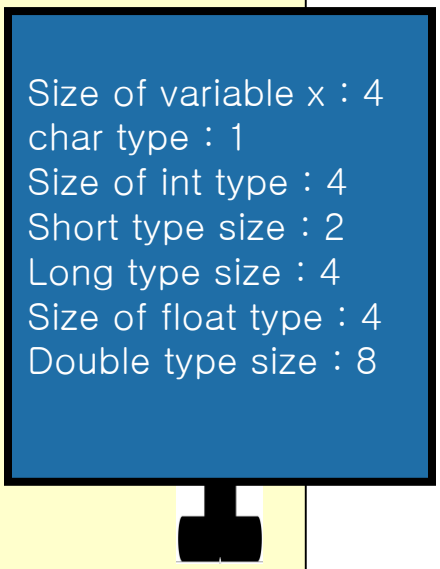
Size of Data type

- To find out the size of a data type, use sizeof You can use the operator. sizeof is an operator that returns the size of a variable or data type in bytes.



Example : Size of data type

```
#include < stdio.h >

int main( void )
{
        int x;

        printf ( " Size of variable x : %d\n" , sizeof (x));

        printf ( "Size of char type : %d\n" , sizeof ( char ));
        printf ( " Size of int type : %d\n" , sizeof ( int ));
        printf ( " Size of short type : %d\n" , sizeof ( short ));
        printf ( " Size of long type : %d\n" , sizeof ( long ));
        printf ( " Size of float type : %d\n" , sizeof ( float ));
        printf ( " Size of double type : %d\n" , sizeof ( double ));

        return 0;
}
```

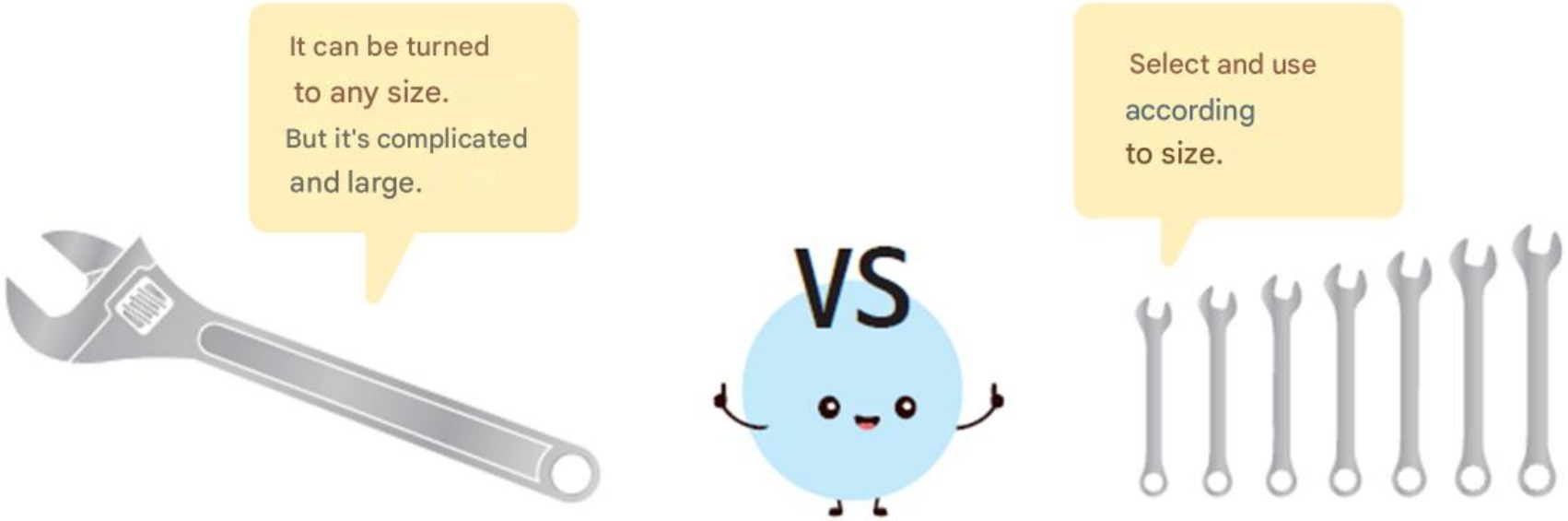


Integer

data type		bit	range
integer type	short	16 bit	-32768~32767
	int	32 bit	-2147483648~2147483647
	long		
	long long	64 bit	-9,223,372,036,854,775,808 ~9,223,372,036,854,775,807

Why are there so many different types of integers in C?

- The idea is to allow programmers to select and use them according to their intended use.
- the number of bits can expand the range of integers, but requires more memory space.



Integer type range

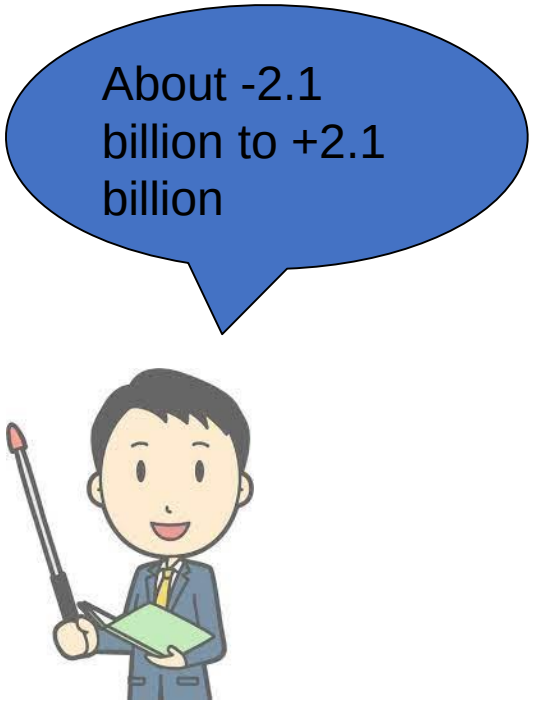
- int type

$$-2^{31}, \ldots, -2, -1, 0, 1, 2, \ldots, 2^{31} - 1$$
(-2147483648 ~ +2147483647)

- short type

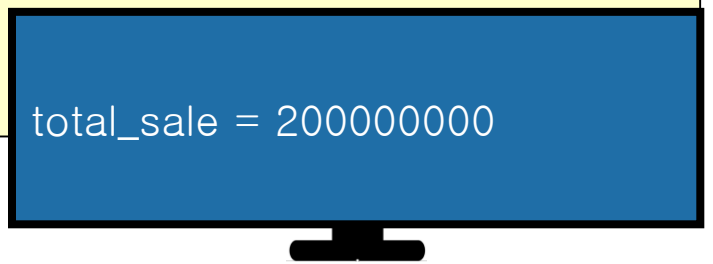
$$-2^{15}, \ldots, -2, -1, 0, 1, 2, \ldots, 2^{15} - 1$$
(-32768 ~ +32767)

- long type
  - Usually the same as int type



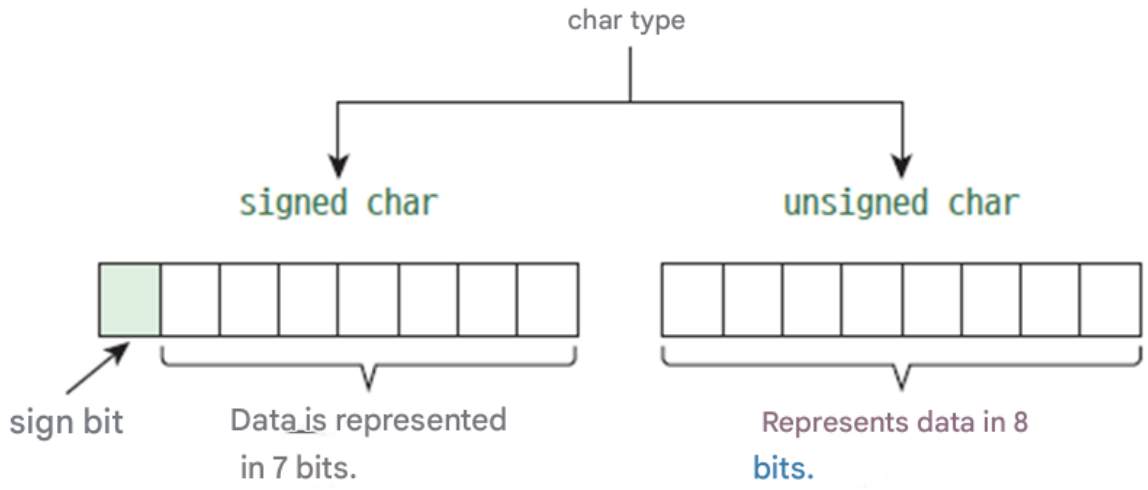
Example

```
/* Program to calculate the size of an integer data type * /
#include < stdio.h >

int main( void )
{
        short year = 0; // Initialize to 0 .
        int sale = 0; // Initialize to 0 .
        long total_sale = 0; // Initialize to 0 .
        long long large_value ; // 64- bit data type

        year = 10; // Be careful not to exceed about 32,000
        sale = 200000000; // Be careful not to exceed about 2.1 billion
        total_sale = year * sale; // Be careful not to exceed about 2.1 billion

        printf ( " total_sale = %d \n" , total_sale );

         return 0;
}
```



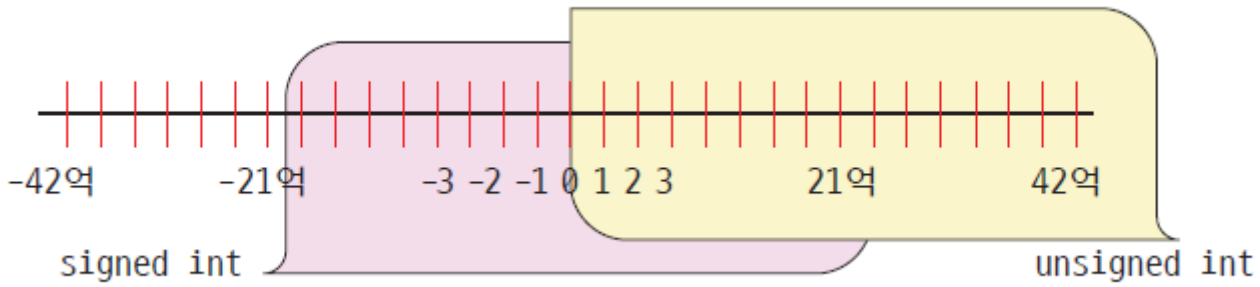
signed, unsigned modifiers

- unsigned
  - Means that only non-negative values are represented
  - unsigned int
- signed
  - Means that it represents a value with a sign
  - Commonly omitted



unsigned int

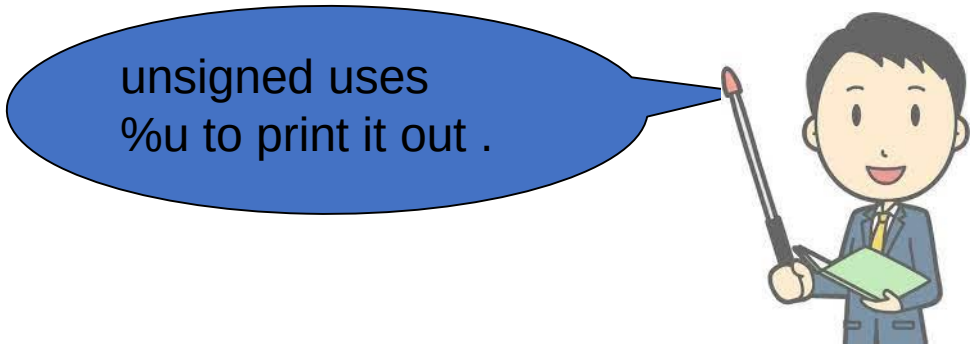
$0, 1, 2, \ldots, 2^{32} - 1$
(0 ~ +4294967295)



unsigned example

```
unsigned int speed; // unsigned int type
unsigned distance; // unsigned int distance and It's the same .
unsigned short players; // unsigned short type
```

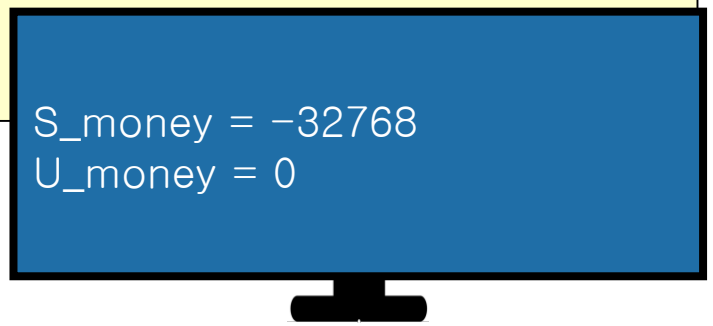
```
unsigned int sales = 2800000000; // about 2.8 billion
printf ( " %u \n" , sales); // If you use %d , it will be printed as a negative number
```



Overflow

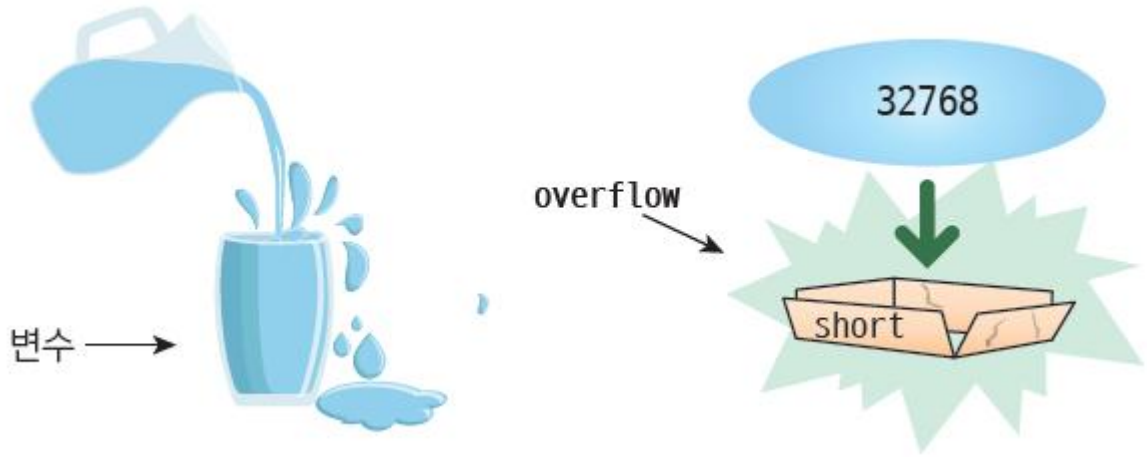
```c
#include < stdio.h >
#include < limits.h >

int main( void )
{
    short s_money = SHRT_MAX; // Initialize to maximum value . 32767
    unsigned short u_money = USHRT_MAX; // Initialize to maximum value . 65535

    s_money = s_money + 1;
    printf ( " s_money = %d" , s_money );

    u_money = u_money + 1;
    printf ( " u_money = %d" , u_money );
    return 0;
}
```

Overflow occurred !!



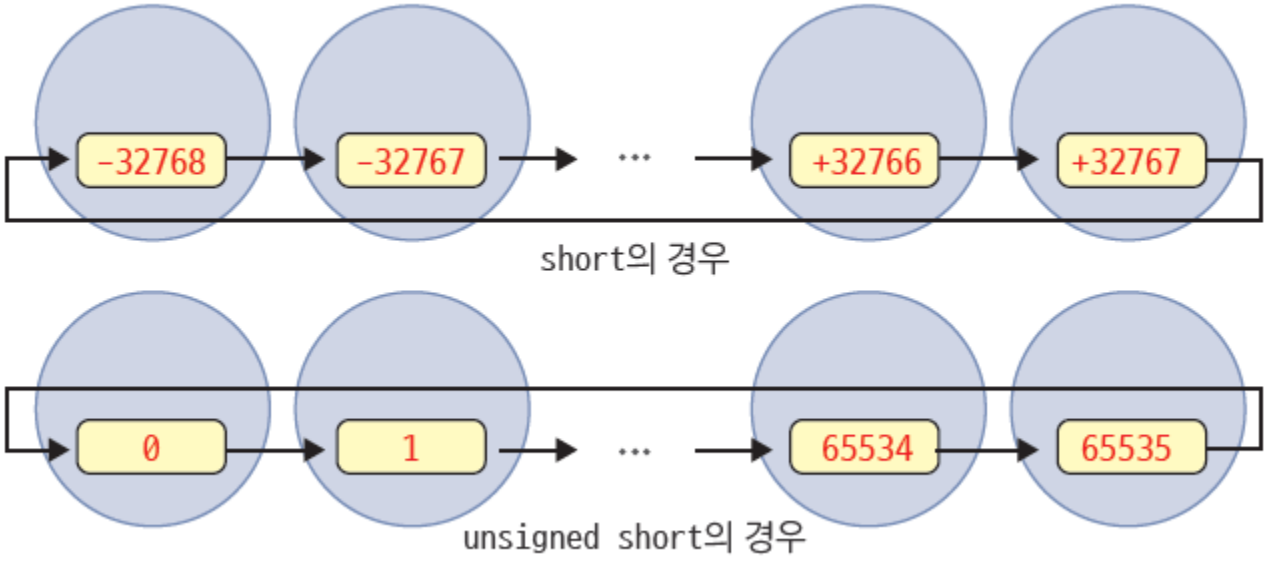
Overflow

- Overflow : Occurs when you try to store a number that excee ds the range that a variable can represent



Overflow

- There is a pattern .
  - It works similarly to a water meter or a car odometer.



reference

Note

The maximum and minimum values for each data type are defined in limits.h.

```
#define CHAR_MIN    (-128)
#define CHAR_MAX    127

#define SHRT_MIN    (-32768)          /* minimum (signed) short value */
#define SHRT_MAX      32767           /* maximum (signed) short value */
#define USHRT_MAX     0xffff          /* maximum unsigned short value */

#define INT_MIN     (-2147483647 - 1) /* minimum (signed) int value */
#define INT_MAX       2147483647      /* maximum (signed) int value */
#define UINT_MAX      0xffffffff      /* maximum unsigned int value */
```

Integer constant

- Basically, when you write a number, It becomes int type.
  - sum = 123; // 123 is int type
- the data type of a constant, do the following :
  - sum = 123L; // 123 is long type

suffix	data type	yes
u or U	unsigned int	123u or 123U
l or L	long	1231 or 123L
ul or UL	unsigned long	123ul or 123UL

Decimal , octal , hexadecimal

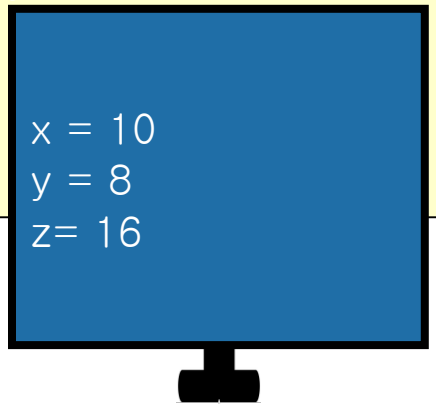
- Octal
  - $012_{8} = 1 \times 8^{1} + 2 \times 8^{0} = 10$
- Hexadecimal
  - $0xA_{16} = 10 \times 16^{0} = 10$

decimal	octal	hexadecimal
0	00	0x0
1	01	0x1
2	02	0x2
3	03	0x3
4	04	0x4
5	05	0x5
6	06	0x6
7	07	0x7
8	010	0x8
9	011	0x9
10	012	0xa
11	013	0xb
12	014	oxc
13	015	0xd
14	016	0xe
15	017	0xf
16	020	0x10
17	021	0x11
18	022	0x12

Example

```
/* Integer constant program * /
#include < stdio.h >

int main( void )
{
      int x = 10; // 10 is a decimal number, is of type int , and has a value of 10 in decimal .
      int y = 010; // 010 is an octal number, of type int , and its value is 8 in decimal .
      int z = 0x10; // 010 is a hexadecimal number, int type, and its value is 16 in decimal .

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

      return 0;
}
```

x = 10
y = 8

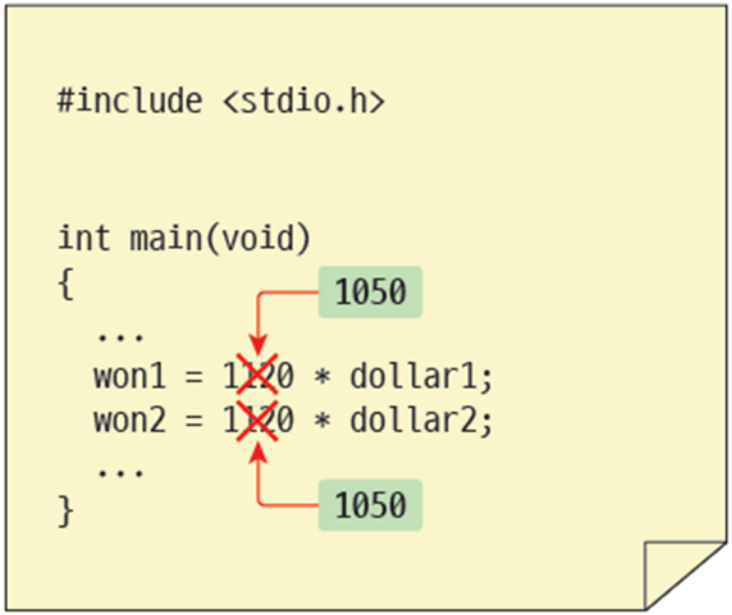


Symbolic Constant

- *constant :* A constant expressed using symbols.
- (example)
  - won = 1120 * dollar; // (1) Use actual value
  - won = EXCHANGE_RATE * dollar; // (2) Use of symbolic constants

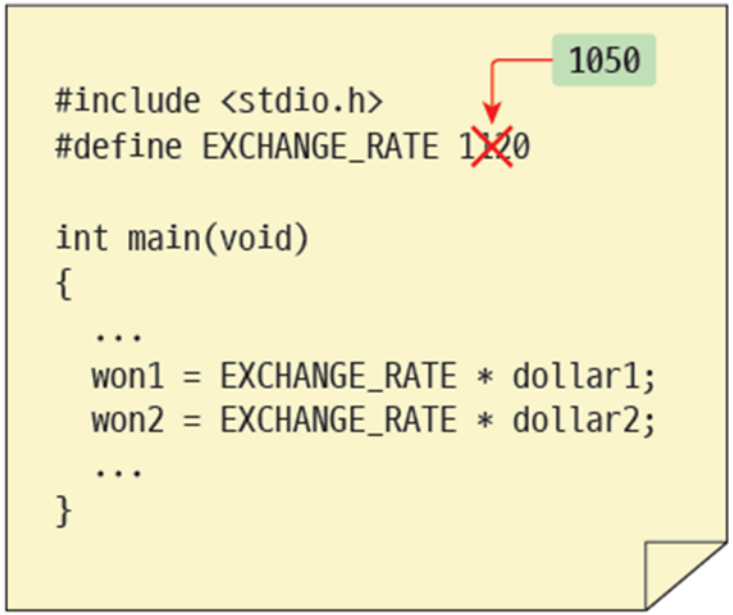
- Advantages of symbolic constants
  - Readability is improved .
  - The values can be easily changed .

Advantages of symbolic constants



When using literal constants:

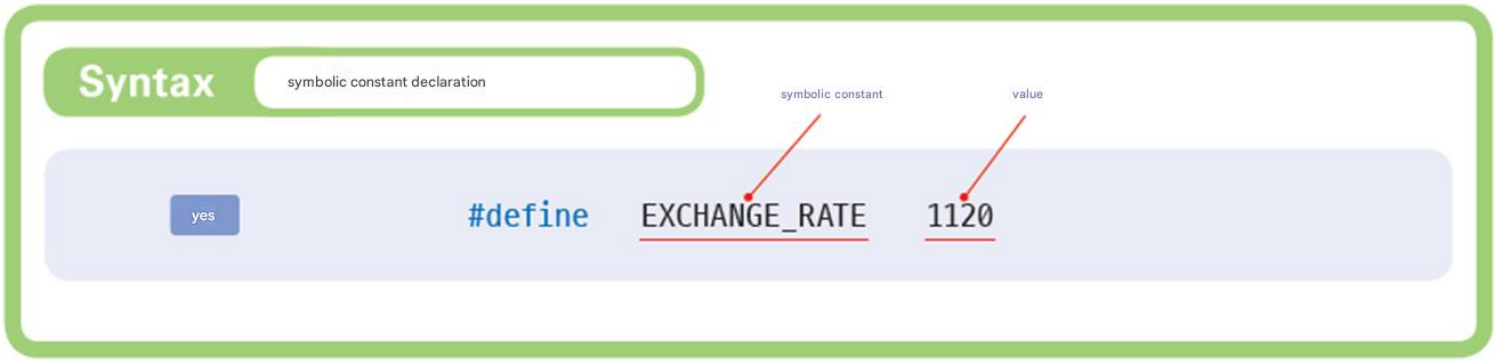
Every place it appears must be corrected.

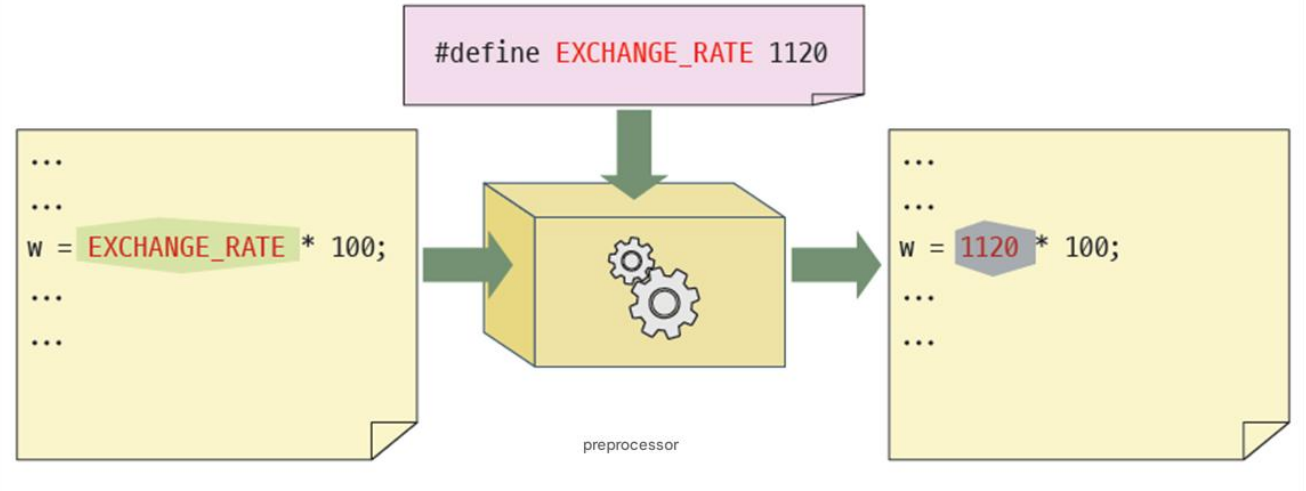


When using symbolic constants:

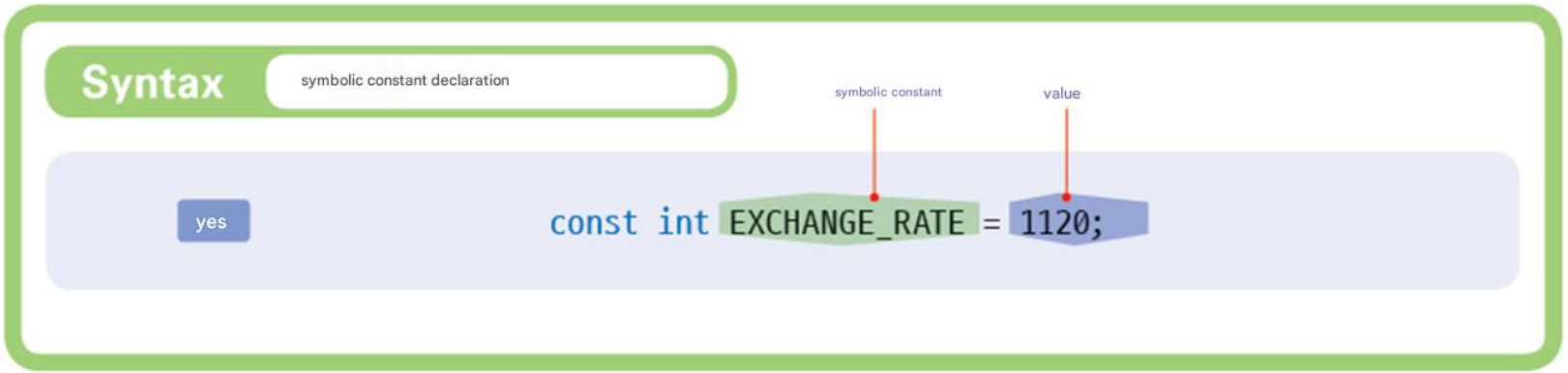
You only need to modify the places where symbolic constants are defined.

How to create symbolic constants #1





How to create symbolic constants #2



Example : Symbolic Constants

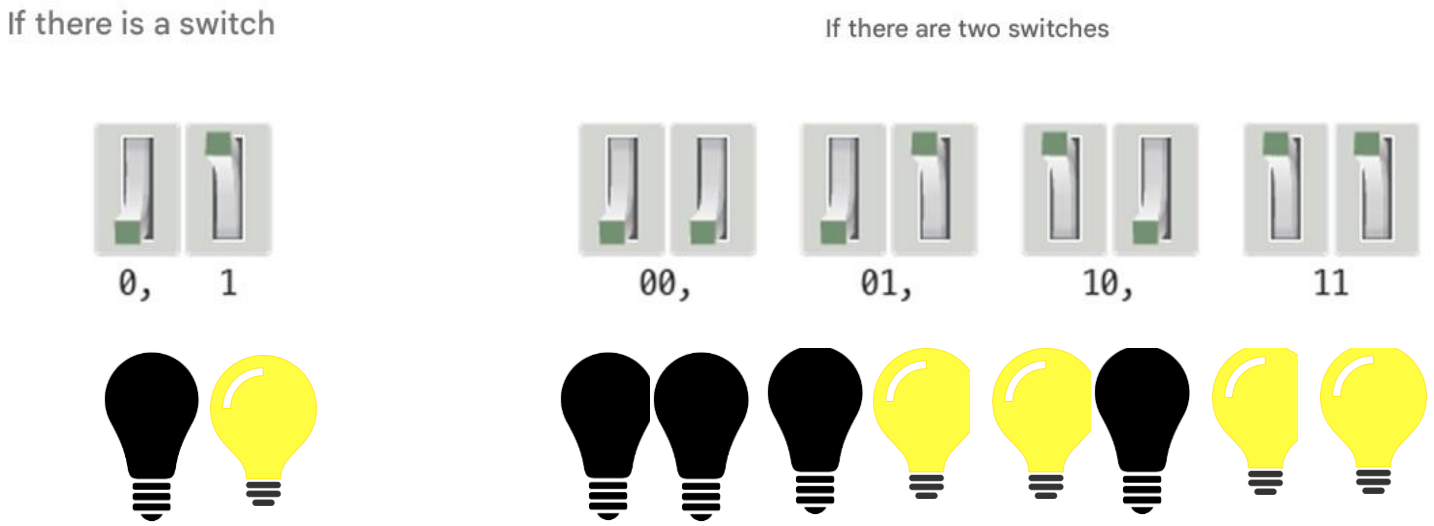
```
#include < stdio.h >
#define TAX_RATE 0.2

int main( void )
{
        const int MONTHS = 12;
        int m_salary , y_salary ; // Declare variables

        printf ( " Enter your salary : " ); // Input instructions
        scanf ( "%d" , & m_salary );
        y_salary = MONTHS * m_salary ; // Calculate net income
        printf ( " Your annual salary is %d ." , y_salary );
        printf ( " Tax is %f ." , y_salary *TAX_RATE);

 return 0;
}
```

Symbol constant

Enter your salary : 100
The annual salary is 1200 .
The tax is 240.000000 .

Integer representation method

- In computers, integers are represented in binary form, and bi nary numbers are represented by electronic switches .

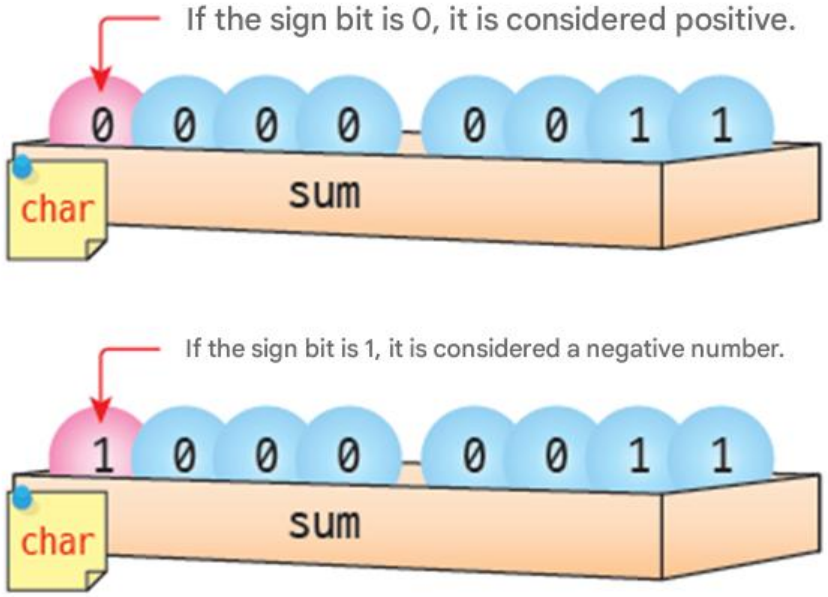


short type

beat pattern	integer	note
0000000000000000	0	positive integer
0000000000000001	1	
0000000000000010	2	
0000000000000011	3	
0000000000000100	4	
0000000000000101	5	
...	...	

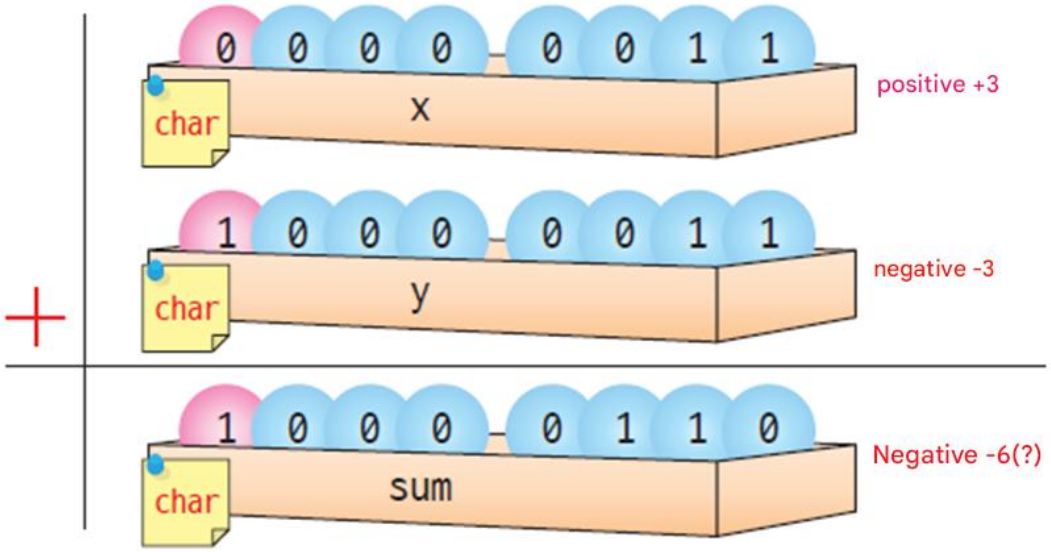
Integer representation method

- Positive
  - You can convert decimal to binary and save it .
- negative
  - Usually the first bit is used as the sign bit .
  - A problem arises .



First way to express negative numbers

- The first method is to consider the very first bit as the sign bit.
- When performing addition operations on positive and negative numbers , the results are inaccurate.
  - (Example) +3 + (-3)



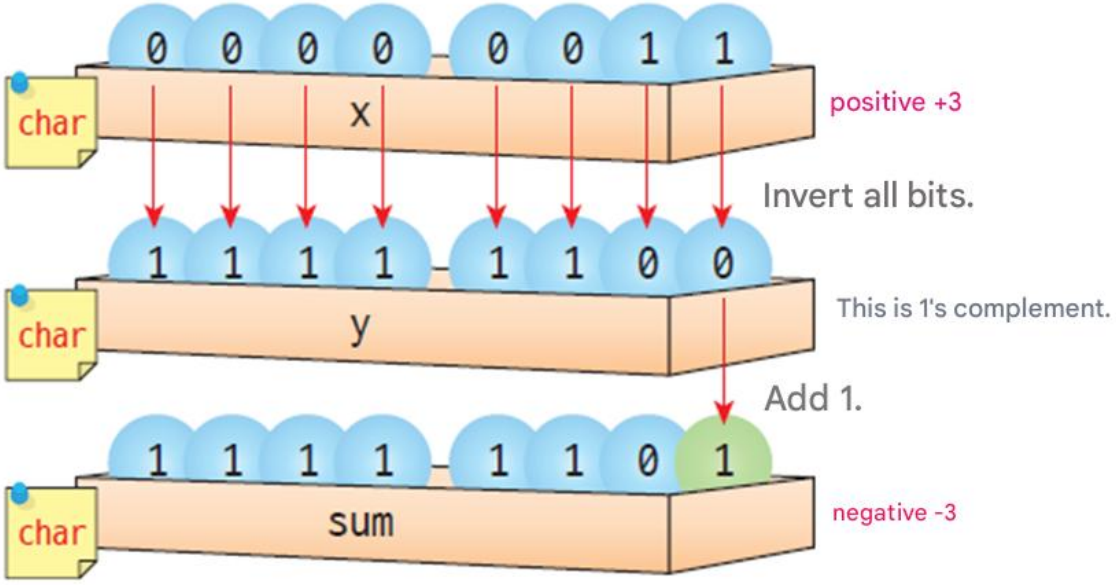
Computers can only do addition

- Computers only have addition circuits to reduce the size of the circuit.
- Subtraction is converted to addition as follows: Handle it.

$$3-3 = 3+(-3)$$

Second way to express negative numbers

- Representing negative numbers with 2 's complement.
  -> Standard way to represent negative numbers
- 2's complement : How to make (-3)
  1. Invert all bits
  2. Add 1

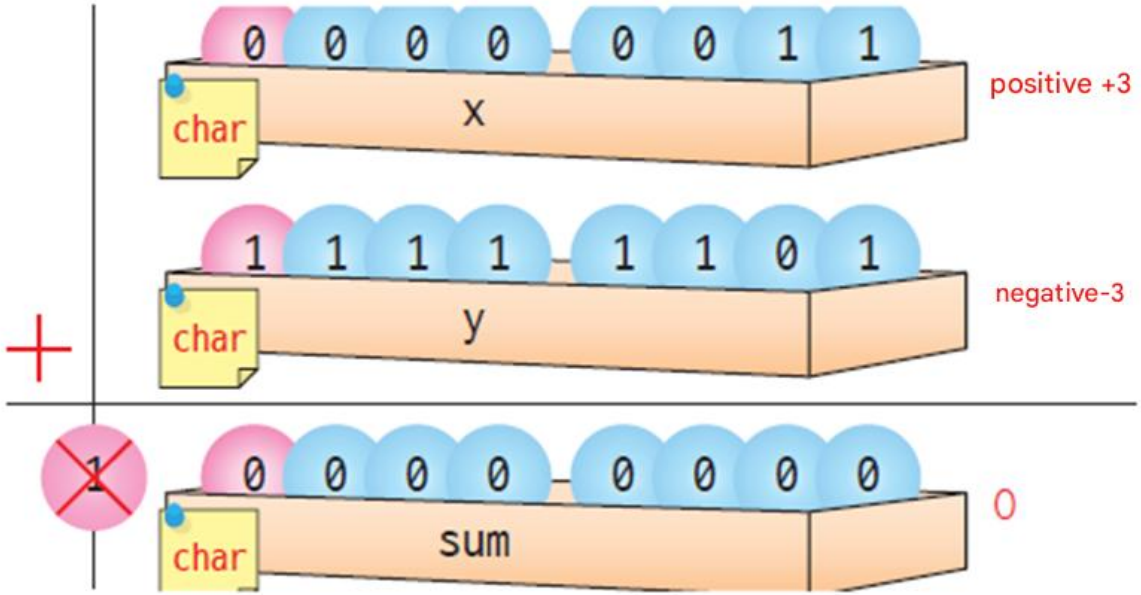


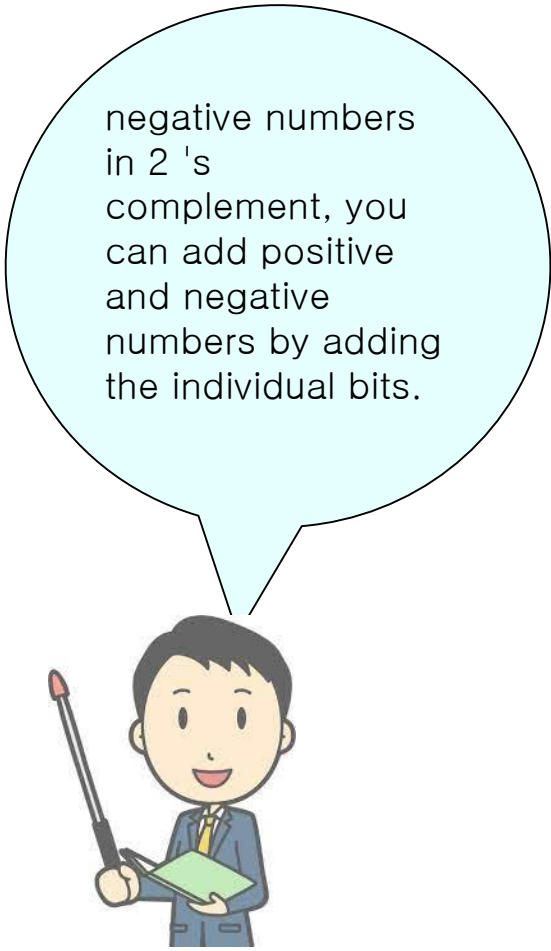
2's complement

bit	unsigned integer	Signed integer (2's complement)
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

bit	unsigned integer	Signed integer (2's complement)
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
0111 1110	126	126
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
1000 0010	130	-126
1111 1110	254	-2
1111 1111	255	-1

2 's complement,

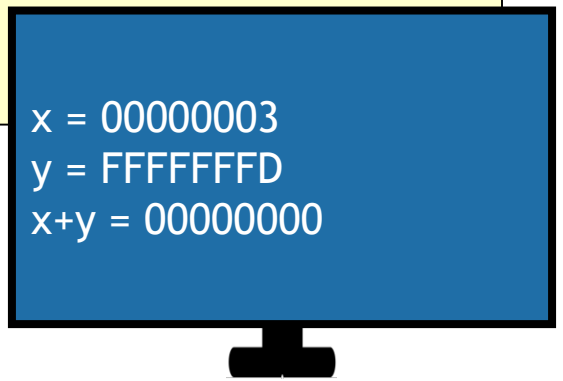




Example

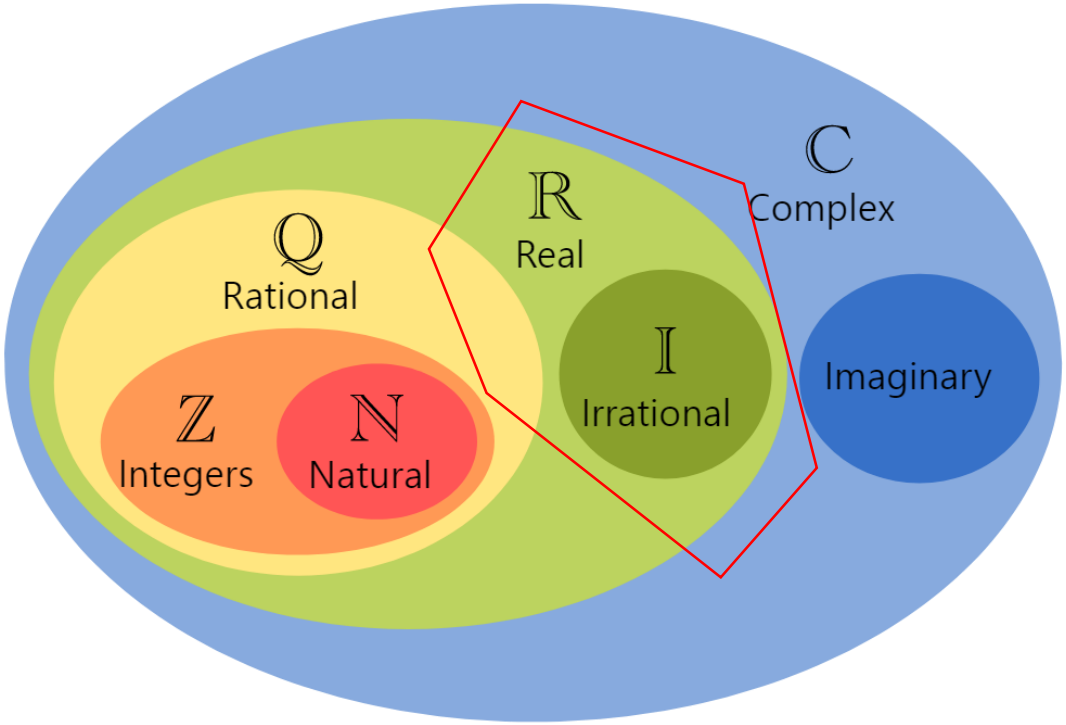
```
/* 2 's complement program * /
#include < stdio.h >

int main( void )
{
        int x = 3;
        int y = -3;

        printf ( "x = %08X\n" , x); // Print as 8 - digit hexadecimal number .
        printf ( "y = %08X\n" , y); // Print as 8- digit hexadecimal number .
        printf ( " x+y = %08X\n" , x+y ); // Print as 8 - digit hexadecimal number .

        return 0;
}
```

negative numbers are represented in 2 's complement .



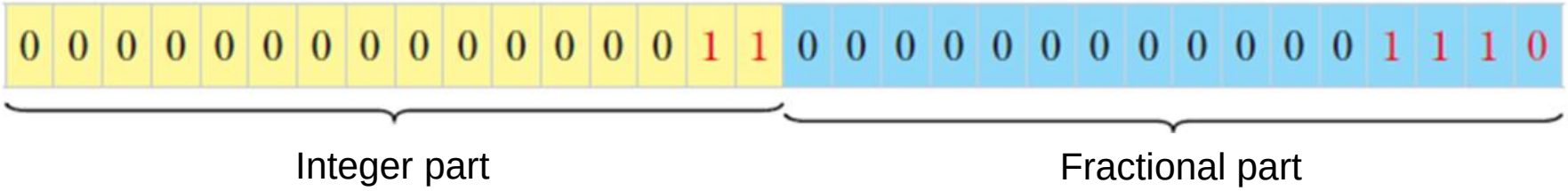
How to indicate a real number

- In mathematics, a real number is a number with a decimal point, such as 3.14. Real numbers are an essential element when writing applications in science or engineering that deal with very large or very small numbers.



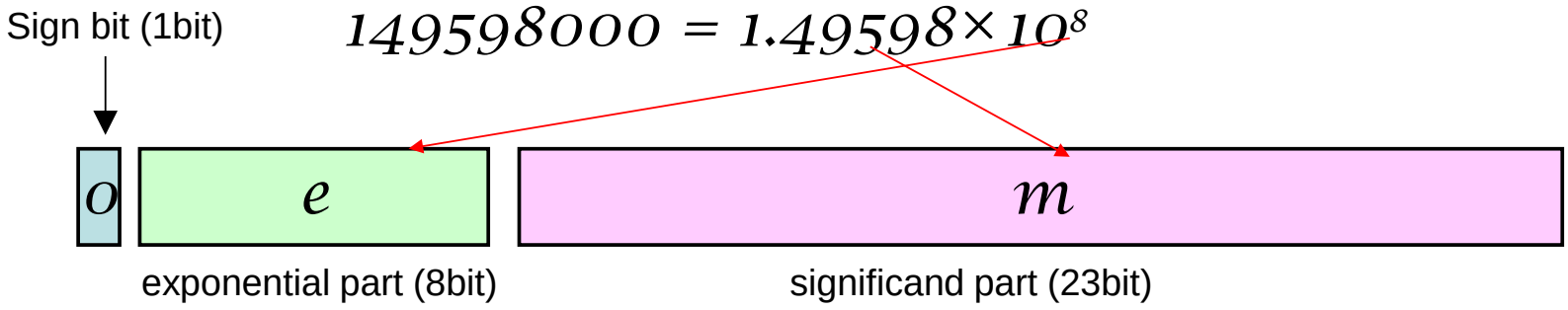
How to express a real number #1

- Allocate a certain number of bits for the integer part and a certain number of bits for the fractional part.
- the total is 32 bits, 16 bits are allocated for the integer part and 16 bits for the fractional part.
- It cannot express the very large numbers required in science and engineering.



How to express a real number #2

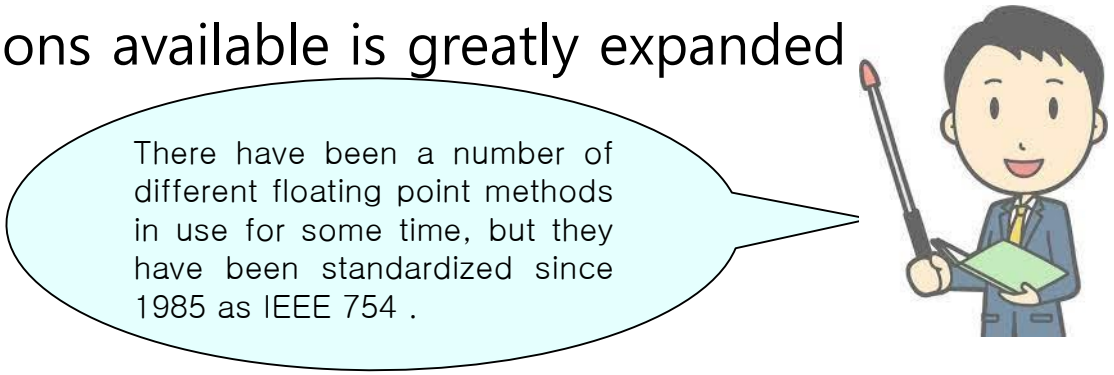
- Floating point method



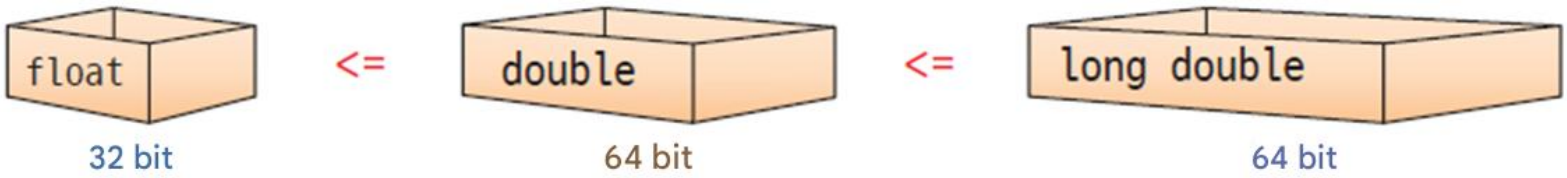
Value of real number $= (-1)^s * (1.m) * 2^{e-127}$

- The range of expressions available is greatly expanded

  $10^{-38} \sim 10^{+38}$



Floating point type



data type	designation	size	range
float	single-precision floating point	32 bit	$\pm 1.17549 \times 10^{-38} \sim \pm 3.40282 \times 10^{+38}$
double long double	double-precision floating point	64 bit	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{+308}$

Format specifier for printing real numbers

- %f
  - printf (“%f”, 0.123456789); // Prints 0.123457
- %e
  - printf (“%e”, 0.123456789); // Prints 1.234568e-001

Example

```
/* Calculating the size of floating point data types * /
#include < stdio.h >
int main( void )
{
        float x = 1.23456789012345678 9;
        double y = 1.23456789012345678 9;

        printf ( " Size of float =%d\n" , sizeof ( float ));
        printf ( " Size of double =%d\n" , sizeof ( double ));

        printf ( "x = %30.25f\ n" ,x );
        printf ( "y = %30.25f\ n" ,y );
       return 0;
}
```

```
size of float = 4
size of double =8
x = 1.234567880630493200000000 0
y = 1.234567890123456700000000 0
```

Floating point constant

mistake	exponential notation	meaning
123.45	1.2345e2	1.2345×10^{2}
12345.0	1.2345e4	1.2345×10^{4}
0.000023	2.3e-5	2.3×10^{-5}
2,000,000,000	2.0e9	2.0×10^{9}

```
1.23456
2.          // You can just add a decimal point.
.28         // The integer part is not required.
2e+10       // + or - signs can be added to the exponent.
9.26E3      // 9.26×10^3
0.67e-9     // 0.67×10^-9
```

Floating point overflow

```
#include < stdio.h >

int main( void )
{
  float x = 1e39;
  printf ( "x = %e\ n" ,x );
}
```

Overflow occurs due to large number

Floating point underflow

```
#include <stdio.h>

int main( void )
{
        float x = 1.23456e-38;
        float y = 1.23456e-40;
        float z = 1.23456e-46;

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

Underflow occurs

```
x = 1.234560e-038
y = 1.234558e-040
z = 0.000000e+000
```

Precautions when using floating point numbers

- There may be errors !

```c
#include < stdio.h >

int main( void )
{
    float value = 0.1;

    printf ( "%.20f \n" , value); // %.20f means to print with 20 decimal places .
    return 0;
}
```

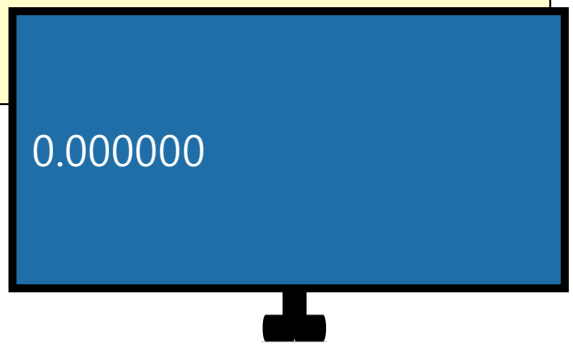
This is because there are values that cannot be represented exactl y in binary . 0.1 is one of them .

Precautions when using floating point numbers

- There may be errors !

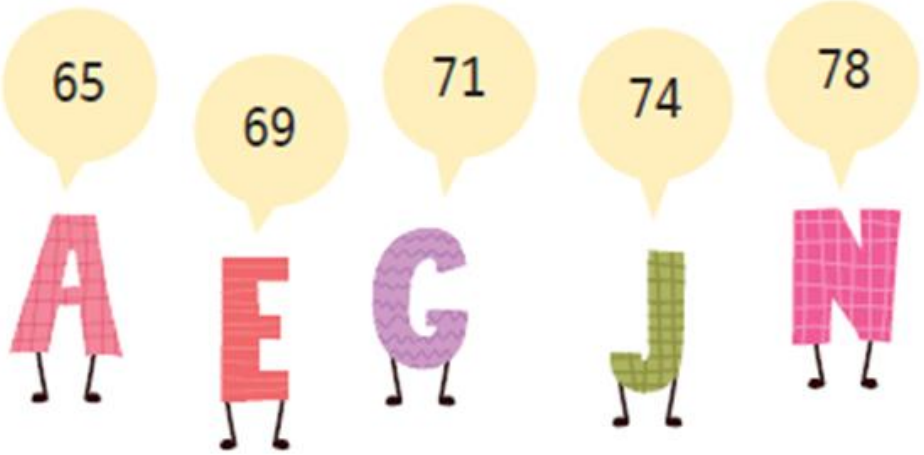
```
#include <stdio.h>

int main( void )
{
        double x;

        x = (1.0e20 + 5.0)-1.0e20;
        printf( "%f \n" ,x);
        return 0;
}
```

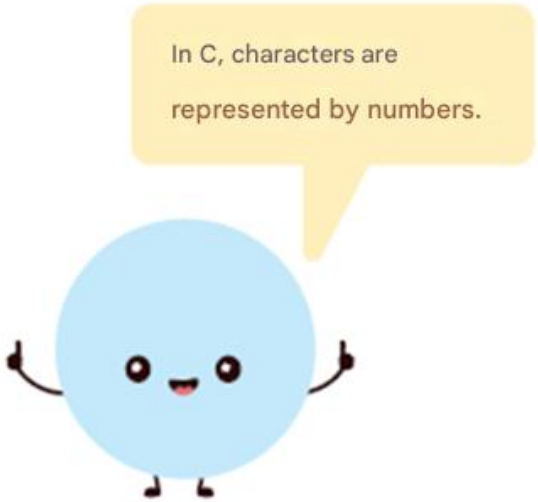
Floating point operations introduce errors .
It is calculated as 0 , not 5.0 .



Text type

- Text is more important to humans than to computers
- Letters are also expressed using numbers
- A common standard is needed.
- ASCII code (**ASCII:** American Standard Code for Information Interchange)





ASCII Code table (some)

Dec	Hex	문자	Dec	Hex	문자	Dec	Hex	문자	Dec	Hex	문자
0	0	NULL	20	14	DC4	40	28	(	60	3C	<
1	1	SOH	21	15	NAK	41	29	)	61	3D	=
2	2	STX	22	16	SYN	42	2A	*	62	3E	>
3	3	ETX	23	17	ETB	43	2B	+	63	3F	?
4	4	EOL	24	18	CAN	44	2C	,	64	40	@
5	5	ENQ	25	19	EM	45	2D	-	65	41	A
6	6	ACK	26	1A	SUB	46	2E	.	66	42	B
7	7	BEL	27	1B	ESC	47	2F	/	67	43	C
8	8	BS	28	1C	FS	48	30	0	68	44	D
9	9	HT	29	1D	GS	49	31	1	69	45	E
10	A	LF	30	1E	RS	50	32	2	70	46	F
11	B	VT	31	1F	US	51	33	3	71	47	G
12	C	FF	32	20	space	52	34	4	72	48	H
13	D	CR	33	21	!	53	35	5	73	49	I
14	E	SO	34	22	"	54	36	6	74	4A	J
15	F	SI	35	23	#	55	37	7	75	4B	K
16	10	DLE	36	24	$	56	38	8	76	4C	L
17	11	DC1	37	25	%	57	39	9	77	4D	M
18	12	DC2	38	26	&	58	3A	:	78	4E	N
19	13	DC3	39	27	'	59	3B	;	79	4F	O

ASCII Code table (some)

Dec	Hex	문자
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c

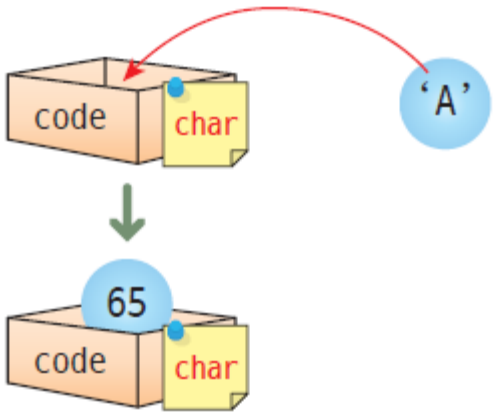
Dec	Hex	문자
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w

Dec	Hex	문자
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	¦
125	7D	}
126	7E	~
127	7F	DEL

Character variable

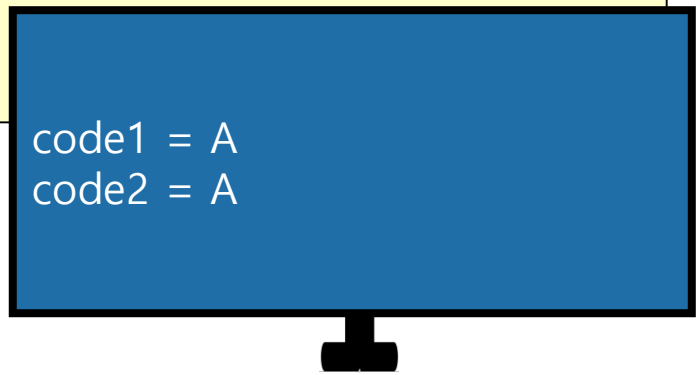
- char type Use to store text .

```
char code;
code = 'A' ;
```



Example

```
/* Character variables and character constants * /
#include < stdio.h >

int main( void )
{
        char code1 = 'A' ; // Initialize to a character constant
        char code2 = 65; // Initialize to ASCII code

        printf ( “code1 = %c\n" , code1);
        printf ( “code2 = %c\n" , code2);
}
```



Tip: Korean How to express

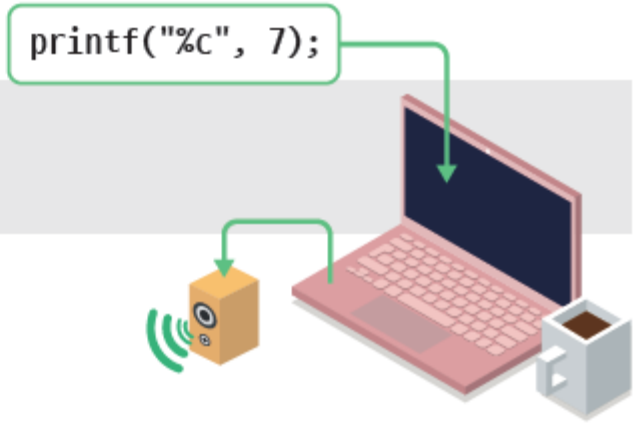
How to express Hangul

Hangul cannot be expressed in 8 bits. This is because there are more characters than English letters. There are two main ways to express Hangul on a computer. The first is to assign a code to each letter. For example, '가' is assigned the code 0xb0a1. The number of letters that can be expressed in Hangul is 11,172, so 8 bits (28=256) are not enough to assign codes to these letters, and 16 bits (216=65,536) are required. This is called a complete form. A representative code system is Unicode. Unicode is an industry standard designed to consistently express and handle all characters in the world on computers. It is established by the Unicode Consortium, and the current latest version is Unicode 10.0. This standard includes character sets, character encodings, character information databases, and algorithms for handling characters. Another method is to use the same 16 bits, but assign 5 bits to each of the initial consonant, medial vowel, and final consonant of the letter, and use the first bit as a symbol to distinguish between alphanumeric characters and Hangul. In other words, if the first bit is 1, it is Hangul, and if it is 0, it is an alphanumeric character. This method of expressing Hangul is called a combinational method.

Control characters

- Characters used for control purposes rather than printing pur poses.
  - (Example) Line break character , tab character , ringtone character ,

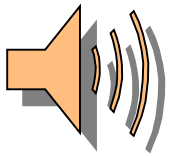
```
char beep = 7;
printf("%c", beep);
```



How to represent control characters

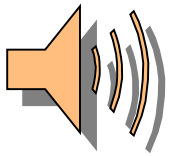
- Use ASCII code directly

```
char beep = 7;
printf ("%c", beep);
```



- Using escape sequences

```
char beep = '\a';
printf ("%c ", beep);
```



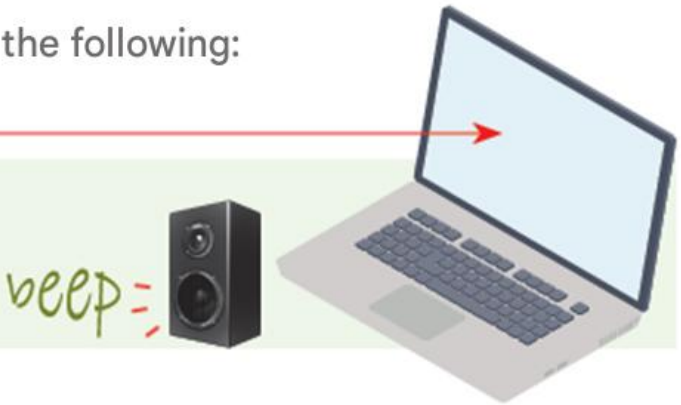
Escape Sequence

control character	name	meaning
\0	null character	
\a	warning(bell)	A "beep" warning sound occurs
\b	backspace	Moves the cursor one character back from its current position.
\t	horizontal tab	Moves the cursor position to the next tab position set on the current line.
\n	newline	Moves the cursor to the beginning of the next line.
\v	vertical tab	Move the cursor to the next set vertical tab position
\f	form feed	It is mainly used in printers to force the printer to turn to the next page.
\r	carriage return	Moves the cursor to the beginning of the current line.
\"	double quotation marks	The original double quotes themselves
\'	single quote	The original single quote itself
\\	backslash	The original backslash itself

How to make a program beep ?

To make your program emit a beep using a special string, do the following:

```
char beep = '\a';
printf("%c", beep);
```



Backslash ₩

- Backslash before characters with special functions If you plac e ₩ , the special meaning of the character is lost .

```
printf (" \" My own Hollywood \" UCC craze ");
```

“My Own Hollywood” UCC Craze

```
printf (" \\ is used to display control characters . ");
```

\ is used to indicate control characters .

Example

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

	printf ( " Please enter your ID and password as 4 digits :\n" );

	printf ( "id: ____ \b\b\b\b " );
	scanf ( "%d" , &id);

	printf ( "pass: ____ \b\b\b\b " );
	scanf ( "%d" , &pass);
	printf ( “\a The entered ID is \”%d\” and the password is \”%d\” ." , id, pass);

	return 0;
}
```

```
your ID and password using 4 digits :
id: 1234
pass: 5678
The entered ID is "1234" and the password is "5678" .
```

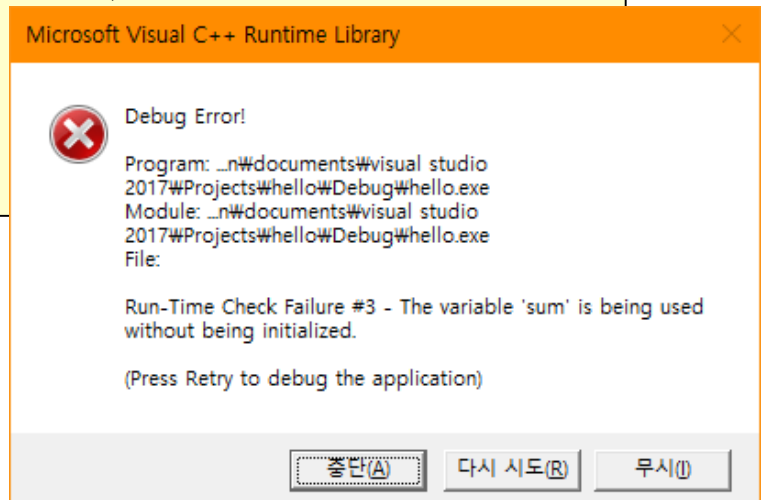
char type as integer

- The char type can be used to store 8- bit integers .

```
#include < stdio.h >

int main( void )
{
        char code = 'A' ;
        printf ( "%d %d %d \n" , code, code + 1, code + 2); // 65 66 67 is printed .
        printf ( "%c %c %c \n" , code, code + 1, code + 2); // ABC is printed .
        return 0;
}
```

Lab: Variables Initial value

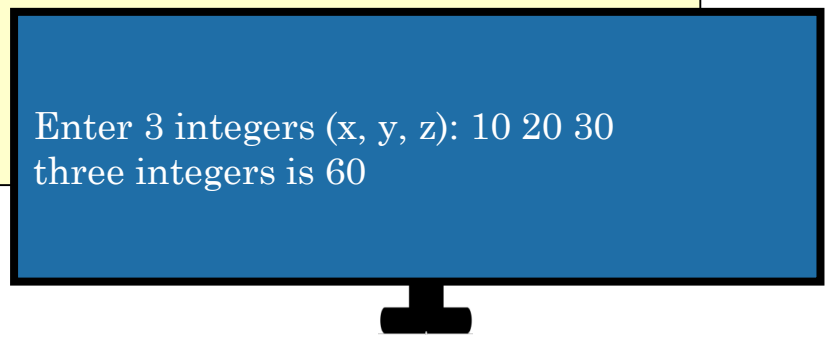
```
#include < stdio.h >
int main( void )
{
        int x, y, z, sum;

        printf (" Enter three integers (x, y, z): ");
        scanf ("%d %d %d", &x, &y, &z);
        sum += x;
        sum += y;
        sum += z;
        printf (" The sum of three integers is %d\n", sum);
        return 0;
}
```



What could be the problem ?

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

        sum = 0;
        printf (" Enter three integers (x, y, z): ");
        scanf ("%d %d %d", &x, &y, &z);
        sum += x;
        sum += y;
        sum += z;
        printf (" The sum of three integers is %d\n", sum);

        return 0;
}
```

Variables must be initialized before use !



Q & A