Ch.3 C Program Components

What you will learn in this chapter

* annotation
* Variables, constants
* Function
* sentence
* Output function printf()
* Input function scanf()
* Arithmetic operations
* Assignment operation

General program form

- receiving data (input stage),
- processing the data (processing stage)
- and then displaying the results on the screen (output stage) .

Addition Program #1

add1.c

```c
/* Program to calculate the sum of two numbers */
#include <stdio.h>

int main(void)
{
        int x;      // Variable to store the first integer
        int y;      // Variable to store the second integer
        int sum;    // Variable that stores the sum of two integers

        x = 100;
        y = 200;

        sum = x + y;
        printf("Sum of two numbers: %d", sum);

        return 0;
}
```

annotation

preprocessor

function

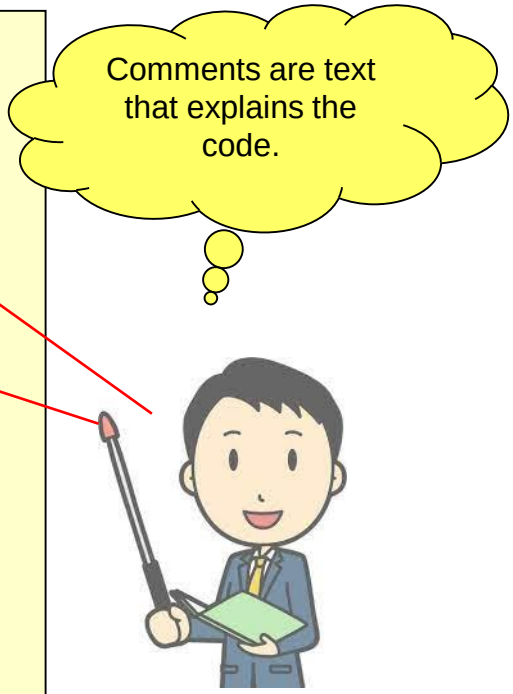
variable declaration

calculation

Sum of two numbers : 300

Comment

```
/* Program to calculate the sum of two numbers * /
#include <stdio.h>
int main( void )
{
        int x; // variable to store the first integer
        int y; // variable to store the second integer
        int sum; // Variable that stores the sum of two integers
        x = 100;
        y = 200;
        sum = x + y;
        printf("Sum of two numbers : %d" , sum);
        return 0;
}
```



2 ways to comment

```
/* single line comment * /

/* several
In a line
Comment done* /
```

```
// This entire line is a comment .
int x; // From here to the end of the line is a comment .
```

The importance of comments

- When someone else looks at the program, it is much easier to understand the contents of the program if there are comments. If a lot of time has passed, even the creator may not
  remember the contents well.

- A good comment does not repeat or explain the code.
  It should clearly state the intent of the code.

Annotation style

```
/*
File name : add.c
Description : Program to add two numbers
Author : Hong Gil-dong
* /
```

```
/************************************
* File name : add.c
* Description : Program to add two numbers
* Author : Hong Gil-dong
************************************ /
```

Indentation

- *Indentation :* Indenting sentences on the same level a few characters from the left end.

```
#include <stdio.h>

int main(void)
{
    int x;      // Variable to store the first integer
    int y;      // Variable to store the second integer
    int sum;    // Variable that stores the sum of two integers

    ...

    return 0;
}
```

Insert blank lines to differentiate between meanings.

Explain the intent of the program in comments.

If the content is the same, indentation is applied.

Without comments and indentation ..

```
#include < stdio.h >

int main( void ) {

int x; int y; int sum; x = 100; y = 200; sum = x + y;
printf ( " Sum of two numbers : %d" , sum); return
0;

}
```

Running is However,
it is difficult to know what kind of processing the program is doing, and it is also difficult to distinguish sentences on the same level because there is no indentation.

Preprocessor

- stdio.h contains definitions of library functions for standard input and output .

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

- Preprocessor
  # Starts with a symbol

Preprocessor

```
/* First program * /

#include < stdio.h >

int main( void )
{
    printf ( "Hello World!" );
    return 0;
}
```

hello.c

```
// stdio.h
...
int printf ( char *,...);
...
```

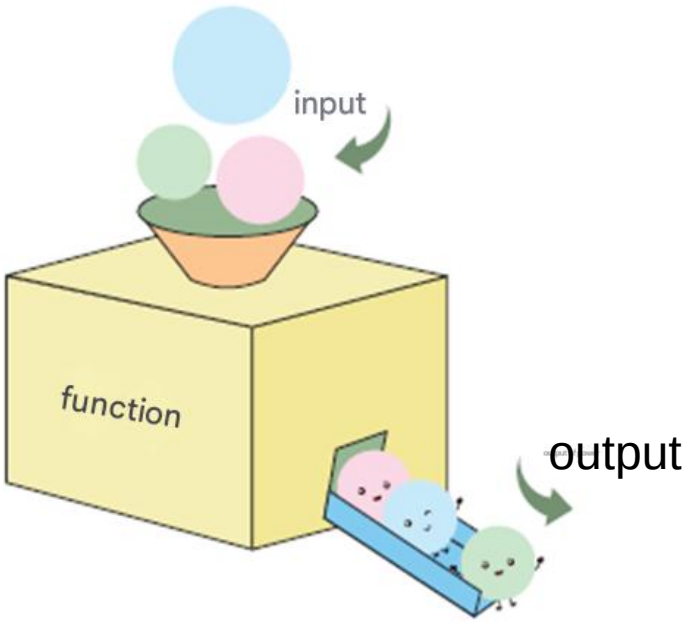
stdio.h

Function

- Function : A set of processing steps that perform a specific function, grouped in parentheses and named accordingly.

- Functions are the basic units that make up a program.

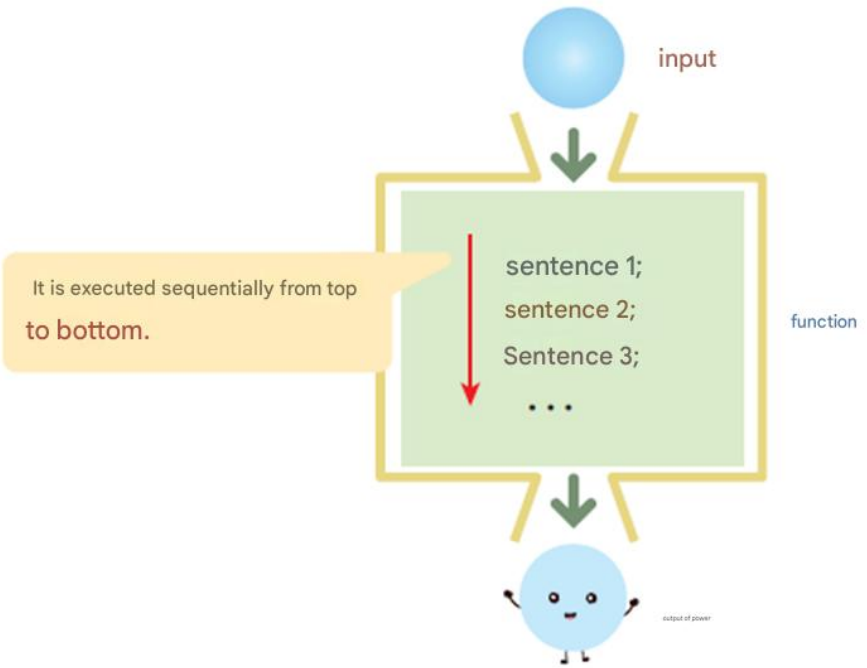
```
int main(void)
{
        ...
        ...
}
```



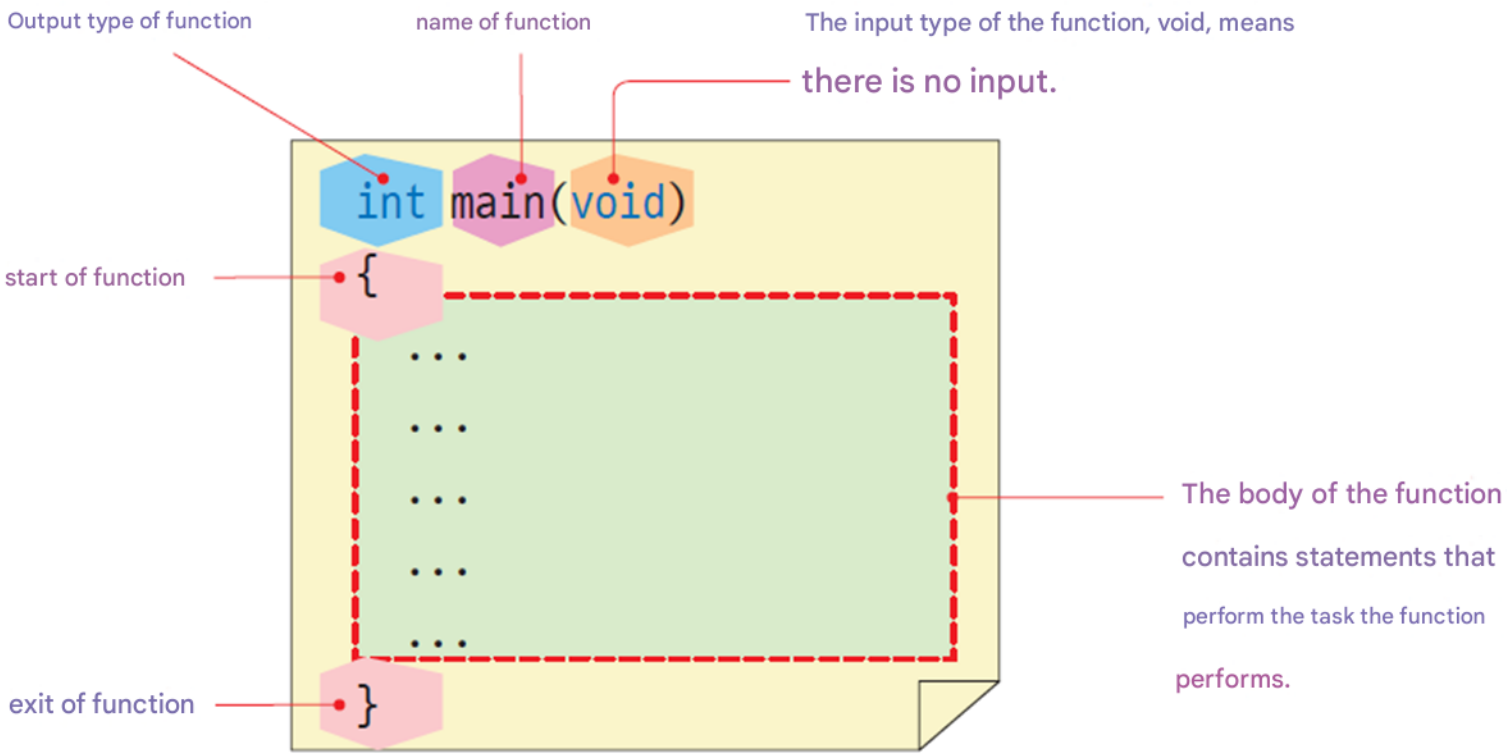
What's inside the function

Q) So what is inside the function ?

A) Inside the function, the processing steps (statements) that the function processes are listed in curly brackets.

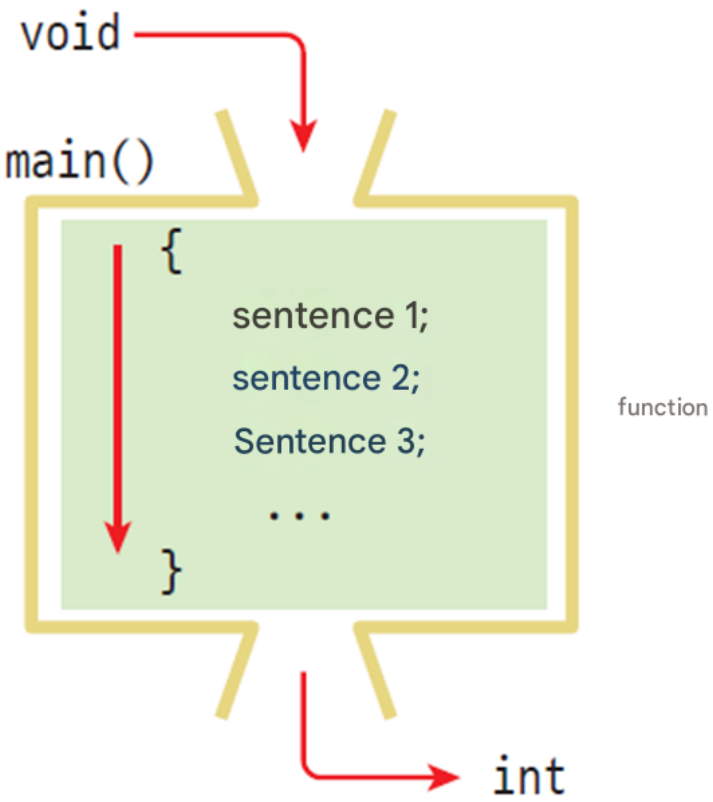


Structure of a function



Function

- The statement that performs the task must be placed inside a function.

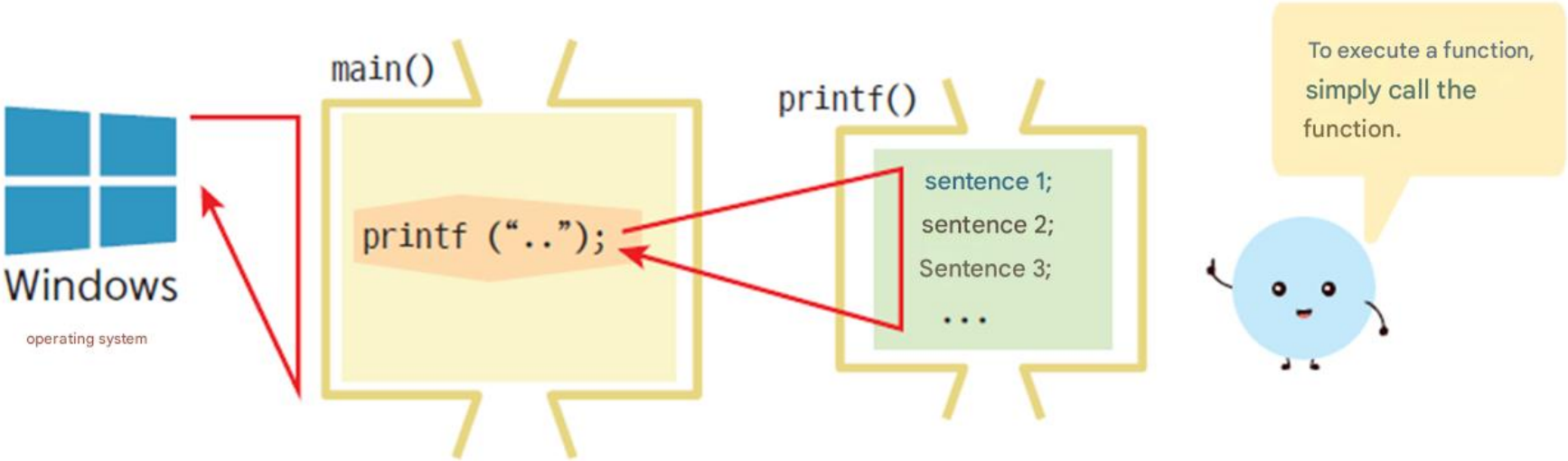


return statement

- return is a keyword that returns a value while terminating a function .
- To return a value, write the return value after return .

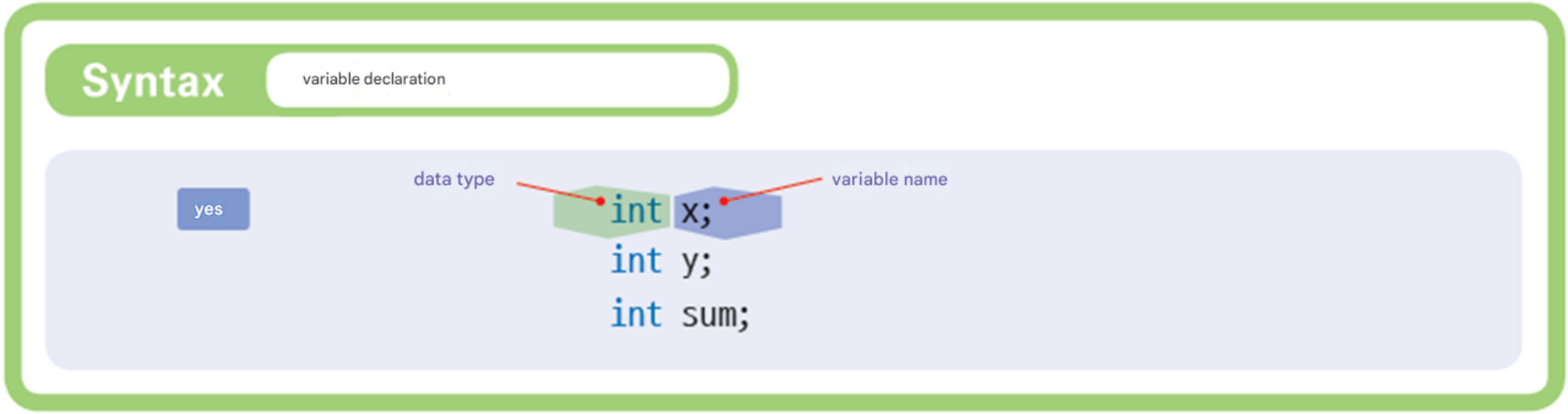
```
#include < stdio.h >

int main( void ) {
        ...
        ...
        return 0;
}
```

Who will call main() ?



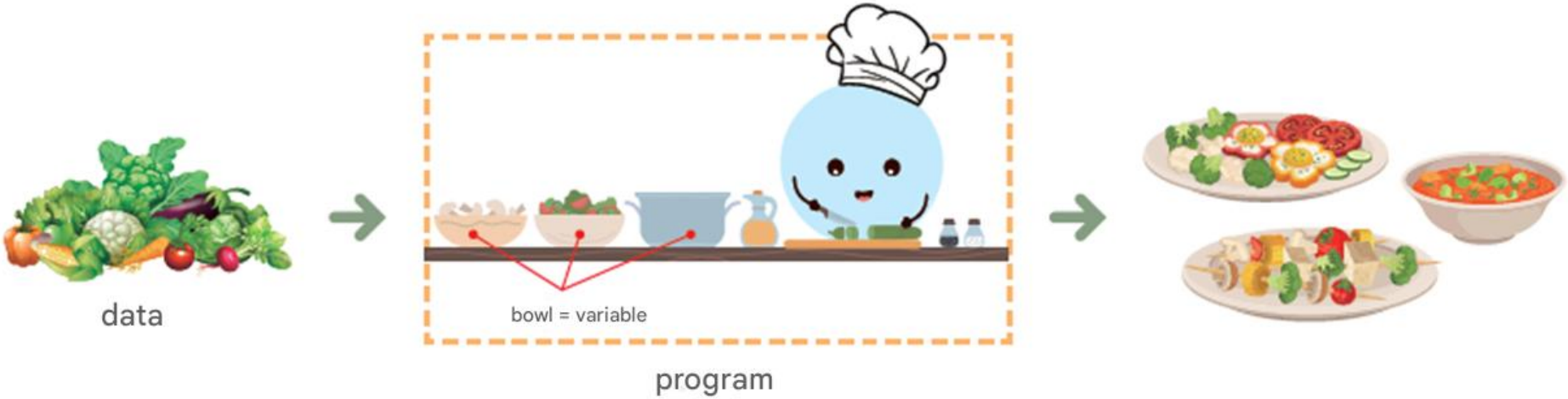
Variable

- Memory space used for temporarily storing data used by a program.



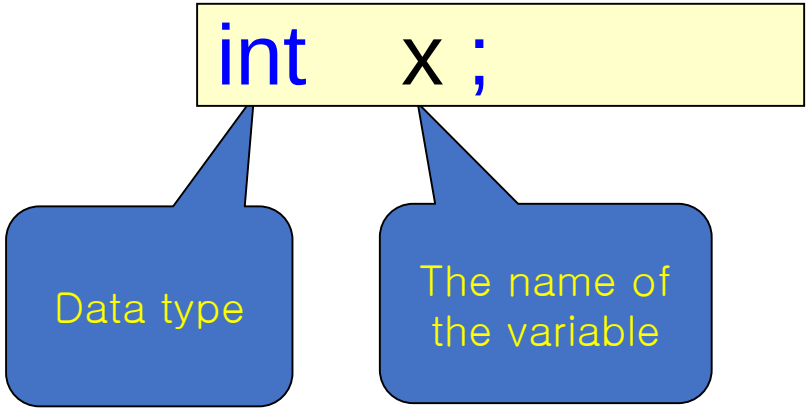
Why are variables needed?

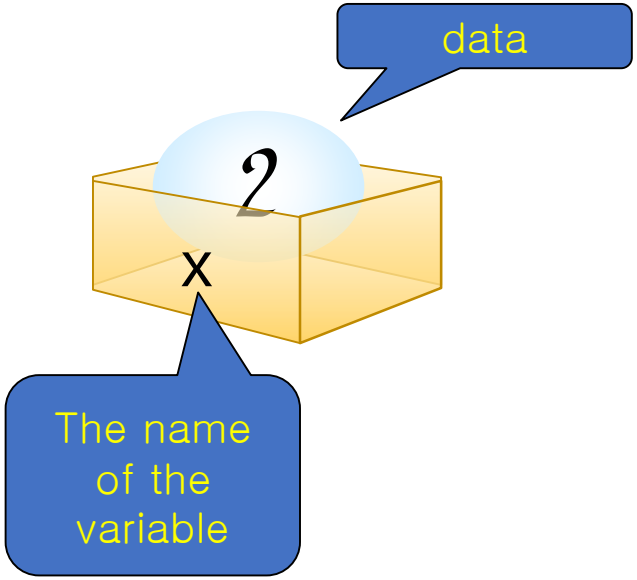
- Variables serve to temporarily store data values .



Types of variables

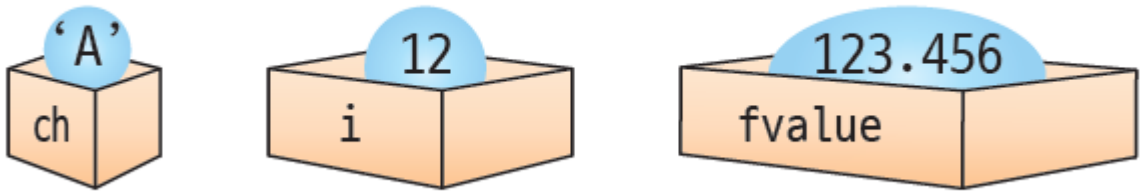
- A variable can be thought of as a box that holds data .





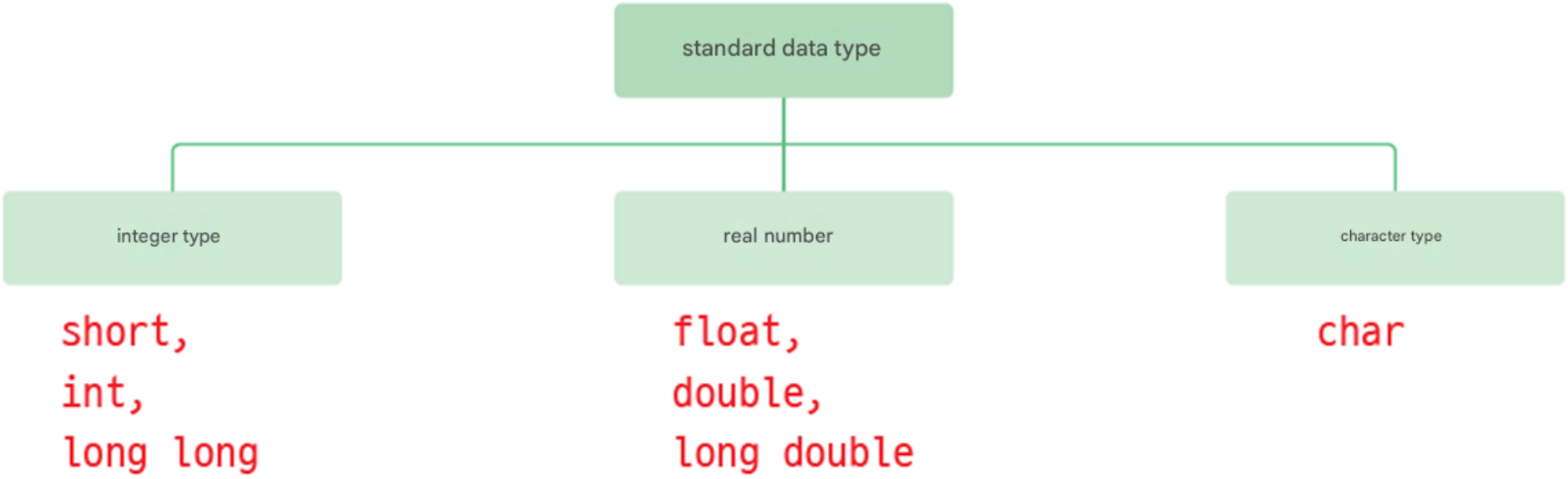
Types of variables

- Variables have several types depending on the type of data .



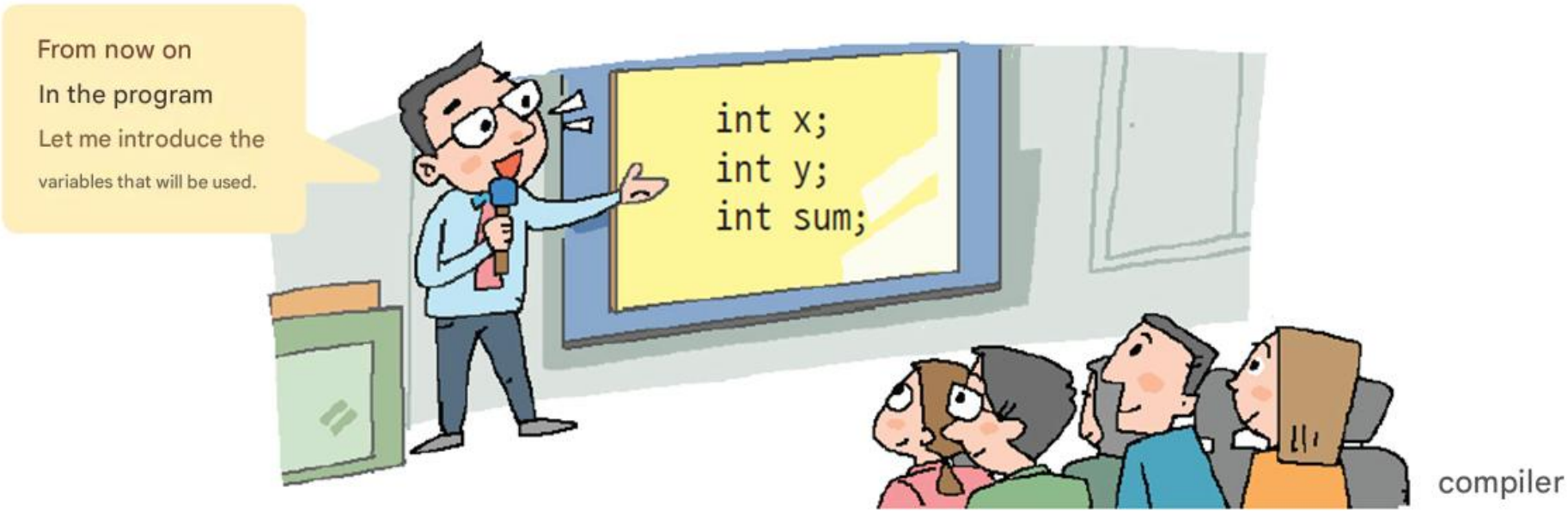
Data type

- It specifies whether the data to be stored in the variable is an integer, a real number, or some other data.
- Data types include integers, floating point numbers (real numbers), and character types.



Declaring variables

- Variable declaration : Telling the compiler in advance what type of variable will be used.



Declaring variables

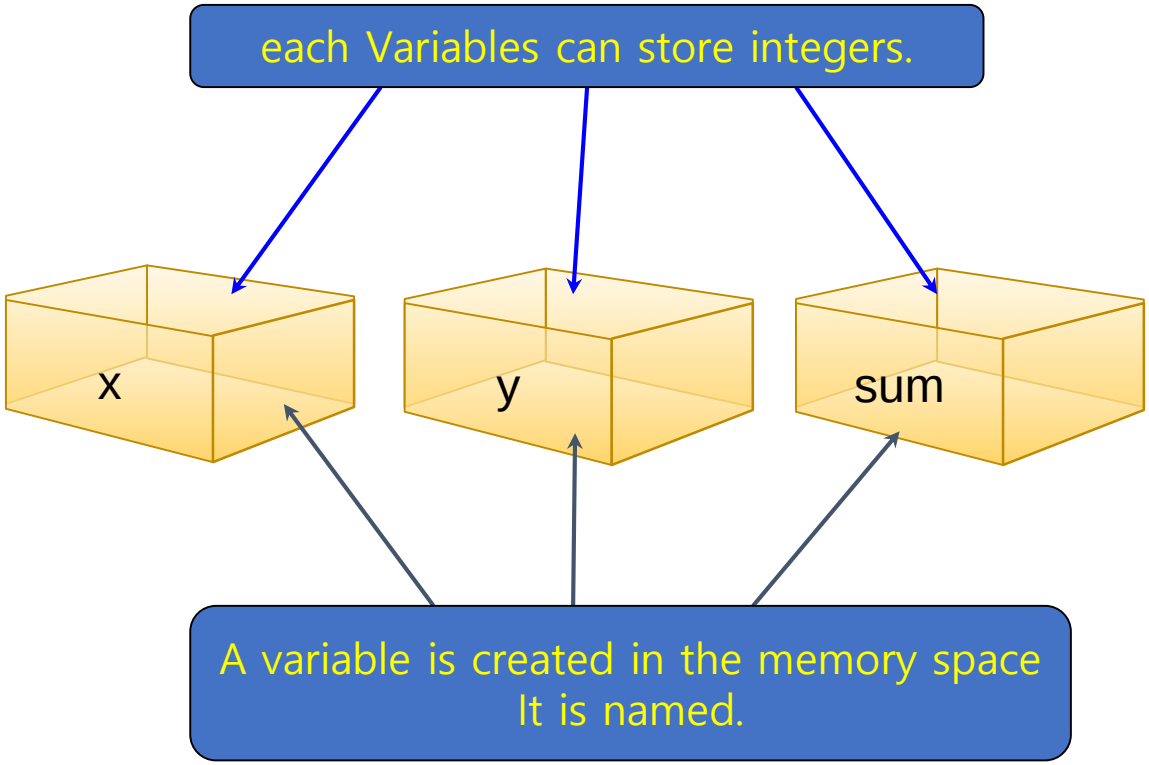
Data type is integer

variable name

```
int x;      // Variable to store the first integer
int y;      // Variable to store the second integer
int sum;    // A variable that stores the sum of two integers.
```

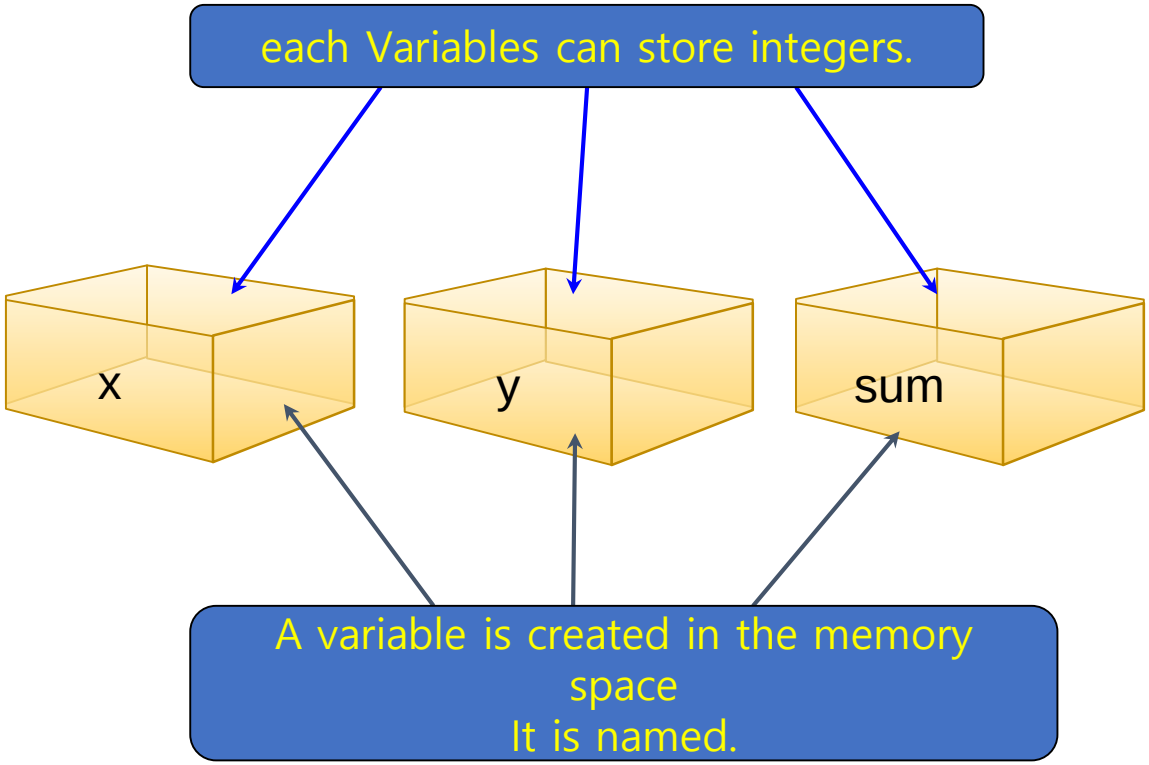
Declaring variables

```
int x ;  // first Integer Save Variable
int y ;  // second Integer Save Variable
int sum; // two Integrity Sum Save Variable
```



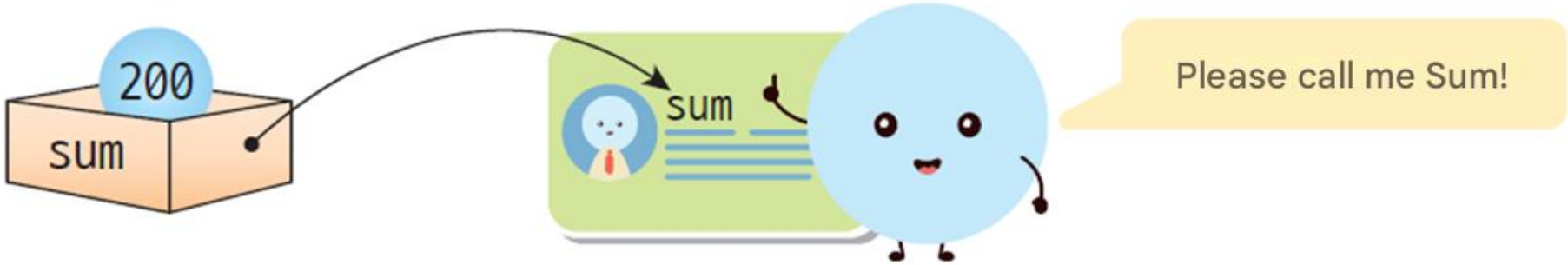
Declaring multiple variables on one line

```
int x, y, sum;  // possible !!
```



Name of variable

- Identifier : The programmer can name variables as he pleases , but he must follow some rules. Just as names such as " Hong Gil-dong" and " Kim Young-hee" identify people, variable names identify variables.



Name of variable

- Rules for creating
  - Identifiers consist of English letters, numbers , and the underscore character _ .
  - There must be no spaces in the middle of the identifier .
  - The first character of an identifier must be a letter or the underscore symbol _ . An identifier cannot start with a number .
  - Uppercase and lowercase letters are distinguished . Therefore, the variables index , Index, and INDEX are all different variables .
  - Identifiers that are identical to keywords in the C language are not allowed .

Function name

Variable name

```
int sub(void)
{
        int x;
}
```

Keyword

- Keyword : A special word that has its own meaning in the C language . Also called reserved words.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

The name of the variable

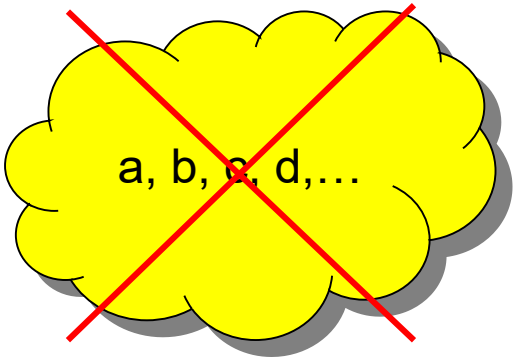
- *sum // Starts with an English alphabet letter*
- *_count // can start with an underscore character .*
- *number_of_pictures // You can put an underscore character in the middle .*
- *King3 // You can also put numbers in, as long as it's not the very first one .*

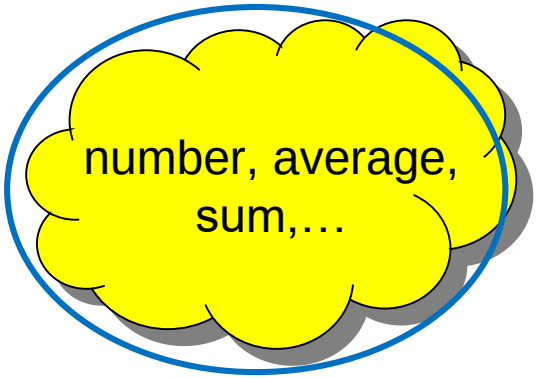
- *2nd_base(X) // Cannot start with a number .*
- *money# // Symbols such as # cannot be used .*
- *double // double is a keyword in the C language .*

Good variable names

- You should choose a name that best describes the role of the variable.
  - i , j, k (X)
  - year, month, date (O)

- How to create a multi-word name
  - underscore Method : bank_account
  - Capitalize the first letter of the word : BankAccount





Initialization of variables

- You can give initial values to variables .
  - int x = 10;
  - int y = 20;
  - int sum = 0;

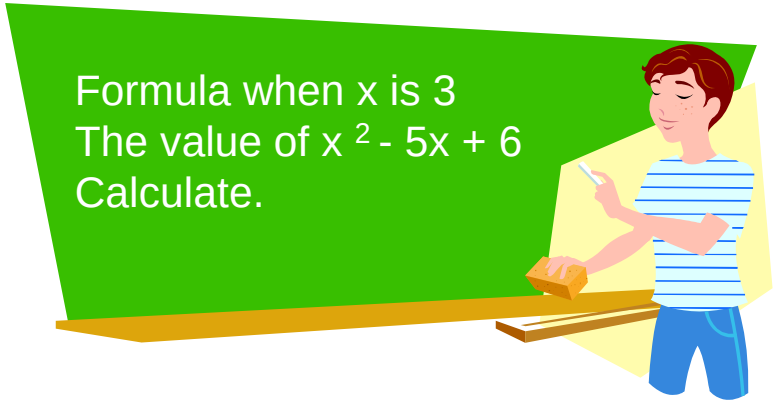
- If the variables are of the same type , you can declare and initialize them on the
  same line.
  - int width = 100, height = 200;

- Initializing as follows is not syntactically incorrect, but should be avoided :
  - int width, height = 200;

“width” is not initialized.

formula

- Expression : Operands and Operators consisting of expression
- A formula has **a result** .

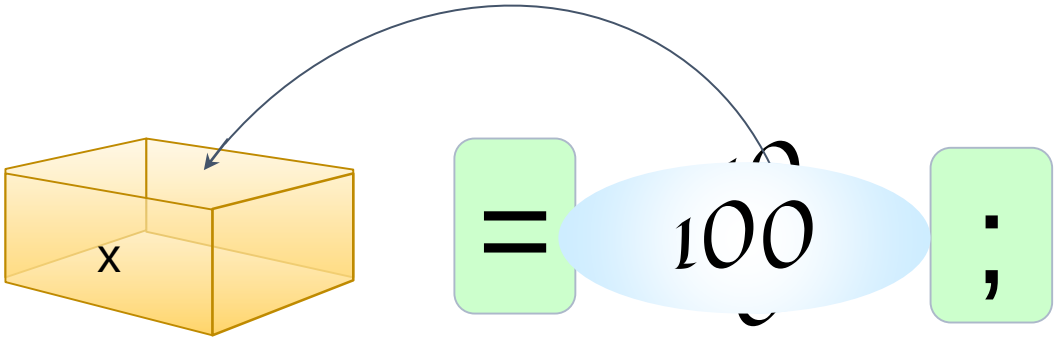



```
int x, y;

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

Storing values in variables

```
x = 100;
```

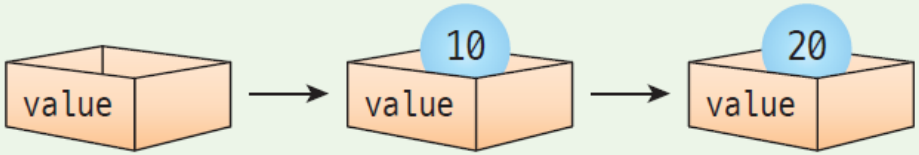
- Assignment operation : An operation that stores a value in a variable.
- Assignment operation : =



various Assignment operation

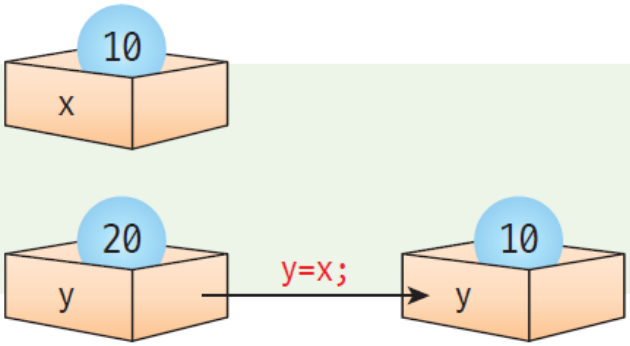
- A variable can store a value using the = symbol, and the value of the variable can be changed as many times as you like .

```
int value;
value = 10;
value = 20;
```



-

```
int x = 10;
int y = 20;
y = x;              // y는 10이 된다.
```



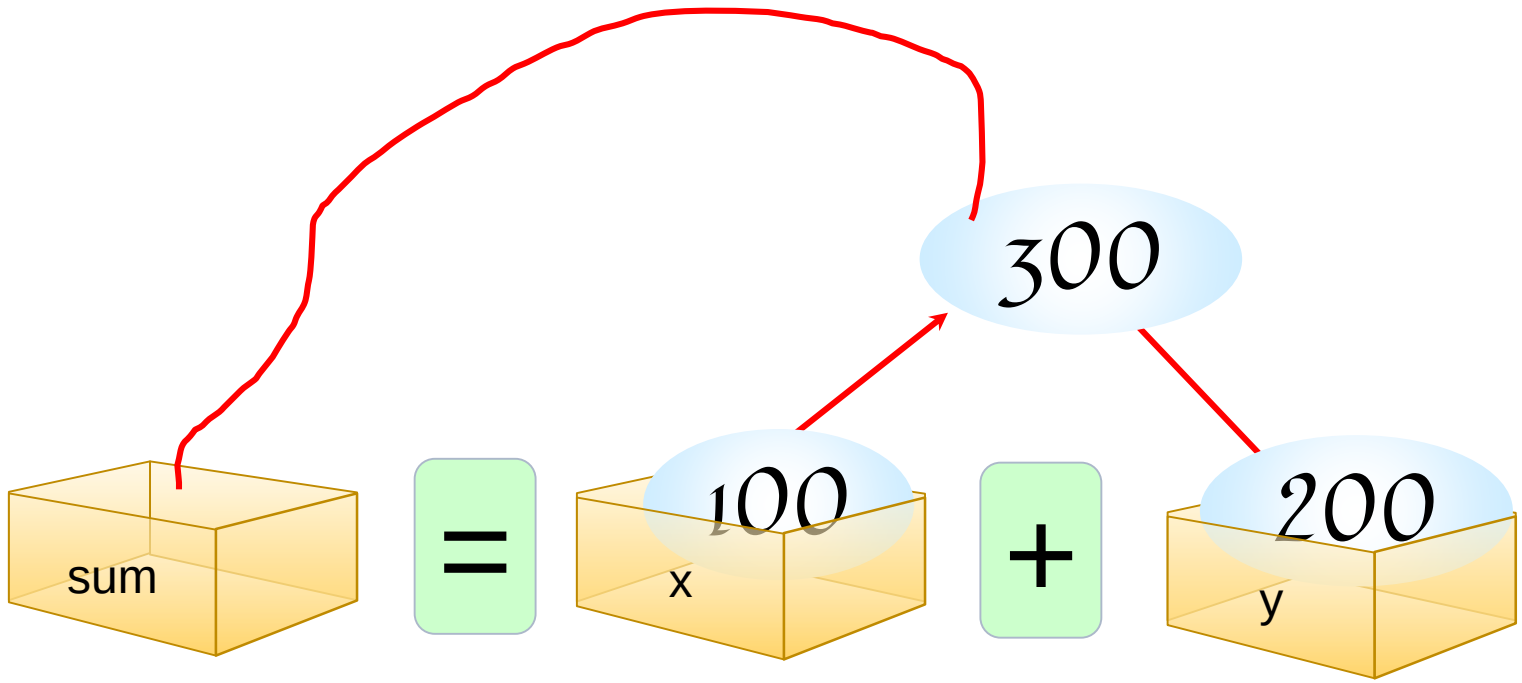
Arithmetic operations

- Arithmetic operators are similar to the operation symbols commonly used in mathematics .

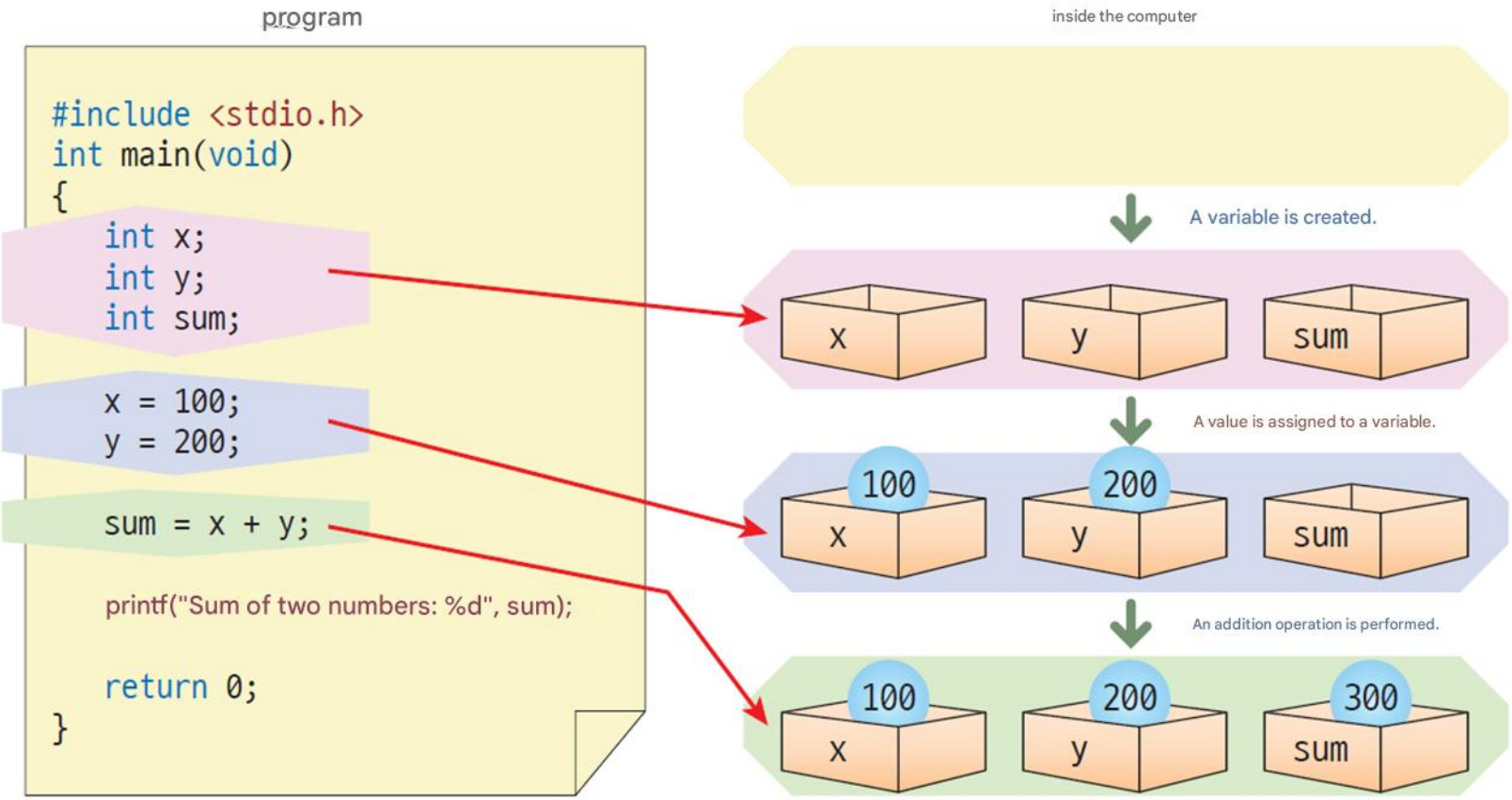
calculation	operator	C formula	symbols in mathematics
addition	+	x+y	x+y
subtraction	–	x – y	x − y
multiplication	*	x*y	xy
division	/	x / y	x/y or s or x÷y y
remain	%	x%y	x mod y

Arithmetic operations

```
sum = x + y;
```



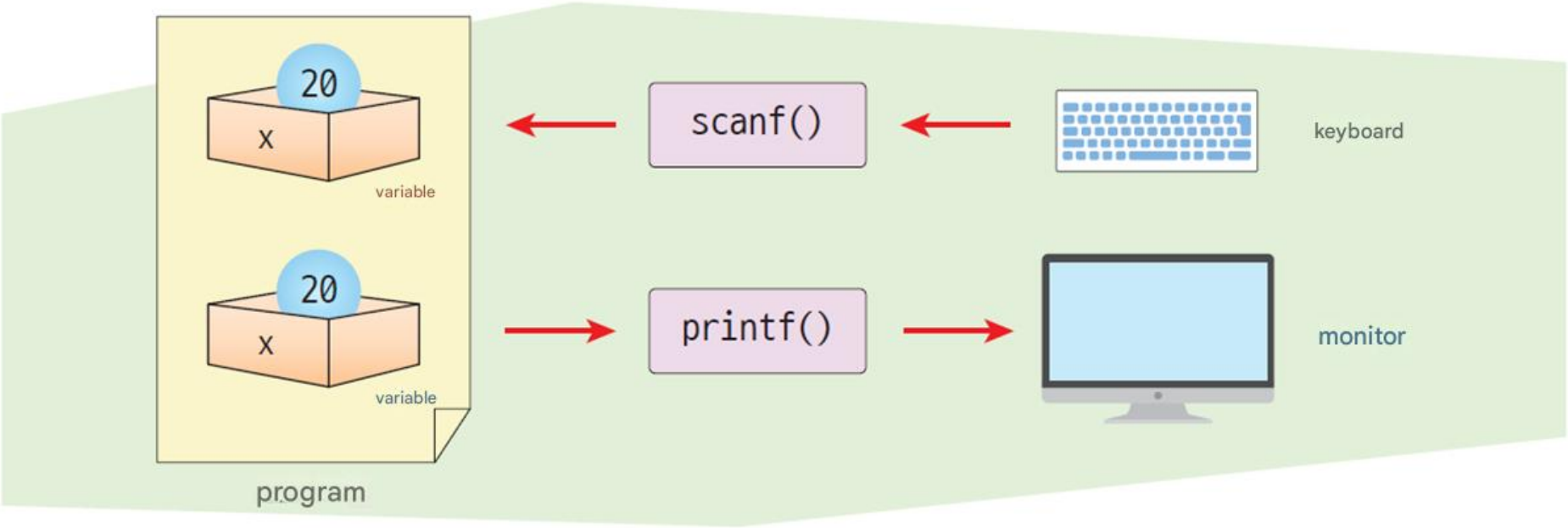
Organize


program
#include <stdio.h>
int main(void)
{
int x;
int y;
int sum;
x = 100;
y = 200;
sum = x + y;
printf("Sum of two numbers: %d", sum);
return 0;
}
inside the computer
A variable is created.
x
y
sum
A value is assigned to a variable.
100
200
x
y
sum
An addition operation is performed.
100
200
300
x
y
sum

Library functions

- Library functions : Library functions are functions that the compiler provides for programmers to use.

  - printf(): Standard output function for printing to the monitor.
  - scanf( ): Standard input function for input from the keyboard.



String output

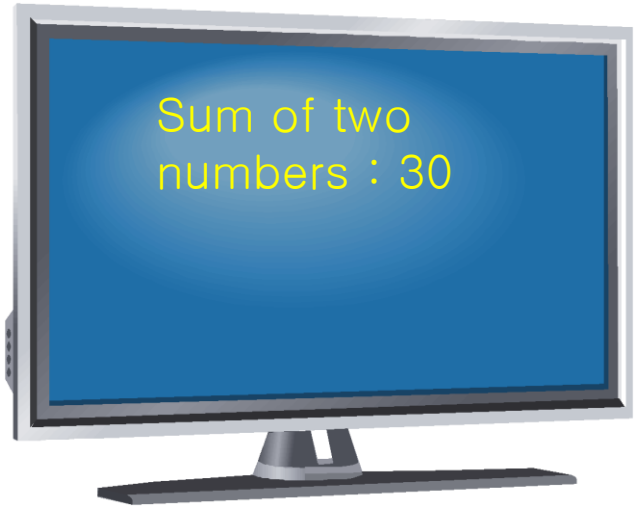
```
printf ("Hello World!\n ");
```

string : “Hello World!\n” and
A list of several characters
together

Output variable values

Output Format

printf (“Sum of two numbers : %d \n, sum");



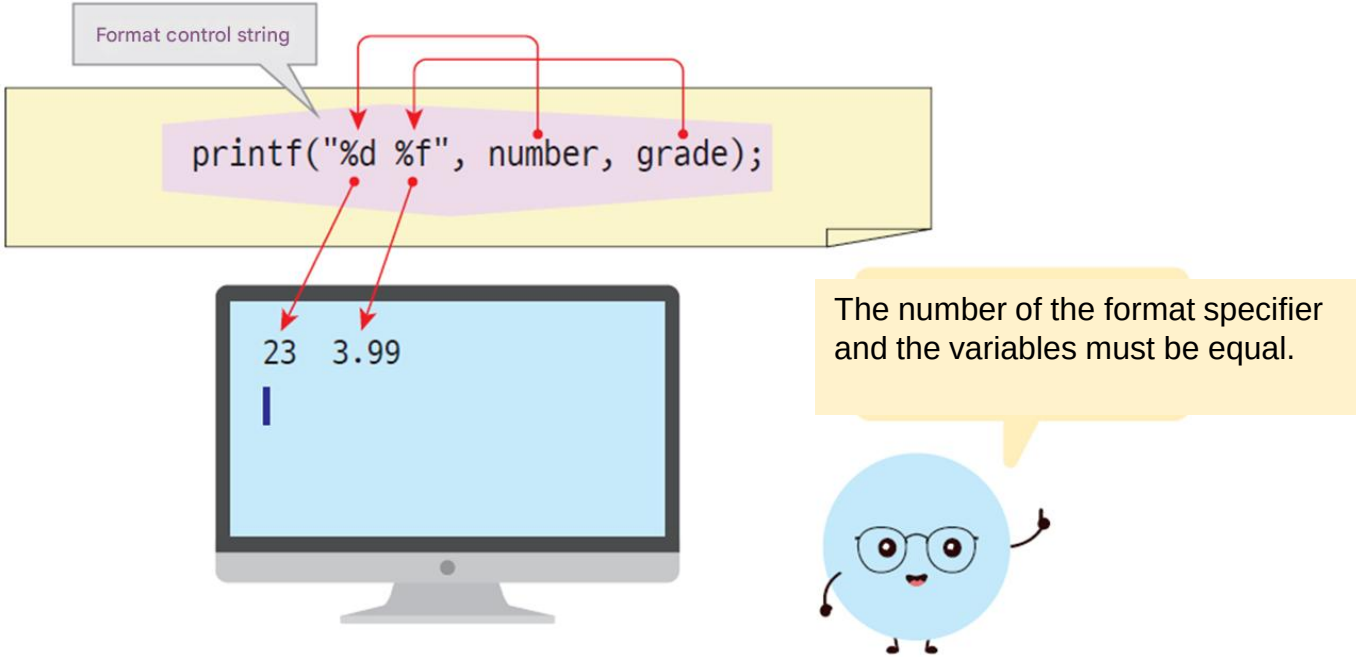
Format specifier

- Format specifier : Specifies the format in which values are printed in printf().

format specifier	meaning	yes	execution result
%d	Output as a decimal integer	printf("%d \n", 10);	10
%f	Output as a float	printf("%f \n", 3.14);	3.14
%c	Output as a character	printf("%c \n", 'a');	a
%s	Output as a string	printf("%s \n", "Hello");	Hello

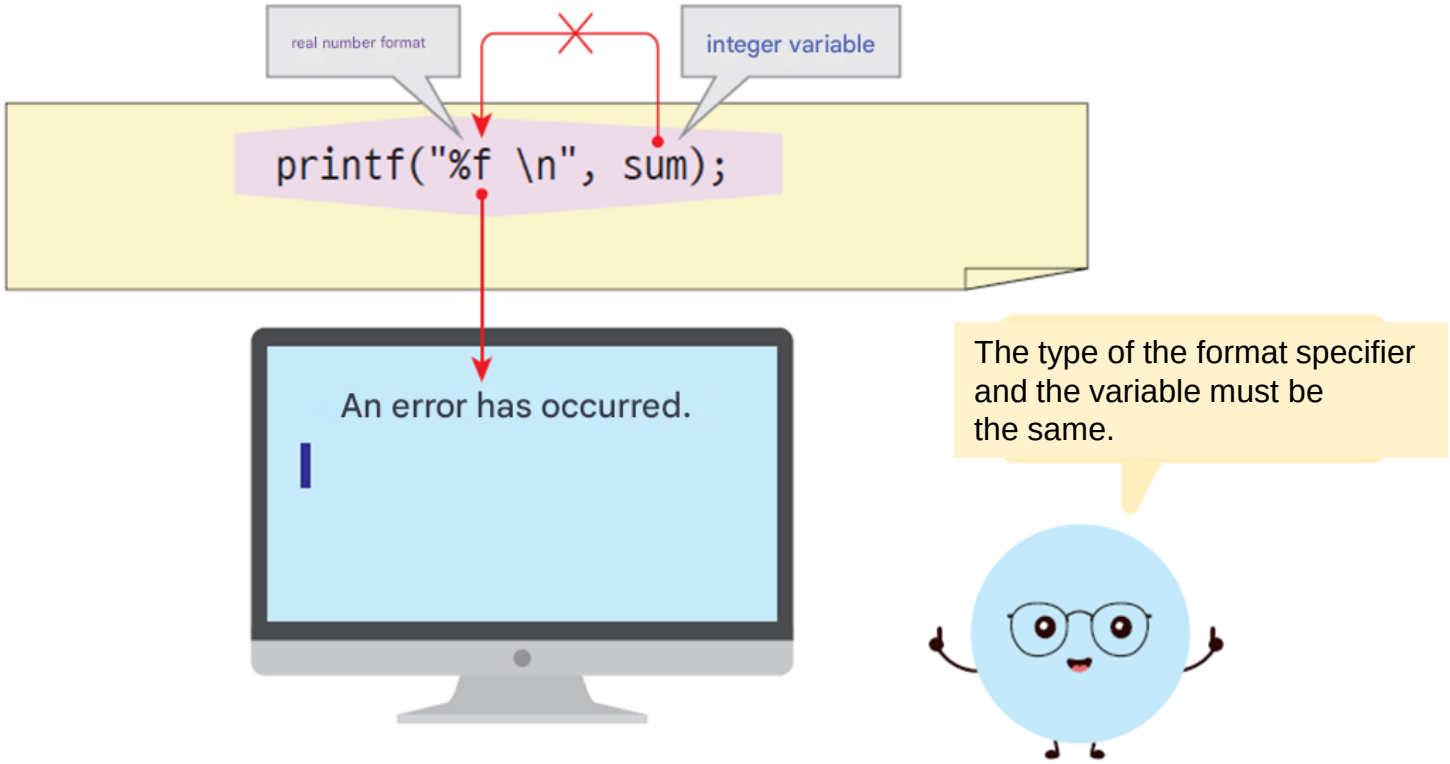
Output multiple variable values

- You can think of it as the value of the variable being substituted in the place of the format specifier and then printed.



Caution !

- The data types of the format and variables must match.



Field width and precision

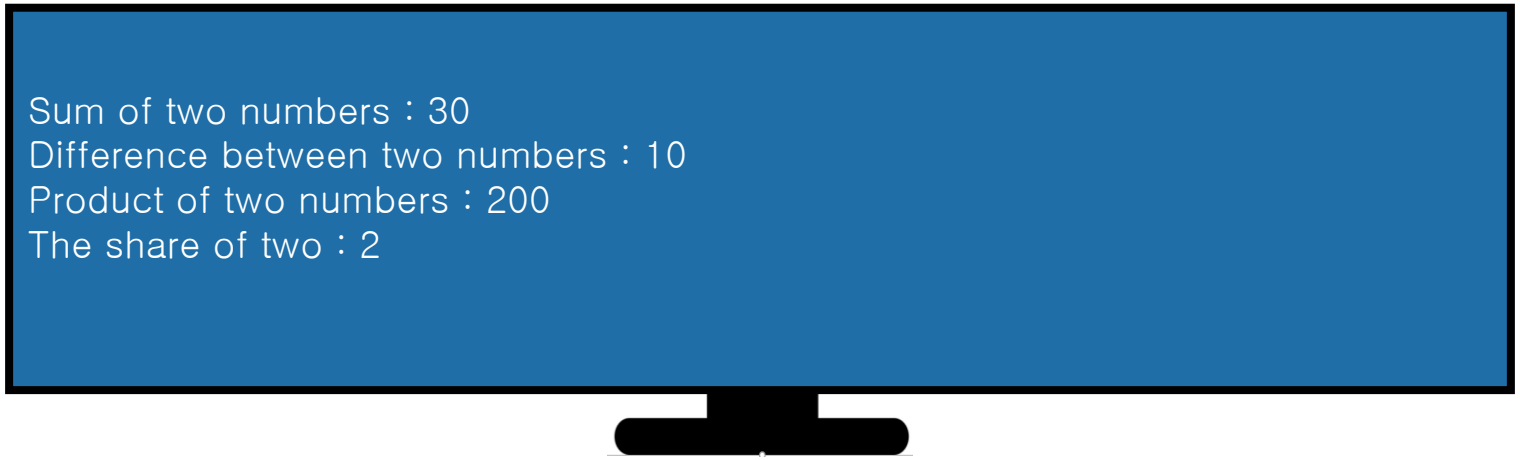
- When printing using printf(), you can specify the size of the field in which data is printed.

output statement	output result	explanation
`printf("%10d", 123);`	[ ][ ][ ][ ][ ][ ][ ][1][2][3]	Width 10, right aligned
`printf("%-10d", 123);`	[1][2][3][ ][ ][ ][ ][ ][ ][ ]	Width 10, left aligned

output statement	output result	explanation
`printf("%f", 1.23456789);`	[1][.][2][3][4][5][6][8][ ][ ]	6 decimal places
`printf("%10.3f", 1.23456789);`	[ ][ ][ ][ ][ ][1][.][2][3][5]	3 decimal places
`printf("%-10.3f", 1.23456789);`	[1][.][2][3][5][ ][ ][ ][ ][ ]	Left Align
`printf("%.3f", 1.23456789);`	[1][.][2][3][5][ ][ ][ ][ ][ ]	Display only decimal places

Lab: Four basic arithmetic calculation

- Stores 20 and 10 in variables x and y, calculates x+y, x-y, x*y, x/y, stores them in variables, and prints these variables on the screen.



Solution

```
// Program to calculate addition, subtraction, multiplication and division between integers
#include < stdio.h >

int main( void )
{
    int x; // variable to store the first integer
    int y; // variable to store the second integer
    int sum, diff, mul , div; // Variables that store the results of operations between two
integers

   x = 20; // store 2 in variable x
   y = 10; // store

   sum = x + y; // Store the result of (x + y)
   diff = x - y; // Store the result of (x – y)
   mul = x * y; // variable Store the result of (x * y)
   div = x / y; // Store the result of  (x / y)
```

Solution

```
  printf ( " Sum of two numbers : %d\n" , sum); // Print the value of sum
  printf ( " Difference between two numbers : %d\n" , diff); // Print the value of diff
  printf ( " Product of two numbers : %d\n" , mul ); // variable Print the value of mul
  printf ( " Quotient of two numbers : %d\n" , div); // Print the value of div

  return 0;
}
```

scanf()

- Gets a value from the keyboard and stores it in a variable.
- Requires the address of the variable .

Format specifier

The value will be saved at address of the variable.

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

Why do I need an address?

- When we buy a product on the Internet and have it delivered to our home, we have to tell the shopping mall our address .


```
& x
```

The & operator calculates the address of a variable .

Format specifiers for scanf ()

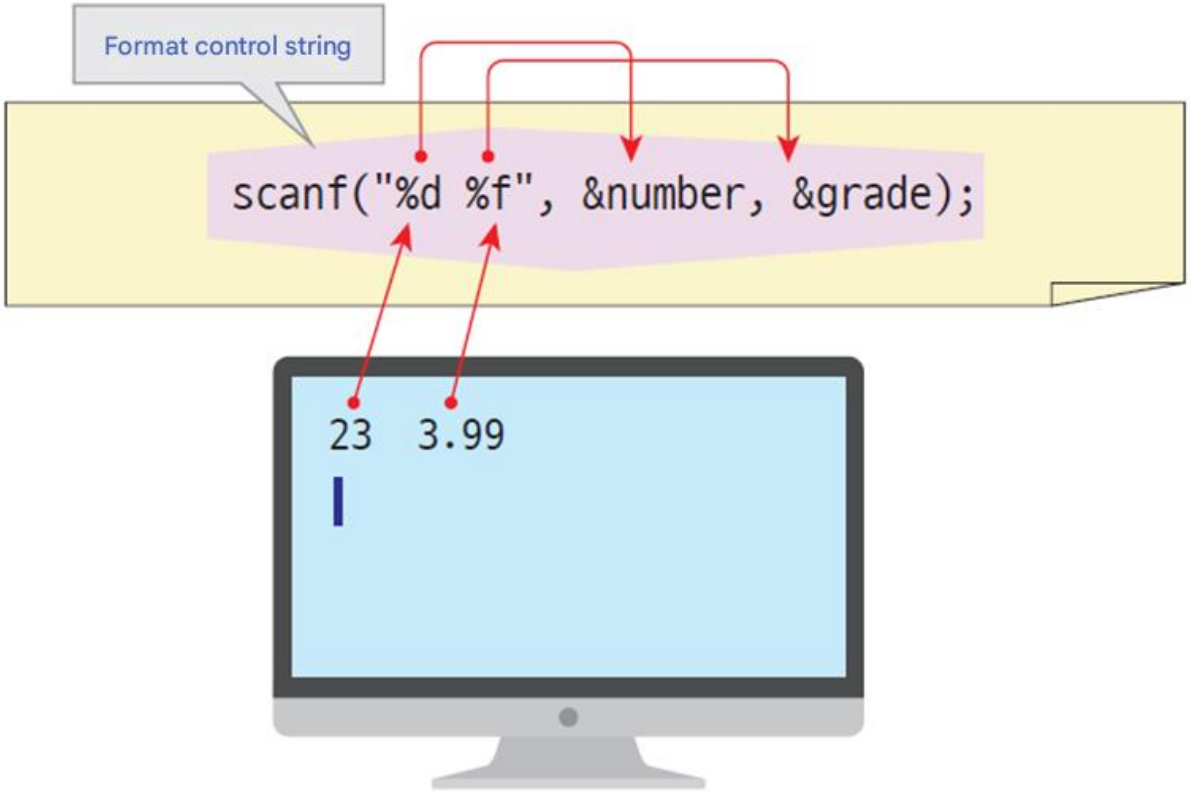
- Mostly it is similar to printf().

format specifier	meaning	yes
%d	Enter a decimal integer	`scanf("%d", &i);`
%f	Enter a real number of float type.	`scanf("%f", &f);`
%lf	Enter a real number of type double.	`scanf("%lf", &d);`
%c	Enter one character.	`scanf("%c", &ch);`
%s	Enter a string.	`char s[10]; scanf("%s", s);`

Not studied yet!

Don't worry too much!

scanf()



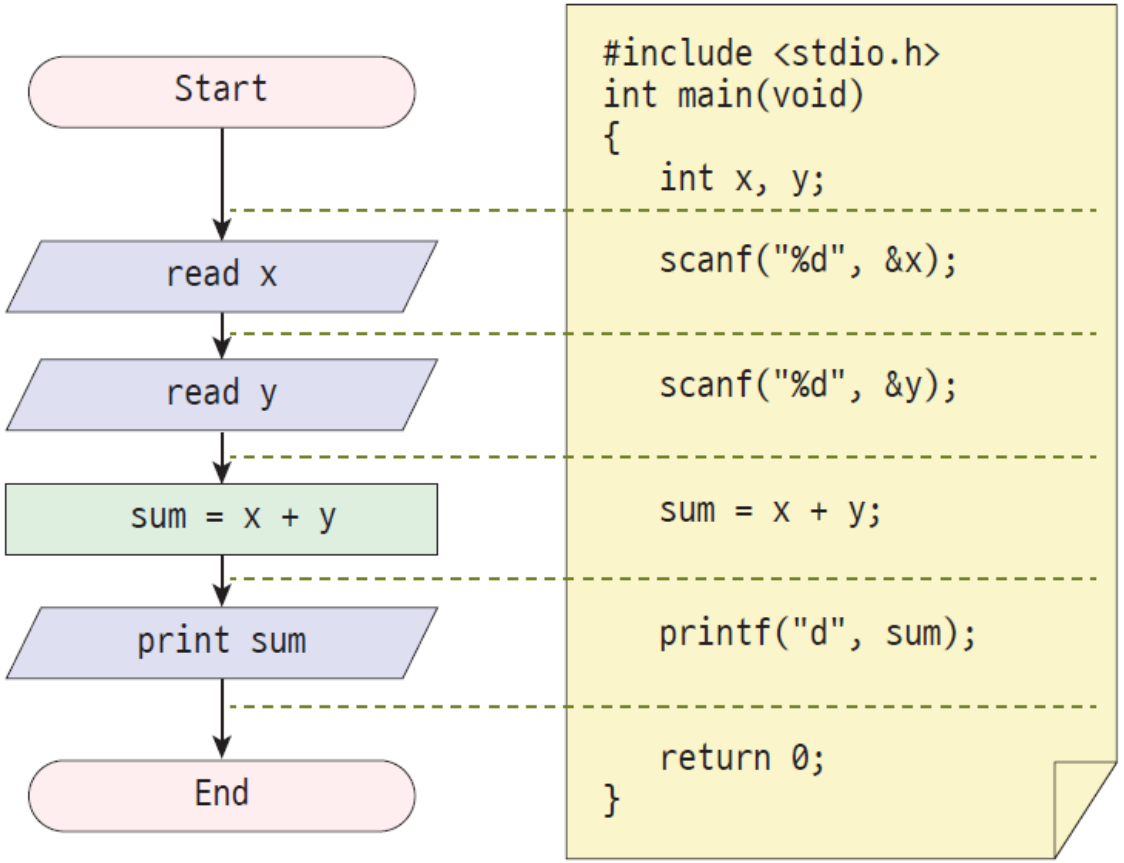
The number of the format specifier and the variables must be equal.

Addition Program #2

- Let's get input from the user .

```
Enter the first number : 10
Enter the second number : 20
Sum of two numbers : 30
```

Algorithm

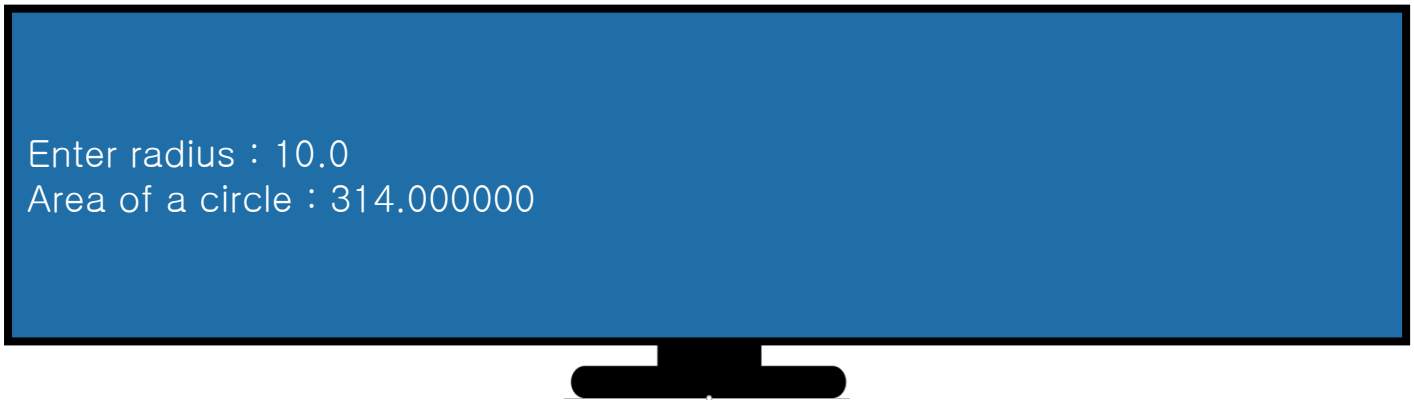

Start
read x
read y
sum = x + y
print sum
End
#include <stdio.h>
int main(void)
{
int x, y;
scanf("%d", &x);
scanf("%d", &y);
sum = x + y;
printf("d", sum);
return 0;
}

Second addition program

```
// Calculate and output the sum of

#include < stdio.h >

int main( void )
{
        int x; // variable to store the first integer
        int y; // variable to store the second integer
        int sum; // Variable to store the sum of two integers

        printf ( " Enter the first number : " ); // Output input guidance message
        scanf ( "%d" , &x); // Receive an integer and store it in

        printf ( " Enter the second number : " ); // Output input guidance message
        scanf ( "%d" , &y); // Receive an integer and store it in

        sum = x + y; // Add two variables .
        printf ( " Sum of two numbers : %d" , sum); // Print the value of sum in decimal

        return 0; // return 0 to the outside
}
```

Circle Area Calculation Program

- It receives the radius of a circle from the user, calculates the area of the circle, and then displays it on the screen .



Circle Area Calculation Program

```
#include < stdio.h >

int main( void )
{
        float radius; // radius of the circle
        float area; // area

        printf ( " Enter the radius : " );
        scanf ( "%f" , &radius);
        area = 3.14 * radius * radius;
        printf ( " Area of the circle : %f\n" , area);

        return 0;
}
```

Currency conversion program

- Let's write a program that converts the won input by the user into dollars and outputs it .

```
Enter the exchange rate : 1400
Enter the amount in Won : 1000000
1,000,000 Won is equal to 714.285,714 US Dollars .
```

```c
/* Program to calculate exchange rates * /
#include < stdio.h >

int main( void )
{
	double rate; // won / dollar exchange rate
	double usd ; // dollar
	int krw ; // Won is declared as an integer variable

	printf ( " Enter the exchange rate : " ); // Input guidance message
	scanf ( "%lf" , &rate); // Input exchange rate

	printf ( " Enter the amount in Korean Won : " ); // Input guidance message
	scanf ( "%d" , &krw ); // Enter the amount in Korean Won

	usd = krw / rate; // Convert to dollar

	printf ( " %d won is %lf dollar .\n" , krw , usd ); // Print calculation result

	return 0; // Return the function result value
}
```

Program to calculate average

- Write a program that receives three double type real numbers from the user, calculates the sum and average, and displays them on the screen.

```
Enter 3 numbers : 10.2 21.5 32.9
Total =64.60
Average =21.53
```

Program to calculate average

```
#include < stdio.h >

int main( void )
{
        double num1, num2, num3;
        double sum, avg;

        printf ( " Enter 3 real numbers : " );
        scanf ( "% lf % lf % lf " , &num1, &num2, &num3); // Input 3 real numbers

        sum = num1 + num2 + num3;
        avg = sum / 3.0;

        printf ( " total = %.2lf\n" , sum); // Display decimal point with 2 digits
        printf ( " average =%.2lf\n" , avg);

        return 0;
}
```

Q & A