Programming in C

Lecture 2

Data types

- Why should we classify data types？
- **In daily learning and life, we will encounter various types of information, which can be stored in various forms such as text, graphics, sound, video, etc. These data themselves are very rich. At present, computers are also capable of processing these diverse types of data**
- **how computers store these data?**

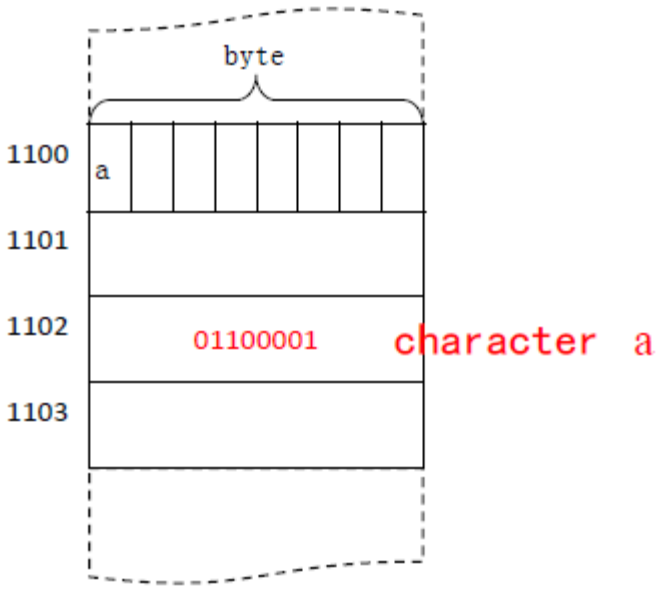
  data and instructions are stored in binary form in computers

  Bit is the most basic storage unit in a computer，eight binary bits make up one byte

  Each byte of storage unit has a unique serial number，we call this serial number address. By using this address, data can be saved and read from each byte in memory
-

Data types

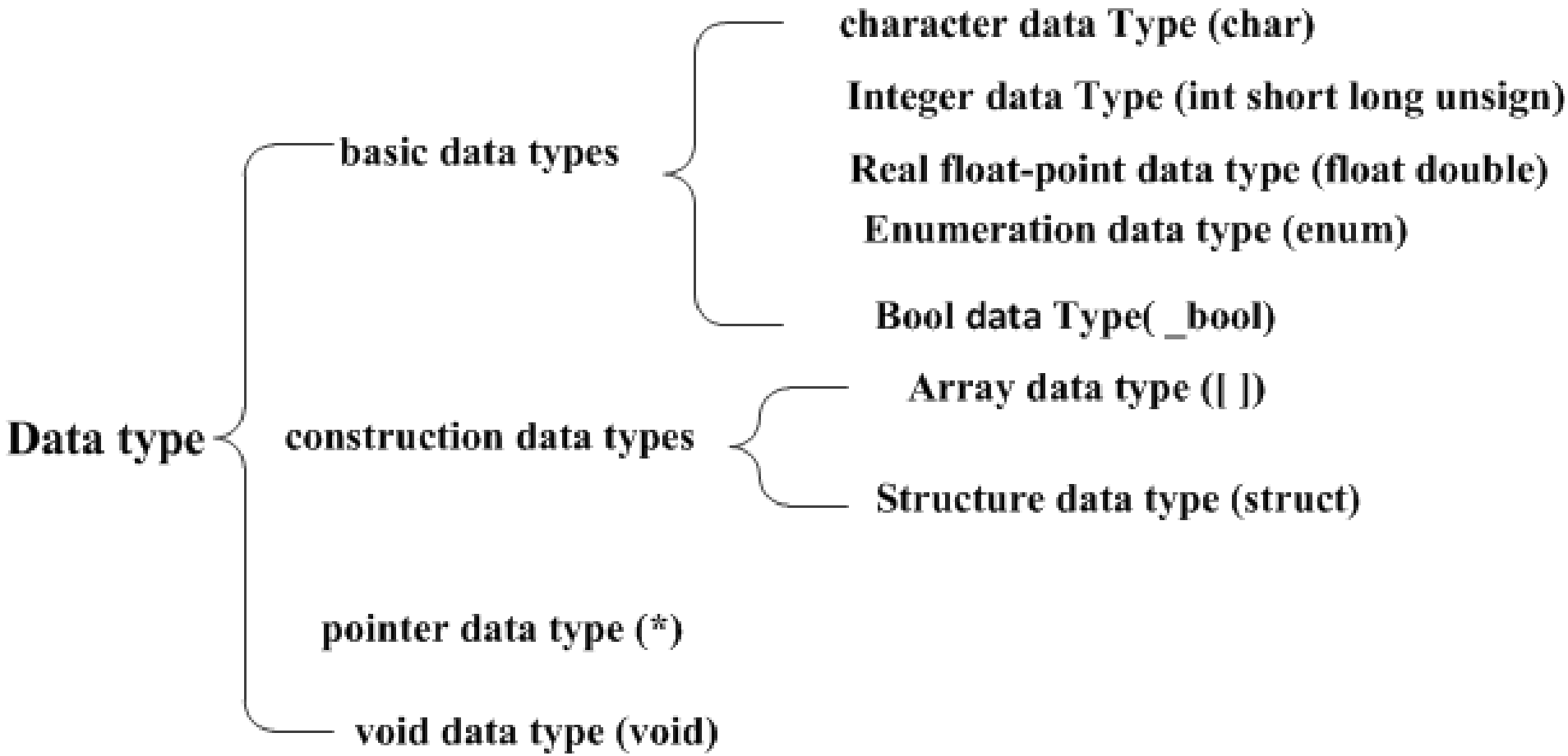
- **Now let‘s assume we want to store a character， the lowercase letter a, with an address of 1102.**
- **The value corresponding to the lowercase letter “a” is decimal number 97, whose binary representation is 01100001.**



Data types

- **In C language, data types are divided into different types based on their defined properties, representation, and storage space usage, mainly including basic types, construction types, pointer types, and void types.**
- **If we are in a certain shopping mall, the name of the product is usually text, which can be represented by character data in C language. The quantity of the product is an integer, which can be represented by integer data in C language. However, the price of the product may have a decimal part, which needs to be represented by float-piont date in C language. If you want to calculate the sales revenue of a certain product, you need to simultaneously present the product name, price, and sales quantity**

Data types


Data type
basic data types
character data Type (char)
Integer data Type (int short long unsign)
Real float-point data type (float double)
Enumeration data type (enum)
Bool data Type(_bool)
construction data types
Array data type ([ ])
Structure data type (struct)
pointer data type (*)
void data type (void)

Data types

- **Basic data types in C: `int`, `float`, `double`, `char`, and `_Bool`.**
- Data type `int:` can be used to store integer numbers (values with no decimal places)
- Data type type `float:` can be used for storing floating-point numbers (values containing decimal places).
- Data type `double:` the same as type `float`, only with roughly twice the precision.
- Data type `char:` can be used to store a single character, such as the letter *a*, the digit character *6*, or a semicolon.
- Data type `_Bool:` can be used to store just the values 0 or 1 (used for indicating a true/false situation). This type has been added by the C99 standard (was not in ANSI C)

Example: Using data types

```
#include <stdio.h>
int main (void)
{
    int integerVar = 100;
    float floatingVar = 331.79;
    double doubleVar = 8.44e+11;
    char charVar = 'W';
    _Bool boolVar = 0;
    printf ("integerVar = %i\n", integerVar);
    printf ("floatingVar = %f\n", floatingVar);
    printf ("doubleVar = %e\n", doubleVar);
    printf ("doubleVar = %g\n", doubleVar);
    printf ("charVar = %c\n", charVar);
    printf ("boolVar = %i\n", boolVar);
    return 0;
}
```

The basic data type `int`

- **the number of students in a class is 45**
- **the number of majors in a certain school is 83**
- **the sales volume of a certain product is 400**
- **the page count of a certain textbook is 256**

- **Integers can also be expressed in a base other than decimal (base 10): octal (base 8) or hexa (base 16).**

Octal notation for integers

- **Octal notation** (base 8): If the first digit of the integer value is 0, the integer is taken as expressed in *octal* notation. In that case, the remaining digits of the value must be valid base-8 digits and, therefore, must be 0–7.
- Example: Octal value `0111` represents the decimal value 73 ($1 \times 8^2 + 1 \times 8 + 1$).
- 0777 is also a legal octal, 01270 is also a legal octal, and 08 010 is illegal

Hexadecimal notation for integers

- **Hexadecimal notation** (base 16): If an integer constant is preceded by a zero and the letter *x* (either lowercase or uppercase), the value is taken as being expressed in hexadecimal. Immediately following the letter *x* are the digits of the hexadecimal value, which can be composed of the digits 0–9 and the letters a–f (or A–F). The letters represent the values 10–15, respectively.
- Example: hexadecimal value **0x12b** represents the decimal value 299 ($1 \times 16^2 + 2 \times 16 + 11$).
- The format characters %x, %X , %#x, or %#X display a value in hexadecimal format

Integer data type

- According to different storage spaces of integer data, C language divides integer data into
- short integer data(short int): 2 bytes
- basic integer data(int): 4 bytes
- long integer data(long int): 4 bytes
- Integer data sometimes needs to represent negative numbers. According to the presence or absence of signed bits, integer data can be divided into
- Signed integer data
- unsigned integers data

The floating number type

- a man's height of 1.83 meters
- a score of 92.5 in a subject
- a price of 35.8 yuan
- a stock increase of 2.77%
- a protein content of 60% in a certain food
- Float point data can be represented in two forms:
- decimal form : 3.1415926 0.123 .8 8. " ." ×
- exponential form: 3.14e2 (=314) -5.12e-6 12e-1.2 × e-7 ×

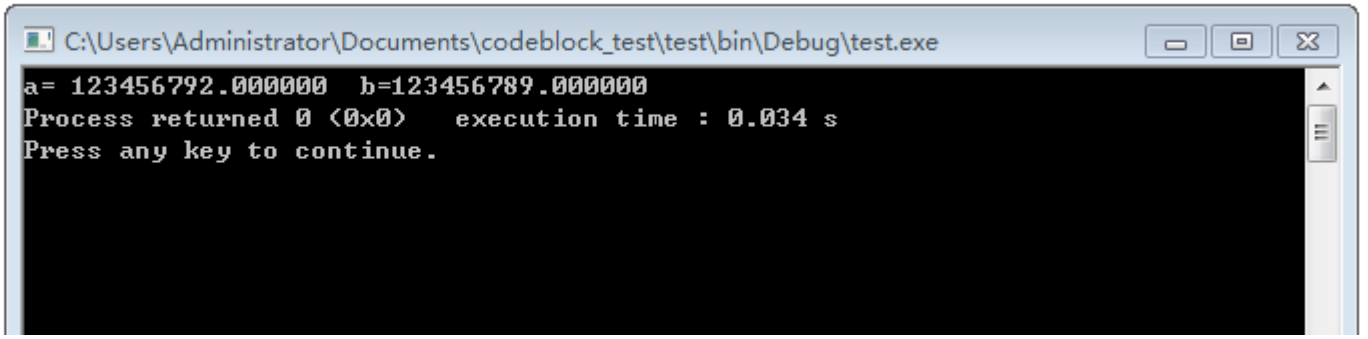
- According to the number of decimal digit that float-point data can accurately represent, float-point data can be divided into three types based on accuracy:
- float: 7 valid digits
- double: sixteen valid digits
- long double: At least 16 significant digits

The floating number type

```
#include <stdio.h>
int main()
{
    float a;
    double b;
    a= 0.0123456789e10;
    b= 0.0123456789e10;
    printf("a= %f  b=%f", a, b);
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    float a;
    double b;
    a= 0.0123456789e10;
    b= 0.0123456789e10;
    printf("a= %f  b=%f", a, b);
    return 0;
}
```

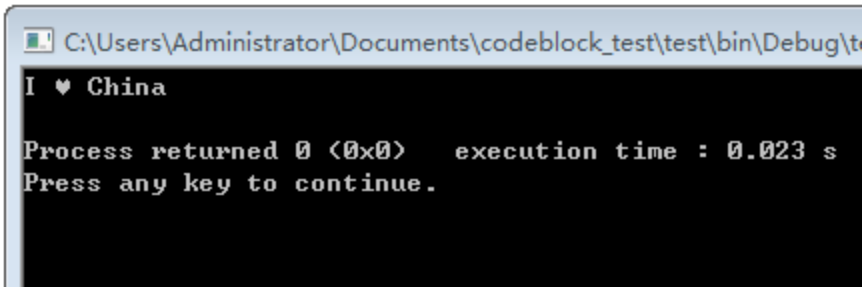


The character type char

- #include <stdio.h>

- int main()
- {
- char a=3;
- printf("I ");
- printf("%c ",a);
- printf("China\n");
- return 0;
- }

```
1   #include <stdio.h>
2
3   int main()
4   {
5     char a=3;
6     printf("I ");
7     printf("%c ",a);
8     printf("China\n");
9     return 0;
10  }
11
```



The character type char

- Character type data
- In C language, character type data includes character form and string form.

  A char variable can be used to store a single character.
- A character constant is formed by enclosing the character within a pair of single quotation marks.

  ‘a’（√）、 ‘#’（√）、 ‘\101’（√）、 ‘7’（√）

  ‘ab’（×）、 ‘\’（×）、 ‘’ ’ （×）、 ‘” ’ （×）、 “a”（×）

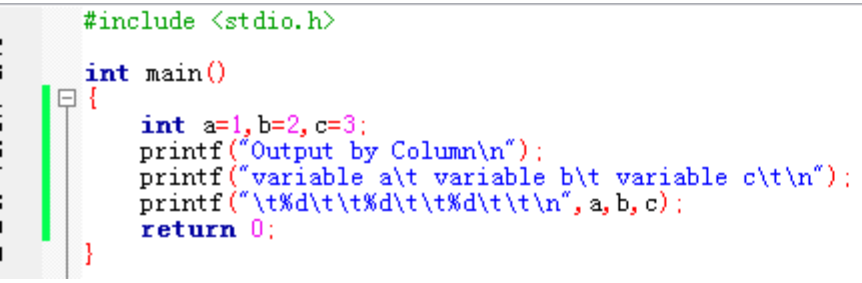
The character type char

escape sequences	Meaning
'\n'	Newline
'\t'	Horizontal tab.
'\v'	Vertical tab.
'\b'	position back one space
'\r'	Carriage return
'\\'	Backslash (\ )
'\''	Single quote (')
'\"'	Double quote (").
'\0dd'	Octal value. (d represents an octal digit.)
'\xhh'	Hexadecimal value. (h represents a hexadecimal digit.)

The character type char

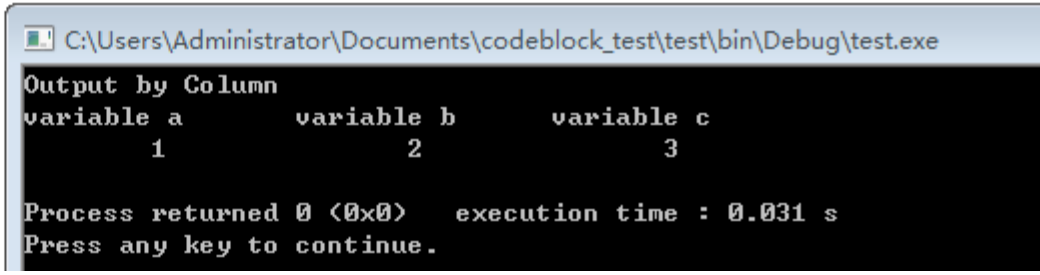
```
#include <stdio.h>

int main()
{
    int a=1,b=2,c=3;
    printf("Output by Column\n");
    printf("variable a\t variable b\t variable
c\t\n");
    printf("\t%d\t\t%d\t\t%d\t\t\n",a,b,c);
    return 0;
}
```



```
#include <stdio.h>

int main()
{
    int a=1,b=2,c=3;
    printf("Output by Column\n");
    printf("variable a\t variable b\t variable c\t\n");
    printf("\t%d\t\t%d\t\t%d\t\t\n",a,b,c);
    return 0;
}
```



```
C:\Users\Administrator\Documents\codeblock_test\test\bin\Debug\test.exe
Output by Column
variable a       variable b      variable c
        1               2               3

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

The character type char

- string type data
- String format: A valid sequence of characters enclosed in double quotes.
- for example：
- “abc”（√）、“123456”（√）、“\\#?”（√）、“a”（√） ‘abc’（×）、 ‘a*”（×）
- Storage of 'a' in memory: The ASCII value of 'a' is: 97, and the binary representation is: 01100001

0	1	1	0	0	0	0	1

- Storage of '+' in memory:The ASCII value of '+' is: 43, and the binary representation is: 00101011

0	0	1	0	1	0	1	1

The character type char

Escape sequences

Escape sequences can also be represented by ASCII code values in memory.

for example：

Storage of ‘/b' in memory:

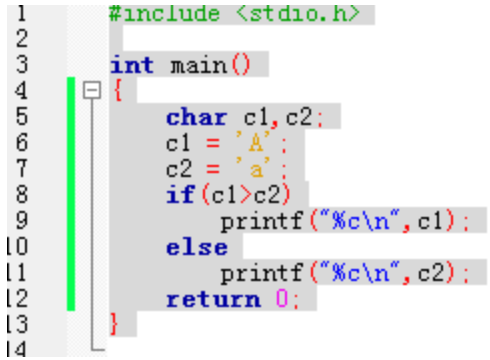
The ASCII value of '/d' is: 13, and the binary representation is: 0000101

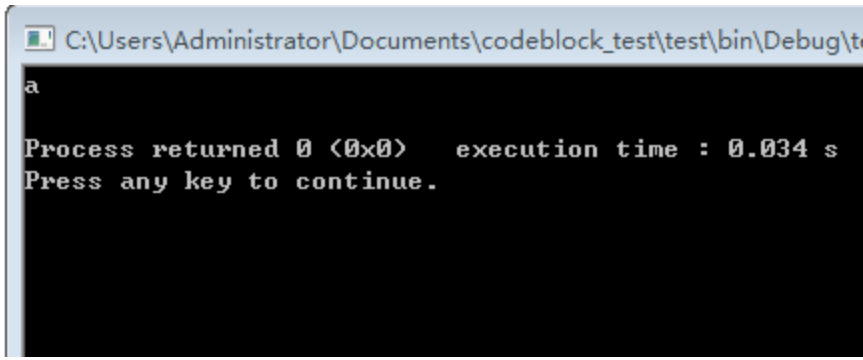
Storage of ‘/n' in memory:

The ASCII value of ‘/n' is: 10, and the binary representation is: 00101011

The character type char

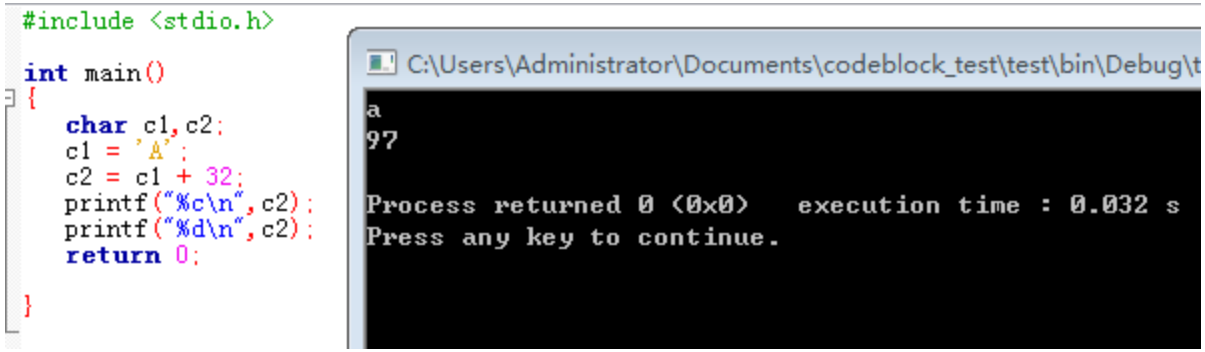
```
#include <stdio.h>
int main()
{
    char c1,c2;
    c1 = 'A';
    c2 = 'a';
    if(c1>c2)
        printf("%c\n",c1);
    else
        printf("%c\n",c2);
    return 0;
}
```





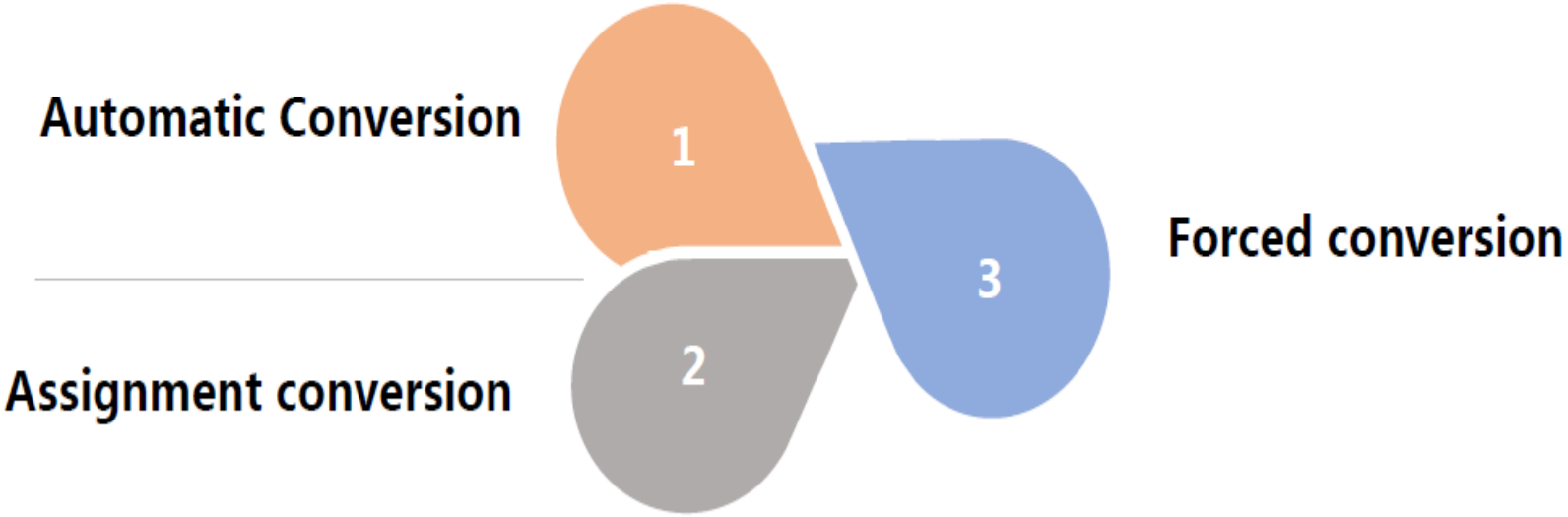
The character type char

```
#include <stdio.h>
int main ( )
{
   char c1,c2;
   c1=’A’;
   c2=c1+32;
   printf("%c\n",c2);
   printf(”%d\n”,c2);
   return 0;
}
```



Data Type Conversion

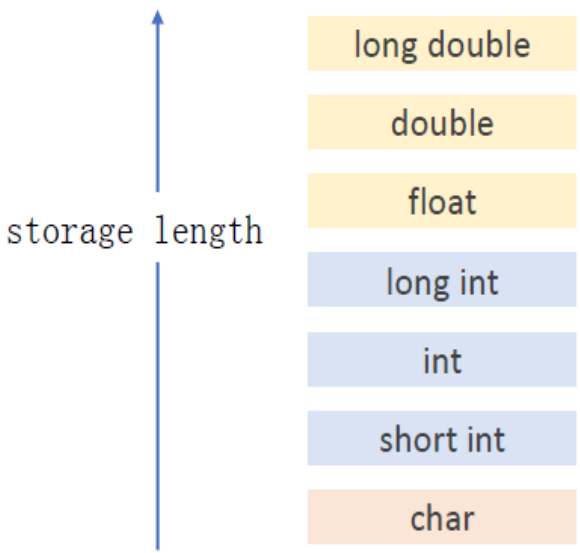
Data Type Conversion

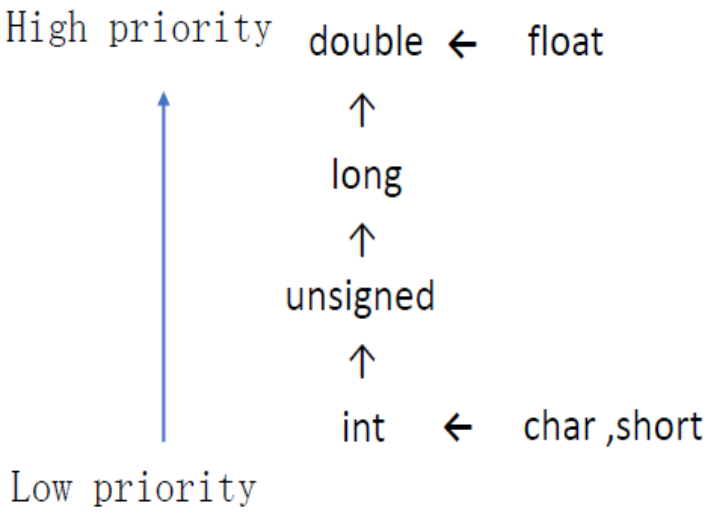


Data Type Conversion

Data Type Conversion

【automatic type conversion】Conversion principle: Convert data with short storage length into data with long storage length to ensure that the accuracy of the data is not reduced.



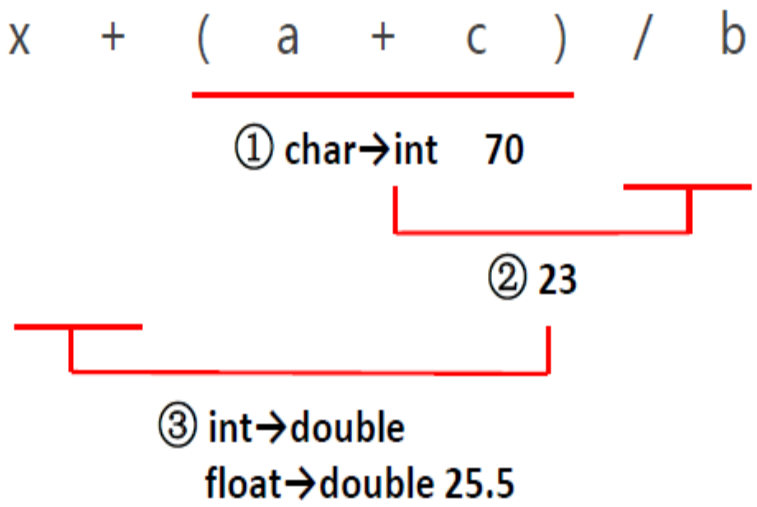


Data Type Conversion

Data Type Conversion

For example:

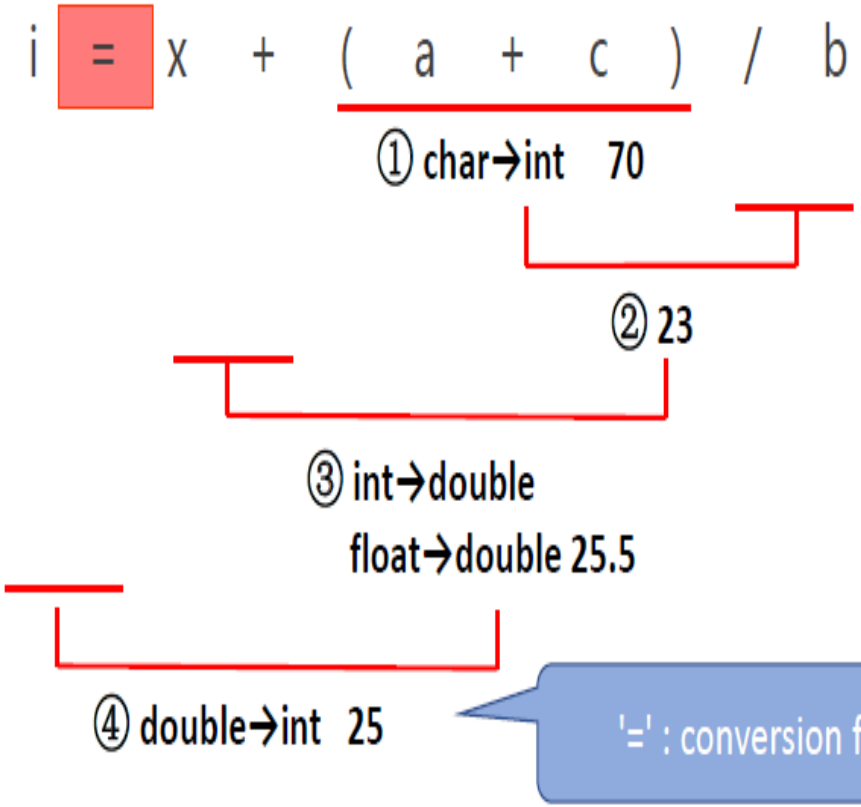
int a=5, b=3; float x=2.5; char c= 'A' ; the value of x+(a+c)/b?



Data Type Conversion

【Assignment type conversion】If the data types on both sides of the assignment are different, convert the type on the right side of the assignment to the type of the variable on the left

for example : int a=5, b=3, i ; float x=2.5; char c= 'A' ; the value of i=x+(a+c)/b ?



Data Type Conversion

【Force data type conversion】Use type conversion operators to force a data or expression to be converted to a specific type.

General Form :

(Data type converter) expression

for example :

int a=5, b=2, i ; float y Find the value of the following equation

i=a+b; 7

y=a/b; 2.0

y=(float)a/b; 2.5

y=(float)(a/b); 2.0

constants and variables

Example: Input the radius of a circle and output the circumference of the circle.
If the radius is r and the circumference is c, then:
S=2 * 3.1415926 * r

```
#include<stdio.h>
#define PI 3.1415926
main()
{
        float r,s;
        scanf("%f",&r);
        s=2*PI*r;
        printf("c=%f\n",s);
}
```

In C language, data is divided into two categories: constant and variable based on whether it can be changed in a program

constants and variables

Constant - refers to the quantity whose value cannot be changed during program operation

- **Direct Constants - Refers to numerical and character type constants in C language**
- **Symbolic constant - refers to a constant defined by an identifier in C language**

Variable - refers to the quantity whose value can change, named after a certain identifier

Direct constant: In C language, it appears in its own representation form, including integer constant, real constant, character constant, string constant, and so on.

For example:

F=i+2	**B=3.14 * a**
Printf ("hello world");	**'*', 'a'**

constants and variables

Symbolic constant: In C language, an identifier can be defined as a constant.

Format:#Define identifier constant data

For example:

#Define PI 3.1415926

#Defint F "false“

After defining a symbolic constant, whenever the identifier is encountered during program processing, it will be replaced with the corresponding constant.

constants and variables

Variables

In C language, variables represent a storage space in computer memory that can hold different types of data.

The use of variables follows the principle of defining before using.

The general format for defining variables is:

Type identifier variable name 1, variable name 2,... variable name n;

For example:

```
Int a, b// Define two integer variables a and b
Float x, y// Define two real variables x and y
Char c// Define a character variable c
```

Type identifier - Declares the type of the variable, determines the number of bytes of memory units allocated by the program for the variable, and the value range of the variable

constants and variables

Variable name
Following the naming convention for identifiers, identifiers can only consist of numbers, letters, and underscores, with the first digit being either a letter or an underscore.

example
a1（√）、book_1（√）、_no（√）、7_paa（×）
Variable names are case sensitive.
For example, ADD, add, and Add are three different variable names
Variable names cannot use keywords specified in C language

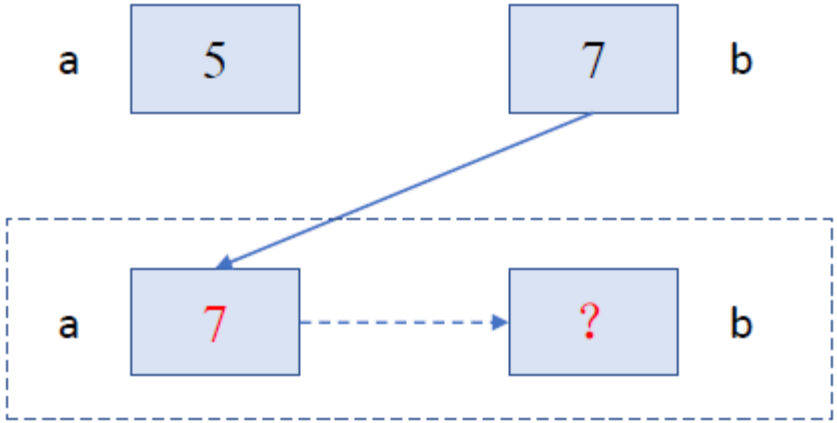
constants and variables

Keywords in C language

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

constants and variables

Example: Define two integer variables a and b, assign values 5 and 7 respectively, and exchange their values.



When the value of variable b is placed in variable a, what is the value of variable a? What value will variable b obtain?

```
#include<stdio.h>
main()
{
inta,b,c;
a=5;b=7;
c=a;
a=b;
b=c;
printf("a=%d,b=%d",a,b);
}
```

constants and variables

Initialization of variables

After defining variables, the program allocates corresponding storage space for the variables in memory, but at this point, the data in the storage space is random. We need to reassign variables, which is called variable initialization.

How to initialize variables

Simultaneous assignment of declaration

```
int a = 10;
float x = 2.5;
char c = '*';
```

Declare before assigning values

```
inta;
float x;
char c;
a = 10;
x = 2.5;
c = '#';
```

Multiple assignments

```
int a,b,c;
a=b=c=10;
a=(b=(c=10));
```

From right to left

constants and variables

The 'lifecycle' of variables

When a variable is defined, the program allocates corresponding storage space for the variable in memory
Obtain the initial value of a variable through an assignment statement
During program operation, the values stored in variables may change
Release the memory space occupied by variables after the program ends

Classification of variables
local variable and global variable

constants and variables

Example: Modifying errors in the following program

```
#include<stdio.h>
main()
{ int a;
a=3.14;
float f;
f=1.23;
printf(“a=%d\n”,a);
printf(“f=%f\n”,f);
}
```

```
#include<stdio.h>
main()
{ int a;
float f;
a=3;
f=1.23;
printf(“a=%d\n”,a);
printf(“f=%f\n”,f);
}
```

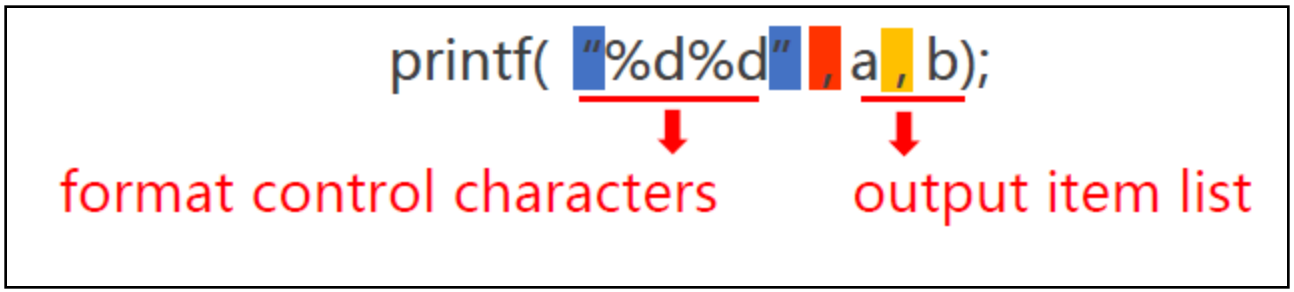
warning C4244: '=' : conversion from 'constdouble ' to 'int', possible loss of data

error C2065: 'f' : undeclared identifier

Printf and Scanf

Standardized output statements:
Printf (format control characters, parameter 1, parameter 2...);
Standardized input statements:
Scanf (format control characters, parameter 1, parameter 2...);

Printf (format control characters, output item list);
Function: Display the corresponding parameter values on a standard output device (such as a display) according to the specified output format.
For example:



Printf and Scanf

Printf() function - format control characters

%d Output a decimal integer data
%o Output an octal integer data
%x Output a hexadecimal integer data
%c Output a character
%s Output a string
%f or %e Output a real data float point data.

example
printf(“%d”,a); //Output a decimal integer data.
If the value of a is 1234, then output 1234
printf(“%f”,a); //Output a float point data.
If the value of a is 12.34, then output 12.34

Printf and Scanf

Modifier	Meaning
flag	The five flags (`-`, `+`, space, `#`, and `0`) are described in Table 4.5. Zero or more flags may be present.
	Example: `"%-10d"`.
digit(s)	The minimum field width. A wider field will be used if the printed number or string won't fit in the field.
	Example: `"%4d"`.
.digit(s)	Precision. For `%e`, `%E`, and `%f` conversions, the number of digits to be printed to the right of the decimal. For `%g` and `%G` conversions, the maximum number of significant digits. For `%s` conversions, the maximum number of characters to be printed. For integer conversions, the minimum number of digits to appear; leading zeros are used if necessary to meet this minimum. Using only `.` implies a following zero, so `%.f` is the same as `%.0f`.
	Example: `"%5.2f"` prints a `float` in a field five characters wide with two digits after the decimal point.

Printf and Scanf

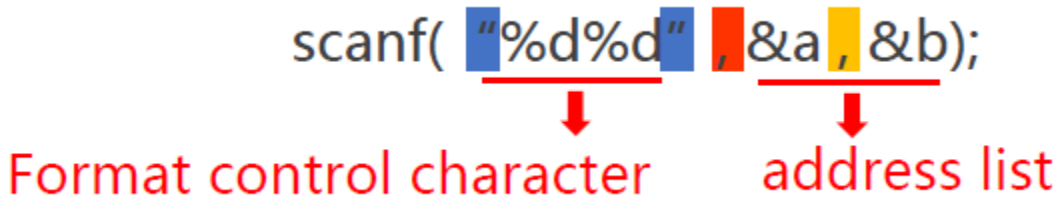
For example:
Printf ("% 3d\ n", a);
If the value of a is 1234, the output result is ␣1234
If the value of a is -1, the output result is ␣-1

Printf ("a=%+7.2f b=% f \ n", a, b);
If the values of a and b are both 1.23456,
the output result is a=␣␣ +1.23 b=1.234560

Printf ("a=% e\ n", a);
If the value of a is double precision 123.456789
the output result is a=1.234568e+002

Printf and Scanf

Scanf (Format Control Characters, Address List);
Function: Receive user input data from the keyboard, perform type conversion according to the format control requirements, and then send it to the variable unit specified by the corresponding parameter.
for example



Note: The scanf function must specify the address of the variable used to receive data, and store the obtained data in the designated variable unit.

Printf and Scanf

scanf() function - format control characters

%d Input a decimal integer data
%o Input an octal integer data
%x Input a hexadecimal integer data
%c Input a character,
Including spaces, carriage returns, and tab characters
%s Input a string
Space, carriage return, and tab characters are considered as input ends
%f or %e Input a real data float point data.

Printf and Scanf

For example:
Scanf ('% d',&a)// Enter a decimal integer data
To obtain a value of 1234 for a, enter 1234.
Scanf ("% f",&a)// Enter a real data
To obtain a value of 12.34 for a, enter 12.34.
Note: If the data type does not match during input, the scanf() function will stop processing and cause a program error.

Printf and Scanf

```
#include<stdio.h>
main()
{ int a;
   char c;
Scanf(“%d%c”,&a,&c)
printf(“a=%d\n”,a);
printf(“c=%c\n”,c);
}
```

```
Scanf(“a=%d c=%c”, &a, &c)
Scanf(“a=%d c=%c”, a, c)
```

Printf and Scanf

The scanf() function - the interval between input data
Example:
the following inputs all need to achieve a value of 12 for a and 34 for b

Statement	Input Format
scanf("%d%d",&a,&b);	12␣34 or 12↙34
scanf("%d,%d",&a,&b);	12,34
scanf("%d %d",&a,&b);	12␣34
scanf("a=%db=%d",&a,&b);	a=12b=34

Input and output character

Getchar()// Enter a character from the input device and press Enter to end the input
Putchar()// Output a character to the current cursor position on the screen

```
#include <stdio.h>
main()
{
char c;
c=getchar();
c=c-32;
putchar(c);
putchar('\n');
}
```

```
a
A
Press any key to continue
```

operator

The symbol representing various operations is called an operator. For example:+, -, */

Types of Operators	operator
arithmetic operator	+、-（negation）、*、/、%、++、--
Relational Operators	>、<、==、>=、<=、！=
Logical operator	&&、||、！
Bit operator	<<、>>、~、|、^、&
Assignment Operators	=、+=、-=、*=、/=
Conditional Operators	?：
Comma Operator	,
Byte operator	sizeof
Pointer operators	*、&
Other Operators	( )、[ ]、(->,.)

operator

[Priority of operator]

When there are multiple operators in an expression, the calculation has a priority order, which is called the priority of the operators.
For example: a+b * c; The priority of '*' is higher than that of '+', so '*' is calculated first and then '+' is calculated.

[Associativity of Operators]

When there are multiple operators with the same priority in an expression, the direction of combination between the operator and the operand is called operator associativity.For example: a+b+c; The associativity of '+' is' from left to right ', where' a+b 'is calculated first and then'+c 'is calculated.

[Number of operands for operator]

For example: a+b; '+' is an operator of two operands that require two operands to execute.

expression

The formula connected by operators is called an expression.
For example, a formula connected by arithmetic operators is called an arithmetic expression: a+b+c
In C language, the operands connected by operators can be of various types such as constants, variables, functions, etc., and parentheses can also be used to change the order of operations.

example
a + (3 + getchar())

expression

Arithmetic operators and expressions

Operator	meaning	Required operands	direction
+	Add	two	from left to right
-	Subtract （negative）	two (one)	from left to righ
*	Multiply	two	from left to right
/	Divide	two	from left to right
%	modular	two	from left to right
++	increment	one	from right to left
--	decrement	one	from right to left

expression

Arithmetic operators and expressions

The operands on both sides of the arithmetic operator must be of the same data type. If the data types are different, the data type conversion is performed first before the operation

For example:

Y=5/2; 2//Divide integer data, and if the result has decimal places, directly round and discard decimals

Y=5.0/2; 2.5

Y=5.0/2.0; 2.5

Specifically, the two operands of the '%' operator must be integer data.

y=8%3 2 y=8%3.0 × y=8.0%3 ×

expression

The operator '-' can represent both subtraction and negative value operations. When there is only one operand, it represents taking negative, and when there are two operands, it represents subtraction.

The operands of the self increasing (++) and self decreasing (--) operators can only be variables, and their purpose is to make the value of the variable+1 or -1

++i (--i) indicates that before using i, the value of i is first increased by 1 (minus 1)
i++(i--) indicates that after using i, the value of i is increased by 1 (minus 1)

For example, assuming a=5

	value of a	value of b
b=++a;	6	6
b=a++;	6	5

expression

Relational Operators and Expressions

The relational operator is used to represent the size relationship between two operands. When the return value is true, it is represented by "1", and when it is false, it is represented by "0".

Operator	meaning	Required operands	direction
>	greater than	two	from left to right
<	less than	two	from left to righ
>=	greater than or equal	two	from left to right
<=	less than or equal	two	from left to right
==	equal to	two	from left to right
!=	Not equal to	two	from left to right

expression

The priority of relational operators is lower than that of arithmetic operators.

For example	
2+3<5	**0**
2+(3<5)	**3**
2<3<5	**1**
2<(3<5)	**0**

expression

logical operators and expressions

Operator	meaning	Required operands	direction
&&	and	two	from left to right
||	or	two	from left to righ
!	Not	one	from right to left

Logical operation truth table

operand a	operand b	a&&b	a||b	!a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

expression

The operating objects of logical operators can be integer data, real data, and character data. In C language, non zero operands are all true, and zero operands are all false

For example
-1&&1 1
-1&&0 0

In logical operators, the '!' operator has higher priority than '&&' and '||'. Logical operators have lower priority than arithmetic operators. There are only two results for the value of a logical expression. When the value of the expression is true, it is represented by "1", and when the value of the expression is false, it is represented by "0".

For example, given a=3, b=4, c=5, d=6, find the value of the following equation:

a<b&&c<d	1	a<b||c<d	1
a<b&&c>d	0	a<b||c>d	1
a>b&&c<d	0	a>b||c<d	1
a>b&&c>d	0	a>b||c>d	0

The function of the assignment operator "=" is to assign a known value to a defined variable.Any form of variable can be assigned using '='.

For example:
int a; A=7;
A=7<9;
When "=" is combined with "+", "-", "*", "/" to form "+=", "-="*=,/=operators

example
int a=3;
a+=2; Equivalent to a=a+2;
int a=3;
a+=a-=a*a; Equivalent to
a=a+（a-=a*a） Equivalent to
a=a+(a=a-a*a);

a=a+(a=a-a*a);

a*a

a=a+ a =a- 9

-6 -6

-12 -12

Conditional Operators and Expressions

Conditional operator: **? :**

The form of a conditional expression

Condition? Expression 1: Expression 2

The function of a conditional operator is similar to an if... else statement, which selects one of two options based on the value of the condition. It requires three operands

The operation order of conditional operators: from left to right, that is, first operate on the operands to the left of "?", and then determine which value of the left and right sides of the operator ":" is the value of the expression based on the result of the operands.

If the value of the condition is true, then the value of the conditional expression is the value of expression 1If the value of the condition is false, then the value of the condition expression is the value of expression 2

Conditional operators have lower priority than arithmetic operators, relational operators, logical operators, and higher priority than assignment operators

For example, assuming inta=1 and b=2; The value of the expression a>b?a:b is ()

```
#include <stdio.h>
void main()
{
int a=1,b=2;
printf("%d\n",a>b?a:b);
}
```

```
2
Press any key to continue
```

```c
#include <stdio.h>
void main()
{
int a=97;
char b='b';
printf("%d\n",5<3?a:b);
}
```

```
98
Press any key to continue
```

If the types of values in expression 1 and expression 2 are different, the type of the entire conditional expression value is consistent with the higher priority data type in both.

Comma operators and expressions

The form of the comma operator: ,
Comma expression: The comma operator can connect two or more expressions as one expression

The form of a comma expression: expression 1, expression 2,... expression n

The operation order of the comma operator: from left to right, first operate on the expression on the left, and then sequentially operate on the expression on the right.
The value of the comma expression: the value of the last expression n.

The priority of the comma operator is the lowest among all operators

For example:

1) Int a=1, b=3, c; In the expression c=(a+1, b+2, a+3), the value of c is 4

2) Int a=1, b=3, c; In the expressions c=a+1, b+2, a+3, the value of c is 2.