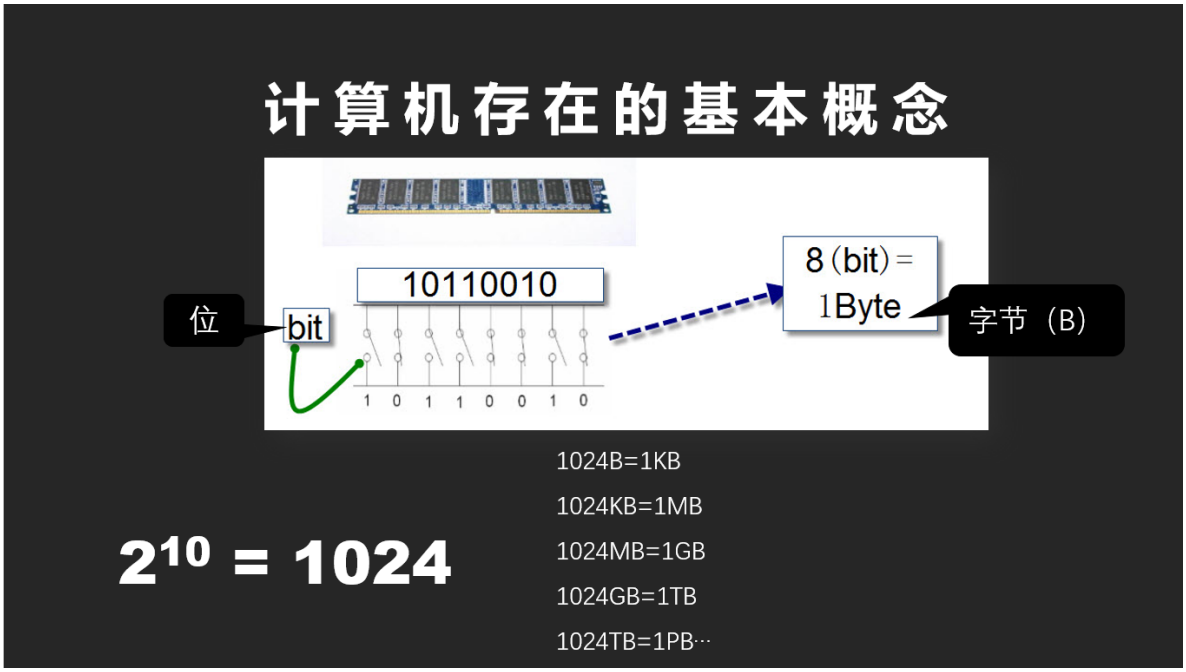
C语言——面向过程编程

单元一、流程结构

第四次课：数据类型与表达式

一、数据存储的基本概念




```
基本数据单位换算：
1B（Byte字节）=8bit
1KB (Kilobyte 千字节)=1024B,
1MB (Mega byte 兆字节 简称“兆”)=1024KB,
1GB (Giga byte 吉字节 又称“千兆”)=1024MB,
1TB (Tera byte 万亿字节 太字节)=1024GB，其中1024=2^10 ( 2 的10次方),
1PB（Peta byte 千万亿字节 拍字节）=1024TB,
1EB（Exa byte 百亿亿字节 艾字节）=1024PB,
1ZB (Zetta byte 十万亿亿字节 泽字节)= 1024 EB,
1YB (Yotta byte 一亿亿亿字节 尧字节)= 1024 ZB,
1BB (Bronto byte 一千亿亿亿字节)= 1024 YB
1NB(Nona byte )= 1024BB
1DB(Dogga byte)= 1024NB
```

二、 C语言基本数据类型

C语言基本数据类型

	类型关键字	类型名称	字节数	数据范围	字面值后缀	打印格式
1	signed char	有符号字符型	1	-128~127	-	%c
2	unsigned char	无符号字符型	1	0~255	-	%hhu
3	signed short int	有符号短整型	2	-32768~32767	-	%hd
4	unsigned short int	无符号短整型	2	0~65535	-	%hu
5	signed int	有符号整型型	4	-2147483648~2147483647	-	%d或%i
6	unsigned int	无符号整型	4	0~4294967295	U或u	%u
7	singed long int	有符号长整型	4	-2147483648~2147483647	L或l	%ld
8	unsigned long int	无符号长整型	4	0~4294967295	UL或ul	%lu
9	signed long long int	有符号长长整型	8	-9223372036854775808 ~9223372036854775807	LL或ll	%lld
10	unsigned long long int	无符号长长整型	8	0~18446744073709551615	ULL或ull	%llu
11	float	单精度浮点型	4	3.4e-38~3.4e38	F或f	%f或%e
12	double	双精度浮点型	8	1.7e-308~1.7e308	-	%f或%e
13	long double	长双精度类型	8	1.7e-308~1.7e308	-	%lf或%e

1、整型相关

```
    int iValue = 9;
    unsigned int uiValue=9u;
    long int lValue = 9l;
    unsigned long int ulValue=95ul;
    long long int llValue = 95ll;
    unsigned long long int ullValue = 95ull;
    short int sValue=9;
    unsigned short int usValue=9u;
    unsigned char ucValue=48u;
```

A：类型关键字

1. 整型类型关键字中 signed符号 通常省略
2. int为基准类型 ， int和别的类型关键字组合时，通常int可以省略。
3. long与int在字节数上可能都是4个字节。在不同的编译器下,不同的操纵系统都有可能不同的,不是固定值。类型关键词通常由C++委员会制定，厂商参照设计。

```
C++真正正式公布的标准就三个：
C++98、C++03、C++11。
其中C++98是第一个正式C++标准，C++03是在C++98上面进行了小幅度的修订，C++11则是一次全面的大进化（之前称C++11为C++0x，以为会在08~09年公布，没想到拖到了11年）。
其实C++在第一个标准C++98之前就已经广为使用了。只不过那时候还没有一个官方的统一标准，后来才开始起草标准草案，起草了n年，直到98年才最终发布。（所以你可以看到一些早期的C++编译器对标准的支持程度很差，比如经典的VC6.0，在研发的时候C++的第一个标准还没公布呢）
至于什么C89、C99这些，是C语言的标准，不是C++的标准，很容易被一些人混淆
```

B：名称规范

```
1 . 匈牙利命名：

```

开头字母用变量类型的缩写，其余部分用变量的英文或英文的缩写，要求单词第一个字母大写。

ex:
int iMyAge; “i”是int类型的缩写；
char cMyName[10]; “c”是char类型的缩写；
float fManHeight; “f”是float类型的缩写；

其他：
前缀类型 a b by c cb cr cx,cy dw fn h i l lp m_ n np p s sz w （一一对应关系）
数组 (Array) 布尔值 (Boolean) 字节 (Byte) 有符号字符 (Char) 无符号字符 (Char Byte，没有多少人用) 颜色参考值 (ColorRef) 坐标差（长度 ShortInt） Double Word 函数 Handle（句柄） 整型 长整型 (Long Int) Long Pointer 类的成员 短整型 (Short Int) Near Pointer Pointer 字符串型 以 null 做结尾的字符串型 (String with Zero End) Word

2 . 驼峰式命名法：

又叫小驼峰式命名法。
第一个单词首字母小写，后面其他单词首字母大写。

ex:
int myAge;
char myName[10];
float manHeight;

3 . 帕斯卡命名法：

又叫大驼峰式命名法。
每个单词的第一个字母都大写。

ex:
int MyAge;
char MyName[10];
float ManHeight;

4 . 还有些许其他的命名规范，如：下划线命名法。

C：字面值 及 前、后缀

字面值是指在程序中无需变量保存，可直接表示为一个具体的数字或字符串的值。

十进制、八进制、 十六进制字面值

前缀指定基数：0x 或 0X 表示十六进制，0 表示八进制，不带前缀则默认表示十进制。

整数常量也可以带一个后缀，后缀是 U 和 L 的组合，U 表示无符号整数（unsigned），L 表示长整数（long）。后缀可以是大写，也可以是小写，U 和 L 的顺序任意，但是后缀不可以重复。

```
    int iValue = 09;//09 八进制 字面值
    unsigned int uiValue=0x9u;//0x9u 十六进制 无符号 字面值
    long int lValue = 0X9Al;//0X9Al 十六进制 长整型 字面值
    unsigned long int ulValue=95ul;//95ul 十进制 无符号 长整型 字面值
    long long int llValue = 95ll;//95ll 十进制   大长整型 字面值

```

D：左值与右值

> C/C++语言中可以放在赋值符号左边的变量，即具有对应的可以由用户访问的存储单元，并且能够由用户去改变其值的量。左值表示存储在计算机内存的对象，而不是常量或计算的结果。或者说左值是代表一个内存地址值，并且通过这个内存地址，就可以对内存进行读并且写（主要是能写）操作；这也就是为什么左值可以被赋值的原因了。相对应的还有右值：当一个符号或者常量放在操作符右边的时候，计算机就读取他们的“右值”，也就是其代表的真实值。简单来说就是，左值相当于地址值，右值相当于数据值。右值指的是引用了一个存储在某个内存地址里的数据。

E：进制转换

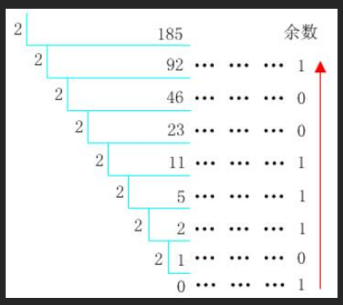
2/8/16 进制转换成十进制

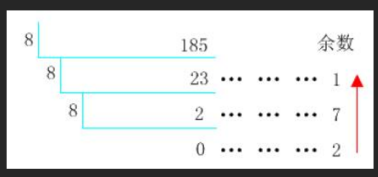
$$a_n \times r^n + a_{n-1} \times r^{n-1} + \ldots + a_1 \times r^1 + a_0 \times r^0$$

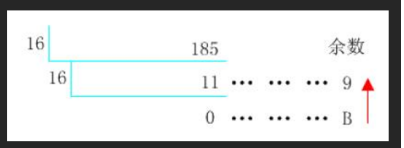
- $(1011011)_2 = 1\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0 = 91$
- $(356)_8 = 3\times8^2+5\times8^1+6\times8^0 = 238$
- $(2FB)_{16} = 2\times16^2+15\times16^1+11\times16^0 = 763$

10进制转换成 2/8/16进制

方法：除 r 取余数，直至商为零，余数倒序排序。





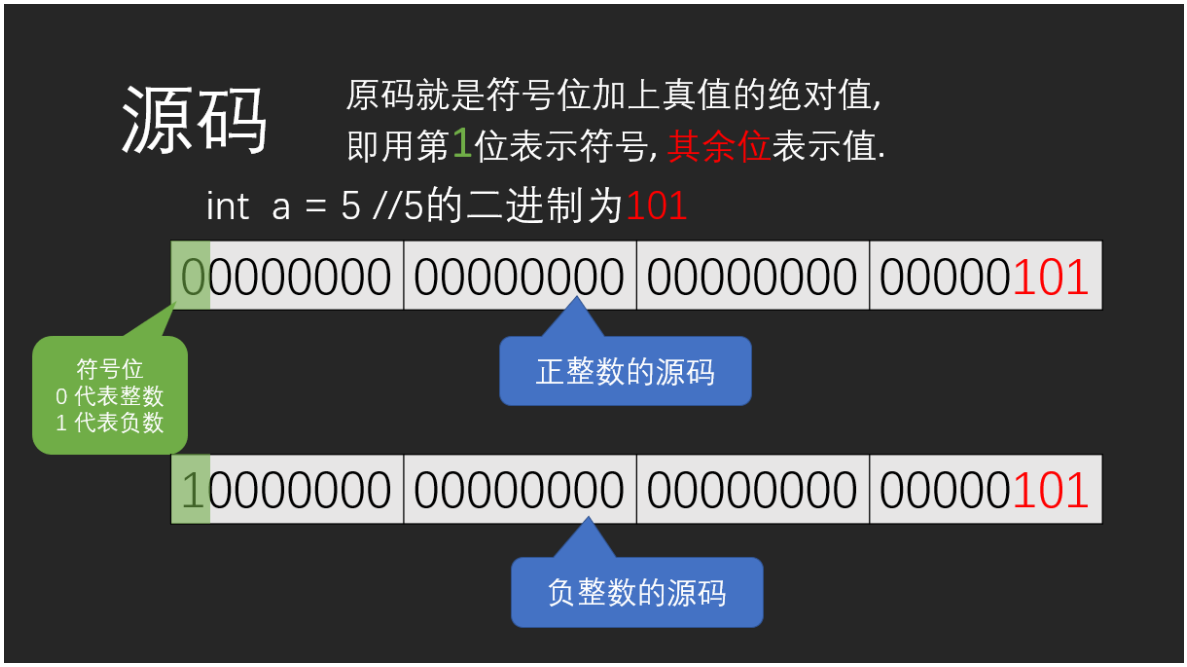


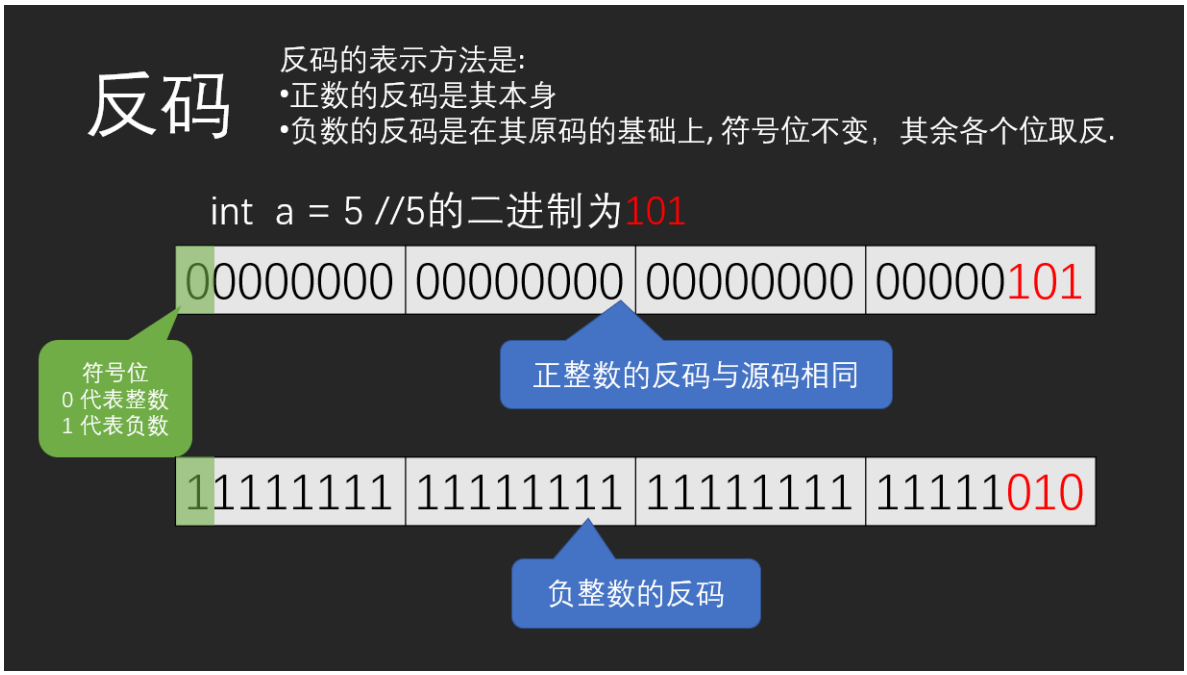
十进制数字：0 1 2 3 4 5 6 7 8 9

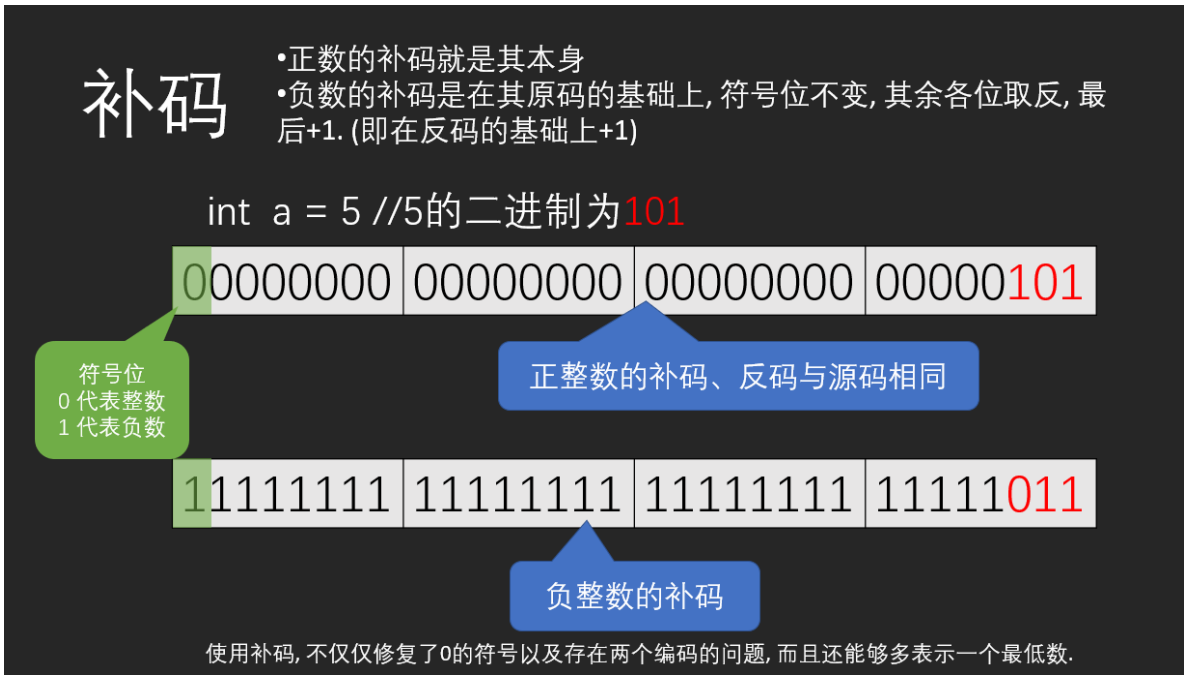
0八进制数字：0 1 2 3 4 5 6 7

0X十六进制数字：0 1 2 3 4 5 6 7 8 9 aA bB cC dD eE fF

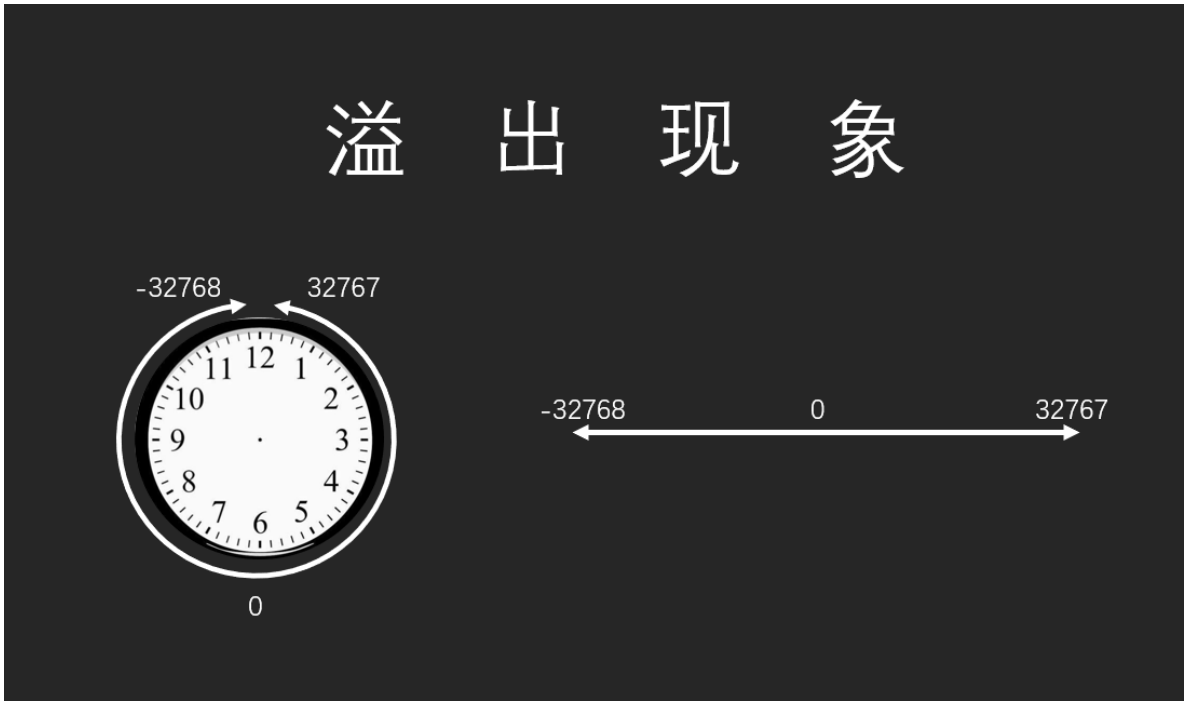
F：数据存储原理


源码
原码就是符号位加上真值的绝对值,
即用第1位表示符号, 其余位表示值.
int a = 5 //5的二进制为101
00000000 00000000 00000000 00000101
符号位
0 代表整数
1 代表负数
正整数的源码
10000000 00000000 00000000 00000101
负整数的源码


反码
反码的表示方法是:
•正数的反码是其本身
•负数的反码是在其原码的基础上, 符号位不变, 其余各个位取反.
int a = 5 //5的二进制为101
00000000 00000000 00000000 00000101
符号位
0 代表整数
1 代表负数
正整数的反码与源码相同
11111111 11111111 11111111 11111010
负整数的反码


补码
•正数的补码就是其本身
•负数的补码是在其原码的基础上, 符号位不变, 其余各位取反, 最后+1. (即在反码的基础上+1)
int a = 5 //5的二进制为101
00000000 00000000 00000000 00000101
符号位
0 代表整数
1 代表负数
正整数的补码、反码与源码相同
11111111 11111111 11111111 11111011
负整数的补码
使用补码, 不仅仅修复了0的符号以及存在两个编码的问题, 而且还能够多表示一个最低数.

G: 溢出现象的原理




```
    short   aa= 32767;//2个字节最大数+1 将溢出得到最小数
    printf("%hhd\n",aa+1);

    aa= -32768;//2个字节最小数-1 将溢出得到最大数
    printf("%hhd\n",aa-1);
```

H: printf的用法查表

https://www.cplusplus.com/reference/cstdio/printf/

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

promotion or conversion is performed, if allowed).

	specifiers						
length	d i	u o x X	f F e E g G a A	c	s	p	n
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

2、字符型相关

十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号
0	0000 0000	NUL	32	0010 0000	[空格]	64	0100 0000	@	96	0110 0000	`
1	0000 0001	SOH	33	0010 0001	!	65	0100 0001	A	97	0110 0001	a
2	0000 0010	STX	34	0010 0010	"	66	0100 0010	B	98	0110 0010	b
3	0000 0011	ETX	35	0010 0011	#	67	0100 0011	C	99	0110 0011	c
4	0000 0100	EOT	36	0010 0100	$	68	0100 0100	D	100	0110 0100	d
5	0000 0101	ENQ	37	0010 0101	%	69	0100 0101	E	101	0110 0101	e
6	0000 0110	ACK	38	0010 0110	&	70	0100 0110	F	102	0110 0110	f
7	0000 0111	BEL	39	0010 0111	'	71	0100 0111	G	103	0110 0111	g
8	0000 1000	BS	40	0010 1000	(	72	0100 1000	H	104	0110 1000	h
9	0000 1001	HT	41	0010 1001	)	73	0100 1001	I	105	0110 1001	i
10	0000 1010	LF	42	0010 1010	*	74	0100 1010	J	106	0110 1010	j
11	0000 1011	VT	43	0010 1011	+	75	0100 1011	K	107	0110 1011	k
12	0000 1100	FF	44	0010 1100	,	76	0100 1100	L	108	0110 1100	l
13	0000 1101	CR	45	0010 1101	-	77	0100 1101	M	109	0110 1101	m
14	0000 1110	SO	46	0010 1110	.	78	0100 1110	N	110	0110 1110	n
15	0000 1111	SI	47	0010 1111	/	79	0100 1111	O	111	0110 1111	o
16	0001 0000	DLE	48	0011 0000	0	80	0101 0000	P	112	0111 0000	p
17	0001 0001	DC1	49	0011 0001	1	81	0101 0001	Q	113	0111 0001	q
18	0001 0010	DC2	50	0011 0010	2	82	0101 0010	R	114	0111 0010	r
19	0001 0011	DC3	51	0011 0011	3	83	0101 0011	S	115	0111 0011	s
20	0001 0100	DC4	52	0011 0100	4	84	0101 0100	T	116	0111 0100	t
21	0001 0101	NAK	53	0011 0101	5	85	0101 0101	U	117	0111 0101	u
22	0001 0110	SYN	54	0011 0110	6	86	0101 0110	V	118	0111 0110	v
23	0001 0111	ETB	55	0011 0111	7	87	0101 0111	W	119	0111 0111	w
24	0001 1000	CAN	56	0011 1000	8	88	0101 1000	X	120	0111 1000	x
25	0001 1001	EM	57	0011 1001	9	89	0101 1001	Y	121	0111 1001	y
26	0001 1010	SUB	58	0011 1010	:	90	0101 1010	Z	122	0111 1010	z
27	0001 1011	ESC	59	0011 1011	;	91	0101 1011	[	123	0111 1011	{
28	0001 1100	FS	60	0011 1100	<	92	0101 1100	\	124	0111 1100	|
29	0001 1101	GS	61	0011 1101	=	93	0101 1101	]	125	0111 1101	}
30	0001 1110	RS	62	0011 1110	>	94	0101 1110	^	126	0111 1110	~
31	0001 1111	US	63	0011 1111	?	95	0101 1111	_	127	0111 1111	DEL

l：打印整型所占字节数的代码

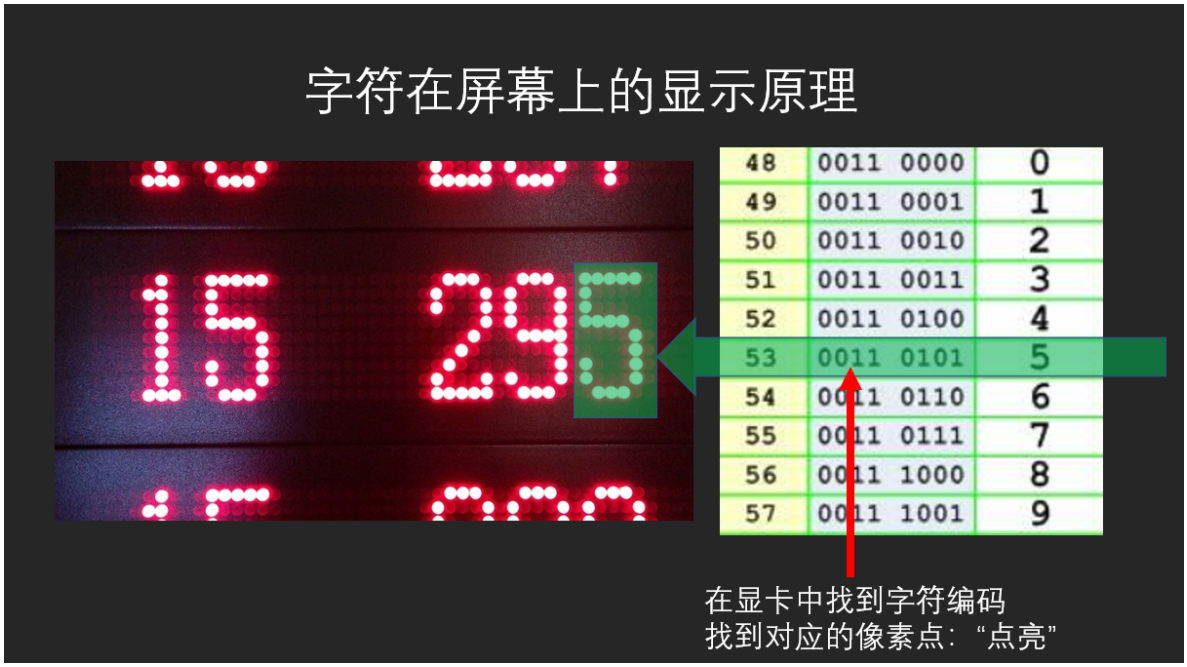
```
int iValue = 9;
unsigned int uiValue=9u;
long  lValue = 9L;
unsigned long  ulValue=95ul;
long long  llValue = 95LL;
unsigned long long  ullValue = 95ull;
short  sValue=9;//小于int 按int
unsigned short  usValue=9u;
unsigned char ucValue=48u;

printf("signed int 所占字节数 %d %d %d\n ", sizeof( signed int ),sizeof(iValue),sizeof(9)  );
printf("unsigned int 所占字节数 %d %d %d \n", sizeof(unsigned int ),sizeof(uiValue),sizeof(9u)  );

```

```
printf("long int 所占字节数 %d %d %d \n", sizeof(long int ),sizeof(lValue),sizeof(9L)  );
printf("long long int 所占字节数 %d %d %d \n", sizeof(long long int ),sizeof(llValue),sizeof(9LL)  );

printf("short int 所占字节数 %d %d %d \n", sizeof(short int ),sizeof(sValue),sizeof(9)  );
printf("char 所占字节数 %d %d %d \n", sizeof(char ),sizeof(ucValue),sizeof(48u)  );
```

II: 打印不同整型数据范围的代码

```
unsigned int a= -1;//4
unsigned short b=-1;//2
unsigned char c = -1;//1
unsigned long long int d = -1LL;//8
printf("char的数据范围:  %hhd %hhd %hhu\n",-(c/2+1),c/2,c);
printf("short的数据范围:  %hd %hd %hu\n",b/2+1,b/2,b);
printf("int的数据范围:  %d %d %u\n",a/2+1 ,a/2,a);//a/2+1 溢出现象 最大整数+1得到最下整数
printf("long long的数据范围:  %lld %lld %llu\n",d/2+1,d/2,d);
```

A: 字符在屏幕上的显示原理



B：字符存储原理

与整型存储原理完全相同，且可之间进行算术运算。区别：

① 由于编码对应的文字图像是有限制的。所以所占字节较少，有一个字节的也有2个字节的。一个字节去掉符号位，最多显示256个字符，两个字节去掉符号位最多可显示65536个字符，基本覆盖世界任何国家的常用文字。

②常见的字符编码表：一个字节的如：ASCII码、两个字节的如：unicode码。

③常见的字符集：

ASCII字符集&编码
ASCII(American Standard Code for Information Interchange, 美国信息交换标准代码)是基于拉丁字母的一套电脑编码系统。它主要用于显示现代英语，而其扩展版本EASCII则可以部分支持其他西欧语言，并等同于国际标准ISO/IEC 646。

字符集范围
ASCII一共定义了128个字符,包括33个控制字符,和95个可显示字符。大部分的控制字符已经被废弃。

GB2312字符集&编码
GB 2312 或 GB 2312-80 是中华人民共和国国家标准简体中文字符集，全称《信息交换用汉字编码字符集·基本集》，又称GB0，由中国国家标准总局发布，1981年5月1日实施。GB 2312编码通行于中国大陆；新加坡等地也采用此编码。中国大陆几乎所有的中文系统和国际化的软件都支持GB 2312。

GBK字符集&编码
汉字内码扩展规范，称GBK，全名为《汉字内码扩展规范(GBK)》1.0版，由中华人民共和国全国信息技术标准化技术委员会1995年12月1日制订，国家技术监督局标准化司和电子工业部科技与质量监督司1995年12月15日联合以《技术标函[1995]229号》文件的形式公布。
GBK的K为汉语拼音Kuo Zhan（扩展）中“扩”字的声母。英文全称Chinese Internal Code Extension Specification。

Unicode字符集&编码
Unicode（中文：万国码、国际码、统一码、单一码）是计算机科学领域里的一项业界标准。它对世界上大部分的文字系统进行了整理、编码，使得电脑可以用更为简单的方式来呈现和处理文字。
Unicode伴随着通用字符集的标准而发展，同时也以书本的形式对外发表。Unicode至今仍在不断增修，每个新版本都加入更多新的字符。目前最新的版本为2016年6月21日公布的9.0.0,已经收入超过十万个字符（第十万个字符在2005年获采纳）。Unicode涵盖的数据除了视觉上的字形、编码方法、标准的字符编码外，还包含了字符特性，如大小写字母。

Unicode发展由非营利机构统一码联盟负责，该机构致力于让Unicode方案替换既有的字符编码方案。因为既有的方案往往空间非常有限，亦不适用于多语环境。

Unicode备受认可，并广泛地应用于电脑软件的国际化与本地化过程。有很多新科技，如可扩展置标语言、Java编程语言以及现代的操作系统，都采用Unicode编码。

UTF-8 编码
UTF-8（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字符编码，也是一种前缀码。其编码中的第一个字节仍与ASCII兼容，这使得原来处理ASCII字符的软件无须或只须做少部分修改，即可继续使用。

C：字符与整型转换

```c
#include<stdio.h>
int main()
{
    /*'0' 字符0 对应的 ASCII编码：48
      给字符变量赋值48与赋值'0'是等价的
    */
    char c=48;
    char a='0';
    int w='a';
    char xing='张';//字符 张占两个字节，char型只有一个字节空间 保存不了一个汉字

    printf("%c %d\n",c,c );//字符变量c以%c展示就是字符0， 以%d展示就是数字编码
```

```
    printf("%c %d\n",97,'a');//打印%c：97对应的字符（a），  打印%d，  字符'a'对应的编码：97
    printf("%c %d\n",97-32,'a'-32);//'a'-32:小写变大写。   大写如变小写 +32
    printf("%d \n %c\n",'张',xing);
    return 0;
}
```

D: 汉字字符

由于汉字用两个字节或称双字节才能保存，因此char一个字节不能完整保存一个汉字。所以保存汉字用双字节字符类型——wchar_t类型。wchar_t称为宽字符类型，长度为两个字节，主要用在国际 Unicode 编码中

```
#include <stdio.h>
#include<locale.h>//设置本地化
int main ()
{
    wchar_t w = L'张';//L前缀表示字符常量用宽字符即2个字节存储。
    setlocale(LC_ALL, "chs");//设置本地语言：简体中文
    printf("\n%wc %d \n",w,sizeof(w));//用%wc,输出wchar_t类型数据
    return 0;
}
```

E: 转义字符

C语言中字符的表述方式可以分为五类：

1. 字符本身：如 'a' 'A' '0'
2. 十进制的编码 如：97 65 48
3. 八进制的编码如：'\141' '\101' '\60' 或'\060'
4. 十六进制编码如：'\x61' '\x41"x30'
5. 助记符号：

转义字符	意义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	换行(LF)，将当前位置移到下一行开头	010
\r	回车(CR)，将当前位置移到本行开头	013
\t	水平制表(HT)	009
\v	垂直制表(VT)	011
\'	单引号	039
\"	双引号	034
\\	反斜杠	092

3、浮点型相关

A：类型关键字

浮点型也可以称之为小数型，分为单精度float与双精度double。单精度占4个字节、双精度占8个字节。

实型	有	float	32	3.4e-38~3.4e38
	有	double	64	1.7e-308~1.7e308

B：float浮点数据存储形态

一个float型实数在内存中占4个字节，即32个二进制bit，从低位到高位依次叫第0位到第31位。这32位可以分为3个部分：符号位（第31位），阶码（第30位到第23位共8位），尾数（最低23位）。

1、符号位。最高位也就是第31位表示这个实数是正数还是负数,为0表示正数或0,为1表示负数.

2、阶码。第30位到第23位这8个二进制位表示该实数转化为规格化的二进制实数后的指数与127(127即所谓偏移量)之和即所谓阶码.

规格化的二进制实数的指数只能在-127----+127之间,所以,一个float型数的最大值在+2^127即+3.4*10^38,最小值在-2^127即-3.4*10^38.

3、尾数。其他最低的23位即第22位到第0位表示该实数转化为规格化的二进制实数后小数点以后的其余各位即所谓尾数.

例如,将十进制178.125表示成机器内的32个字节的二进制形式.

第一步:将128.125表示成二进制数:(178.125)(十进制数)=(10110010.001)(二进制形式);

第二步:将二进制形式的浮点实数转化为规格化的形式:(小数点向左移动7个二进制位可以得到)

10110010.001=1.0110010001*2^7 因而产生了以下三项:

符号位：该数为正数,故第31位为0,占一个二进制位.

阶码：指数为7,故其阶码为127+7=134=(10000110)(二进制),占从第30到第23共8个二进制位.

尾数为小数点后的部分， 即0110010001.因为尾数共23个二进制位,在后面补13个0,即01100100010000000000000

所以,178.125在内存中的实际表示方式为:

0 10000110 01100100010000000000000

再如,将-0.15625表示成机器内的32个字节的形式.

第一步:将-0.15625表示成二进制形式: (-0.15625)(十进制数)=(-0.00101)(二进制形式);

第二步:将二进制形式的浮点数转化为规格化的形式:(小数点向右移动3个二进制位可以得到)

-0.00101=-1.01*2^(-3) 同样,产生了三项:

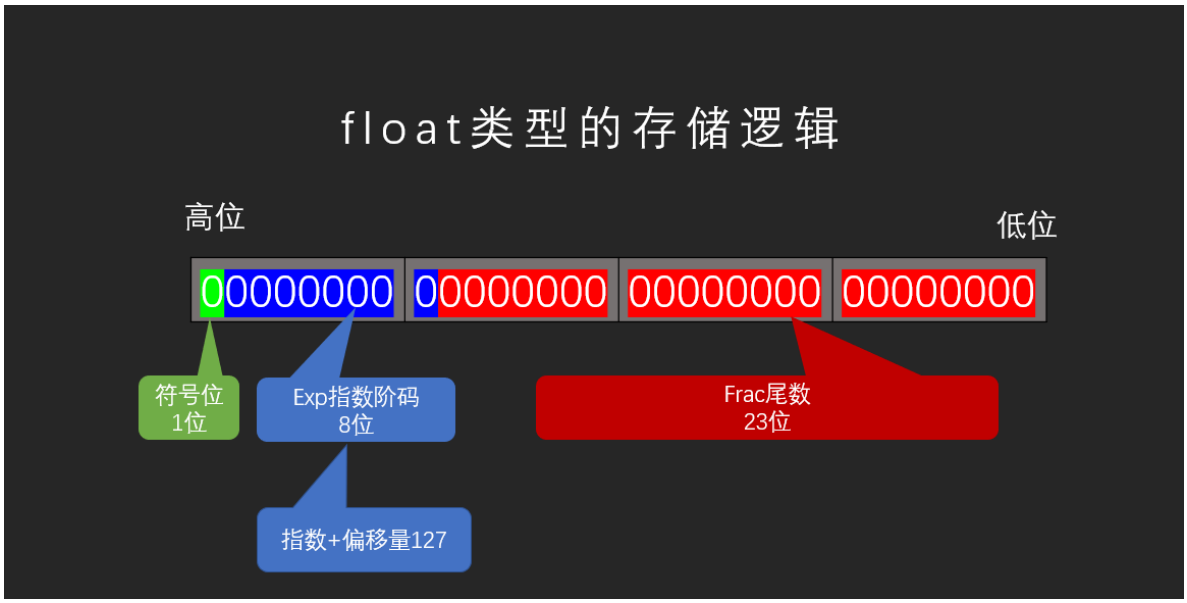
符号位：该数为负数,故第31位为1,占一个二进制位;

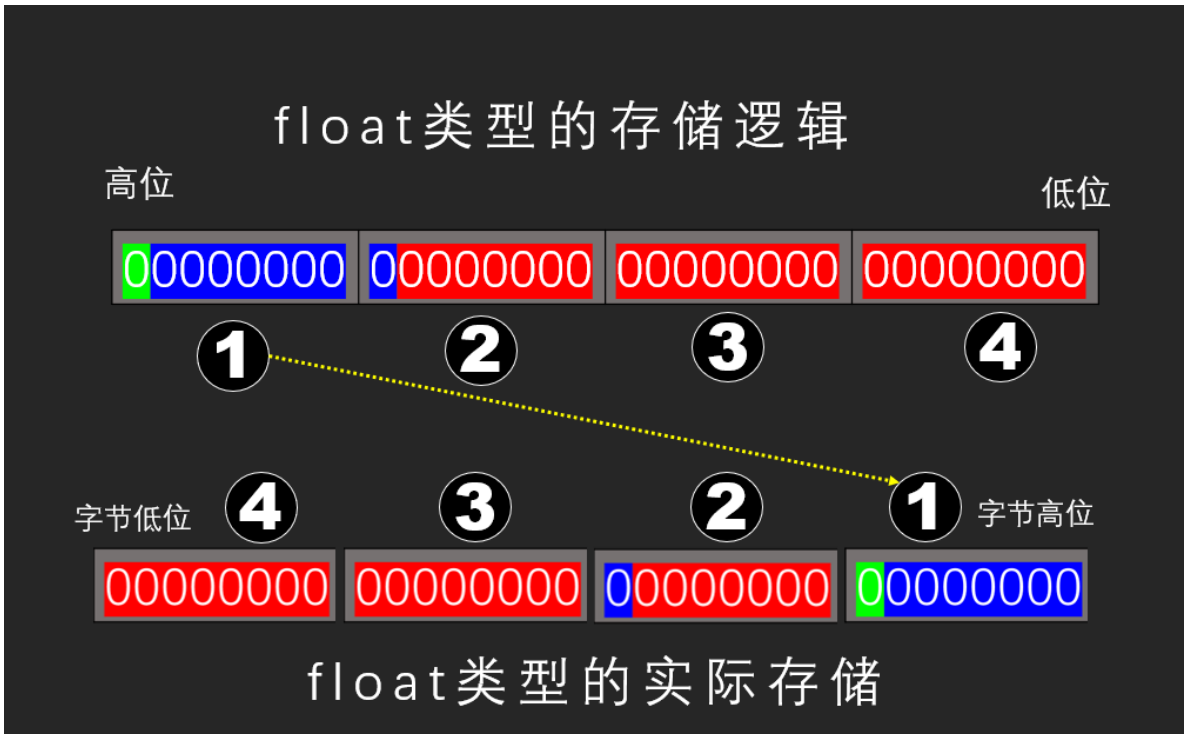
阶码：指数为-3,故其阶码为127+(-3)=124=01111100,占从第30到第23共8个二进制位;

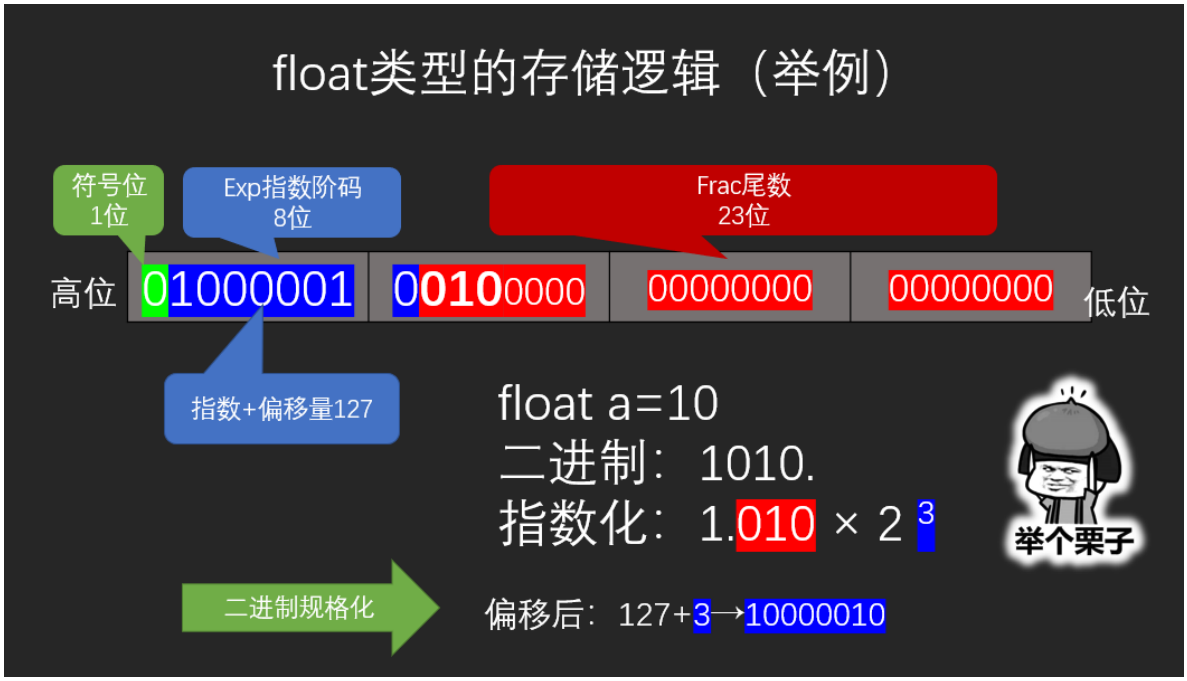
尾数为小数点后的01,当然后面要补21个0;

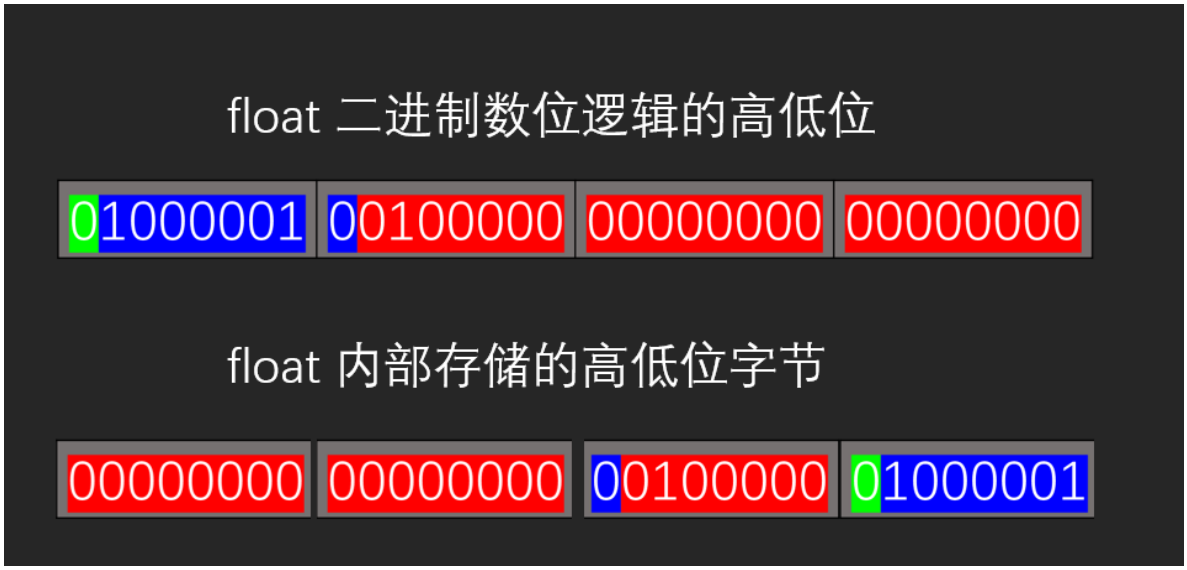
所以,-0.15625在内存中的实际表示形式为:

```
1 01111100 01000000000000000000000
```



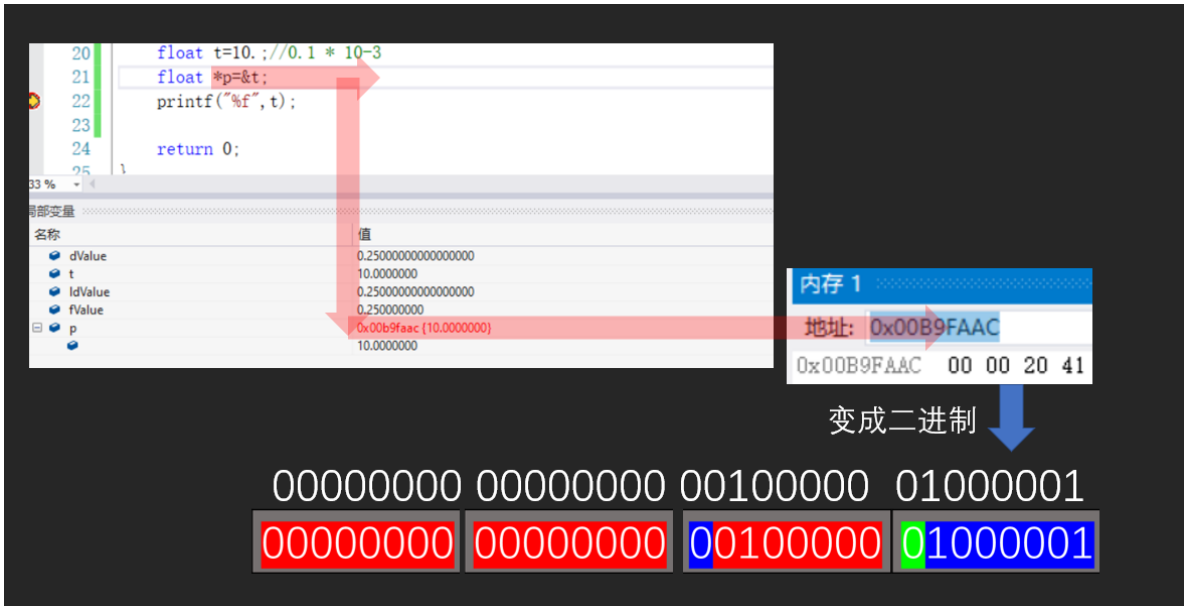






测试一下float 型数据10.0的真实计算机存储

```
float t=10.;
float *p=&t;//取 变量t的内存地址
```



	符号域	指数域	小数域
单精度浮点数	1 位[31]	8位[30-23]	23位[22-00]
双精度浮点数	1 位[63]	11 位[62-52]	52 位[51-00]

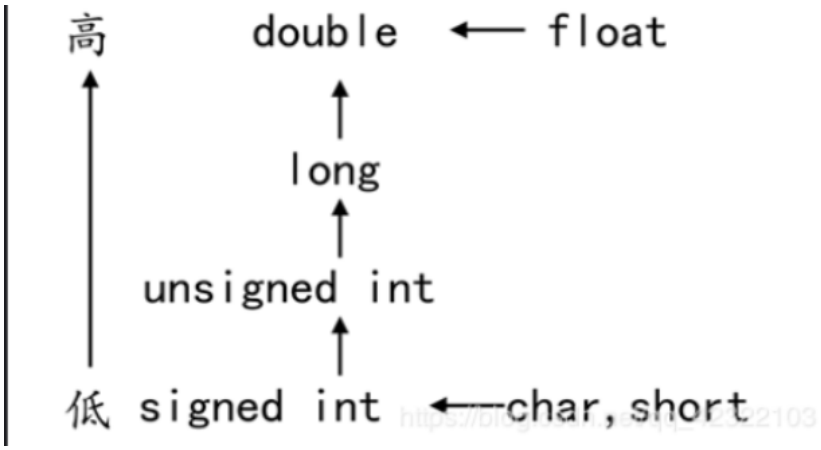
```
float有效数字位为6 – 7位，字节数为4，指数长度为8位，小数长度为23位。取值范围为 3.4E-38～3.4E+38。

double有效数字位为15 – 16位，字节数为8，指数长度为11位，小数长度为52位。取值范围为1.7E-308～1.7E+308。
```

C: 浮点型字面值表示方式：

```
    //地球距离太阳的距离 ：  149597870千米
    double distanceA;
    double distanceB;
    //小数形式：
    distanceA = 149597870.0;//.后面的0 可以省略
    //指数形式：
    distanceB = 1.49597870e8;//8代表10的8次方
    printf("指数形式打印：  %e,%e\n",distanceA,distanceB);//默认6位尾数
    printf("指数形式打印：  %.8e,%.8e\n",distanceA,distanceB);//制定8位尾数
    printf("小数形式打印：  %f,%f\n",distanceA,distanceB);
    printf("相对短形式打印：  %g \n",distanceA );//小数与指数哪个短，显示哪个。
    distanceA = 3.14 ;
    printf("相对短形式打印：  %g \n",distanceA );//小数与指数哪个短，显示哪个。
```

4、不同数据类型之间的转换

A：自动类型转换（隐式方式）



转换的规律：

①整型存储形态向浮点型存储形态转换

②同一存储形态：字节少的向字节多的转换

③同一类型：有符号向无符号的转换

```
//整型类型级别从低到高依次为：
int -> unsigned int -> long -> unsigned long -> long long -> unsigned long long
```

```
//浮点型级别从低到高依次为：
float -> double
```

B：强制类型转换（显示方式）

```
int  a = 3.14;//自动类型转换
int b =(int) 3.14;//强制类型转换
```

强制类型转换的写法： **（类型名称）表达式**

由于( )的运算级别最高。因此强制类型转换的计算顺序第一。

三、 C语言的计算表达式

1、 基本概念：

① 运算符：可计算出结果的计算符号如：+ - > <.....

②操作数：运算时需要的数据包括：字面值、变量、其它表达式等...。根据运算符所需的操作数个数可分为：单目，双目、三目运算。

③表达式：运算符配上操作数形成完整可计算的数学式子

④结合性：表达式中不同运算符的计算方向：从左向右算如：+ - * / ，从右向左算： = 赋值运算。

⑤优先级：复杂表达式中，运算符号不止一个，是由很多不同计算符号组成，计算机在计算时按优先级不同进行分步骤计算。例如：算术运算中，先算乘除后算加减.

【！！！牢记】：只要是表达式，必须返回一个计算结果。因此对于不同的表达式，必须能够手动演算其计算结果。

2、运算符总表：

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	左到右	--
	()	圆括号	(表达式）/函数名(形参表)		--
	.	成员选择（对象）	对象.成员名		--
	->	成员选择（指针）	对象指针->成员名		--
2	-	负号运算符	-表达式	右到左	单目运算符
	~	按位取反运算符	~表达式		
	++	自增运算符	++变量名/变量名++		
	--	自减运算符	--变量名/变量名--		
	*	取值运算符	*指针变量		
	&	取地址运算符	&变量名		
	!	逻辑非运算符	!表达式		
	(类型)	强制类型转换	(数据类型)表达式		--
	sizeof	长度运算符	sizeof(表达式)		--
3	/	除	表达式/表达式	左到右	双目运算符
	*	乘	表达式*表达式		
	%	余数（取模）	整型表达式%整型表达式		
4	+	加	表达式+表达式	左到右	双目运算符
	-	减	表达式-表达式		
5	<<	左移	变量<<表达式	左到右	双目运算符
	>>	右移	变量>>表达式		
6	>	大于	表达式>表达式	左到右	双目运算符
	>=	大于等于	表达式>=表达式		
	<	小于	表达式<表达式		
	<=	小于等于	表达式<=表达式		
7	==	等于	表达式==表达式	左到右	双目运算符
	! =	不等于	表达式!= 表达式		
8	&	按位与	表达式&表达式	左到右	双目运算符
9	^	按位异或	表达式^表达式	左到右	双目运算符
10	|	按位或	表达式|表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12	||	逻辑或	表达式||表达式	左到右	双目运算符
13	?:	条件运算符	表达式 1? 表达式 2: 表达式 3	右到左	三目运算符
	=	赋值运算符	变量=表达式		--
	/=	除后赋值	变量/=表达式		--
	*=	乘后赋值	变量*=表达式		--
	%=	取模后赋值	变量%=表达式		--

14	+=	加后赋值	变量+=表达式	右到左	--
	-=	减后赋值	变量-=表达式		--
	<<=	左移后赋值	变量<<=表达式		--
	>>=	右移后赋值	变量>>=表达式		--
	&=	按位与后赋值	变量&=表达式		--
	^=	按位异或后赋值	变量^=表达式		--
	|=	按位或后赋值	变量|=表达式		--
15	,	逗号运算符	表达式,表达式,...	左到右	--

3、运算符特殊特性

A: 算术运算

```
//1. + - 两个操作数时为加法
printf("%d",3+5);//打印 8
//2. + - 一个操作数时为+（正号） - 负号或相反数。
printf("%d",-3);
printf("%d",-(-3));
printf("%d",-(-(-3)));//从右向左算
//3. () 优先级最高的计算符号。
```

B: 赋值运算

①：= 赋值号也有返回值。返回结果为左值（左侧变量存储空间的数据）。

因此可以有如下代码：

```
int a,b,c,d;
a=b=c=d=1;
```

它的计算顺序：

1. d=1//把字面值1赋给左侧变量d，并返回d=1这个赋值表达式的值 1（即d变量的存储值）
2. c=d// 把d的存储值再赋值给c， 如下同理
3. b=c
4. a=b

②：复合赋值如：+= -= ...的计算过程（以+=为例）

```
int a = 10;
a+=5;//a变量自增5
// += 执行两个运算 1 +运算 2 = 赋值
//即：先增加后保存，简称自增
```

③： ++ -- 的计算过程（以++为例）

```c
int a = 10;
int b = 10;
int m,n;
m=a++;
n=++b;
printf("a=%d b=%d m=%d n=%d",a,b,m,n);
//打印结果：a=11 b=11 m=10 n=11
```

m=a++运算：包括两个步骤：第一步：m=a 第二步 a=a+1。

因此m得到a的初始值10，然后自增1变成11

n=++b 运算：包括两个步骤：第一步：b=b+1 第二步 n=b。

因此b自增1，从初始值10变为了11。再把11赋值给n

++表达式的计算结果的总结：

a++: a在前 ++在后 则：**先给后加**

++a : ++在前，a在后 则：**先加后给**

-- *运算亦如此*

C：关系运算

【注意】C语言没有boolean类型，因此 关系运算、逻辑运算返回值为0或1。0代表假 1 代表真

```c
printf("%d\n",3>5);//计算结果 0
printf("%d\n",3<5);//计算结果 1
```

D：逻辑运算

① 逻辑运算:

1. **！逻辑非**：从右向左计算，取相反逻辑即 0 相反是1，1相反是0
2. **&& 逻辑与**：与（并且之意） 1 && 1 两侧都是1（即真）则结果为 1真， 有一侧0则结果为0 假
3. **|| 逻辑或**： 或（或者之意）0 || 1 两侧都为0 则结果为0 有一侧为1 结果就为1 即真

   逻辑运算符的优先级：！优于 && 优于 || 。

   【注意】： && 和 || 并非同级别运算符。

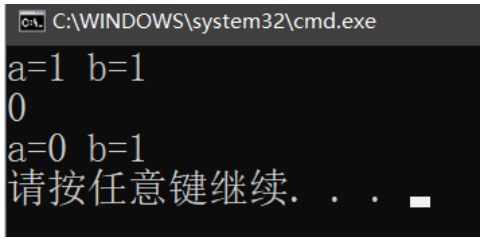
   【逻辑运算、关系运算混合时的优先级：】

   ！ 优于 关系运算(> >= < <= == !=) 优于&& 优于 ||

② && 和 || 的“短路”问题

```
    int a,b;
    b=a=1;
    printf("a=%d b=%d\n",a,b);
    printf("%d\n",--a&&++b);
    printf("a=%d b=%d\n",a,b);
```

打印结果：



根据结果发现：--a执行后，a自减变为了0。 但 ++b并没有执行。因为b还是初始值1。

【原因】：&& 在执行时，发现左侧为0假，右侧就不进行判断执行了。因此++b表达式并没有被执行。

【同理】：|| 在执行时，发现左侧为1真，右侧也不进行任何执行操作。

E：条件运算符

C语言中唯一三目运算符

【格式】： 关系逻辑表达式？真返回的值 ：假返回值

例如：

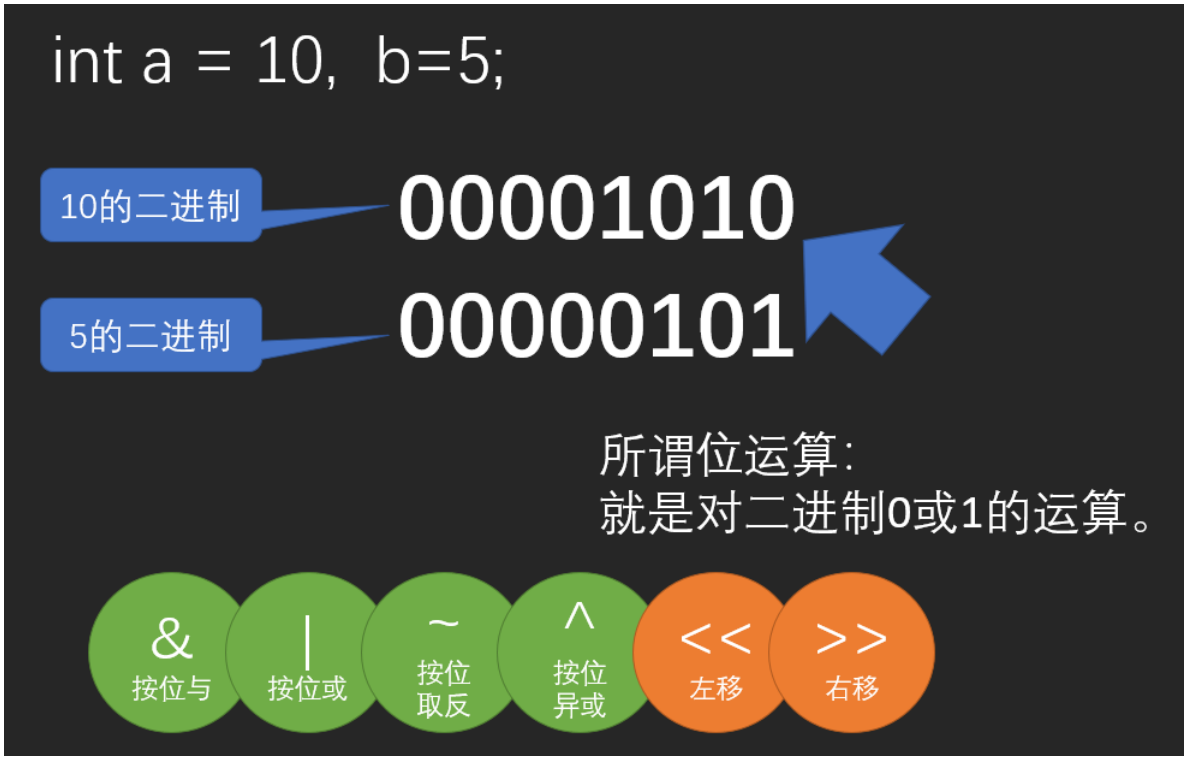
```
    int a=10;
    char r ;
    r= a%2==0?  'T':'F';//如果 a是2的倍数，返回T 否则返回F
    printf("%c\n",r);
    printf("%c\n", r>='A'&&r<='Z' ? r+32:r);//r切换成小写字母
```

条件运算符的嵌套：

```
    int age=32;
    int sex=1;
    printf("%c\n", sex?  age>=22?'Y':'N'  : age>=20? 'Y' :'N');
```

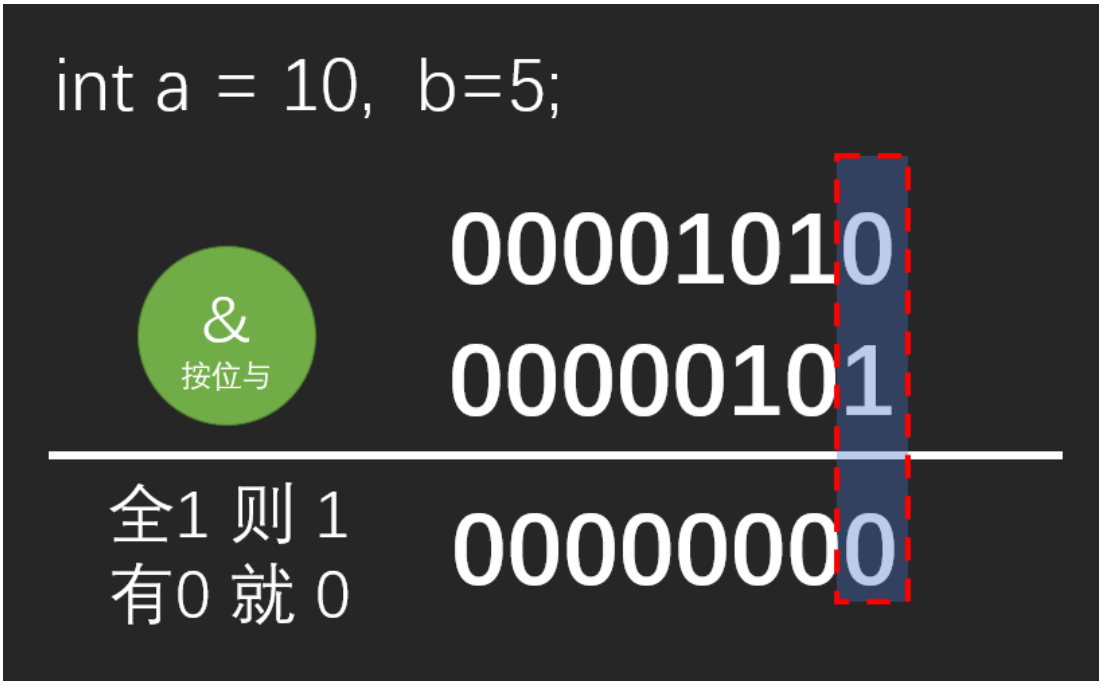
F：位运算符

①：位运算的基本概念

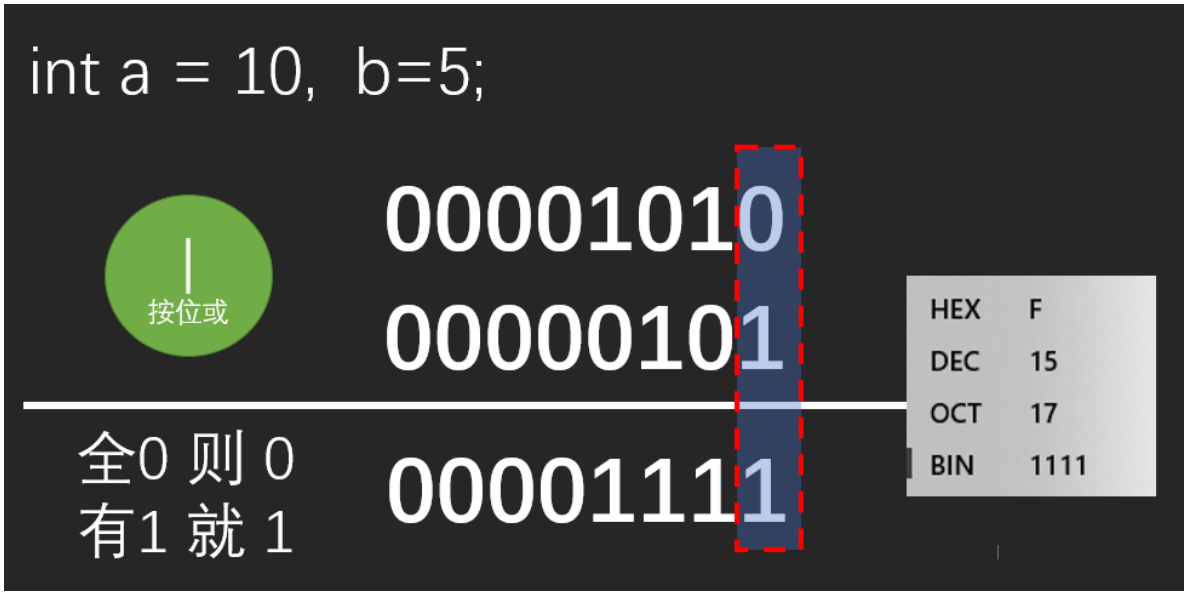


②：典型位运算符

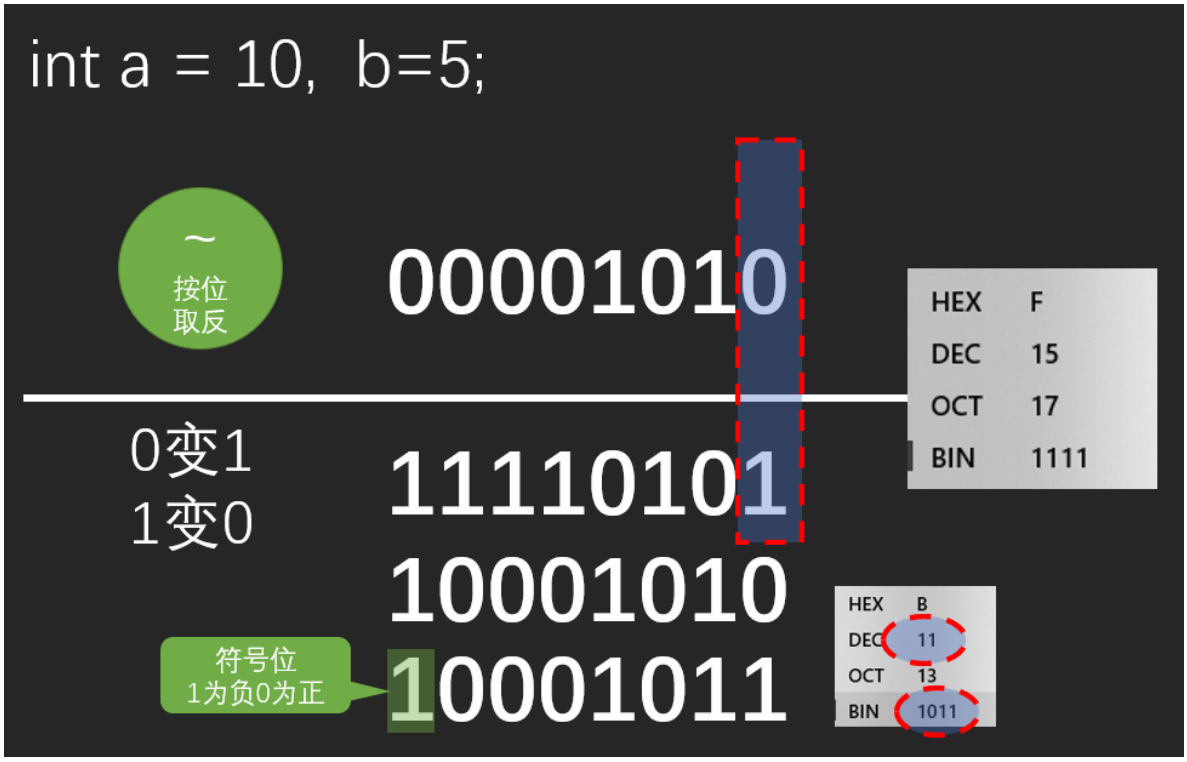
```
    int a=10;
    int b=5;
    printf("%d\n",a&b);//结果为0
```



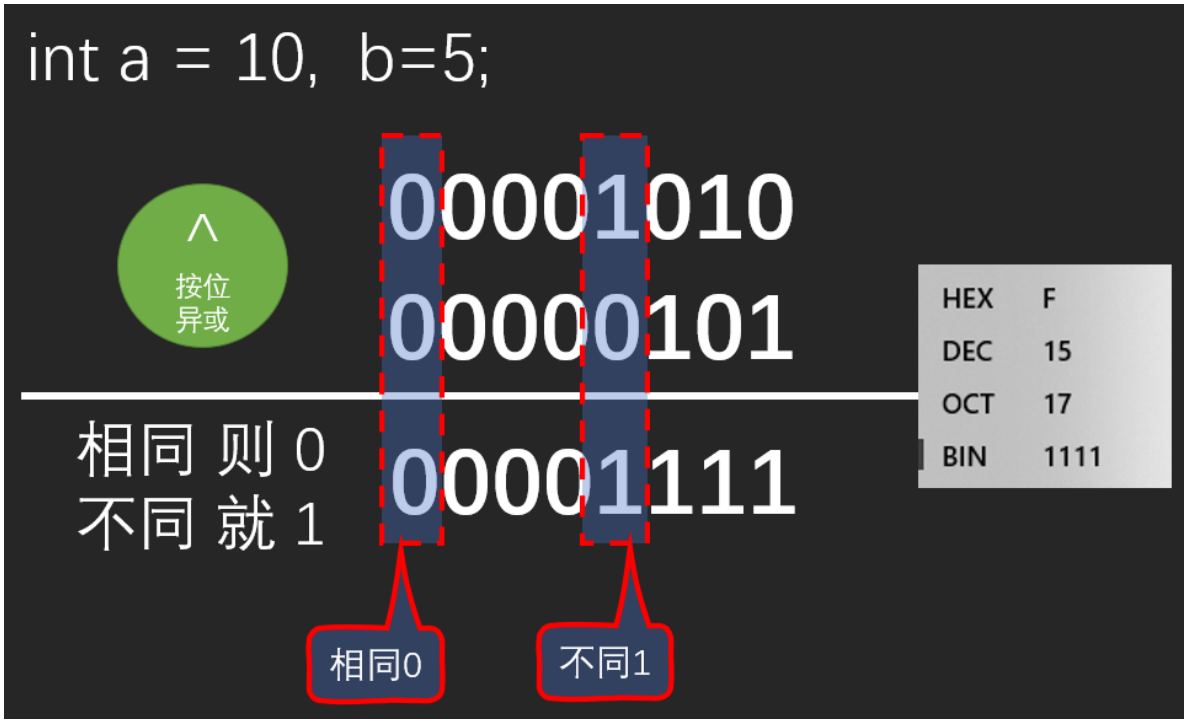

```
    int a=10;
    int b=5;
    printf("%d\n",a|b);//结果 15
```




```c
    int a=10;
    printf("%d\n",~a );//结果为-11
```

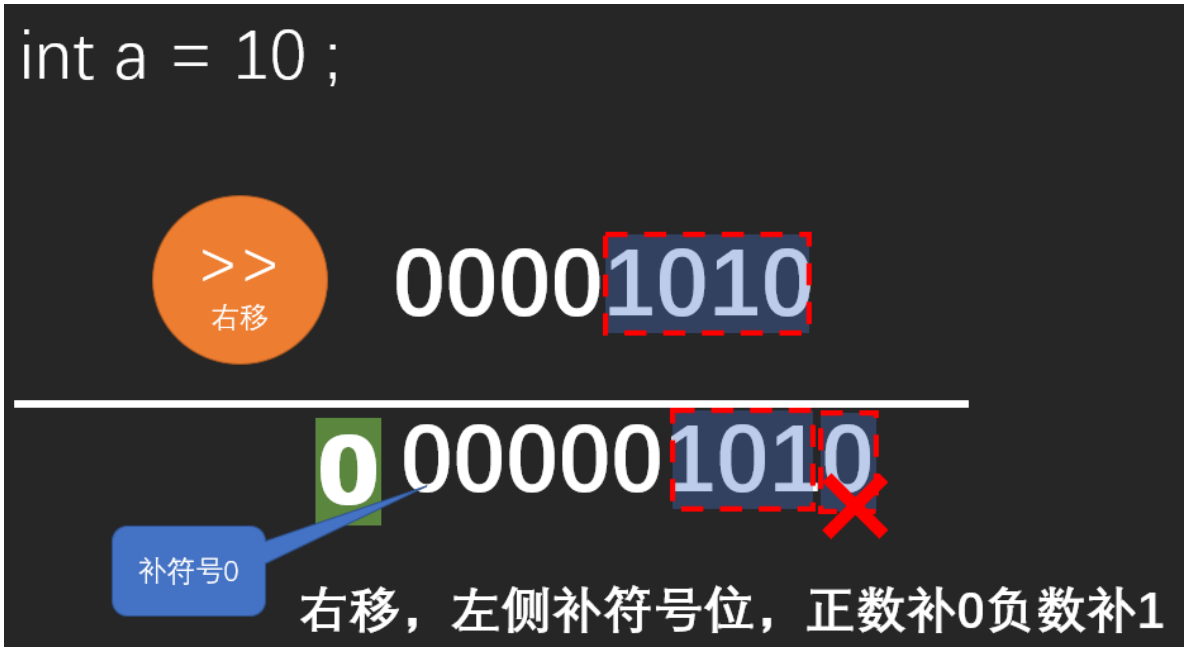



```c
    int a=10;
    int b=5;
    printf("%d\n",a^b );//结果15
```




```c
int a=10;
printf("%d\n",a<<1 );//结果是 20
```


```c
int a= 10;
printf("%d\n",a>>1 );//5
a= -10;
printf("%d\n",a>>1 );//-5
```



③ 位运算的运用demo：

1. **交换**：两个变量的值

```
不允许你使用额外的辅助变量来完成交换
两个相同的数异或之后结果会等于 0，即 n ^ n = 0。并且任何数与 0 异或等于它本身，即 n ^ 0 = n。
```

```
 int x = 10,y = 20;
 x = x^y;
 y = x^y;
 x = x^y;
printf("%d %d\n",x,y);
```

2. **判断奇偶数**

```
int x = 10;
printf("%s\n", x&2? "偶数":"奇数");
```

3. **计算x的绝对值**

```
int y ;
int x= -15;
y = x >> 31 ;
printf("%d\n", (x^y)-y);
```

G：逗号运算符

```c
int a,b,c;
int m;
/*
 逗号运算符的优先级别最低，()里用","连接了4个表达式。
 整个（）里称为一个，号表达式。逗号表达式返回值为最后一个表达式的计算结果，即m=12。
*/
m=(a=3,b=4,c=5,a+b+c);
printf("a=%d,b=%d,c=%d,m=%d",a,b,c,m);//打印结果：a=3,b=4,c=5,m=12
```

H：sizeof运算符

sizeof是C语言中保留关键字，也可以认为是一种运算符，单目运算符。用于取得一个变量或一个数据类型所占内存空间的字节数。

```c
int a=10;
int s[5] ;
char *p="abcde";
char str[]="abcde";
printf("字面值 10的字节数 %d\n",sizeof(10));//结果：4 解释：10为int型，占4个字节
printf("字面值 10LL的字节数 %d\n",sizeof(10LL));//结果：8解释：10LL为longlong类型，占8给字节
printf("double类型的字节数 %d\n",sizeof(double));//结果：8解释：类型名做参数，double占8个字节
printf("长度为5的int型数组的字节数 %d\n",sizeof(s));//结果：20解释：s为int型数组 一个int变量4个字节 5个是20字节
printf("字符指针p的字节数 %d\n",sizeof(p));//结果：4解释：p为指针变量，32位系统自然p保留地址长度4个字节
printf("字符数组str的字节数 %d\n",sizeof(str));//结果：6解释：字符数组，根据字符串常量统计长度，一个char型一个字节，abcde\0（\0为字符串结束标记，也占一个字节）共6个字节
```

I：内存地址运算符

& 为单目运算时，称为取地址运算符

* 为单目运算时，称为地址空间内操作运算符，空间内操作包括：取值、赋值。 &与*为互逆运算。

```c
int a = 0 ;
int *p = &a;//& 在单目（即一个操作数时）为取得某变量内存地址的作用
scanf("%d",&a);//扫码键盘，从输入缓冲中取得一个整数，存入到 a的地址空间中。
printf("%d\n",a);//打印，验证 输入的值是否正确
```

```c
int a =0;
int *p = &a;//计算a变量的地址并赋给指针p。
*p=1;//利用指针p对地址空间赋值1
printf("%d\n",a);//打印：a变量的值查看是否改变
printf("%d\n",*p);//*为单目运算符时，其功能是取得p指向地址里的int值
```