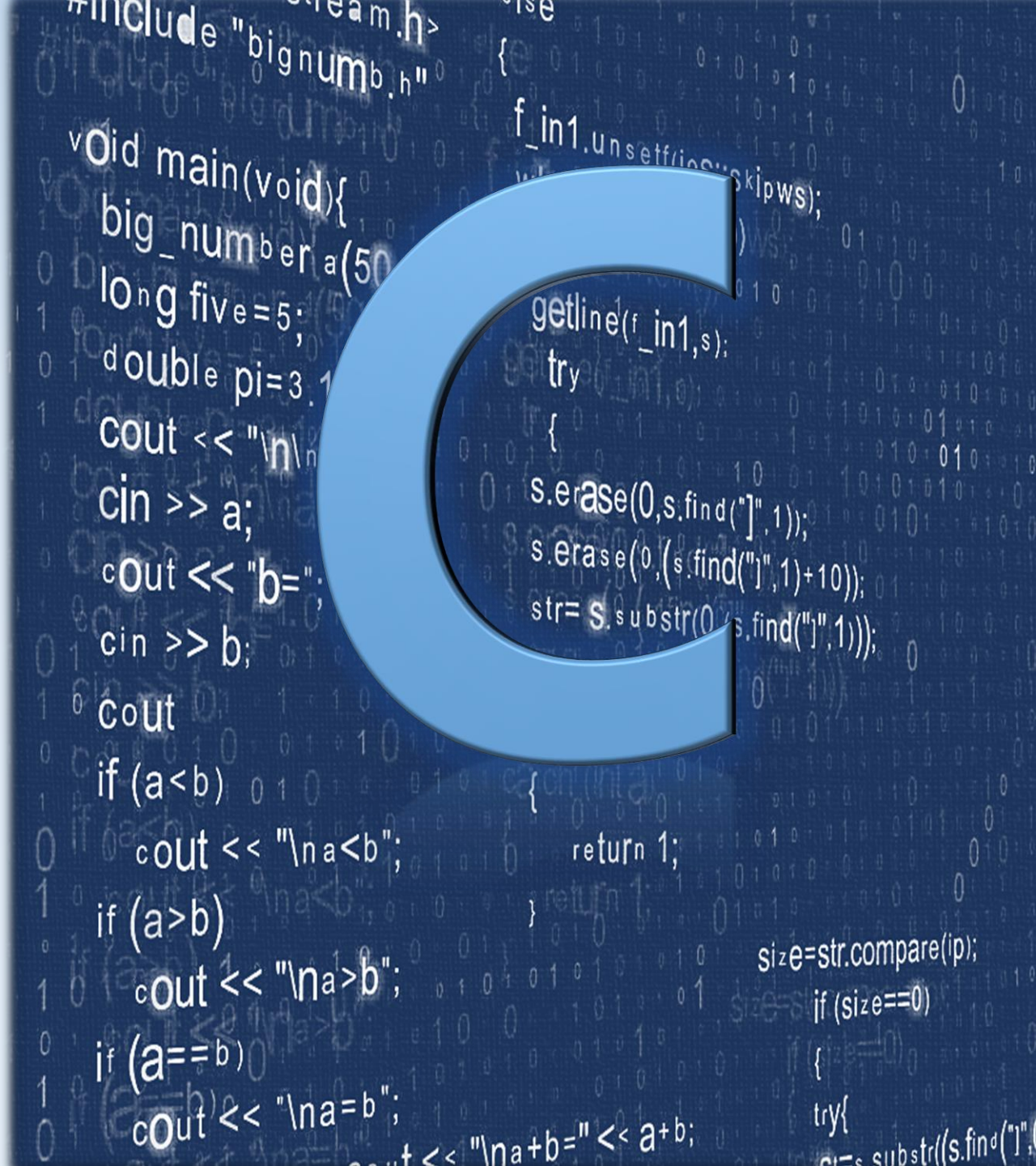


# 《C语言程序设计》

丁盟  
C语言课程组



# 上一讲知识复习

- ◆掌握C语言中的数据类型及区别
- ◆掌握定义变量的方法
- ◆掌握命名规则
- ◆掌握不同类型字面值的写法。
- ◆掌握输入与输出的方式

# 本讲教学目标

- ◆理解左值及右值。
- ◆掌握运算符的种类、重点掌握运算符优先级。
- ◆熟悉各种运算符的功能及相关表达式的求值方法。
- ◆了解sizeof运算符。
- ◆了解表达式副作用。
- ◆掌握显式类型转换的方法，了解隐式转换。
- ◆掌握溢出的计算方法，了解在什么情况下可能会造成溢出。

# 本章授课内容



# C语言的“单词”

## ❖ 标准定义了多种类型的“单词”：

- 关键词(**keywords**)：return、for、void、...
- 标识符(**identifiers**)：x、y、z、...
- 字面值(**literal**)：3、4、"Hello World!"、...
- 注释(**comment**)：/\*This is a comment.\* /、...
- 操作符(**operator**)：+、-、\*、/、...
- 分隔符(**separator**)：{、}、...
- ...

# C语法的基本概念总结

- ❖ C语言的数据表示：变量，常量，字面值等。
- ❖ C语言的运算符：加，减，乘，除，取余等。
- ❖ 由**单个或多个**操作数、运算符组成的符合C语言规则的式子叫做**表达式**。
- ❖ 表达式的值就是经过计算以后所得到的结果。
- ❖ C语言中数据表示的分类：
  - 左值：**可以写的内存块儿表示**，变量
  - 右值：**只可以读的内存块儿表示**，常量，字面值



# C语言中的运算符(I)

❖ C语言运算符可分为以下几类：

- ◆ 算数运算符：加(+)、减(-)、乘(\*)、除(/)、求余或称模运算(%)、自增(++)、自减(--)
- ◆ 关系运算符：包括大于(>)、小于(<)、等于(==)、大于等于(>=)、小于等于(<=)、不等于(!=)
- ◆ 逻辑运算符：与(&&)、或(||)、非(!)
- ◆ 赋值运算符：
  - 简单赋值(=)
  - 复合算术赋值(+=, -=, \*=, /=, %=)
  - 复合位运算赋值(&=, |=, ^=, >>=, <<=)

# C语言中的运算符(II)

- ◆ 条件运算符：(?:).这是唯一——一个三目运算符。
- ◆ 位运算符：与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)
- ◆ 逗号运算符：( , )
- ◆ 指针运算符：(\*)和(&)。
- ◆ 求字节数运算符：(sizeof)。
- ◆ 特殊运算符：括号(), 下标[], 成员(→, .)



# C语言中运算符的优先级 (I)

优先级	运算符	结合律	类别	备注
1(最低)	,	从左到右	二元	顺序求值运算符 (逗号运算符)
2	=、+=、-=、*=、/=、<<=、>>=、 %=、^=、&=、 =	从右到左	二元	赋值运算符
3	?:	从右到左	三元	条件运算符
4		从左到右	二元	逻辑或运算符
5	&&	从左到右	二元	逻辑与运算符
6		从左到右	二元	按位或运算符
7	^	从左到右	二元	按位异或运算符
8	&	从左到右	二元	按位与运算符
9	==、!=	从左到右	二元	判定相等/不等的运算符
10	<、>、<=、>=	从左到右	二元	小于/大于/小于等于/大于等于
11	<<、>>	从左到右	二元	左移/右移运算符
12	+、-	从左到右	二元	加法/减法运算符
13	*、/、%	从左到右	二元	乘法/除法/求余运算符
14	(数据类型)	从右到左	一元	类型转换运算符

# C语言中运算符的优先级 (II)


15	*	从右到左	一元	间接访问运算符
	&	从右到左	一元	取地址运算符
	-, +	从右到左	一元	算术负/正运算符
	!	从右到左	一元	逻辑非运算符
	~	从右到左	一元	按位取反运算符
	sizeof	从右到左	一元	取长度运算符
	++, --	从右到左	前缀	自增/自减运算符
16(最高)	(数据类型){数值字面值}	从左到右	后缀	复合字面值
	++, --	从左到右	后缀	自增/自减运算符
	->	从左到右	后缀	间接选择运算符
	.	从左到右	后缀	直接选择运算符
	f(...)	从左到右	后缀	函数调用
	x[i]	从左到右	后缀	数组下标
	标识符、字面值	无	初等	简单记号

# C语言中的运算符

❖学习一个运算符一定要解决的问题：

- ◆ 优先级
- ◆ 结合性
- ◆ 几元运算符
- ◆ 运算规则
- ◆ 结果是什么
  - 变量
  - 字面值
- ◆ 特殊规则

# 本章授课内容



运算符的基本规则

算术运算符



赋值运算符

关系运算符

逻辑运算符

# 算数运算符

- ❖ 算数运算符二元运算符，即应有两个操作数参与运算，其相应的表达式形式为：
  - ◆ 操作数1 + 操作数2。
  - ◆ 操作数1 - 操作数2
  - ◆ 操作数1 \* 操作数2
  - ◆ 操作数1 / 操作数2
  - ◆ 操作数1 % 操作数2
- ❖ 运算规则同数学运算
- ❖ 运算结果是“常量字面值”，右值

**注意：对于%来说，结果的符号同被取余数相同而且两个操作数必须为整数.**

# 算术运算符

- ❖ 设有 `int x = 4, y;` 请分析语句 `y = x*5+4;` 执行完后, `y` 的值是多少?
- ❖ 设有 `int x = 4, z = 5, y;` 请分析语句 `y = z + x*(9 - z);` 执行完后, `y` 的值是多少?
- ❖ 设有 `int x = 4, y = 3, z;` 请分析语句 `z = x%y;` 执行完后, `z` 的值是多少?
- ❖ 设有 `int x = 4, y = 3; double z1, z2;` 请分析语句 `z1 = x / y; , z2 = x % y;` 执行完后, `z1` 和 `z2` 的值是多少?



# 算数运算符

❖ 自增运算符(**++**)：一元运算符，有两种：

自增运算符	前++	后++
表达式形式	++操作数	操作数++
运算元	一元运算符	
操作数	必须为左值	
运算规则	1、当前变量自动加1 2、然后参与当前表达式运算	1、先计算当前表达式的值 2、当前表达式计算完毕后， 变量自动加1
运算结果	变量	常量字面值

例 设有int x = 3, y; , 请分析语句  
y = ++x;执行后，x、y 的值？

例 设有int x = 3, y = 4, z; , 请分  
析语句z = x++ + y;执行后，x、y、  
z 的值？

# 算术运算符

❖ 自减运算符(--): 一元运算符，操作形式有两种：

自增运算符	前--	后--
表达式形式	--操作数	操作数--
运算元	一元运算符	
操作数	必须为左值	
运算规则	1、当前变量自动减1 2、然后参与当前表达式运算	1、先计算当前表达式的值 2、当前表达式计算完毕后， 变量自动减1
运算结果	变量	常量字面值

例 设有int x = 3, y = 4, z; , 请分析语句z = x++ + --y;执行后，x、y、z 的值？

# 算数运算符

## ❖总结：

- ◆ ++、--运算符的操作数**必须是左值**
- ◆ ++运算符和运算符++的使用规范（同理--）
- ◆ ++、--运算符的副作用
  - `a=c+++b+++c++;` → a?
  - `c=(i++)+(i++);` → c ?
  - `printf( "%d\t%d\n" ,i,i++);` → 结果？

## 注意：

- 1.尽量分多行写.
- 2.尽量使用多使用（ ）.
- 3.尽量不要再一行语句中对一个变量多次使用自增自减.

# 运算符和表达式总结

❖求代码结果。

```
#include <stdio.h>

int main(void)
{
    int x = 3;
    int a;

    a = ++x + ++x + ++x;
    printf("a = %d", a);

    return 0;
}
```

# 本章授课内容



# 赋值运算符

❖ 简单赋值 ( = )，二元运算符，结合性从右到左

◆ 不是判断是否相等

◆ 操作数1 = 操作数2

◆ 运算规则

- 把操作数2的值取出
- 修改操作数1的值为操作数2的值

◆ 运算结果，左值，变量

- 操作数1



# 赋值运算符

❖ 复合赋值运算符（`op=`），二元运算符

◆ 操作数1 `op=` 操作数2

◆ 运算规则

- 操作数1 = 操作数1 `op` 操作数2

◆ 运算结果

- 操作数1

❖ 例：int x=3，y=4，请分析`x += y`；执行后，求x，y？

# 本章授课内容



# 关系、判等运算符


## ❖ 关系运算符：

- ◆ 大于(>)、小于(<)、等于(==)、大于等于(>=)、小于等于(<=)、不等于(!=)
- ◆ 一般形式：操作数1 运算符 操作数2
- ◆ 运算规则：同数学运算规则
- ◆ 运算结果
  - 0 or 1

# 关系、判等运算符

- ❖ 设有 `int x = 3, y = 2, z;` , 请分析 `z = x > y;` 执行后, `z` 的值?
- ❖ 设有 `int x = 3, y;` , 请分析 `y = 2 < x >= 8;` 执行后, `y` 的值?
- ❖ 设有 `int x = 3, y = 3, z = 5;` , 请分析表达式: `x == y`、`x != y`、`y != z`、`x == 3`、`x != 4` 的计算结果?
- ❖ 设有 `int x = 3, y = 3, z;` , 请分析语句 `z = x == y != 4;` 执行后, `z` 的值?

# 本章授课内容



运算符的基本规则

算术运算符

赋值运算符

关系运算符

逻辑运算符



# 逻辑运算符

❖ 逻辑运算符：与(&&)、或(||)、非(!)

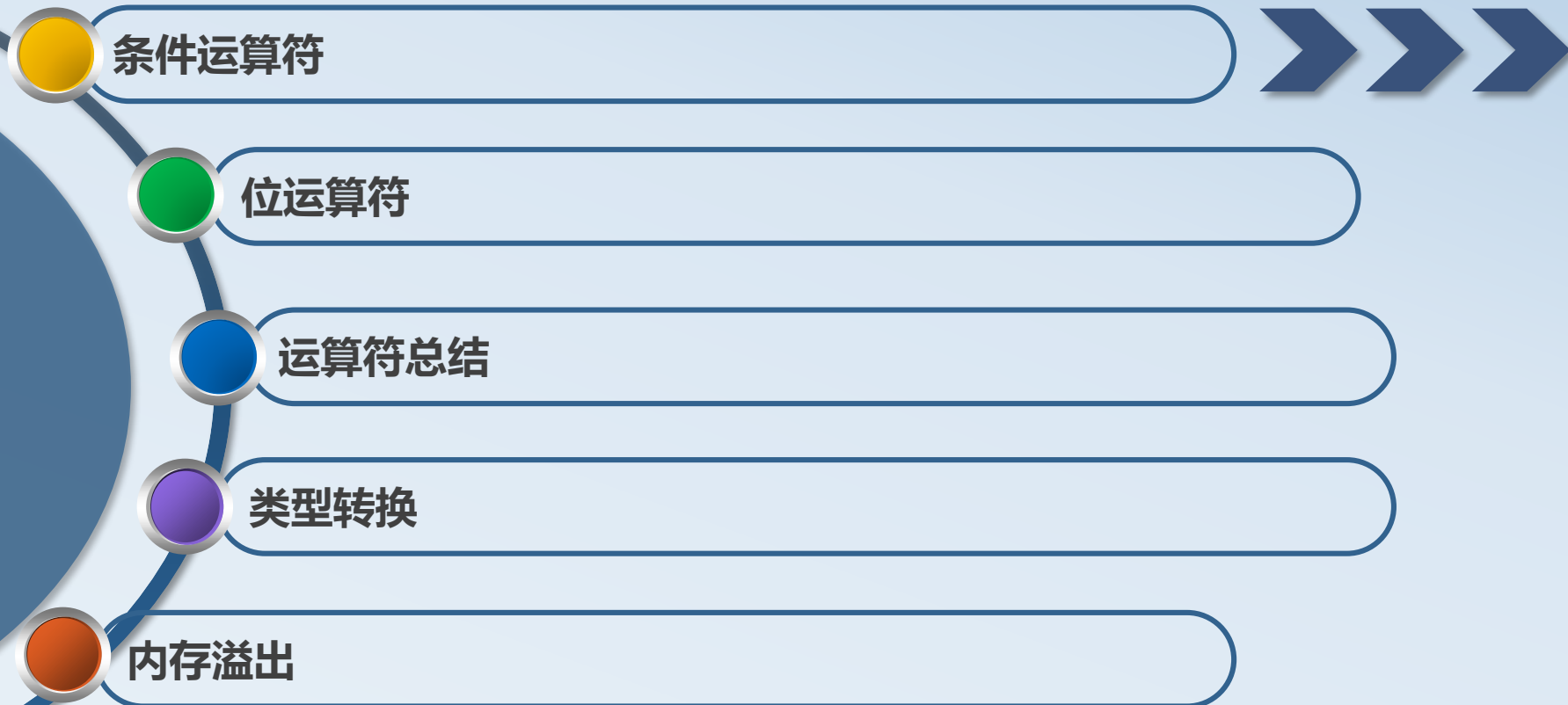
逻辑运算符	非运算	或运算	与运算
形式	!操作数	操作数1  操作数2	操作数1&&操作数2
运算元	一元运算符	二元运算符	
运算规则	真变假 假变真	有一个为真则结果为真 否则为假	有一个为假则结果为假 否则为真
运算结果	0 or 1		
特殊规则 短路规则		第一个操作数为真则 第二个操作数不运算	第一个操作数为假则 第二个操作数不运算



# 逻辑运算符

- ❖ 设有 `int x = 3, y = 0, z;` , 请分析表达式 `x && y`、`x || y`、`!x`、`!y` 的结果？
- ❖ 设有 `int a = 3, b = 2, c = 1, d = 5, e = 6, f;` , 请分析表达式 `f = a < b || b < c && c < d || d < e` 的结果？

# 本章授课内容



# 条件运算符

## ❖ 条件运算符 (?:), 三元运算符

◆ 一般形式：操作数1 ? 操作数2 : 操作数3

◆ 运算过程：

1. 对操作数1求值。
2. 若操作数1的结果非0，则对操作数2求值（**不需要**对操作数3进行求值），并将该值作为条件表达式的最终结果。
3. 若操作数1的结果为0，则**只需**对操作数3求值（而**不需要**对操作数2进行求值），并将该值作为条件表达式的结果。

例 设有int a = 3, b = 4, c; 试分析语句c = a > b ? a : b;执行后，c 的值？

# 本章授课内容



条件运算符



位运算符



运算符总结



类型转换



内存溢出

# 位运算操作运算符

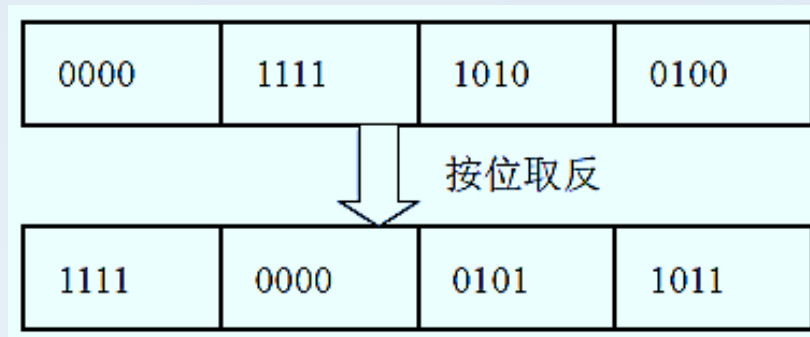
## ❖位运算符：

- 位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)右移(>>)

## ❖~运算符

- 一般形式：~操作数
- 作用：将操作数的二进制表示逐位取反。

例 设有short x = 0x0FA4, y; , 请分析y = ~x;执行后, y 的值?



# 位运算操作运算符

- ❖ &运算符一般形式：操作数1 & 操作数2  
– 对操作数1和操作数2的二进制数，进行**按位求与** **同1为1**
- ❖ |运算符一般形式：操作数1 | 操作数2  
– 对操作数1和操作数2的二进制数，进行**按位求或** **同0为0**
- ❖ ^运算符一般形式：操作数1 ^ 操作数2  
– 对操作数1和操作数2的二进制数，进行**按位求异或** **相同为0，不同为1**

例1 设有int x = 3, y = 4, z; , 请分析语句z = x & y;执行后，z 的值？

例2 设有int x = 0xF4AB, y = 0x1AFC, z; , 请分析语句z = x | y;执行后，z 的值？

例3 设有int x = 0xF4AB, y = 0x1AFC, z; , 请分析语句z = x ^ y;执行后，z 的值？



# 位运算操作运算符

❖ <<运算符一般形式：操作数1 << 操作数2

◆ 对操作数1的**每一位**都**向左**移动操作数2位

❖ **注意：**

◆ 以上操作的操作数1和操作数2应为**整数**类型；

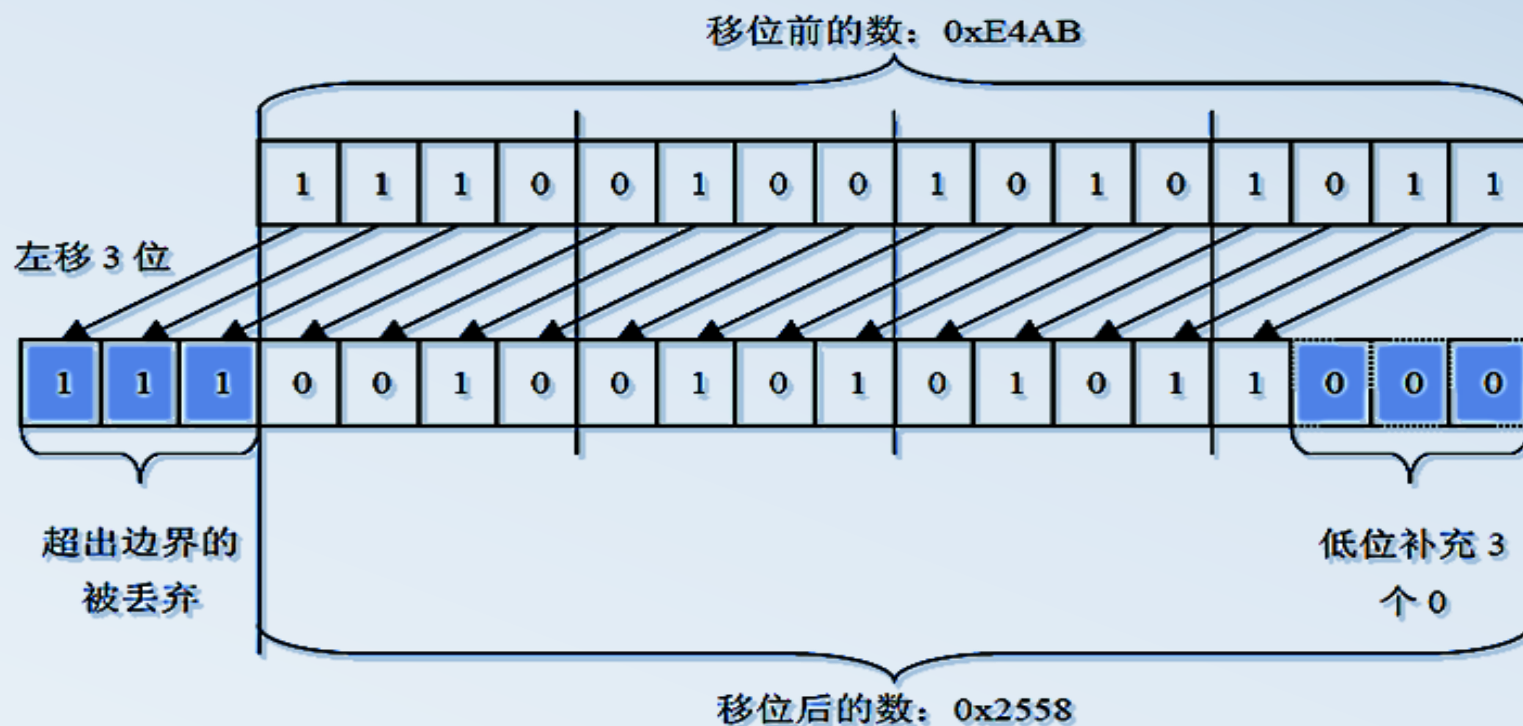
◆ 运算数的各二进位全部左移操作数2指定值的位数，移到边界之外的位被**丢弃**，低位**补0**；

◆ 如操作数2是负数，则移位运算符的结果是未定义的；

◆ 如操作数2的值大于或等于转换后左操作数1数值的位数，则移位运算符的结果也是未定义的。

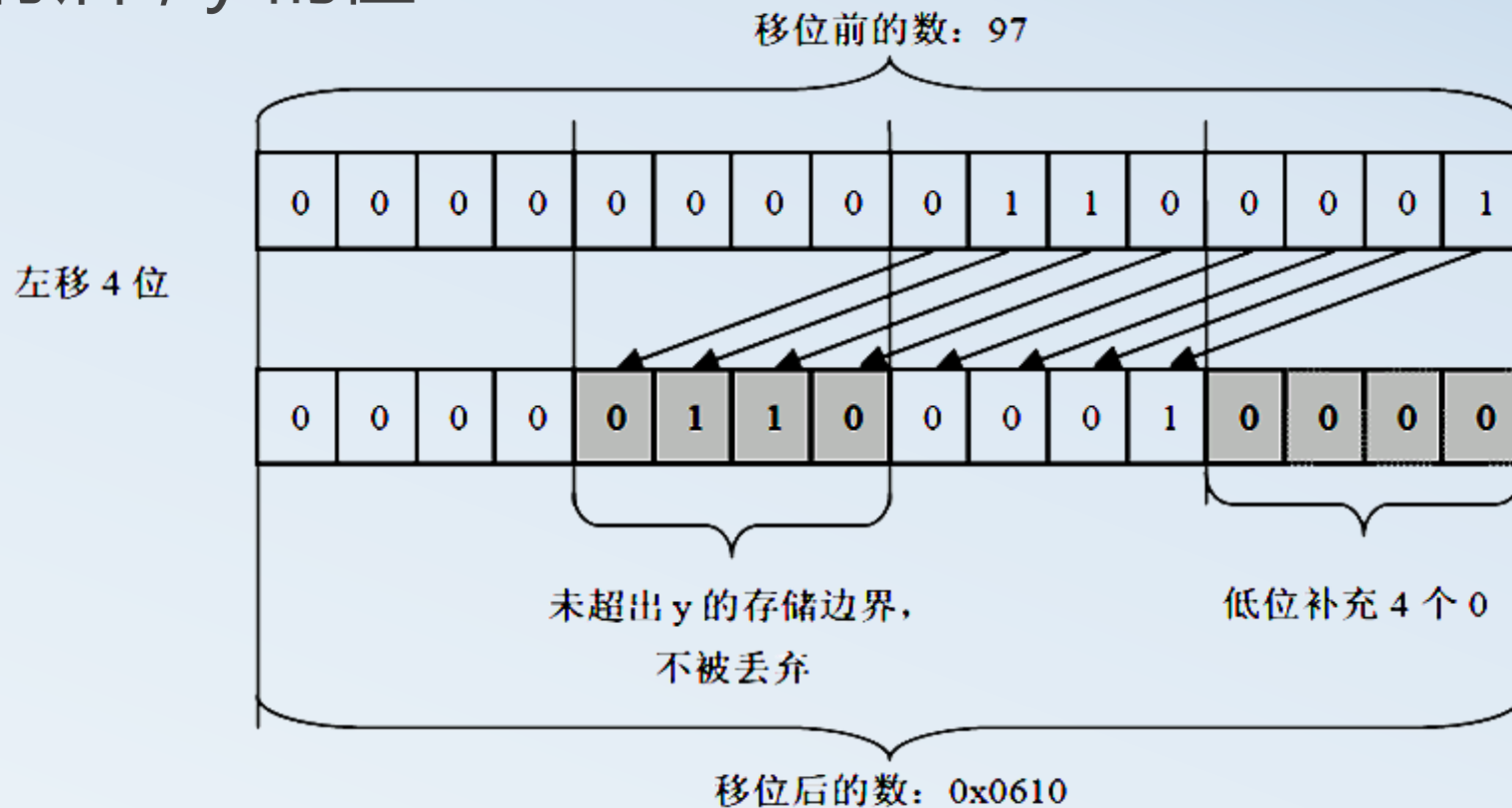
# 位运算操作运算符

❖例1 设有short x = 0xE4AB, y; , 请分析语句y = x << 3;执行后, x、y 的值?



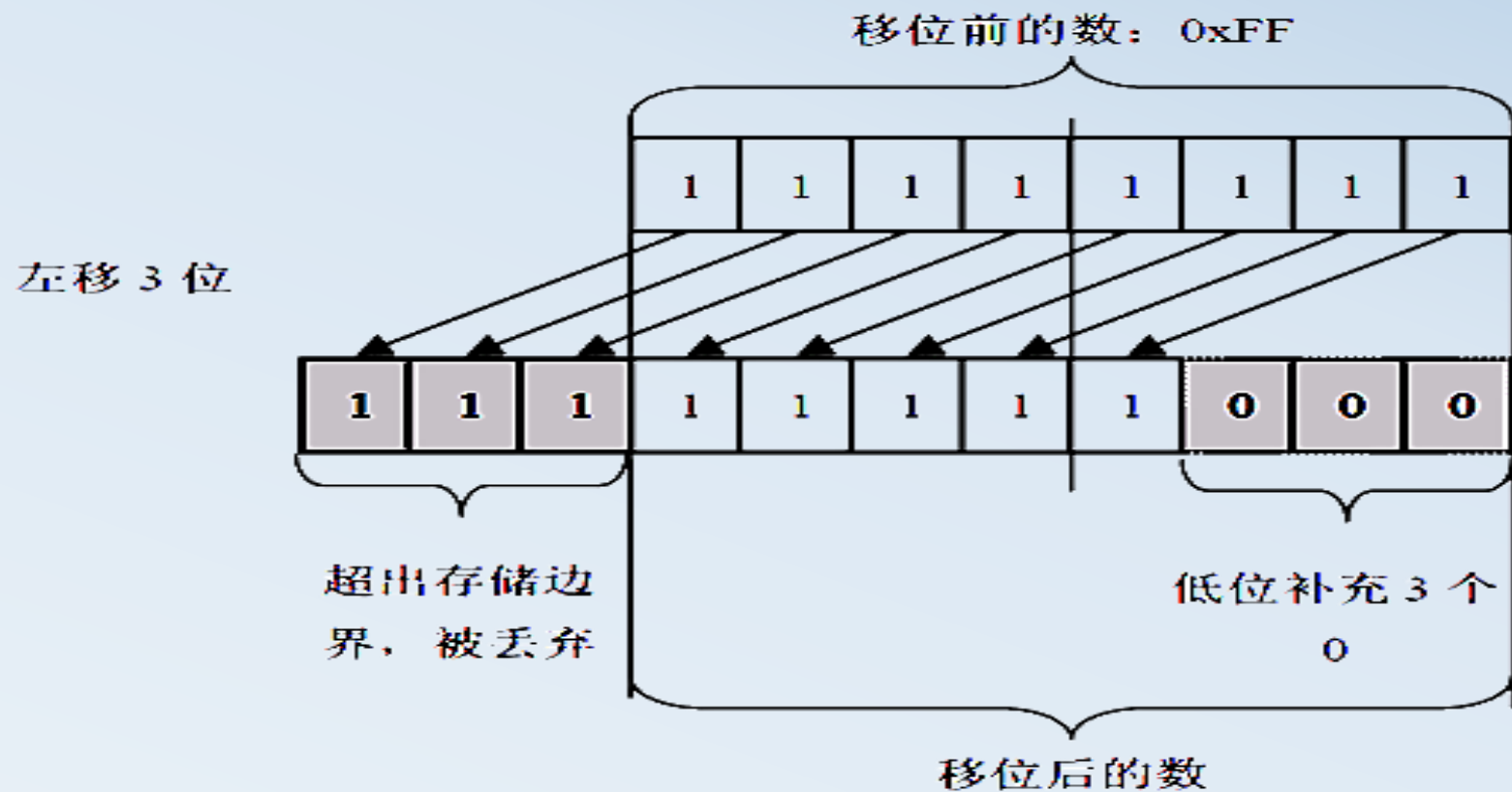
# 位运算操作运算符

- ❖ 例2 设有char x = 'a' , y; , 请分析语句y = x << 4;执行后, y 的值?
- ❖ 例3 设有char x = 'a' ; int y; , 请分析语句y = x << 4;执行后, y 的值?



# 位运算操作运算符

- ❖例4 设有int x = 0xFAFF; char y;请分析语句y = x << 3; 执行后，y的值？



# 位运算操作运算符

❖ >> 运算符一般形式：操作数1 >> 操作数2

◆ 对操作数1的**每一位**都**向右**移动操作数2位

❖ 注意：

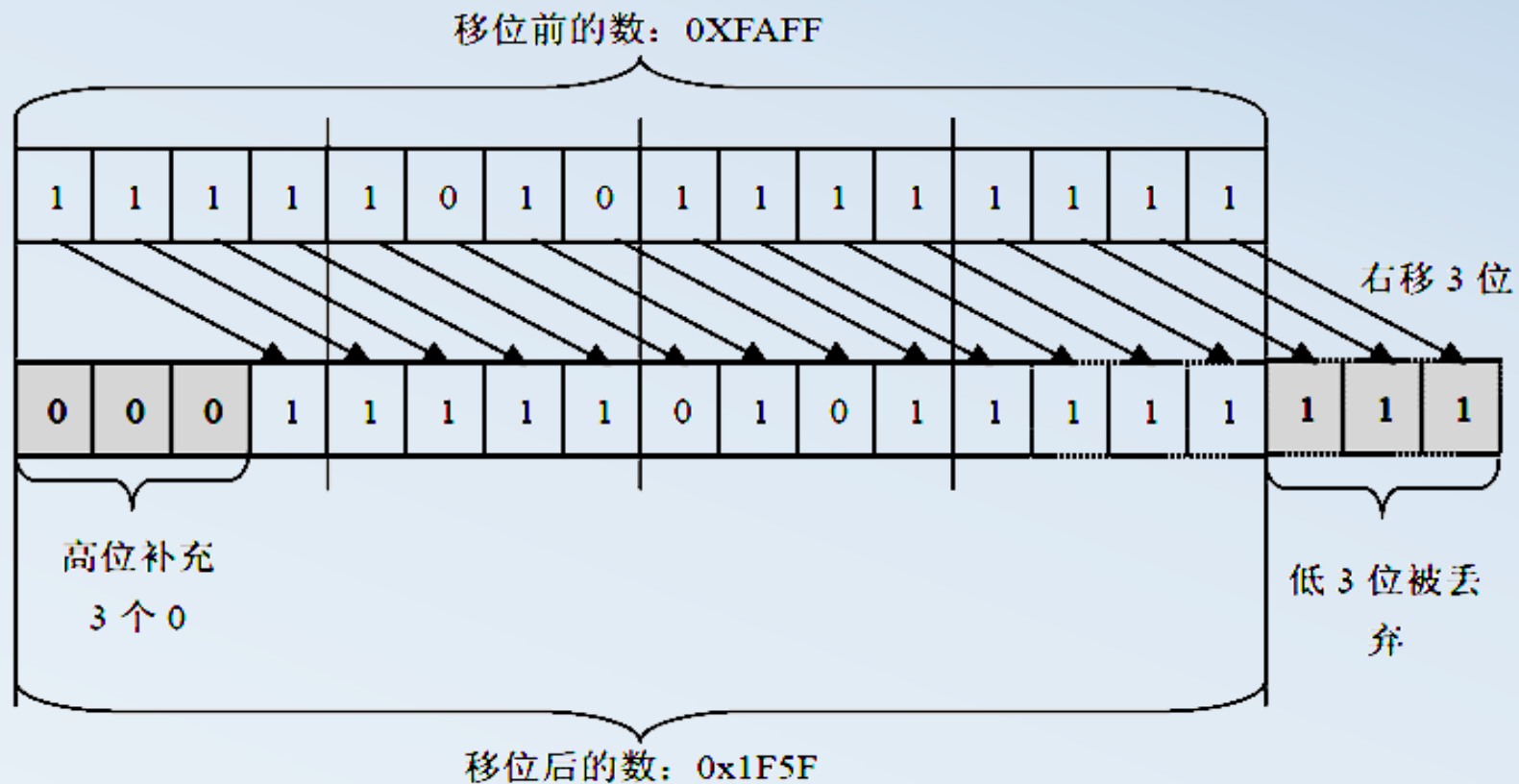
◆ 若操作数1为无符号整型数（或带符号的非负数）时，操作数1的各二进制位右移操作数2指定的位数，**高位补0**。如：将(0100 0110)b 右移两位将得到(0001 0001)b。

# 位运算操作运算符

- ◆ 若操作数1为带符号的负数时，操作数1的各二进制位右移操作数2指定的位数：
  - 有的在高位补0，此时将(1000 0110)→(0010 0001)b。
  - 有的将操作数1移出的低位移入高位，此时将(1000 0110)b→(10100001)b。
  - 含有对带符号的负数进行右移的程序是不可移植的。
- ◆ 如操作数2是负数，则移位运算符的结果是未定义的；
- ◆ 如操作数2的值大于或等于转换后左操作数1数值的位数，则移位运算符的结果也是未定义的。

# 位运算操作运算符

❖ 例 设有 unsigned int x = 0XFAFF, y; , 请分析语句 y = x >> 3; 执行后, y 的值。



# 本章授课内容



条件运算符



位运算符



运算符总结



类型转换



内存溢出



# 逗号运算符

- ❖ 逗号表达式（ 又称顺序表达式 ）包括逗号分开的两个表达式。
  - ◆ 一般形式为：表达式1,表达式2
- ❖ 注意：
  - ◆ 逗号表达式的结合律是从左至右。表达式2的值作为整个逗号表达式的值。
  - ◆ 表达式1和表达式2也可以是逗号表达式。
  - ◆ 并不是有逗号出现的地方，就可以被认为是逗号表达式。

# 逗号运算符

- ❖ 例1 设有 `int x = 3, y;` , 请分析语句 `y = x + 3, 4;` 执行后, `y` 的值?
- ❖ 例2 设有 `int x = 3, y;` , 请分析语句 `y = (x + 3, 4);` 执行后, `y` 的值?
- ❖ 例3 设有 `int x = 3, y;` , 请分析语句 `y = ((x + 3, 4, 5), x + 6);` 执行后, `y` 的值?

# 运算符表达式总结

❖ C语言运算符可分为以下几类：

- 算数运算符：+、-、\*、/、%、++、--
- 关系运算符：>、<、==、>=、<=、!=
- 逻辑运算符：&&、||、!
- 赋值运算符：=、+=、-=、\*=、/=、%=、&=、|=、^=、>>=、<<=
- 条件运算符：?:
- 位运算符：&、|、~、^、<<、>>
- 逗号运算符：,
- 指针运算符：\*、&
- 求字节数运算符：sizeof
- 特殊运算符：括号()、下标[]、成员(→, .)

# 不同类型的变量进行运算

❖ `int x = 5 ; float y = 4.6 ;`

- `float z = x + y;`
- `double z = x + y;`
- `int z = x + y;`

如果把不同类型的变量相互运算会怎样呢？



# 本章授课内容



条件运算符



位运算符



运算符总结



类型转换



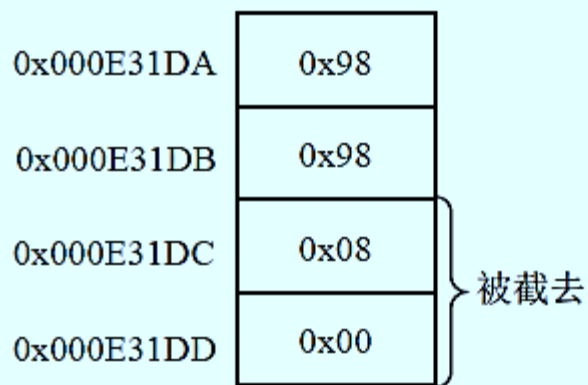
内存溢出



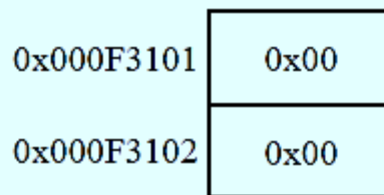
# 类型转换

❖ 类型转换分为**隐式**类型转换和**显示**类型转换。

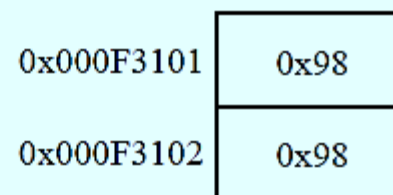
```
int x = 0x89898; /*等价于 int x = 0x00089898;*/  
short y;  
y = x;
```



(a) x 的存储示意图



(b) 赋值前 y 的存储示意图

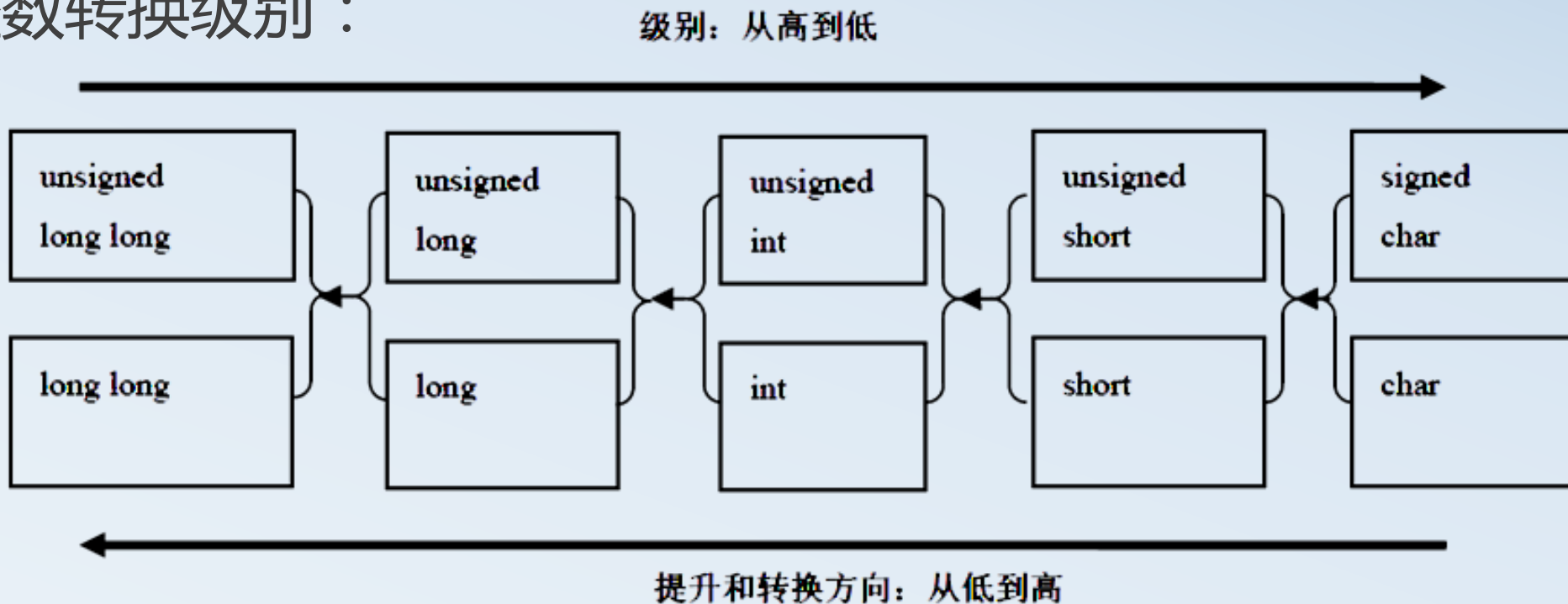


(c) 赋值后 y 的存储示意图

# 隐式类型转换

- ❖ 在表达式中如果有不同类型的变量或字面值参与同一运算时，编译器将在编译时自动按照规定的规则将其转换为相同的数据类型，这种由编译自动完成的转换即是隐式转换。

- ❖ 整数转换级别：



# 隐式类型转换

- ❖ 规则 1：若运算符两边的操作数类型相同则不需要任何转换。
- ❖ 规则 2：若两边都是带符号类型或都是无符号类型，则把低级别类型的操作数转换为高级别的类型。如：设有 `long long x; long y;`，则计算 `x + y` 时先将 `y` 的类型转换为 `long long`。
- ❖ 规则 3：若一个操作数是带符号的（设为 `S`），另一个是无符号的（设为 `U`），且 `U` 的级别高于或等于 `S` 的级别，则把 `S` 转换成 `U` 的类型。如：设有 `unsigned int x; int y;`，则计算 `x + y` 时先将 `y` 的类型转换为 `unsigned int`。
- ❖ 规则 4：若一个操作数是带符号的（设为 `S`），另一个是无符号的（设为 `U`），且 `S` 的类型能够表示 `U` 的类型的数值，则把 `U` 转换为 `S` 的类型。
- ❖ 规则 5：若一个操作数是带符号的（设为 `S`），另一个是无符号的（设为 `U`），且以上规则都不适用，则把 `S` 和 `U` 转换为与 `S` 的类型级别相同的无符号类型。



# 隐式类型转换

## 注意：

- ❖ 隐式类型转换是编译器自动进行的.
- ❖ 隐式类型转换一般是向较大的类型转变.
- ❖ 在一个表达式中尽量避免带符号和无符号的数同时出现

# 显示类型转换

❖ 显式转换表达式的一般形式为：

◆ (希望转换的类型) 操作数

❖ 例 设有 `int x = 3, y = 0xFFFFFFFF; double f;` ,  
请分析下列语句执行后，f 的值。

语句	f 的值
<code>f = x + y;</code>	2.000000
<code>f = (unsigned int)(x+y);</code>	2.000000
<code>f = x + (double)y;</code>	2.000000
<code>f = (double)x + y;</code>	2.000000
<code>f = (double)(x + y);</code>	2.000000

# 显示类型转换

## 注意：

- ❖ 显示类型转换是程序员手动进行的.
- ❖ 基本可以进行任意类型的转换.

# 本章授课内容



条件运算符



位运算符



运算符总结



类型转换



内存溢出



# 内存溢出

- ❖ 由于数值数据类型都有数值范围，当两个数据发生运算时，其结果就有可能超出结果类型的数值范围，这种现象称为**溢出**。
- ❖ 例如：`short x = 32767 ; x = x + 1 ; x ?`
- ❖ 例如：`unsigned short x = 65535 ;  
x++ ; x ?`

# 内存溢出

❖例 设有int x = 0x7FFFFFFF, y= 1, z;请分析  
z=x+y;执行后z的值？

	01111111	11111111	11111111	11111111
+	00000000	00000000	00000000	00000001
<hr/>				
	10000000	00000000	00000000	00000000

# 内存溢出

## 注意：

- ❖ 溢出是未定义的行为.
- ❖ 程序员要尽量避免溢出.

Thank You !