



《计算概论A》 程序设计部分

C 语言的构成成分——运算成分

李 戈

北京大学 信息科学技术学院 软件研究所

lige@sei.pku.edu.cn

C 语言中的运算符

■ C 语言的运算符范围很宽

- ◆ 求字节数运算符: **sizeof**
- ◆ 下标运算符 **[]**
- ◆ 赋值运算符 **=**
- ◆ 算术运算符 **+ - * / %**
- ◆ 关系运算符 **< > == >= <= !=**
- ◆ 逻辑运算符 **! && ||**
- ◆ 条件运算符 **? :**
- ◆ 逗号运算符 **,**
- ◆ 位运算符 **>> ~ | ^ &**
- ◆ 指针运算符 ***, &**
- ◆ 强制类型转换运算符: **(类型)**
- ◆ 分量运算符 **. →**

赋值运算符

- “=” 赋值运算符

- ◆ 给赋值号左边的变量赋予数值

- 在变量定义的同时可以为变量赋初值。如：

- ◆ `int a=3;`

- 相当于：

- `int a; a=3;`

- ◆ `int a, b, c = 5`

- 表示只给 c 赋初值。相当于

- `int a, b, c;`

- `c = 5;`

- ◆ `int a = b = c = 5;` (Is this right?)

赋值运算符

■ 要点一：两边类型不同

- ◆ 若 = 两边的类型不一致，赋值时要进行类型转换。
- ◆ 不管 = 右边的操作数是什么类型，都要转换为 = 左边的类型

```
int main()
{
    int int_i = 64.12345;
    char char_i = int_i;
    float float_i = char_i;
    bool bool_i = float_i;
    cout << showpoint << int_i << " " << char_i <<
        " " << float_i << " " << bool_i << endl;
    return 0;
}
```



64 0 64.00000 1

赋值运算符

■ 要点二：长数赋给短数

◆ 截取长数的低n位送给短数

0 1 1 0 1 1 0 0 0 0 1

0 1 1 0 0 0 0 1

```
int main()
{
    char char_a = ' ';
    int int_i = 0x361;
    cout << hex << int_i << endl;
    char_a = int_i;
    cout << char_a << endl;
    return 0;
}
```

361

a

赋值运算符

■ 举例: `short = long`

- 截取长整型数的低16位送给short字符。
- 如果最高位为1, 则得到负数, 否则得到正数;

0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

```
int main()
```

```
{
```

```
    long int long_i = 0x2AAAAAAAAA;
```

```
    cout << long_i << endl;
```

```
    short short_j = long_i;
```

```
    cout << hex << short_j << endl;
```

```
    cout << dec << short_j << endl;
```

```
    return 0;
```

```
}
```

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

715827882

aaaa

-21846

赋值运算符

■ 要点三：短数赋给长数

- ◆ 最好处理的情况！原来是什么数，现在还是什么数！
- ◆ `short int a = -1; int b = a;`

计算机的处理过程：

- ◆ 若short 型数为**无符号数**：
 - short 型16位到long型低16位，long型高16位补0；
- ◆ 若 short型数为**有符号数**：
 - short型16位到 long型低16位；
 - 若short型最高位为0，则long型高16位补0；
 - 若short型最高位为1，则long型高16位补1；

示例

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    short short_i = -123;
    cout << hex << short_i << endl;
    int int_j = short_i;
    cout << hex << int_j << endl;
    cout << dec << int_j << endl;
    return 0;
}
```



```
ff85
ffffffff85
-123
```


■ 要点四：符号位的赋值处理

- ◆也很好处理的情况！直接搬运！
- ◆直接赋值，不管符号位还是数字位！

例如：

- ◆ **unsigned = int 或 long**
 - 直接赋值，符号位当作数字。
- ◆ **int = unsigned int**
 - 直接赋值，数字当作符号位。

signed

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

unsigned

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    unsigned int u;
    cout << unsigned
    signed int sign
    cout << hex <<
    cout << dec <<
    return 0;
}
```

```
2863311530
aaaaaaaa
-1431655766
```

```
unsigned int unsigned_int_i = 0xAAAAAAAA;
cout << unsigned_int_i << endl;
signed int signed_int_j = unsigned_int_i;
cout << hex << signed_int_j << endl;
cout << dec << signed_int_j << endl;
return 0;
```

signed

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

unsigned

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0



赋值运算总结

- 当：两边类型不同
 - ◆ 自动完成类型转换！
- 当：长数赋给短数
 - ◆ 截取长数的低位送给短数！
- 当：短数赋给长数
 - ◆ 原来是什么数，现在还是什么数！
- 当：符号位的赋值处理
 - ◆ 直接赋值，不管符号位还是数字位！



赋值运算总结

- 当：两边类型不同
 - ◆ 自动完成类型转换！
- 当：长数赋给短数
 - ◆ 截取长数的低位送给短数！
- 当：短数赋给长数
 - ◆ 原来是什么数，现在还是什么数！
- 当：符号位的赋值处理
 - ◆ 直接赋值，不管符号位还是数字位！

表达式

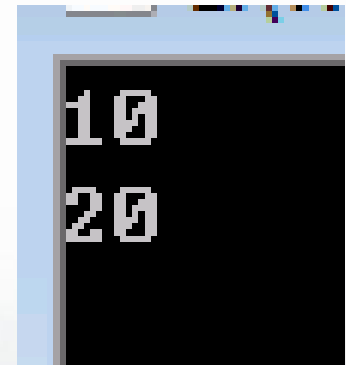
■ 什么是表达式

- ◆ 程序中由运算符、操作数和括号组成等所组成的计算式，是计算求值的基本单位。

`a * b + c;` `123 < 10;` `'a'*3.14f == 1;` `a = b;`

- ◆ 表达式是有“值”的，赋值语句也不例外；

```
int main()
{
    int i = 0;
    cout << (i = 10) << endl;
    cout << (i = i + i) << endl;
    return 0;
}
```



```
10
20
```

赋值表达式

■ 复合的赋值运算

◆ 在赋值符号前加上其它运算符号则构成复合赋值运算;

◆ 例如:

● $a += 3;$

等价于 $a = a + 3;$

● $x * = y + 8;$

等价于 $x = x * (y + 8);$

● $x \% = 3;$

等价于 $x = x \% 3;$

赋值表达式

■ 连续的赋值运算

◆ 自右而左的结合顺序

● $a = b = 5;$ //a, b均为5

● $a = b = c = 5;$ //a, b, c均为5

◆ $\text{int } a=b=c=5;$ //编译错!

◆ $a = (b = 4) + (c = 6);$ //a为10, b为4, c为6

■ 举例:

$a += a -= a * a$ (设a为12)

$\swarrow \quad \searrow$

$a = a - a * a$ (a为12-12 * 12=-132)

\swarrow

$a += -132 \rightarrow a = a + (-132) \rightarrow a = -264$

算术运算符和算术表达式

■ 算术运算符和算术表达式

◆ 基本的算术运算+、-、*、/、%

- % 是模运算，即，求余运算，必须是整数

- ◆ 如 $7 \% 4 = 3$

■ 注意：

◆ 整数运算，结果仍为整数

- $5/3$ 的结果仍是整数，小数部分被忽略

◆ 实数运算，结果为double型

- $5.3/3$ 或 $5/3.0$ 的结果为double型

◆ 舍入的方向随编译器的不同而不同

算术表达式

■ 算术运算符的优先级

()

* / %

+ -

◆ 在同一级别中，采取由左至右的结合方向

● 如： $a - b + c$ 相当于 $(a - b) + c$

● 如： $a \% b * c / d$ 相当于 $((a \% b) * c) / d$

◆ 当数据类型非常复杂时

● 如： $123\%s + (i + '@') + i * u - f / d$

其中 `short s; int i; float f; double d; unsigned u;`

算术表达式

■ 剪刀法求表达式的值

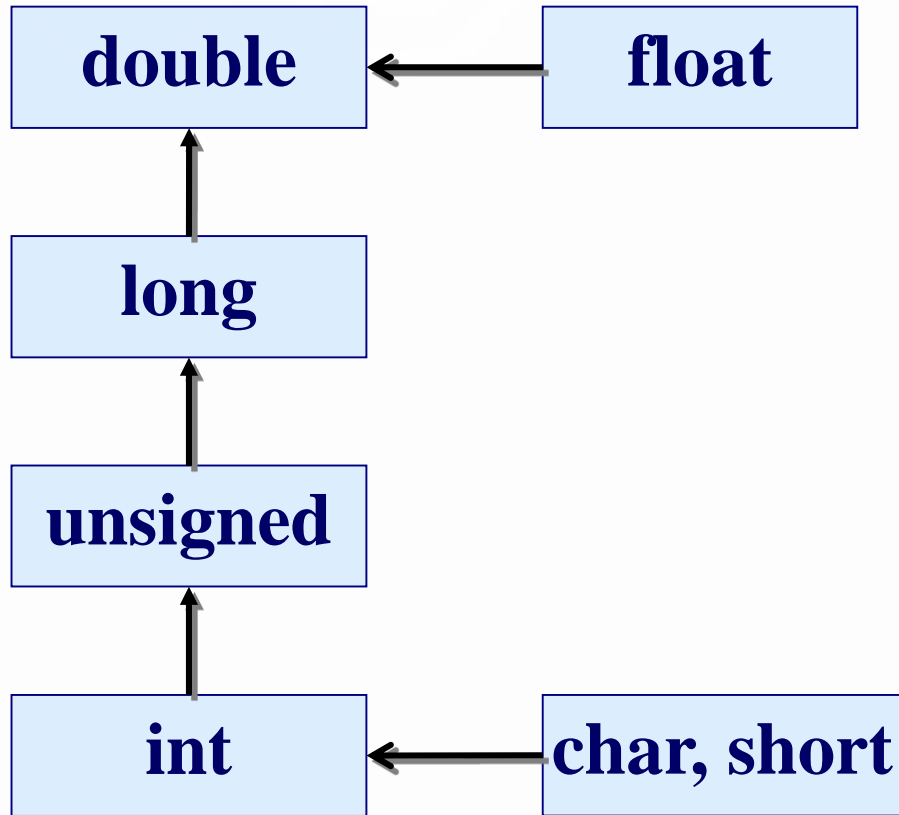
short s; int i; float f; double d; unsigned u;

123%s + (i+'@') + i*u - f/d

Handwritten annotations for the expression above:

- Red vertical lines with circled numbers 1, 2, 3, 4, 5 mark the operators: %, +, +, *, and -.
- Red handwritten labels indicate the type of each operand:
 - 123**: int
 - s**: int
 - (i+'@')**: int
 - i**: int
 - u**: unsigned int
 - f**: float
 - d**: double
- Red horizontal lines and brackets show the evaluation order and resulting types:
 - A line under **123** and **s** is labeled **int**.
 - A line under **(i+'@')** is labeled **int**.
 - A bracket under **i** and **u** is labeled **unsigned int**.
 - A large bracket under the entire expression **123%s + (i+'@') + i*u - f/d** is labeled **double**.

计算过程中的类型转换



```
short s; int i; float f; double d; unsigned u;
```

123%s + (i+'@') + i*u + f/d

int int unsigned double

int unsigned double

算术表达式

■ 自增、自减运算符：使变量的值加1或减1

◆ $++i$, $--i$

● 在使用 i 之前，先将 i 的值加（减）1

◆ $i++$, $i--$

● 在使用 i 之后，再将 i 的值加（减）1

■ 例如： i 的值为3，则

◆ $j = ++i$;

◆ $j = i++$;

◆ $\text{cout} << ++i$;

◆ $\text{cout} << i++$;

自增、自减运算符

■ 举例：

◆ 设i的值为3； 则

```
cout<<-i++<<endl;
```

```
cout<<-++i<<endl;
```

```
cout<<(-i)++<<endl;
```

```
cout<<++i++<<endl;
```

■ 注意：

◆ ++ 和--只能用于变量

自增、自减运算符

1	0	
2	0	
4	6	
4		
3	2	2

```
int main()
```

```
{
```

```
    int a = 0, b = 0, c = 2, d = 0, e = 2, f = 2;
```

```
    cout << a << " " << a++ << " " << endl;
```

```
    cout << ++b << " " << b++ << " " << endl;
```

```
    cout << c << " " << (c++) + (++c) << " " << endl;
```

```
    cout << (d = f++) + (e = f) << endl;
```

```
    cout << f << " " << d << " " << e << endl;
```

```
    return 0;
```

```
}
```

关系运算符

■ C语言提供6种关系运算符:

① < (小于)

② <= (小于或等于)

③ > (大于)

④ >= (大于或等于)

⑤ == (等于)

⑥ != (不等于)

优先级相同

高

优先级相同

低



关系运算表达式

■ 关系运算表达式的值:

“真” / “假”

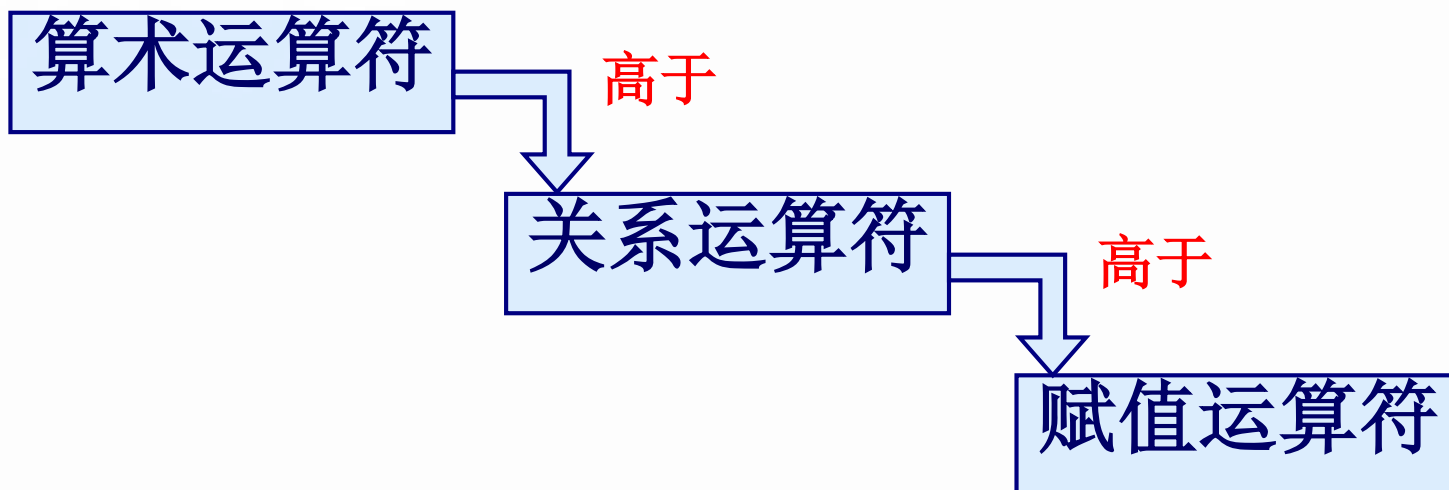
◆ 如 $a = 3$, $b = 4$

$a > b$ 的值为 0

$a \neq b$ 的值为 1

$a == b$ 的值为 0

运算符的优先级



■ 例如:

◆ $1 + 2 \% 3 * 4 > 5 / 6 - 7$

◆ $1 + 2 \% (3 * 4) > (5 / 6 - 7) == 8$

◆ $X = 1 + 2 \% 3 * 4 > 5 / 6 - 7 == 8$

◆ $1 > 2 == 3 > 4$

逻辑运算符

■ 三种逻辑运算符：

◆ 逻辑与 &&

◆ 逻辑或 ||

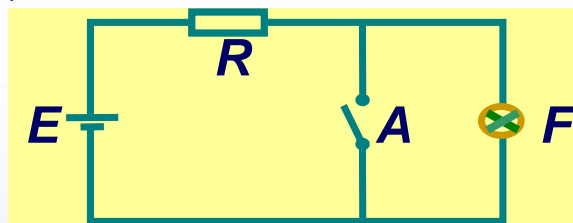
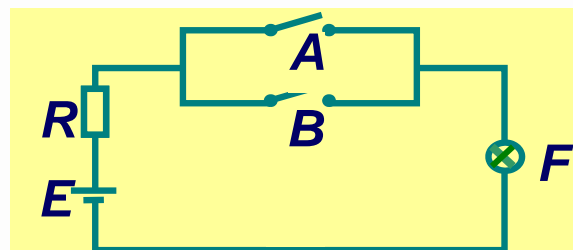
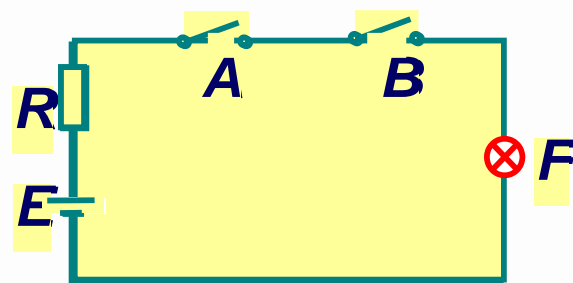
◆ 逻辑非 !

【注】：

◆ 若A、B为真，则 $F=A \&\&B$ 为真。

◆ 若A、B之一为真，则 $F=A || B$ 为真。

◆ 若A为真，则!A为假。





逻辑表达式的值

- 以0代表“假”；以非0代表“真”。
 - ◆ 若 $a = 4$ ，则 $!a$ 的值为0。
 - ◆ 若 $a = 4$ ， $b = 5$ ，则 $a \&\& b$ 的值为1。
 - ◆ 若 $a = 4$ ， $b = 5$ ，则 $a \parallel b$ 的值为1。
 - ◆ 若 $a = 4$ ， $b = 5$ ，则 $!a \parallel b$ 的值为1。
 - ◆ 表达式 $4 \&\& 0 \parallel 2$ 的值为1。

逻辑运算符优先级

■ 逻辑表达式

◆ 一个逻辑表达式中若包含多个逻辑运算符，则按以下的优先次序：

● **!(非)→&&(与)→|| (或)**，即“!”优先级最高。

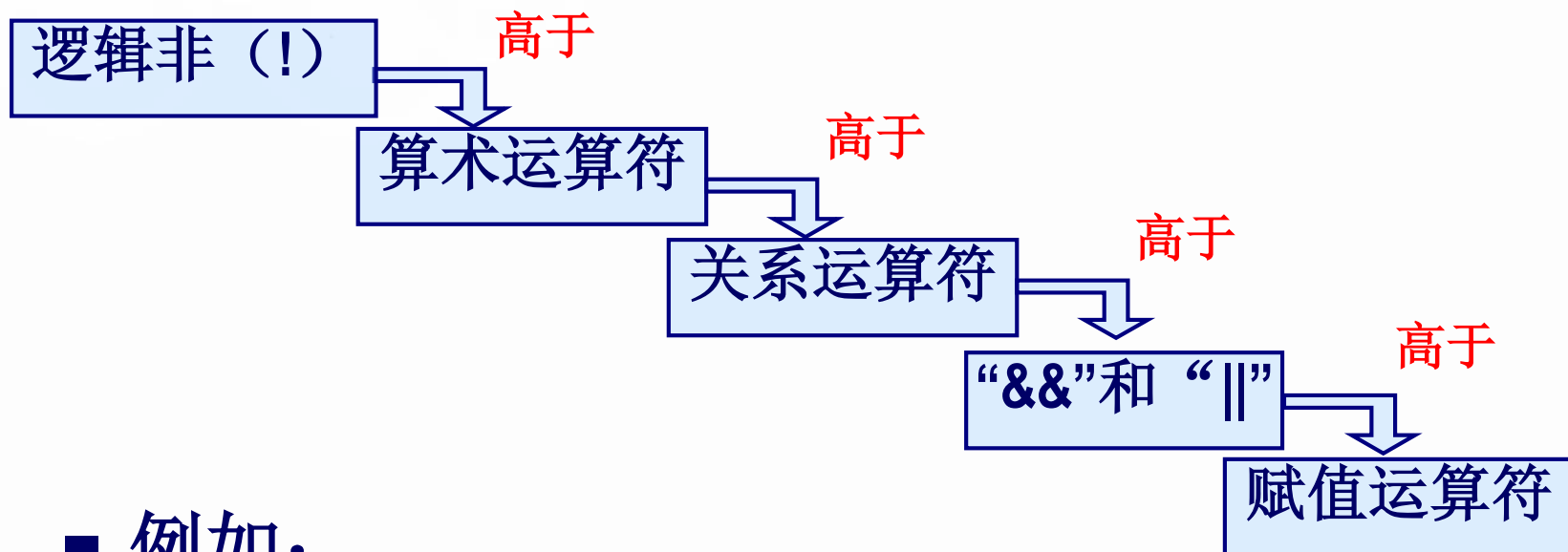
◆ 例如：

!a&&b||c;

! a && b || x>y && c;

! a && b || x>y && c + 1;

混合运算的优先级



■ 例如：

- ◆ $a > b \ \&\& \ x > y$ 可写成 $(a > b) \ \&\& \ (x > y)$
- ◆ $a == b \ || \ x == y$ 可写成 $(a == b) \ || \ (x == y)$
- ◆ $!a \ || \ a > b$ 可写成 $(!a) \ || \ (a > b)$

思考题

```
#include<iostream>
using namespace std;
int main()
{
    int a = 0, b = 0;
    a = 5 > 3 && 2 || 8 < 4 - (b = !0);
    cout<<a<<" "<<b;
    return 0;
}
```

逻辑运算的取舍

- 逻辑表达式求解中，并不总是执行所有的运算
 - ◆ 只有在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符！
- 对于表达式 $a \ \&\& \ b \ \&\& \ c$
 - ◆ 只有a为真(非0)时，才需要判别b的值，
 - ◆ 只有a和b都为真的情况下才需要判别c的值。
- 对于表达式 $a \ || \ b \ || \ c$
 - ◆ 只要a为真(非0)，就不必判断b和c；只有a为假，才判别b；a和b都为假才判别c。



逻辑运算的取舍举例

```
#include<iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{
```

```
    int i = 0, a = 1, b = 2, c = 3;
```

```
    i = ++a || ++b || c++;
```

```
    cout<<i <<" "<<a<<" "<<b<<" "<<c<<endl;
```

```
    return 0;
```

```
}
```




运算对象的扩展

- 逻辑运算符两侧可以是任何类型

- ◆ 如：字符型、实型或指针型等；

- ◆ 系统最终以 0 和 非0 来判定它们，例如

`'c' && 'd'`

- ‘c’和 ‘d’的ASCII值都不为0，按“真”处理。

- 思考：

- ◆ `'a' == 'b' && ! 'c'`



示例

```
#include<iostream>
using namespace std;
int main()
{
    int a = 0, b = 1;
    a = 8>4 - (b = !'c') && 5>3 + 'a' % 6 == 'b';
    cout<<a<<" "<<b;
    return 0;
}
```

应用示例

- 问题：要判别某一年`year`是否闰年。闰年的条件是符合下面二者之一：
 - ◆ ①能被4整除，但不能被100整除。
 - ◆ ②能被100整除，又能被400整除。
- 求解：
 - ◆ 可以用如下逻辑表达式来判别`year`是否为闰年：
$$(\text{year \% } 4 == 0 \ \&\& \ \text{year \% } 100 != 0) \ || \ \text{year \% } 400 == 0$$
 - ◆ 也可以用如下逻辑表达式判别`year`是否非闰年：
$$(\text{year \% } 4 != 0) \ || \ (\text{year \% } 100 == 0 \ \&\& \ \text{year \% } 400 != 0)$$

逗号运算和逗号表达式

■ 用逗号将两个表达式连起来

- ◆ 表达式1, 表达式2, 表达式3, 表达式n
- ◆ 先求表达式1, 再求表达式2,, 再求表达式n, 整个表达式的值为表达式n的值

$a = 3 * 5, a * 4;$

■ 下式中是否相同?

● $x = (a = 3, 6 * 3);$

● $x = a = 3, 6 * 3;$



条件运算符

表达式1 ? 表达式2 : 表达式3

■ 求值规则：

- ◆ 如果 表达式1 的值为真，则以表达式2的值作为条件表达式的值；否则以表达式3的值作为整个条件表达式的值

$\text{max} = (a > b) ? a : b ;$

相当于：

$\text{if}(a > b) \text{ max} = a ;$

$\text{else max} = b ;$



强制类型转换

■ 形式:

◆ (类型名) (表达式)

■ 举例:

◆ (double)a

将a的**值**转换成double类型

◆ (int)(x+y)

将x+y的**值**转换成int类型

◆ (float)(5/3)

将5/3的**值**转换成float类型

■ 注意:

◆ 强制类型转换后, **被转换的量**的类型并没有发生变化



选讲内容——位运算



位运算

■ 位运算

- ◆ 所谓位运算是指进行二进制位的运算。

■ C++语言中的位运算符

- ◆ 按位与 (&) 双目运算符
- ◆ 按位或 (|) 双目运算符
- ◆ 按位异或 (^) 双目运算符
- ◆ 取反 (~) 单目运算符
- ◆ 左移 (<<) 单目运算符
- ◆ 右移 (>>) 单目运算符

位运算符 (1)

■ “按位与” 运算符(&)

◆ 参加运算的两个数据，各位均独立进行“与”运算。

■ 例如：对于表达式： $a = 3 \& 5$ ，有：

$3 = 00000011$

$(\&) 5 = 00000101$

00000001

因此， $3\&5$ 的值为： 1

位运算符 (2)

■ “按位或” 运算符 (|)

◆ 参加运算的两个数据，各位均独立进行“或”运算。

■ 例如：对于表达式： $a = 3 | 5$ ，有：

$3 = 00000011$

$(|) 5 = 00000101$

00000111

因此， $3 | 5$ 的值为： 7

位运算符 (3)

■ “异或”运算符 (\wedge)

◆ 异或运算符 \wedge 也称XOR运算符。参加运算的两个数，各位均独立进行异或运算：

◆ 即 $0 \wedge 0 = 0$ ； $0 \wedge 1 = 1$ ； $1 \wedge 0 = 1$ ； $1 \wedge 1 = 0$ ；

例如：

	00111001	(十进制数57，八进制数071)
(\wedge)	00101010	(十进制数42，八进制数052)
<hr/>		
	00010011	(十进制数19，八进制数023)

即 $071 \wedge 052$ ，结果为023(八进制数)。

位运算符 (4)

- 取反运算符“ \sim ”是一个单目(元)运算符，用来对一个二进制数按位取反，

- ◆ 即将0变1，1变0。

- ◆ 例如：

000000000010101

(\sim)

↓

111111111101010

因此， $\sim(025)_8$ 的值为 $(177752)_8$

位运算符 (5)

■ 左移运算符(<<)

- ◆ 用来将一个数的各二进制位全部左移若干位。
- ◆ 高位左移后溢出，舍弃不起作用。
- ◆ 若 $a=15$ ，即二进制数00001111，左移2位得00111100，即十进制数60

$$a = a \ll 2$$

- ◆ 左移1位相当于该数乘以2，左移2位相当于该数乘以 $2^2=4$ 。
 - 只适用于该数左移时被溢出舍弃的高位中不包含1的情况。

位运算符 (6)

■ 右移运算符(>>)

- ◆ 用来将一个数的各二进制位全部右移若干位。
- ◆ 移到右端的低位被舍弃，对无符号数，高位补0。
- ◆ 若 $a=15$ ，即二进制数00001111，右移2位得00000011，即十进制数3；

$$a = a \gg 2$$

- ◆ 当没有非零数位被抛弃时，右移一位相当于除以2，右移 n 位相当于除以 2^n 。



关于位运算的几个问题(1)

■ 右移运算符符号位的处理

- ◆ 对无符号数，右移时左边高位移入0。
- ◆ 对于有符号的值，
 - 若原来符号位为0(该数为正)，则左边移入0；
 - 若符号位原来为1(即负数)，则左边移入0还是1，要取决于所用的计算机系统。
 - ◆ 若移入0，称为“逻辑右移”或“简单右移”
 - ◆ 若移入1，称为“算术右移”(VC)



关于位运算的几个问题(2)

■ 不同长度的数据进行位运算

◆ 如果两个数据长度不同进行位运算时，系统会将二者按右端对齐。

● 如 $a \& b$ ，而 a 为 `int` 型， b 为 `short` 型

◆ 如何补位

● 如果 b 为无符号整数型，则左侧添满 0。

● 如果 b 为有符号整数型：

◆ 如果 b 为正数，则左侧 16 位补满 0。

◆ 如果 b 为负数，则左端 16 位补满 1。

关于位运算的几个问题(3)

■ 位运算赋值运算符

◆ 位运算符与赋值运算符可以组成复合赋值运算符如: $\&=$, $|=$, $>>=$, $<<=$, $\wedge=$

● $a \&= b$ 相当于 $a = a \& b$

● $a |= b$ 相当于 $a = a | b$

● $a >>= 2$ 相当于 $a = a >> 2$

● $a <<= 2$ 相当于 $a = a << 2$

● $a \wedge= b$ 相当于 $a = a \wedge b$

关于位运算的几个问题(4)

■ 算符优先级

- ◆ 取反运算符 “ \sim ”
- ◆ 算术运算符
- ◆ 左移<< 右移>>
- ◆ 关系运算符
- ◆ 按位与&
- ◆ 按位异或 \wedge
- ◆ 按位或|
- ◆ 逻辑运算符

高

低

位运算的使用 (1)

■ “按位与” 运算的用途

◆ 任意存储单元与0进行“按位与”运算可**清零**；

◆ **取一个数中某些指定位**

● 如有一个整数a，想要其中的低字节。只需将a与 $(377)_8$ 按位与即可。

a	00 10 11 00	10 10 11 00
b	00 00 00 00	11 11 11 11
c	00 00 00 00	10 10 11 00

■ “按位或” 运算

◆ 对一个数据的某些位取定值为1；

位运算的使用 (2)

■ “异或”运算符的使用

(1) 使特定位翻转

- ◆ 使01111010低4位翻转（即1变为0，0变为1）可将其与00001111进行 \wedge 运算，即：

$$\begin{array}{r} 01111010 \\ (\wedge) \quad 00001111 \\ \hline 01110101 \end{array}$$

(2) 使特定位保持不变

- ◆ 与0相 \wedge ，保留原值如 $012 \wedge 00 = 012$

$$\begin{array}{r} 00001010 \\ (\wedge) \quad 00000000 \\ \hline 00001010 \end{array}$$

“异或”运算符示例

■ 交换两个值，而不必使用临时变量

◆ 假如 $a = 3$, $b = 4$ 。想将 a 和 b 的值互换，可以用以下赋值语句实现：

$a = a \wedge b;$

$b = b \wedge a;$

$a = a \wedge b;$

设： $a = 3, b = 4$

$a = 011$

$(\wedge) \quad b = 100$

$a = 111$ ($a \wedge b$ 的结果, a 已变成 7)

$(\wedge) \quad b = 100$

$b = 011$ ($b \wedge a$ 的结果, b 已变成 3)

$(\wedge) \quad a = 111$

$a = 100$ ($a \wedge b$ 的结果, a 变成 4)