

运算与表达式

Wang Houfeng

wanghf@pku.edu.cn

EECS, PKU

内容

➤ 赋值

- 运算与表达式
- 位运算

常量表示

- 常量：在程序运行过程中，其值保持不变的量
 - 字面常量
 - -1, 0, 123, 4.6, -1.23;
 - 符号常量
 - 用标识符代表一个常量称符号常量

```
#include <iostream>
using namespace std;
int main()
{
    const float PI = 3.14159f;
    float r, area;
    cin >> r;
    area = r * r * PI;
    cout << "Area = " << area;
    return 0;
}
```

直接常量的类型表示

- 整型常量的后缀

- `n = 10000L;` //长整型常量
- `m = -0x88abL;` //长整型十六进制常量
- `k = 10000U;` //无符号整型常量
- `i = 07777LU;` //无符号长整型八进制常量

- 浮点型常量的后缀

- `x = 3.1415F` //单精度浮点型常量
- `y = 3.1415L` //长双精度浮点型常量

- 说明

- 浮点型常量默认为double 型;
- U, L, F均可以小写;

C语言中的运算

- 求字节数运算符: **sizeof**
- 赋值运算符 **=**
- 算术运算符 **+ - * / %**
- 关系运算符 **< > == >= <= !=**
- 逻辑运算符 **! && ||** 非, 与, 或
- 条件运算符 **? :** **?:** 是一个运算符
- 逗号运算符 **,**
- 位运算符 **>> ~ | ^ &** 右移, 取反, 或, 异或, 与
- 下标运算符 **[]**
- 指针运算符 ***, &**
- 强制类型转换运算符: **(类型)**
- 分量运算符 **. →**

C语言中的运算

- 求字节数运算符: **sizeof**
- 赋值运算符 **=** 赋值不能改变右边的值
- 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 **>> ~ | ^ &**
- 下标运算符 **[]**
- 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

- “=” 赋值运算符
 - 给赋值号左边的变量赋予右边的结果值
- 在变量声明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;` (正确吗?) X

赋值运算的特点之一

- ◆ 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 = 左边变量的类型

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int int_i = 64.12345;
    char char_i = int_i;
    float float_i = char_i;
    bool bool_i = float_i; 非0即为真
    cout << showpoint << int_i << " " << char_i <<
        " " << float_i << " " << bool_i << endl;
    return 0;
}
```

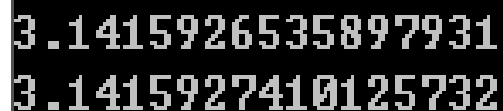


64 0 64.00000 1

类型转换特点（1）

- 如果=右边的操作数具有较高级别的类型，则在类型转换时，进行截断取舍
- 类型转换后可能会损失精度！

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    double double_i = 3.14159265358979323846;
    cout << setprecision(30) << double_i << endl;
    float float_i = double_i;
    cout << float_i << endl;
    return 0;
}
```



```
3.1415926535897931
3.1415927410125732
```

类型转换特点 (2)

- 如果=右边的操作数类型级别较低，则在类型转换时，采用补齐方式
- 类型转换后不会损失精度

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main()
```

$$\{$$

```
float float_i = 3;
```

```
cout << showpoint << float_i << endl;
```

```
double double_i = float_i;
```

```
cout << setprecision(20) << double_i << endl;
```

```
return 0;
```

}

```
3.000000
3.000000000000000000000000
```

赋值运算的特点之二

◆ 整数长数赋给短数：截取长数的低n位送给短数



361

al

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

赋值运算的特点之二（续）

- ◆ 整数长数赋给短数：如截取后的高位为1，则变为负数

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 1 0 1 0 1 0

```
int main()
```

```
{
```

```
    long int long_i = 0x2AAAAAAA;
```

```
    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

赋值运算的特点之三

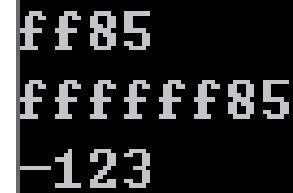
- ◆ 短数赋给长数：原来是什么数，现在还是什么数！

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**：
 - 首先将short型16位赋到 long的低16位；
 - long的高16位处理如下：
 - 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
```

数据变长
结果不变

赋值运算的特点之四

- ◆ 符号位的赋值处理：直接将符号位搬过去
 - ◆ 直接赋值，不管高位是符号位还是数字位！
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	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	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

示例

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    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;
}
```

```
2863311530
aaaaaaaa
-1431655766
```

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 1 0 1 0

赋值运算总结

- 赋值号两边类型不同
 - 自动完成类型转换！
- 赋值号右边长数赋给赋值号左边短数
 - 取长数的低位送给短数！
- 赋值号右边短数赋给赋值号左边长数
 - 原来是什么数，现在还是什么数！
- 符号位的赋值处理
 - 直接赋值，不管是符号位还是数字位！

内容

➤ 赋值

➤ 运算与表达式

- 位运算

C/C++的表达式

- 一般定义
 - 基始：由运算符、操作数（包括函数）和括号等所组成的计算式，是计算求值的基本形式
- 递归定义
 - 基始：
 - ❖ 变量可以作为表达式；如， a
 - ❖ 常量可以作为表达式；如, 10
 - ❖ 具有结果值的函数也可以作为表达式； $\max(x,y)$
 - 归纳：由运算符联接的表达式构成更复杂的表达式
 - ❖ 例： $a+10-\max(x,y)$

一种特殊表达式：赋值表达式

- 由赋值运算符连接的表达式
 - 注意赋值表达式与赋值语句的区别

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

复合型赋值

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 $x = x \% 3;$

注意：右边作为整体参与运算！

连续赋值

- 连续的赋值运算

- 自右而左的结合顺序

理解赋值语句与赋值运算

- $a = b = 5;$ //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)

$\rightarrow a = a - a * a$ (a为12-12 * 12=-132)

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

容易出错，不建议这样写!

算术运算

运算符	运算对象个数	含义
+	双目 (2) / 或者单目 (1)	算术加 (有时也表示正号)
-	双目 (2) / 或者单目 (1)	算术减 (有时也表示负号)
*	双目 (2)	乘法
/	双目 (2)	除法
%	双目 (2)	取余数 (也称取模)
++	单目 (1)	增1运算
--	单目 (1)	减1运算

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
 - $5/3$ (??), $5\%3$ (??)
 - ◆ 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

算术运算优先顺序

- 算术运算中，单目运算的优先级别最高，加减的最低，中间的为乘除（取余）
- 改变常规运算优先级的方法是加括号。

算术运算的优先级

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

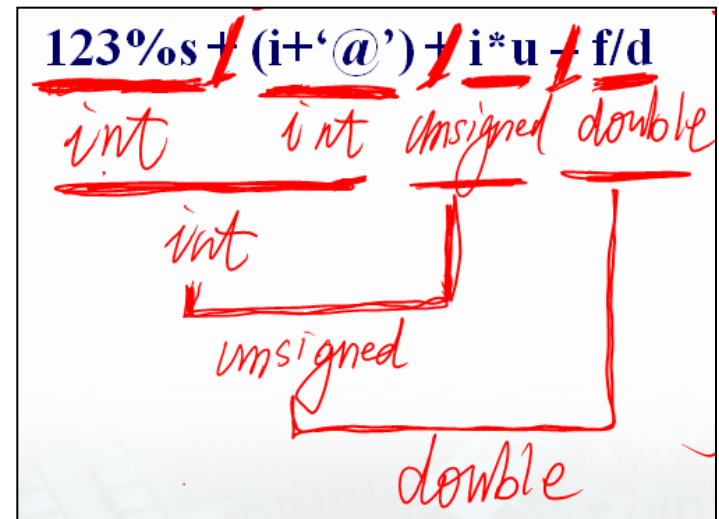
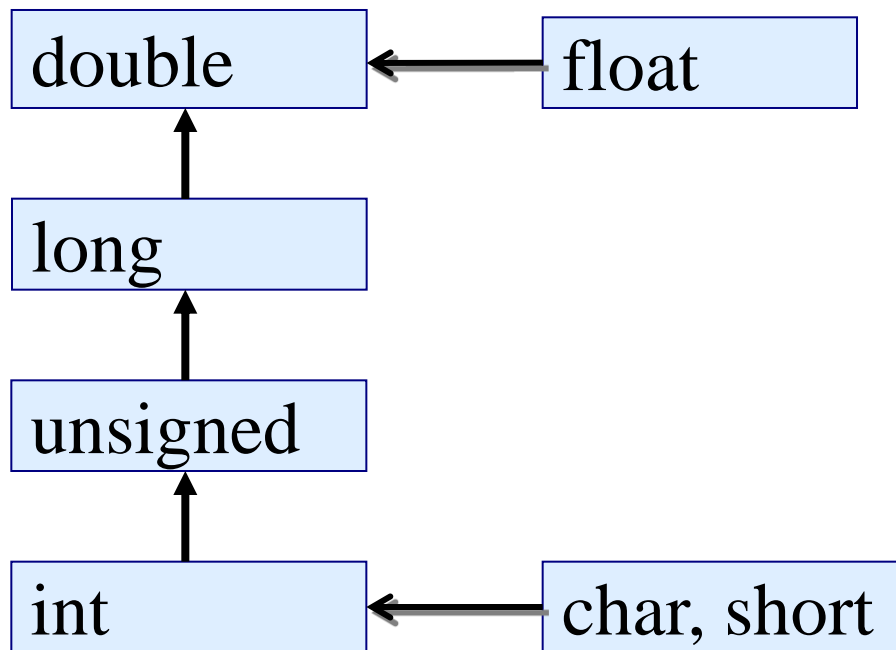
其中 short s; int i; float f; double d; unsigned u;

算术表达式的结合与类型变换

- 剪刀法求表达式的值

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

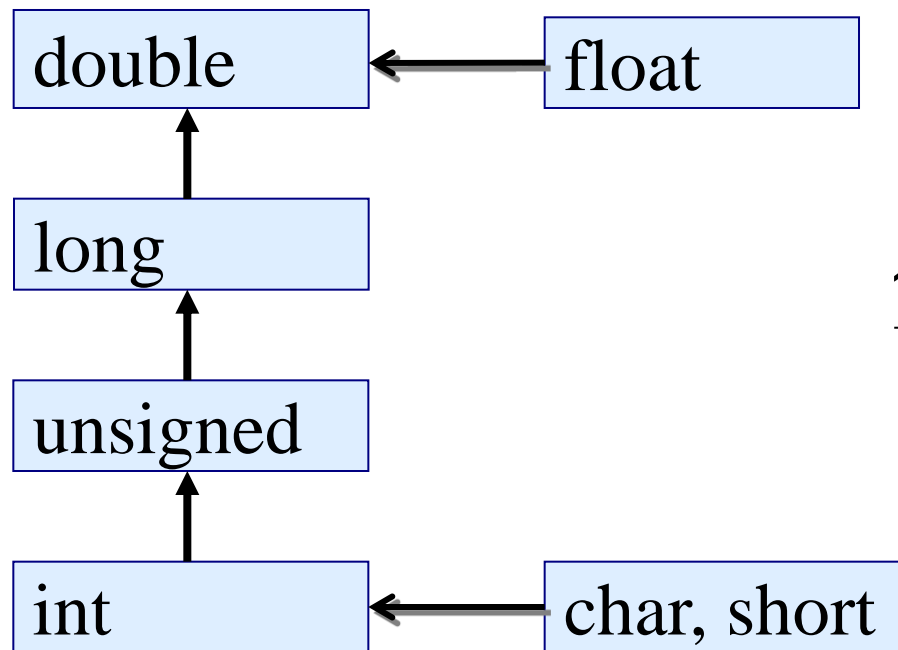
123%s + (i+'@') + i*u - f/d



不同类型之间的混合运算

如: $10 + 'a' + i * f - d / e$

其中 `int i;` `float f;` `double d;` `long e;`



- 数据转换向着**长类型**方向
- 转换总是逐个运算符进行

$$\begin{array}{ccccccc} 10 & + & 'a' & + & i & * & f & - & d/e \\ \hline & & | & & | & & | & & | \\ & & \text{int} & + & \text{double} & & & & \text{double} \\ & & | & & | & & & & | \\ & & \text{double} & - & \text{double} & & & & \\ \hline & & & & & & & & \text{double} \end{array}$$

自增1（减1）的运算

- $j = j + 1$; 等同于 $j++$; 或者 $++j$; 它们都可以作为单独的语句出现
- 增1（减1）只能用于（整型/字符型）变量

❖ $++j$: 表示 j 变量事先增1, 再参与其它运算, 例:

$m = ++j$; // 表示 j 先增1, 再将其结果赋给变量 m

如果 j 的值是3, 则上述运算后 m 是多少? $j=4 \ m=4$

❖ $j++$: 表示 j 先参与其它运算, 再增1; 例:

$m = j++$; // 表示 j 的值先赋给变量 m , 之后, 再将 j 增1

如果 j 的值是3, 则上述运算后 m 是多少? $j=4 \ m=3$

增1（减1）运算举例

■ i 的值为3，则

- ◆ $j = ++i$;
- ◆ $j = i++$;
- ◆ $\text{cout} << ++i$;
- ◆ $\text{cout} << i++$; 3

```
j=3;   k=++j;           // j=4; k=4;
j=3;   k=j++;           // j=??, k=?
j=3;   cout<<++j;       // ? ?
j=3;   cout<<j++;       // ? ?
a=3;b=5;c=(++a)*b;      //c= ? , a=?
a=3;b=5;c=(a++)*b;      //c=??, a=??
```

再看增1（减1）运算举例

编程网格结果

0	0
1	1
2	6
5	
3	2 3

```
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;
}
```

VC结果

1	0
2	0
4	6
4	
3	2 2

重载符 << 运算顺序从右到左，输出
顺序从左到右，平级括号右优先

关系运算

- 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

关系运算表达式与结果值

- 关系运算表达式的值：

“真” / “假”

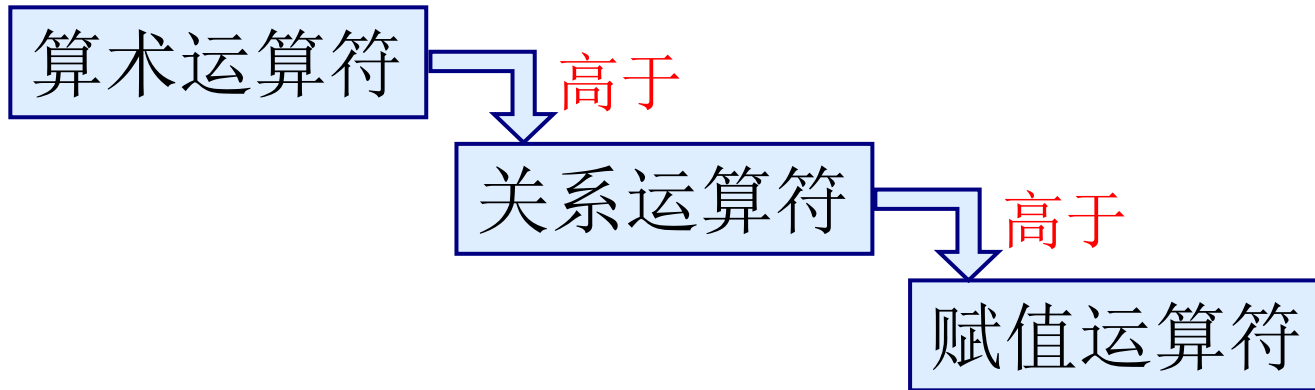
– 如 $a = 3$, $b = 4$

$a > b$ 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

- 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

逻辑表达式的值

- 逻辑表达式
 - 逻辑运算符之间按以下优先次序运算
 - $!(\text{非}) \rightarrow \&\&(\text{与}) \rightarrow \parallel(\text{或})$; 即 “!” 优先级最高

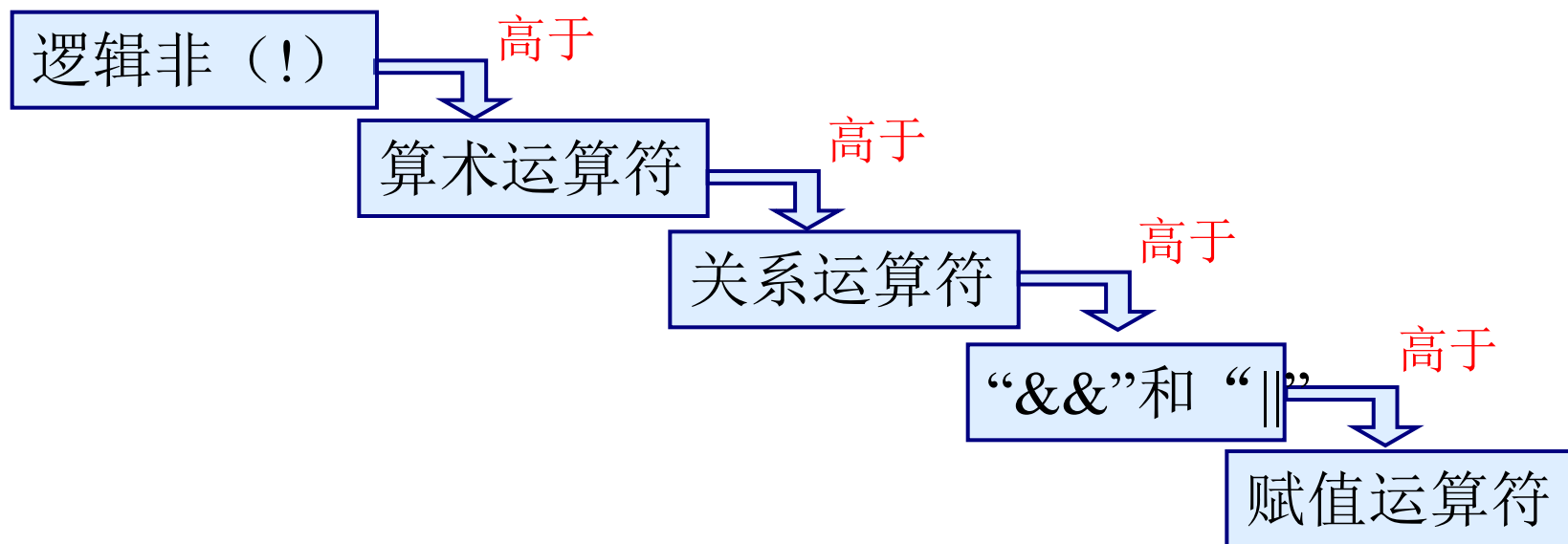
◆ 例如：

$!a \&\&b \parallel c$;

$!a \&\&b \parallel x > y \&\&c$;

$!a \&\&b \parallel x > y \&\&c + 1$;

混合运算优先级



- 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均为真(非0)时，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

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。

逗号, 运算符

- 格式: $e1, e2$
- 意义: 先计算 $e1$, 再计算 $e2$; 其中, $e2$ 的值和类型是整个逗号表达式 $e1, e2$ 的值和类型
- 可以有多个逗号运算, 总是以**最后一个**的值和类型为整个表达式的值和类型。
- 举例:

$j = (k+10, 100, 250*2);$ //结果就是 $j = 250*2$

$j = 3+5, j++, j*2$ // 首先, $j=8$, 其次 $j=9$; 最后, 整个表达式值为 $9*2=18$, 但 j 值为 9 (注意: 是三个表达式)。

少用逗号表达式!

● $x = (a = 3, 6 * 3);$

● $x = a = 3, 6 * 3;$

比较结果

条件表达式

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，**值？**
 - (float)5/3 将5的**值**转换成float类型，再运算，**值？**
- 注意：
 - 强制类型转换后，**被转换的量**的类型并没有发生变化

内容

➤ 赋值

➤ 运算与表达式

➤ 位运算

位运算（一般了解）

- 位运算
 - 所谓位运算是指进行二进制位的运算。
- C/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——按位与 (&)

可以对整数（包括字符型）进行位操作

- 运算规则

- 将两个运算量的每一个位进行逻辑与操作

- 举例：计算 $3 \& 5$ 结果为1

3:	0	0	0	0	0	0	1	1
5: (&)	0	0	0	0	0	1	0	1
<hr/>								
3 & 5:	0	0	0	0	0	0	0	1

- 用途：

- 将某一位清0，其它位不变。例如：(0376: 八进制)

- 将 char 型变量 a 的最低位清 0: $a = a \& 0376;$

- 取指定位。

- 例如：有 char c; int a;

- 取出 a 的低字节，置于 c 中: $c = a \& 0377;$

位运算——按位或 (|)

- 运算规则
 - 将两个运算量的每一位进行逻辑或操作
- 举例：计算 $3 \mid 5$ 结果为7

3: 0 0 0 0 0 0 1 1

5: (|) 0 0 0 0 0 1 0 1

3 | 5: 0 0 0 0 0 1 1 1

- 用途：
 - 将某些位置1，其它位不变。
例如：将 int 型变量 a 的低字节置 1：
 $a = a \mid 0xff;$

位运算——按位异或 (\wedge)

- 运算规则

- 两个操作数进行异或：

- 若对应位相同，则该位结果为 0，

- 若对应位不同，则该位结果为 1，

- 举例：计算 $071 \wedge 052$

071: 0 0 1 1 1 0 0 1

052: (\wedge) 0 0 1 0 1 0 1 0

$071 \wedge 052$: 0 0 0 1 0 0 1 1

位运算——按位异或 (\wedge)

- 用途：
 - 使特定位翻转：**相异为1**（与0异或，则保持原值；与1异或，则取反）

例如：要使 **01111010** 低四位翻转：

$$\begin{array}{r} 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\ (\wedge)\ \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 1} \\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1 \end{array}$$

位运算——取反(\sim)

单目运算符：对一个二进制数按位取反

例： 025: 00000000000010101

\sim 025: 11111111111101010

位运算——移位

- 左移运算 (<<)

左移后，低位补0，高位舍弃。

- 右移运算 (>>)

右移后，低位：舍弃

高位：无符号数：补0

有符号数：补“符号位”（与系统相关）

char a=-8

a>>2

11111000

11111110 值：-2 (除以4)

复合赋值位运算

- 复合表示

$\&=$, $|=$, $\ll=$, $\gg=$, $\wedge=$

- 例子:

注意：右边作为整体参与运算！

➤ $a \&= b$ 相当于 $a = a \& b$

➤ $a |= b$ 相当于 $a = a | b$

➤ $a \gg= 2$ 相当于 $a = a \gg 2$

➤ $a \ll= 2$ 相当于 $a = a \ll 2$

➤ $a \wedge= b$ 相当于 $a = a \wedge b$

再看优先顺序

- 算符优先级

- 取反运算符 “ \sim ”
- 算数运算符
- 左移 \ll 右移 \gg
- 关系运算符
- 按位与 $\&$
- 按位异或 \wedge
- 按位或 \mid
- 逻辑运算符

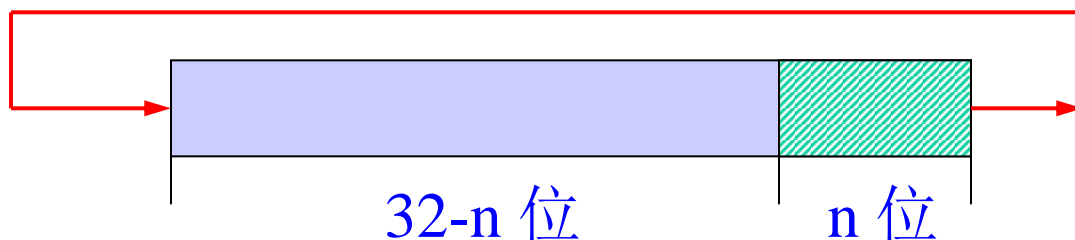
高

低



例子-1：循环右移n位

- 假设 `int` 类型由32位（4字节）表示，试将一个 `int` 类型的变量 `a` 表示的正值循环右移 `n` 位（按二进制移位）



- 先将 `a` 的右边 `n` 位移到 `b` 的最高位： $b = a \ll (32 - n)$
- 再将 `a` 右移 `n` 位，左边高位补0： $c = a \gg n$
- 再将 `c` 与 `b` 按位或运算： $b = b | c$ ；最后结果放在 `b`；

程序实现

```
#include <iostream>
using namespace std;
int main()
{ unsigned a,b,c;
  int n;
  cin>>a>>n;
  b=a<<(sizeof(a)*8-n); // sizeof 用于计算字节
  c=a>>n;
  b=c | b;
  cout<<b<<endl;
  return 0;
}
```