C++的表达式中分为右值和左值：左值和右值不以位置来区分

右值：当一个对象被用作右值时，用的是对象的值（内容）

左值：当一个对象被用作左值时，用的是对象的身份（在内存中的位置）

C++中若运算符优先级相同，遵守左结合律。赋值运算和非为特例，遵守右结合律。

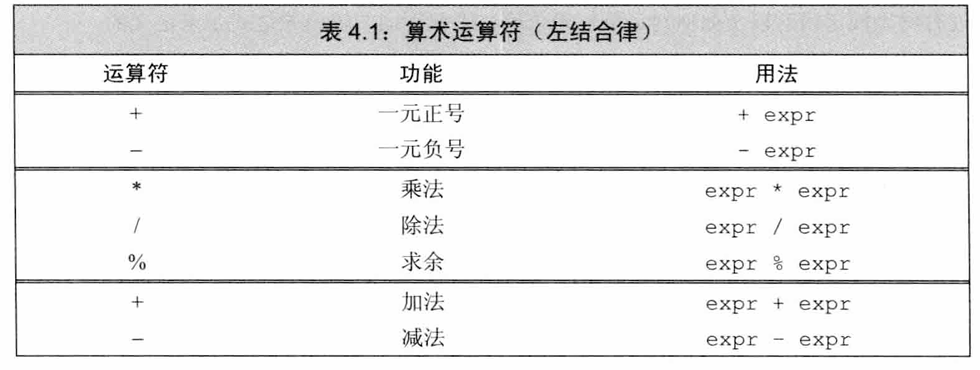
明确规定了运算顺序的运算符：&& > || > ? > ,

一个复合表达式中各个对象求值顺序是不确定的！

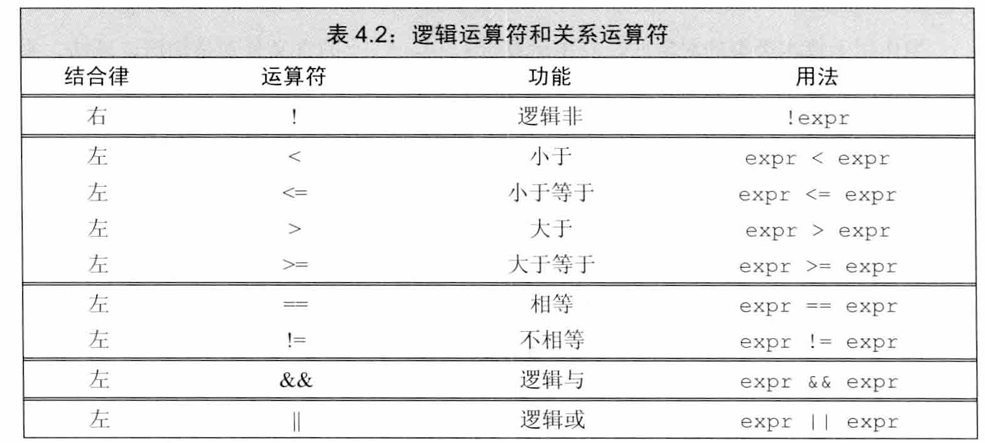
例如 i++ + ++i 这个东西是错误的，为未定义行为

大多数运算符之间都没有明确规定运算顺序，因此在不确定优先级与结合律时加括号（这样也能明确提升代码可读性），同一表达式中最好不要使用某个已被改变值的运算对象（例如 i++ 之后不要再出现改变 i 的值的行为）

算术运算符优先级分为三个档位，优先级从上到下依次下降：



逻辑运算符的五个档位，优先级依次下降：



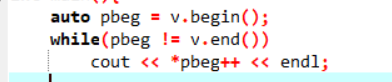
对于逻辑与和逻辑非，存在策略“短路求值”。对于逻辑与，在左边为真时才会运行右边的表达式；对于逻辑或，在左边为假时才会运行右边的表达式。由此可以知道实际上逻辑与和逻辑或中隐含了一个if-else语句。

ps：对于bool类型的字面值，false转换为int时会为0，true转换为int则是1。并非某些语言中true代表非0，而false代表0，不要将算数值和bool作比较。

赋值运算符可以连用，并且满足右结合律，

比如a = b = 1;这句代码是正确的，b先被赋值为1，而后a被赋值为b的值。

在一条语句中混用解引用符和递增运算符：



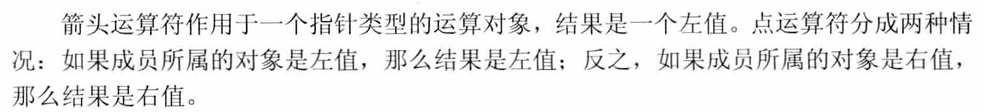
pbeg++将 pbeg的值加一，然后返回 pbeg的初始值的副本作为求值结果用于解引用。(递增运算符优先级高于解引用，这在形式上其实显得反直觉)

ps：关于递增运算符，如果没有向上面这样特殊的需求，尽量使用前置版本，比如++i，因为前置版本直接返回递增后的对象，而后置版本会复制一个递增前的对象的副本并返回。

. 和 ->这两个运算符都能用于访问对象的实例，但是

image.png

ps：解引用运算符优先级低于点运算符



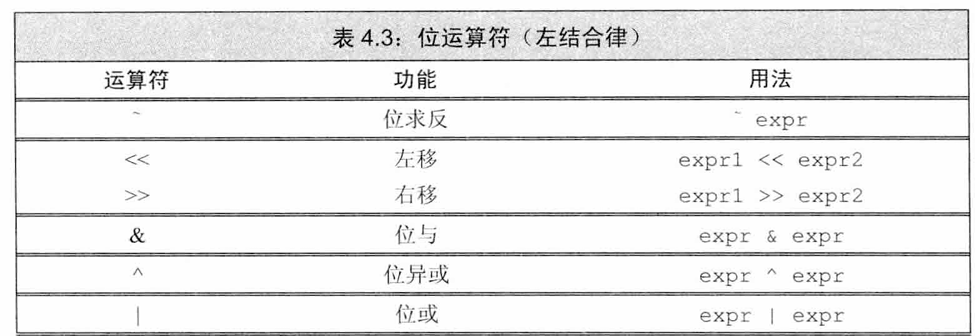
即对于点运算符，如果为被赋值的一方则为左值，为赋值的一方则为右值（例如new Point(x,y),x=5中，实际上隐含了一个赋值号，创建的这个变量为临时变量，在右边理应有一个变量作为左值，因此此刻点运算符的结果为右值）



image.png

尝试使用箭头运算符来访问一个非指针类型的对象的成员。箭头运算符作用于指针，所以只能作为左值。

位运算符，五档优先级：



关于左移和右移，实际上是在尾数间移动小数点，然后改变阶码，下面顺手贴个浮点数和32位整数的存储方式

int32按补码存储

浮点数：



类型转换：

隐式类型转换发生的场景：

1.比 int 类型小的整型值首先提升为较大的整数类型

2.在条件里，把非布尔值转换成布尔值

3.初始化过程中，初始值转换为变量的类型

4.赋值时，右侧运算对象转换成左侧类型

5.算数运算或关系运算的运算对象有多种类型，转换成一种

算术转换：

1.整型提升：把小整数类型（包括 char、bool等）转换成较大的整数类型。如果 int 可以就转换成 int，否则提升成 unsigned int 类型。

2.无符号类型的运算对象：如果一个运算对象为无符号类型，另一个为有符号类型。若无符号类型变量大于有符号类型变量，那么将带符号的转换为无符号的类型（此时若带符号的为负数，会下溢）；若带符号类型变量小于无符号类型变量，如果无符号类型变量存在于带符号类型变量的区间中，无符号类型会转为带符号的类型，反之带符号类型仍会转换成无符号类型。

其他类型的隐式转换：

数组转换成指针：大多数情况下数组自动转换成指向数组首元素的指针。（decltype关键字参数、取地址符(&)、sizeof、typeid 都不会发生这种转换）

指针的转换：0 或 nullptr 都能转换成任意指针类型。指向非常量的指针能转换成 void\*。指向所有对象的指针都能转换成 const void\*。

转换成常量：指向非常量的指针转换成指向相应常量类型的指针（常量指针，底层const）

显式转换（强制转换）：

强制转换的五种方式：强制类型转换，static\_cast、dynamic\_cast、const\_cast、reinterpret\_cast 。

强制类型转换：

image.png

static\_cast：任何类型转换，只要不包含底层 const，都可以用 static\_cast。当把较大的类型转换为较小的类型时，static\_cast 很有用。static\_cast告诉编译器，知道存在精度损失并且不在乎精度损失。编译器在显式转换后就不再警告，在编译时会进行类型检查。

image.png

const\_cast:const\_cast 只能改变对象的底层 const。可以去掉或增加 const 性质。只有const\_cast 能改变表达式的常量属性。

image.png

reinterpret\_cast：通常用于指针类型之间的转换，它不进行任何类型检查，只改变指针的指向，使用时需谨慎。

dynamic\_cast：主要用于将基类指针或引用转换为派生类指针或引用，在运行时进行类型检查，以确保转换的安全性。