C 语言运算符优先级及结合性明细表

优先级	运算符	功能	目数	结合性
1	（）	改变运算优先级或函数调用	双目	从左向右
	[]	访问数组元素		
	.	直接访问数据成员		
	－＞	间接访问数据成员		
2	！	逻辑非	单目	从右向左
	~	按位取反		
	+,－	取正，取负		
	*	间接访问对象		
	&	取对象地址		
	＋＋，－－	增 1，减 1		
	（）	强制类型转换		
	sizeof	测类型长度		
3	*,/,%	乘，除，取余	双目	从左向右
4	＋，－	加，减		
5	<<,>>,>>>	按位左移，按位右移，无符号右移		
6	＜，<=，＞，>=	小于，小于等于，大于，大于等于		
7	＝＝,！＝	等于，不等于		
8	&	按位与		
9	^	按位异或		
10	|	按位或		
11	&＆	逻辑与		
12	||	逻辑或		
13	？：	条件运算符	三目	从右向左
14	＝	赋值	双目	从右向左
	＋＝，－＝	加赋值，减赋值		
	*＝，/＝	乘赋值，除赋值		
	%＝，&＝	取余赋值、按位与赋值		
	^=	按位异或赋值		
	|＝	按位或赋值		
	＜＜=	按位左移赋值		
	＞＞=	按位右移赋值		
15	，	逗号运算符	双目	从左向右

C++ 操作符的优先级

优先级	运算符	叙述	示例	重载性	结合性
1	::	全局作用域	::name	否	由左至右
	::	类作用域	class::name		
	::	名字空间作用域	namespace::name		

2	++	后缀递增	i++		
	--	后缀递减	i--		
	{}	组合	{i++;a*=i;}		
	()	函数调用或变量初始化	c_tor(int x, int y) : _x(x), _y(y * 10) {}		
	[]	数组访问	array[4] = 2;		
	.	以对象方式访问成员	obj.age = 34;	否	
	->	以指针方式访问成员	ptr->age = 34;		
	dynamic_cast	运行时检查类型转换(C++专有)	Y& y = dynamic_cast<Y&>(x);	否	
	static_cast	未经检查的类型转换(C++专有)	Y& y = static_cast<Y&>(x);	否	
	reinterpret_cast	重定义类型转换(C++专有)	int const* p = reinterpret_cast<int const*>(0x1234);	否	
	const_cast	更改非常量属性(C++专有)	int* q = const_cast<int*>(p);	否	
	typeid	获取类型信息(C++专有)	std::type_info const& t = typeid(x);	否	
3	++	前缀递增	++i		由右至左
	--	前缀递减	--i		
	+	一元正号	int i = +1;		
	-	一元负号	int i = -1;		
	! not	逻辑非 !的备用拼写	if (!done) …		
	~ compl	按位取反 ~的备用拼写	flag1 = ~flag2;		
	(*type*)	转换为给定的类型	int i = (int)floatNum;		
	*	取指针指向的值	int data = *intPtr;		
	&	某某的地址(参考)	int *intPtr = &data;		
	sizeof	某某的大小	size_t s = sizeof(int);	否	
	new	动态内存分配(C++专有)	long* pVar = new long;		
	new[]	动态数组内存分配(C++专有)	long* array = new long[20];		
	delete	动态内存释放(C++专有)	delete pVar;		
	delete[]	动态数组内存释放(C++专有)	delete [] array;		
4	.*	成员对象选择(C++专有)	obj.*var = 24;	否	由左至右
	->*	成员指针选择(C++专有)	ptr->*var = 24;		
5	*	乘法	int i = 2 * 4;		
	/	除法	float f = 10.0 / 3.0;		
	%	模数(取余)	int rem = 4 % 3;		
6	+	加法	int i = 2 + 3;		
	-	减法	int i = 5 - 1;		
7	<<	位左移	int flags = 33 << 1;		

	>>	位右移	int flags = 33 >> 1;		
8	<	小于关系	if (i < 42) …		
	<=	小于等于关系	if (i <= 42) ...		
	>	大于关系	if (i > 42) …		
	>=	大于等于关系	if (i >= 42) ...		
9	== eq	等于关系 ==的备用拼写	if (i == 42) ...		
	!= not_eq	不等于关系 !=的备用拼写	if (i != 42) …		
10	& bitand	位 AND &的备用拼写	flag1 = flag2 & 42;		
11	^ xor	位 XOR(独占 or) ^的备用拼写	flag1 = flag2 ^ 42;		
12	| bitor	位 OR(包含 or) |的备用拼写	flag1 = flag2 | 42;		
13	&& and	逻辑 AND &&的备用拼写	if (conditionA && conditionB) …		
14	|| or	逻辑 OR ||的备用拼写	if (conditionA || conditionB) ...		
15	c?t:f	三元条件运算	int i = a > b ? a : b;	否	由右至左
16	=	直接赋值	int a = b;		
	+=	以和赋值	a += 3;		
	-=	以差赋值	b -= 4;		
	*=	以乘赋值	a *= 5;		
	/=	以除赋值	a /= 2;		
	%=	以取余数赋值	a %= 3;		
	<<=	以位左移赋值	flags <<= 2;		
	>>=	以位右移赋值	flags >>= 2;		
	&= and_eq	以位 AND 赋值 &=的备用拼写	flags &= new_flags;		
	^= xor_eq	以位 XOR 赋值 ^=的备用拼写	flags ^= new_flags;		
	|= or_eq	以位 OR 赋值 |=的备用拼写	flags |= new_flags;		
17	throw	抛出异常	throw EClass(“Message”);	否	
18	,	逗号	for (i = 0, j = 0; i < 10; i++, j++) …		由左至右

详解：

符中它们具有最高的优先级，又由于它们都是从右至左结合的，因此*p++与*(p++)等效是毫无疑问的。

C 语言运算符优先级顺口溜[转]

醋坛酸味灌，味落跳福豆。 （共 44 个运算符）

醋－初等，4 个： ( ) [ ] -> 指向结构体成员 . 结构体成员

坛－单目，9 个： ! ~ ++ -- -负号 (类型)*指针 &取地址 sizeof 长度 （自右向左）

酸－算术，5 个： * / % + -减

味－位移，2 个： << >>

灌－关系，6 个： < <= > >= == 等于 != 不等于

味－位逻，3 个： & 按位与 ^ 按位异或 | 按位或

落－逻辑，2 个： && 逻辑与 || 逻辑或

跳－条件，1 个，三目： ? : （结合方向：自右向左）

福－赋值，11 个： = += -= *= /= %= >>= <<= &= ^= |= （结合方向：自右向左）

豆－逗号，1 个： ,

结合方向自右向左的只有三类：赋值、单目和三目，其它的都是从左至右结合。

注意：

自己今天写程序时候把**关系运算符号和位逻辑的优先级弄反**。尽量用括号，就不出问题。如：if（P1OUT&BIT0==0）P1OUT ^=BIT0; //判断 P1OUT 是否输出为 0，是就翻转，即点亮 LED.这行程序是错误的。可两边加上括号，即 if（（P1OUT&BIT0）==0），或是直接用：P1OUT|=BIT0；

C 语言中，只有 4 个运算符规定了运算方向，它们是&&、| |、条件运算符及赋值运算符。&&、| |都是先计算左边表达式的值，当左边表达式的值能确定整个表达式的值时，就不计算右边表达式的值。如 a = 0 && b; &&运算符的左边位 0，则右边表达式 b 就不再判断。在条件运算符中。如 a?b:c；先判断 a 的值，再根据 a 的值对 b 或 c 之中的一个进行求值。赋值表达式则规定先对右边的表达式求值，因此使 a = b = c = 6;成为可能。