

OPERATORS : operator precedence table, using masks

OPERATOR PRECEDENCE TABLE			
::	scope resolution	class_name::member	
::	global	::name	
.	member selection	object.member	
->	member selection	pointer->member	
[]	subscripting	pointer[ expr ]	
()	function call	expr( expr_list )	
++	post increment	lvalue++	
--	post decrement	lvalue--	
sizeof	size of object	sizeof expr	
sizeof	size of type	sizeof( type )	
++	pre increment	++lvalue	
--	pre decrement	--lvalue	
~	complement	~expr (bitwise negation)	
!	not	!expr	
-	unary minus	-expr	
+	unary plus	+expr	
&	address of	&lvalue	
*	dereference	*expr (value at address)	
new	create (allocate)	new type	
delete	destroy (de-allocate)	delete pointer	
delete[]	destroy array	delete[] pointer	
()	cast (type conversion)	( type )expr	
.*	member selection	object.*pointer-to-member	
->*	member selection	pointer->*pointer-to-member	
*	multiply	expr * expr	
/	divide	expr / expr	
%	modulo (remainder)	expr % expr	
+	add (plus)	expr + expr	
-	subtract (minus)	expr - expr	
<<	shift left	expr << expr	
>>	shift right	expr >> expr	
<	less than	expr < expr	
<=	less than or equal to	expr <= expr	
>	greater than	expr > expr	
>=	greater than or equal to	expr >= expr	
==	equal to	expr == expr	
!=	not equal to	expr != expr	
&	bitwise AND	expr & expr	
^	bitwise exclusive OR	expr ^ expr one or the other, but not both	
	bitwise inclusive OR	expr   expr	
&&	logical AND	expr && expr	

	logical inclusive OR	expr    expr	
-----			
? :	conditional expression	expr ? expr : expr	
-----			
=	simple assignment	expr = expr	
*=	multiply and assign	expr *= expr	
/=	divide and assign	expr /= expr	
%=	modulo and assign	expr %= expr	
+=	add and assign	expr += expr	
-=	subtract and assign	expr -= expr	
<<=	shift left and assign	expr <<= expr	
>>=	shift right and assign	expr >>= expr	
&=	AND and assign	expr &= expr	
=	inclusive OR and assign	expr  = expr	
^=	exclusive OR and assign	expr ^= expr	
-----			
throw	throw exception	throw expr	
-----			
,	comma (sequencing)	expr, expr	
-----			
=====			

## USING MASKS

You can use the bitwise AND and OR operators to determine whether certain bits are 'on' or 'off', or to turn certain bits 'on' or 'off'. Here's an example:

```
// bits.cp
#include <iostream.h>

#define kBit1 01
#define kBit2 02
#define kBit3 04
#define kBit4 010
#define kBit5 020

void myCheckBits( short item );

void main( void )

    short flags = 0;

    // turn bits on
    flags = flags | kBit2;
    flags = flags | kBit4;

    myCheckBits( flags );

void myCheckBits( short item )

    // check if bits are on
    if( item & kBit1 )
        cout << "Bit 1 is on." << endl;
    if( item & kBit2 )
        cout << "Bit 2 is on." << endl;
    if( item & kBit3 )
        cout << "Bit 3 is on." << endl;
    if( item & kBit4 )
        cout << "Bit 4 is on." << endl;
    if( item & kBit5 )
```

```
    cout << "Bit 5 is on." << endl;  
// end bits.cp
```