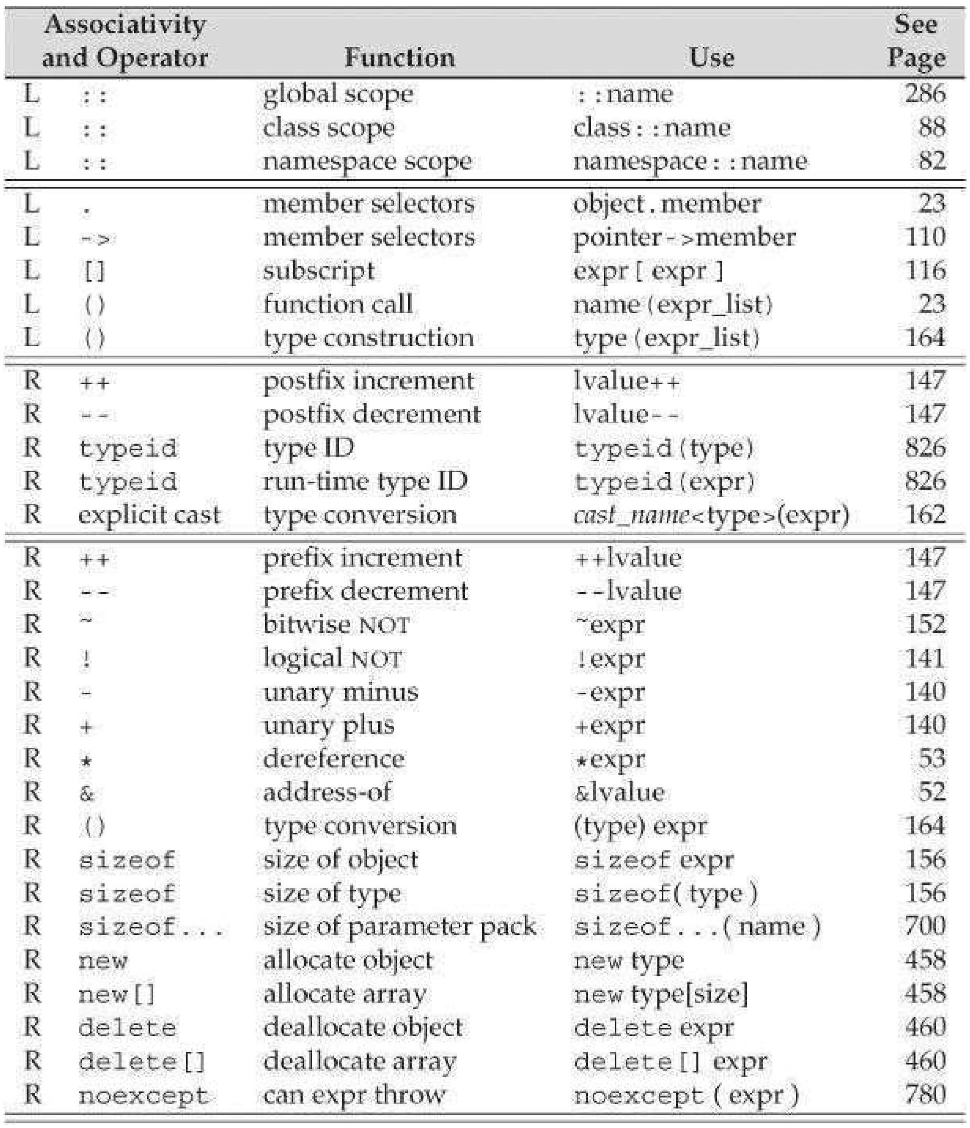
4.12. Operator Precedence Table

Table 4.4. Operator Precedence



Associativity and Operator		Function	Use	See Page
L	`::`	global scope	`::`name	286
L	`::`	class scope	class`::`name	88
L	`::`	namespace scope	namespace`::`name	82
L	`.`	member selectors	object`.`member	23
L	`->`	member selectors	pointer`->`member	110
L	`[]`	subscript	expr`[` expr `]`	116
L	`()`	function call	name`(`expr_list`)`	23
L	`()`	type construction	type`(`expr_list`)`	164
R	`++`	postfix increment	lvalue`++`	147
R	`--`	postfix decrement	lvalue`--`	147
R	`typeid`	type ID	`typeid(`type`)`	826
R	`typeid`	run-time type ID	`typeid(`expr`)`	826
R	explicit cast	type conversion	*cast_name*<type>(expr)	162
R	`++`	prefix increment	`++`lvalue	147
R	`--`	prefix decrement	`--`lvalue	147
R	`~`	bitwise NOT	`~`expr	152
R	`!`	logical NOT	`!`expr	141
R	`-`	unary minus	`-`expr	140
R	`+`	unary plus	`+`expr	140
R	`*`	dereference	`*`expr	53
R	`&`	address-of	`&`lvalue	52
R	`()`	type conversion	(type) expr	164
R	`sizeof`	size of object	`sizeof` expr	156
R	`sizeof`	size of type	`sizeof(` type `)`	156
R	`sizeof...`	size of parameter pack	`sizeof...(` name `)`	700
R	`new`	allocate object	`new` type	458
R	`new[]`	allocate array	`new` type[size]	458
R	`delete`	deallocate object	`delete` expr	460
R	`delete[]`	deallocate array	`delete[]` expr	460
R	`noexcept`	can expr throw	`noexcept (` expr `)`	780

L	->*	ptr to member select	ptr->*ptr_to_member	837
L	.*	ptr to member select	obj.*ptr_to_member	837
L	*	multiply	expr * expr	139
L	/	divide	expr / expr	139
L	%	modulo (remainder)	expr % expr	139
L	+	add	expr + expr	139
L	-	subtract	expr - expr	139
L	<<	bitwise shift left	expr << expr	152
L	>>	bitwise shift right	expr >> expr	152
L	<	less than	expr < expr	141
L	<=	less than or equal	expr <= expr	141
L	>	greater than	expr > expr	141
L	>=	greater than or equal	expr >= expr	141
L	==	equality	expr == expr	141
L	!=	inequality	expr != expr	141
L	&	bitwise AND	expr & expr	152
L	^	bitwise XOR	expr ^ expr	152
L	|	bitwise OR	expr | expr	152
L	&&	logical AND	expr && expr	141
L	||	logical OR	expr || expr	141
R	?:	conditional	expr ? expr : expr	151
R	=	assignment	lvalue = expr	144
R	*=, /=, %=,	compound assign	lvalue += expr, etc.	144
R	+=, -=,			144
R	<<=, >>=,			144
R	&=, |=, ^=			144
R	throw	throw exception	throw expr	193
L	,	comma	expr , expr	157

Chapter Summary

C++ provides a rich set of operators and defines their meaning when applied to values of the built-in types. Additionally, the language supports operator overloading, which allows us to define the meaning of the operators for class types. We'll see in Chapter 14 how to define operators for our own types.

To understand expressions involving more than one operator it is necessary to understand precedence, associativity, and order of operand evaluation. Each operator has a precedence level and associativity. Precedence determines how operators are grouped in a compound expression. Associativity determines how operators at the same precedence level are grouped.

Most operators do not specify the order in which operands are evaluated: The compiler is free to evaluate either the left- or right-hand operand first. Often, the order of operand evaluation has no impact on the result of the expression. However, if