Arithmetic operators [edit]

Operator name		Syntax	Can overload	Included in C	Prototype examples	
					As member of K	Outside class definitions
Basic assignment		`a = b`	Yes	Yes	`R& K::operator =(S b);`	N/A
Addition		`a + b`	Yes	Yes	`R K::operator +(S b);`	`R operator +(K a, S b);`
Subtraction		`a - b`	Yes	Yes	`R K::operator -(S b);`	`R operator -(K a, S b);`
Unary plus (integer promotion)		`+a`	Yes	Yes	`R K::operator +();`	`R operator +(K a);`
Unary minus (additive inverse)		`-a`	Yes	Yes	`R K::operator -();`	`R operator -(K a);`
Multiplication		`a * b`	Yes	Yes	`R K::operator *(S b);`	`R operator *(K a, S b);`
Division		`a / b`	Yes	Yes	`R K::operator /(S b);`	`R operator /(K a, S b);`
Modulo (integer remainder)[a]		`a % b`	Yes	Yes	`R K::operator %(S b);`	`R operator %(K a, S b);`
Increment	Prefix	`++a`	Yes	Yes	`R& K::operator ++();`	`R& operator ++(K a);`
	Postfix	`a++`	Yes	Yes	`R K::operator ++(int);`	`R operator ++(K a, int);`
					Note: C++ uses the unnamed dummy-parameter `int` to differentiate between prefix and suffix increment operators.	
Decrement	Prefix	`--a`	Yes	Yes	`R& K::operator --();`	`R& operator --(K a);`
	Postfix	`a--`	Yes	Yes	`R K::operator --(int);`	`R operator --(K a, int);`
					Note: C++ uses the unnamed dummy-parameter `int` to differentiate between prefix and suffix decrement operators.	

Comparison operators/relational operators [edit]

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Equal to	`a == b`	Yes	Yes	`bool K::operator ==(S const& b);`	`bool operator ==(K const& a, S const& b);`
Not equal to	`a != b` `a not_eq b` [b]	Yes	Yes	`bool K::operator !=(S const& b);`	`bool operator !=(K const& a, S const& b);`
Greater than	`a > b`	Yes	Yes	`bool K::operator >(S const& b);`	`bool operator >(K const& a, S const& b);`
Less than	`a < b`	Yes	Yes	`bool K::operator <(S const& b);`	`bool operator <(K const& a, S const& b);`
Greater than or equal to	`a >= b`	Yes	Yes	`bool K::operator >=(S const& b);`	`bool operator >=(K const& a, S const& b);`
Less than or equal to	`a <= b`	Yes	Yes	`bool K::operator <=(S const& b);`	`bool operator <=(K const& a, S const& b);`

Logical operators [edit]

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Logical negation (NOT)	`!a` `not a` [b]	Yes	Yes	`R K::operator !();`	`R operator !(K a);`
Logical AND	`a && b` `a and b` [b]	Yes	Yes	`R K::operator &&(S b);`	`R operator &&(K a, S b);`
Logical OR	`a || b` `a or b` [b]	Yes	Yes	`R K::operator ||(S b);`	`R operator ||(K a, S b);`

Bitwise operators [edit]

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Bitwise NOT	`~a` `compl a` [b]	Yes	Yes	`R K::operator ~();`	`R operator ~(K a);`
Bitwise AND	`a & b` `a bitand b` [b]	Yes	Yes	`R K::operator &(S b);`	`R operator &(K a, S b);`
Bitwise OR	`a | b` `a bitor b` [b]	Yes	Yes	`R K::operator |(S b);`	`R operator |(K a, S b);`
Bitwise XOR	`a ^ b` `a xor b` [b]	Yes	Yes	`R K::operator ^(S b);`	`R operator ^(K a, S b);`
Bitwise left shift[c]	`a << b`	Yes	Yes	`R K::operator <<(S b);`	`R operator <<(K a, S b);`
Bitwise right shift[c][d]	`a >> b`	Yes	Yes	`R K::operator >>(S b);`	`R operator >>(K a, S b);`

Compound assignment operators [edit]

Operator name	Syntax	Meaning	Can overload	Included in C	Prototype examples	
					As member of K	Outside class definitions
Addition assignment	`a += b`	`a = a + b`	Yes	Yes	`R& K::operator +=(S b);`	`R& operator +=(K a, S b);`
Subtraction assignment	`a -= b`	`a = a - b`	Yes	Yes	`R& K::operator -=(S b);`	`R& operator -=(K a, S b);`
Multiplication assignment	`a *= b`	`a = a * b`	Yes	Yes	`R& K::operator *=(S b);`	`R& operator *=(K a, S b);`
Division assignment	`a /= b`	`a = a / b`	Yes	Yes	`R& K::operator /=(S b);`	`R& operator /=(K a, S b);`
Modulo assignment	`a %= b`	`a = a % b`	Yes	Yes	`R& K::operator %=(S b);`	`R& operator %=(K a, S b);`
Bitwise AND assignment	`a &= b` `a and_eq b` [b]	`a = a & b`	Yes	Yes	`R& K::operator &=(S b);`	`R& operator &=(K a, S b);`
Bitwise OR assignment	`a |= b` `a or_eq b` [b]	`a = a | b`	Yes	Yes	`R& K::operator |=(S b);`	`R& operator |=(K a, S b);`
Bitwise XOR assignment	`a ^= b` `a xor_eq b` [b]	`a = a ^ b`	Yes	Yes	`R& K::operator ^=(S b);`	`R& operator ^=(K a, S b);`
Bitwise left shift assignment	`a <<= b`	`a = a << b`	Yes	Yes	`R& K::operator <<=(S b);`	`R& operator <<=(K a, S b);`
Bitwise right shift assignment[d]	`a >>= b`	`a = a >> b`	Yes	Yes	`R& K::operator >>=(S b);`	`R& operator >>=(K a, S b);`

Member and pointer operators [edit]

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Array subscript	`a[b]`	Yes	Yes	`R& K::operator [](S b);`	N/A
Indirection ("object pointed to by *a*")	`*a`	Yes	Yes	`R& K::operator *();`	`R& operator *(K a);`
Address ("address of *a*")	`&a`	Yes	Yes	`R K::operator &();`	`R operator &(K a);`
Structure dereference ("member *b* of object pointed to by *a*")	`a->b`	Yes	Yes	`R* K::operator ->();`[e]	N/A
Structure reference ("member *b* of object *a*")	`a.b`	No	Yes	N/A	
Member pointed to by *b* of object pointed to by *a*[f]	`a->*b`	Yes	No	`R& K::operator ->*(S b);`	`R& operator ->*(K a, S b);`
Member pointed to by *b* of object *a*	`a.*b`	No	No	N/A	

Other operators [edit]

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Function call *See Function object.*	`a(a1, a2)`	Yes	Yes	`R K::operator ()(S a, T b, ...);`	N/A
Comma	`a, b`	Yes	Yes	`R K::operator ,(S b);`	`R operator ,(K a, S b);`
Ternary conditional	`a ? b : c`	No	Yes	N/A	
Scope resolution	`a::b`	No	No	N/A	
User-defined literals[g] *since C++11*	`"a"_b`	Yes	No	N/A	`R operator "" _b(T a)`
Size-of	`sizeof (a)` [h] `sizeof (type)`	No	Yes	N/A	
Size of parameter pack *since C++11*	`sizeof...(Args)`	No	No	N/A	
Align-of *since C++11*	`alignof (type)` or `_Alignof (type)` [i]	No	Yes	N/A	
Type identification	`typeid (a)` `typeid (type)`	No	No	N/A	
Conversion (C-style cast)	`(type) a` `type(a)`	Yes	Yes	`K::operator R();` Note: for user-defined conversions, the return type implicitly and necessarily matches the operator name.	N/A
static_cast conversion	`static_cast<type>(a)`	No	No	N/A	
dynamic_cast conversion	`dynamic_cast<type>(a)`	No	No	N/A	
const_cast conversion	`const_cast<type>(a)`	No	No	N/A	
reinterpret_cast conversion	`reinterpret_cast<type>(a)`	No	No	N/A	
Allocate storage	`new type`	Yes	No	`void* K::operator new(size_t x);`	`void* operator new(size_t x);`
Allocate storage (array)	`new type[n]`	Yes	No	`void* K::operator new[](size_t a);`	`void* operator new[](size_t a);`
Deallocate storage (*delete* returns *void* so it isn't strictly speaking an operator)	`delete a`	Yes	No	`void K::operator delete(void *a);`	`void operator delete(void *a);`
Deallocate storage (array)	`delete[] a`	Yes	No	`void K::operator delete[](void *a);`	`void operator delete[](void *a);`
Exception check *since C++11*	`noexcept(a)`	No	No	N/A	