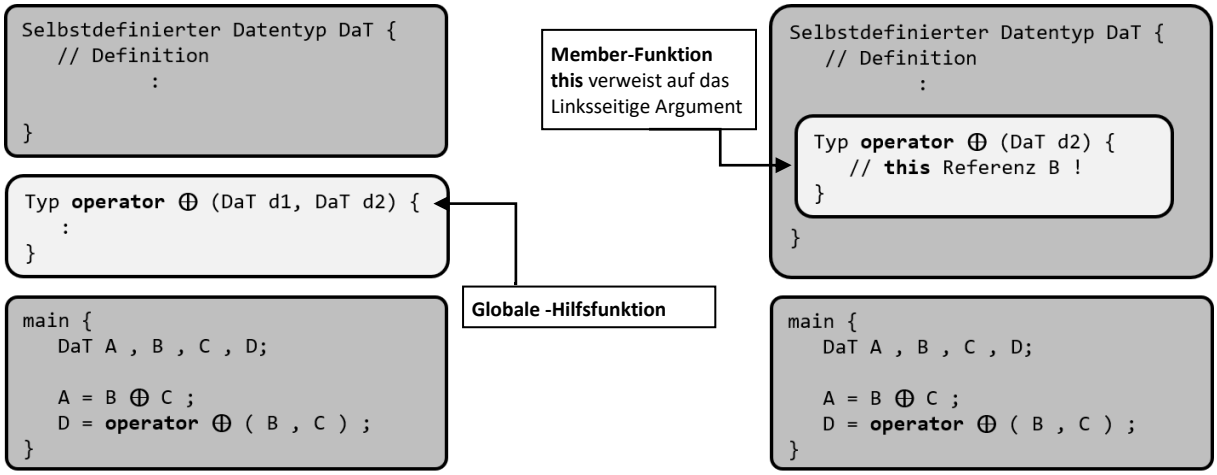
In C++ können Operatoren auf Objekte von selbstdefinierten Datentypen (z.B.: struct, class, …) als Funktion (**operator**-Funktion) dargestellt werden. Somit kann die **operator**-Funktion (wie alle anderen Funktionen auch) überladen werden. Operator-Funktionen können als Element eines selbstdefinierten Typs (Member-Funktion) oder außerhalb einer Typ-Definition (globale Hilfsfunktion) implementiert werden.

```
// ⊕ ist ein beliebiger Operator (+,-,*,…)
// A,B,C sind selbstdefinierte Typen
// dann sind beide Anweisungen identisch
C = A ⊕ B ;
C = operator ⊕ ( A , B ) ;
// C = A + B  ist identisch mit C = + ( A , B )
```

Es können alle C++-Operatoren überladen werden, mit folgenden Ausnahmen und Einschränkungen:

- Folgende Operatoren dürfen nur als Member-Funktion überladen werden: ' = ', ' -> ', ' () ', ' [] ', ' ->* ' und die Konvertierungsoperatoren sowie klassenspezifische Operatoren zur Speicherverwaltung.
- Folgende Operatoren dürfen gar nicht überladen werden: ' ?: ', ' :: ', ' . ', ' .* ', typeid , sizeof und die C++-Cast-Operatoren.
- Mindestens ein Operand muss ein nutzerdefinierter Datentyp sein.

Schematische Darstellung beider Implementierungsarten für binären Operatoren:



Regeln:

- Es nicht möglich, die Operatoren bei den Basisdatentypen zu überladen.
- Es lassen sich keine neuen Operatoren damit erzeugen, es können also nur Operatoren überladen werden, die bereits existieren. Neue Operatoren-Symbole lassen sich hiermit nicht einführen.
- Die Operanden eines Operators können nicht verändert werden. Ein binärer Operator hat nach wie vor zwei Operanden und ein unärer einen.
- Auch die Priorität der Operatoren verändert sich nicht. Zum Beispiel: Der Operator * besitzt immer noch eine höhere Priorität als der Operator + (Punkt-vor-Strich-Regelung).
- Operatoren dürfen außerdem keine Standardargumente enthalten und müssen dieselbe Anzahl an Argumenten wie der ursprüngliche Operator haben.

Operatoren Übersicht (Auszug):

Art	Ausdruck	Operatoren	Member - Funktion	Globale Hilfsfunktion
unär	⊕ a	+ - ! ~	operator ⊕ ()	operator ⊕ (A)
unär	⊕ a	++ -- (präfix)	operator ⊕ ()	operator ⊕ (A)
unär	a ⊕	++ -- (postfix)	operator ⊕ (int)	operator ⊕ (A , int)
binär	a ⊕ b	+ - * / == !=	operator ⊕ (B)	operator ⊕ (A , B)
Zuweisung	a = b	=	operator = (B)	nicht möglich
Stream	cout << a	<<	nicht üblich	operator << (Stream , A)