Zusammenfassung Operatorüberladung 1

Tabelle der Operatoren in C++

+	-	*	/	%	^	&	|
~	!	,	=	<	>	<=	>=
++	--	<<	>>	==	!=	&&	||
+=	-=	*=	/=	%=	^=	&=	|=
<<=	>>=	[ ]	( )	->	->*	new	delete

Überladung ist nicht erlaubt für die Operatoren

.	.*	::	?:	sizeof

- Es ist nicht möglich, neue Operatoren zu definieren.
- Operatorprioritäten bleiben erhalten
- Binäre und unäre Operatoren können nur auf ihre Weise verwendet werden (Vorzeichenoperatoren können nicht als gemeine Operatoren verwendet werden).
- Die Assoziativität kann nicht verändert werden.
- Die Bedeutung von Operatoren bezüglich eingebauter Datentypen lässt sich nicht ändern, Operatoren können nur überladen werden, wenn mindestens ein Parameter (Operand) eines benutzerdefinierten Typs existiert.
- Die Operatoren () und [] können nur als Member- nicht als Friend-Funktionen überladen werden.
- Die Operatorüberladung sollte immer in einer Weise erfolgen, daß die Bedeutung des Operators eindeutig ist.

- Deklarationen von Operatoren in Klassedeklarationen ('@' steht für Operator)
  1) erg_typ operator @ (); // unärer Operator
  2) erg_typ operator @ (arg_typ arg) // binärer Operator
  3) friend erg_typ operator @ (class_typ [&] arg) // unärer Operator als Friend, der Operandentyp // muss der Typ der Klasse, die die Deklaration // enthält sein
  4) friend erg_typ operator @ (class_typ [&] arg1, arg_typ arg2) // binärer Operator als Friend Variante 1
  5)friend erg_typ operator @ (arg_typ arg1, class_typ [&] arg2) // binärer Operator als Friend Variante 2
- Variante 4 und 5 entsprechen den Ausdrücken A@B und B@A (Herstellen von Kommutativität).
- Um einen Operator mit einer Nichtmemberfunktion (friend-Funktion) zu überladen, benötigt man eine Funktion mit 2 Parametern.
- Zu beachten ist, dass unäre Operatoren als friend und binäre Operatoren zu gleichen Parameterlisten führen können.
- Ein unärer Operator als Memberfunktion hat keinen Parameter.
- Das Überladen des Operators + schließt das Überladen des Operators += nicht ein.
- Sind die Parameter der Operatorfunktionen groß, so sollten Const-Referenzen benutzt werden.
- Die Operatoren +, -, ...müssen ein Objekt (temporär) erzeugen. Die Operatoren =, += ... können auch eine Referenz zurückgeben, da sie auf einem bereits existierenden Objekt operieren.

Dynamische Felder und der Operator []

- Der Operator [] kann auf der linken Seite des Zuweisungsoperators stehen, wenn die Operatorfunktion operator[] eine Referenz zurückgibt.
- Die Operatorfunktion solllte den Index prüfen, im Fehlerfall ist es sinnvoll, eine Referenz auf einen Dummy-Wert zurückgegeben, somit kann es zu keinem Fehlverhalten (überschreiben eines undefinierten Bereiches) kommen.
- Indexgrenzen könnten beim Anlegen des Feldes auch noch angegeben werden, es müsste dann der entsprechende Konstruktor vorgesehen werden.
- Die Größe des Feldes steht nicht per Compilezeit fest, sie kann über einen Parameter dem Konstruktor übergeben werden.
- Ein Array könnte auch als verkettete Liste implementiert sein, ohne dass sich das Interface dabei ändern müsste.
- Man kann nur einen Parameter übergeben, beim Überladen von [].
- Soll ein Zugriff [x,y] ermöglicht werden, so müssen die runden Klammern überladen werden.
- Es ist festgelegt, dass [] nur durch eine nicht statische Memberfunktion überladen werden kann, nicht durch eine Friendfunktion.

Konvertierungen zwischen Klassen

- Kann keine Zuordnung zu den Argumenten einer (überladenen) Funktion gefunden werden, so versucht der Kompiler über Standardkonvertierungen eine Zuordnung zu treffen.

Zusammenfassung Operatorüberladung 2

- Standardkonvertierungen haben keine Vorrangregeln.
- Ein Zeiger beliebigen Typs kann einem Argument des Typs void * zugeordnet werden.
- Der Wert 0 kann einem Zeiger oder einem arithmetischen Datentyp zugeordnet werden.
- Die Zuordnung erfolgt so, dass von links nach rechts die bessere Zuordnung getroffen werden kann.
- Es ist möglich, spezielle Konvertierungen für Klassen zu definieren.

Standardkonvertierungen in C / C++

1 Ist einer der Operanden vom Typ **long double**, so wird der andere Operand nach **long double** konvertiert.
2 Trifft 1 nicht zu und ist einer der Operanden vom Typ **double**, so wird der andere Operand nach **double** konvertiert.
3 Treffen 1 und 2 nicht zu und ist einer der Operanden vom Typ **float**, so wird der andere Operand nach **float** konvertiert.
4 Treffen 1 bis 3 nicht zu (keine Gleitkommaoperanden), wird eine Festkommakonvertierung ausgeführt
4.1 Ist ein Operand **unsigned long,** so wird der andere Operand nach **unsigned long** konvertiert.
4.2 Trifft 4.1 nicht zu, und ist einer der Operanden vom Typ **long** und der andere Operand vom Typ **unsigned int**, so wird der Operand vom Typ **unsigned int** nach **long** (16-Bit-Compiler) bzw. nach **unsigned long** (32-Bit-Compiler) konvertiert.
4.3 Treffen 4.1 und 4.2 nicht zu, und ist einer der Operanden vom Typ **long**, so wird der andere Operand nach **long** konvertiert.
4.4 Treffen 4.1 bis 4.3 nicht zu und ist einer der Operanden vom Typ **unsigned int**, so wird der andere Operand nach **unsigned int** konvertiert.
4.5 Treffen 4.1 bis 4.4 nicht zu werden beide Operanden nach **int** konvertiert.

Konvertierung durch Konstruktor

- Ein Konstruktor mit nur einem Parameter wird auch Konvertierungskonstruktor genannt. Er überführt einen Wert des Parametertyps in den Typ der Klasse.

Konvertierungsoperatoren

- Konvertierungsoperatoren haben die Form
  operator typ() const;
- Die Operatorfunktion hat keinen Returntyp.
- Die Operatorfunktion hat keine Parameter.
- Die Operatorfunktion muß eine nichtstatische Memberfunktion sein.
- Benutzt werden kann eine Konvertierungsfunktion durch expliziten Aufruf, wie ein Konstruktor, wie ein cast, durch impliziten Aufruf.
- Der Kompiler kann benutzerdefinierte und Standardkonvertierungen bei einer Operation verarbeiten.
- Konvertierungen sind auch zwischen Klassen möglich, der Zieltyp müßte dann ein Klassentyp sein.

Konvertierung eigene Klasse in anderen Typ : Konvertierungsoperator Konvertierung fremden Typ in eigenen Klassentyp : Konvertierungskonstruktor

- Gibt es mehrere Möglichkeiten, eine Konvertierung durchzuführen, so kommt es zu einem Kompilerfehler, der Kompiler kann nicht entscheiden, welche Möglichkeit benutzt werden soll.
- Bieten mehrere Klassen Möglichkeiten einer Konvertierung, kann es ebenfalls zu Konflikten kommen.