

Grundlagen der Informatik II – 12. Übung

Schwerpunkt: Überladen von Operatoren einer Klasse

Operatoren aus einem anderen Blickwinkel

- Problemstellung:

Bisher gewohnte Operatoren (+, -, =, <, >, ==, ...) sollen auch auf Klassenobjekte angewendet werden. Die Bedeutung von Operatoren ist aber nicht stets intuitiv ersichtlich und macht in manchen Fällen vielleicht auch keinen Sinn.

Operator	Objekte	Bedeutung der Operation
+	zwei Studenten	Addition zweier Studenten hat keinen Sinn
	zwei komplexe Zahlen	Realteile addiert ergeben neuen Realteil, Imaginärteile addiert ergeben neuen Imaginärteil
	zwei Matrizen	komponentenweises Addieren der Matrizen
==	zwei Studenten	Gleichheit des Studienfachs prüfen
	zwei komplexe Zahlen	Realteile und Imaginärteile auf Gleichheit überprüfen
	zwei Matrizen	komponentenweises Vergleichen der Matrizen
=	zwei Studenten	Zuweisung aller Leistungsergebnisse eines Studentenobjektes an ein anderes (z.B. bei Studienfachwechsel – nur bedingt sinnvoll)
	zwei komplexe Zahlen	eine komplexe Zahl an eine andere zuweisen: $c_1 = (c_2 + c_3) \rightarrow (c_2+c_3)$ ist auch komplexe Zahl
	zwei Matrizen	komponentenweises Kopieren einer Matrix in eine andere

- Eigenschaften von Operatoren:

- Anzahl der Operanden (binäre und unäre Operationen)
- Ergebnistyp der Operation (Addition zweier ganzer Zahlen ergibt ganze Zahl, Division zweier ganzer Zahlen kann gebrochene Zahl ergeben)
- Wirkung des Operators (Addition, Subtraktion, Vergleich, Zuweisung,...)

- Frage: Wie implementiert man neue Operatoren in Klassen?

- Antwort: Einführung von Operatormethoden innerhalb einer Klasse

Überladen von Operatoren einer Klasse mit C++

- Allgemein: Operatoren einer Klasse sind Methoden folgender Form:

ergebnistyp Klassenname :: **operator@** (**datentyp** Operand2)

---> @ steht für den jeweiligen Operator

Beispiel: Implementation eines Additionsoperators für komplexe Zahlen

- gegeben sei eine Klasse namens „Complex“ zur Verwaltung komplexer Zahlen:

```
class Complex {
    private:
        int real;
        int imag;
    public:
        void set_real(int r);
        void set_imag(int i);
        int get_real();
        int get_imag();
};
```

- Vorgehensweise:

Ermitteln aller Eigenschaften des zu implementierenden Operators

- Anzahl Operanden: - erster Operand ist Objekt selbst
- zweiter Operand ist Methodenparameter
- Ergebnistyp der Operation: Complex
- Wirkung des Operators: Addition der Realteile getrennt von Imaginärteilen

- resultierende Implementation:

```
Complex operator+ (const Complex & ref_c) const {
    Complex tmp;
    tmp.set_real( ref_c.get_real() + real );
    tmp.set_imag( ref_c.get_imag() + imag );
    return tmp;
}
```

- Anwendung:

```
Complex c1,c2,c3;
c1 = c2 + c3;
c1.operator= ( c2.operator+(c3) );
```

Ausgewählte Operatormethoden am Beispiel der „Complex“-Klasse

Operator	Operatormethode	Bedeutung der Operation
+	<code>Complex :: operator+ (const Complex& ref_c) const;</code>	<ul style="list-style-type: none"> - Erzeugen eines temporären Complex-Objektes tmp - kopieren des Realteils und des Imaginärteils von ref_c nach tmp - Addition von Real- und Imaginärteil des aktuellen Objektes mit denen von ref_c - Rückgabe des temporären Objektes
<	<code>bool :: operator< (const Complex& ref_c) const;</code>	<ul style="list-style-type: none"> - Errechnen des Betrages von ref_c und des Betrages der aktuellen komplexen Zahl und numerischen Vergleich machen - wenn Betrag von ref_c kleiner als Betrag der aktuellen Zahl, dann gib true zurück, sonst false
=	<code>Complex& operator= (const Complex& ref_c);</code>	<ul style="list-style-type: none"> - Real- und Imaginärteil von ref_c in aktuelles Objekt eintragen und aktuelles Objekt zurückgeben: <code>this ... Zeiger auf aktuelles Objekt</code> <code>*this ... aktuelles Objekt</code>
+=	<code>Complex& operator+= (const Complex& ref_c);</code>	<ul style="list-style-type: none"> - Real- und Imaginärteil von ref_m auf aktuelles Objekt aufaddieren und aktuelles Objekt zurückgeben wie bei operator=

● Aufrufbeispiele:

```
Complex c1,c2,c3;

c1 = c2 + c3;    // c1.operator= ( c2.operator+(c3) );

if(c1 < c2)
    cout << "c1 ist kleiner als c2" << endl;

c2 += c3;        // c2.operator+= ( c3 );
```