Inhaltsverzeichnis

1. C++: Überladen von Operatoren....1
1.1. Ziele....1
1.2. Einleitung....1
1.2.1. Aufgabe: z= a + b;....2
1.2.2. Aufgabe: z= a+3....2
1.2.3. Aufgabe: z= 3 + a;....3
1.2.4. Lösung: friend – Funktionen....3
1.3. Friend-Funktionen....3
1.3.1. Die Ausgabe << für eigene Klassen....4
1.3.2. Beispiel: oop1-Firma - Die friend-Funktion operator<<()....4
1.3.3. Aufgabe: oop1-Banksysteme – Die Friend-Funktion operator<<()....5
1.4. Aufgabe: intarray -Überladen v. Operatoren....6
1.4.1. Die Ausgabe << für eigene Klassen....6
1.4.2. Lösung: der überladene operator << (intarray.h,intarray.cpp)....6
1.4.3. Aufgabe: Der Vergleichsoperator ==....7
1.4.4. Lösung: Der Vergleichsoperator ==....7
1.4.5. MAB: > Operator überladen....7
1.4.6. Aufgabe: Der Zuweisungsoperator =....8
1.4.7. Lösung: Der Zuweisungsoperator =....8
1.4.8. Aufgabe: Der Kopierkonstruktor....8
1.4.9. Lösung: Der Kopierkonstruktor....9
1.4.10. Merke....9
1.4.11. Aufgabe: Der index Operator []....9
1.4.12. Aufgabe: return per Referenz oder Wert....10
1.4.13. Lösung: return per Referenz oder Wert....11

1. C++: Überladen von Operatoren

1.1. Ziele

☑ Für neue Klassen die wichtigsten Operatoren (<<, <, =, [], ...) überladen können

☑ Anhand der Klasse MyString soll das Überladen von Operatoren behandelt werden.

- Der/Die SchülerIn soll für selbstdefinierte Klassen Operatoren überladen können.
- Der/Die SchülerIn soll erkennen können, dass bei der Verwendung von "dynamischen Membern" der Kopierkonstruktor und der Zuweisungsoperator zu erstellen sind.

1.2. Einleitung

In C++ werden alle Operatoren für selbstdefinierte Klassen in Form von
☑ globalen C-Funktionen (friend) oder
☑ Klassenmethoden
realisiert.

Am Beispiel der Klasse CBruch wollen wir die beiden Varianten studieren.

Hier die Klasse: CBruch:

```
class CBruch { ..... };
```

```
// drei Objekte erzeugen
CBruch z, a, b;
```

1.2.1. Aufgabe: z= a + b;

Wir wollen folgendes programmieren können:

```
z= a + b;
```

☑ Lösung:
Es stehen 2 Möglichkeiten zur Verfügung:

☑ a) Überladen mit: **globaler C-Funktion**:

```
     aus z= a + b; wird ein normaler Funktionsaufruf durchgeführt

     z= operator+(a,b);
```

☑ b) Überladen mit: **Methodenaufruf**:

```
     aus z= a + b; wird ein Methodenaufruf durchgeführt

     z= a.operator+(b);
```

Bei der Methoden-Variante fällt auf, dass ein Parameter weniger verwendet wird.

Beide Varianten sind möglich.

Allerdings gibt es Fälle, bei denen die Methoden-Variante nicht möglich ist.
Dies ist dann der Fall, wenn ein Operand kein CBruch-Objekt ist. Die folgenden Beispiele zeigen dies.

1.2.2. Aufgabe: z= a+3

Wir wollen
z= a + 3;

verwenden können:

a) Überladen mit: globaler Funktion:
z= operator+(a,3)

b) Überladen mit Methodenaufruf:
z= a.operator+(3)

Alles wie gehabt, aber wie sieht's beim folg. Beispiel aus?

1.2.3. Aufgabe: z= 3 + a;

```
z= 3 + a;
a) Überladen mit: globaler Funktion:
    z= operator+(3,a);


b)  Überladen mit Methodenaufruf:
    z= 3.operator(a) !!!!!!
```

1.2.4. Lösung: friend – Funktionen

Wir sehen, dass in manchen Fällen die Verwendung eines Methodenaufrufes nicht möglich ist. Wir müssen globale Funktionen, also friend-Funktionen in der Klasse CBruch zur Verfügung stellen.

```
class CBruch {
....
public:
     // den + Operator überladen
     friend CBruch operator+ (const CBruch& , const CBruch&);
     friend CBruch operator+ (long , const CBruch&);
     friend CBruch operator+ (const CBruch& , long);

....
}
```

1.3. Friend-Funktionen

Sogenannte **Friend-Funktionen sind einfache C-Funktionen** (also keine Klassen/Objekt-Methoden).

Friend-Funktionen werden vom Programmierer der Klasse in der Header-Datei mit dem **Schlüsselwort friend** deklariert.

Dadurch können Friend-Funktionen **auf den private-Bereich** einer Klasse zugreifen.

Als typisches Anwendungsbeispiel für Friend-Funktionen, wird die Verwendung des Ausgabe-operators "<<" angesehen.

1.3.1. Die Ausgabe << für eigene Klassen

Wir wollen das aus den vorigen Kapiteln OOP-Klassen-Vererbung Projekt Firma/Company stammende CAbteilungs-Objekt namens abteilung auf cout ausgeben.

```
...
// ein ABTEILUNGsobjekt erstellen und die Personenobjekte einfügen

CAbteilung* abteilung= new CAbteilung("Programmierer-Abteilung");

abteilung->addMitarbeiter(ich);
abteilung->addMitarbeiter(sie);
abteilung->addMitarbeiter(er);

// statt der Verwendung von toString(), wollen wir,
// dass die Klasse sich vollständig in das C++ Konzept
// 'einbettet'
// cout << abteilung->toString() << endl;

cout << *abteilung << endl;
…
```

Der Compiler macht aus dem Aufruf

cout << abteilung;

folgendes

```
operator<<(cout, ia);
```

1.3.2. Beispiel: oop1-Firma - Die friend-Funktion operator<<()

Bringen Sie folgende Änderungen/Erweiterungen in ihr Projekt oop1-Firma ein.

☑ cabteilung.h

```
class CAbteilung{
     …
     public:
     …

     // einer Funktion durch das Schlüsselwort friend den
     // Zugriff auf den private-Bereich ermöglichen

     friend ostream& operator<<(ostream& o, const CAbteilung& e);
```

```
};
```

☑ cabteilung.cpp:

```
...
ostream& operator<<(ostream& o, const CAbteilung& e){

     o << e.toString() << endl ;
     return o;
}
…
```

oder ohne Verwendung v. toString()

```
ostream& operator<<(ostream& o, const CAbteilung& e){

    vector<CPerson*>::iterator it;

    o << endl<<endl;
    o << "=========================================================\n";
    o << "=== CFIRMA / MITARBEITER "<<endl;
    o << "=========================================================\n";

    o<< "Abteilung:" << e.name <<endl;
    for(it= (e.mitarbeiter)->begin(); it!=(e.mitarbeiter)->end(); it++){
        o << (*it)->toString(); // !!!!!POLYMORPHISMUS
    }

    return o;
}
```

1.3.3. Aufgabe: oop1-Banksysteme – Die Friend-Funktion operator<<()

Projekt: oop1-Banksysteme

Erweiteren Sie die Klassen Konto und Bank, sodass Objekte davon mit << auf cout ausgegeben werden können.

1.4. Aufgabe: intarray -Überladen v. Operatoren

Im Ordner intarray finden Sie das Programm intarray-main.cpp. Bringen Sie das Programm zum Laufen und kopieren Sie in dieses Skriptum die richtigen Lösungen.

1.4.1. Die Ausgabe << für eigene Klassen

Wir wollen das Intarray-Objekt namens ia auf cout ausgeben.

```
Intarray ia(20);

cout << ia;
```

Der Compiler macht aus dem Aufruf

cout << ia;

folgendes

```
operator<<(cout, ia);
```

1.4.2. Lösung: der überladene operator << (intarray.h,intarray.cpp)

☑ intarray.h

```
class Intarray{
      …
      public:
      …

      // einer Funktion durch das Schlüsselwort friend den
      // Zugriff auf den private-Bereich ermöglichen
      friend ostream& operator<<(ostream& o, const Intarray& e);

};
```

☑ cintarry.cpp:

```
ostream& operator<<(ostream& o, const Intarray& e){

      for (int i=0; i < e.len; i++)
            o << e.a[i] << "," ;

      return o;
}
```

....

1.4.3. Aufgabe: Der Vergleichsoperator ==

```
Intarray ia(20),  ib(20);

...
if (ia == ib) ....
      ...


// if (ia.operator==(ib))
```

1.4.4. Lösung: Der Vergleichsoperator ==

☑ cintarray.h

```
class Intarray{
      …
      public:
      …
      ??????????????????????????????

};
```

☑ intarry.cpp:

```
??????????????????
```

1.4.5. MAB: > Operator überladen

```
Intarray ia(3), ib(3);

if (ia > ib) // zB: (1,3, 2) ist groesser als (1,2,3)
```

☑ intarray.h

```
class Intarray{
      …
      public:
      …
      ??????????????????????????????

};
```

☑ intarry.cpp:

```
??????????????????
```

1.4.6. Aufgabe: Der Zuweisungsoperator =

```
Intarray ia(999),  ib(20);

ib= ia;
```

zu beachten sind hier:
☑ Freigabe des von ib belegten Speichers
☑ Reservieren von Speicher für ib in der Größe von ia
☑ Inhalt von ia nach ib kopieren
☑ !Achtung auf ib= ib;

1.4.7. Lösung: Der Zuweisungsoperator =

☑ cintarray.h

```
class Intarray{
      …
      public:
      …
      ??????????????????????????????

};
```

☑ intarry.cpp:

```
??????????????????
```

1.4.8. Aufgabe: Der Kopierkonstruktor

☑ MERKE:
Bei Klassen, deren member dynamische Speicherverwaltung (new,delete) nutzen **MUSS** man den Kopierkonstruktor erstellen.

1.4.9. Lösung: Der Kopierkonstruktor

☑ Antwort:
??????????????????????????????
Wenn Sie die Antwort nicht wissen. Sie werden Sie in den Übungen erfahren und vergessen Sie nicht die Antwort hier einzutragen.

☑ intarray.h

```
class Intarray{
      …
      public:
      …
      ??????????????????????????

};
```

☑ intarry.cpp:

```
??????????????????
```

1.4.10. Merke

// as a rule, you should ALWAYS define both

// a COPY CONSTRUCTOR and
// an ASSIGNMENT OPERATOR

// whenever your class contains pointer members

1.4.11. Aufgabe: Der index Operator []

Wir wollen folg. Situation beachten:

```
Intarray ia(20);
int val= 17;

Fall1:
ia[1]= val;
      wird intern übersetzt zu
      ia.operator[](1)= val;


Fall2:
val= ia[1];
      wird intern übersetzt zu
      val= ia.operator[](1);
```

Beachte:

Hier ist der Typ der Rückgabe der operator[]-Methode zu beachten:

☑ Rückgabetyp ist eine Referenz. (Fall1)
☑ Rückgabetyp ist ein Wert. (Fall2)

Eines müssen wir noch wissen.

☑ Wenn eine Funktion/Methode einen **Wert zurück** gibt, wird der in der return-Anweisung angegebene Wert **kopiert**. Diese Kopie kann der Aufrufer der Funktion/Methode dann für eine Zuweisung verwenden.

Danach wird diese Kopie zerstört.

☑ Wenn eine Funktion/Methode eine **Referenz zurück** gibt, wird der in der return-Anweisung angegebene Wert **nicht kopiert**, **sondern ein Verweis** auf diese Variable/Member wird an den Aufrufer retourniert.

☑ **Rückgabe per Wert**

```
int Intarray::operator[] (int index) {
   ....
   return a[index];
}
```

Die return-Anweisung erzeugt eine temp. Kopie

☑ **Rückgabe per Referenz**

```
int& Intarray::operator[] (int index) {
   ....
   return a[index];
}
```

Die return-Anweisung liefert die Referenz auf das Element im Array a.

1.4.12. Aufgabe: return per Referenz oder Wert

```
Intarray ia(20);
int val= 17;

Fall1:
ia[1]= val;
     wird intern übersetzt zu
     ia.operator[](1)= val;

Antwort für Fall1: (Streiche/Lösche die falsche Antwort durch)
     operator-Methode verwendet
     a) Referenzrückgabe
     b) Wertrückgabe

Fall2:
val= ia[1];
     wird intern übersetzt zu
     val= ia.operator[](1);

Antwort für Fall2: (Streiche/Lösche die falsche Antwort durch)
```

```
     operator-Methode verwendet
     a) Referenzrückgabe
     b) Wertrückgabe
```

1.4.13. Lösung: return per Referenz oder Wert

☑ intarray.h

```
class Intarray{
     …
     public:
     …
     ??????????????????????????

};
```

☑ intarry.cpp:

```
??????????????????
```