Überladen von Operatoren

- Überladen des Indexoperators [ ]
- Überladen des Ausgabeoperators <<
- Überladen des Eingabeoperators >>

Überladen von Operatoren

ähnlich dem Überladen von Funktionen

Überladen von Operatoren setzt allerdings Klassen voraus.

Syntax der Operatordefinition
(⊗ steht für eines der möglichen C++-Operatorsymbole)

für eine globale Funktion:

ReturnDatentyp `operator`⊗
(*Argumentliste*) { *Funktionscode* }

für eine Elementfunktion:

ReturnDatentyp *Klassenname*`::operator`⊗
(*Argument*) { *Funktionscode* }

Überladen von Operatoren

Operatorfunktionen unterscheiden sich von den bisher bekannten Funktionen lediglich durch zwei Dinge:

- Der Funktionsname einer Operatorfunktion besteht aus dem Schlüsselwort `operator` und dem angehängten Operatorzeichen

- Der Aufruf einer Operatorfunktion wird entsprechend der folgenden Tabelle in einen Funktionsaufruf umgewandelt.

Elementfunktion	Syntax	Ersetzung durch
nein	`x ⊗ y`	`operator⊗(x, y)`
	`⊗ x`	`operator⊗(x)`
	`x ⊗`	`operator⊗(x, 0)`
ja	`x ⊗ y`	`x.operator⊗(y)`
	`⊗ x`	`x.operator⊗()`
	`x ⊗`	`x.operator⊗(0)`
	`x = y`	`x.operator=(y)`
	`x[y]`	`x.operator[](y)`
	`x->`	`(x.operator->())->`

Überladen von Operatoren

Einschränkungen für Operatorfunktionen:

- Es können die üblichen C++-Operatoren wie
  `=` `==` `+=` usw.
  überladen werden, nicht jedoch
  `.` `.*` `::` `?:`
  und andere Zeichen wie
  `$` `§` usw.

- `new` und `delete` sowie `new[]` und `delete[]` können überladen werden.

- Die Definition von neuen Operatoren ist nicht möglich.

- Die vorgegebenen Vorrangregeln können nicht verändert werden.

- Wenigstens ein Argument der Operatorfunktion muss ein `class`-Objekt sein oder die Operatorfunktion muss eine Elementfunktion sein.

Überladen von Operatoren

Überladen des Indexoperators `[]`

Beispiel: eine Klasse `IntArray`

Verwendung der Klasse:

```
#include <iostream>
#include <cstdlib>        // fuer exit()
using namespace std;

int main()
{
  IntArray x(20);

  // Initialisierung der Arrayelemente
  for (int i = 0; i < 20; i++)
    x[i] = i * 2;

  // Ausgabe der Arrayelemente
  for (int i = 0; i <= 20; i++)
    cout << "At index " << i
         << ": value is " << x[i]
         << endl;

  return 0;
}
```

Überladen des Indexoperators `[]`

Schnittstelle der Klasse `IntArray`:

```
class IntArray
{
  public:
    IntArray(int sz = 1);
    IntArray(const IntArray& other);
    ~IntArray();
    IntArray& operator=(const IntArray& other);

    // überladener Indexoperator
    int& operator[](int index);

  private:
    void destroy();
    void copy(const IntArray& other);

    int *data, size;
};
```

Der überladene Indexoperator hat den Rückgabetyp

```
int&
```

Dadurch können Ausdrücke wie

```
x[10]
```

sowohl auf der rechten als auch auf der linken Seite einer Wertzuweisung stehen.

Überladen des Indexoperators `[]`

Implementierung der Klasse `IntArray`:

```
IntArray::IntArray(int sz)
{
  if (sz < 1)
  {
    cout << "IntArray: "
         << "size of array must be >= 1, not "
         << sz << "!" << endl;
    exit(1);
  }
  // Größe notieren und Array erzeugen
  size = sz;
  data = new int [sz];
}

// Kopierkonstruktor
IntArray::IntArray(const IntArray& other)
{
  copy(other);
}

// Destruktor
IntArray::~IntArray()
{
  destroy();
}
```

Überladen des Indexoperators `[]`

Implementierung der Klasse `IntArray`:

```
// überladene Wertzuweisung
IntArray& IntArray::operator=(const IntArray& other)
{
  // Selbstzerstörung vermeiden!
  if (this != &other)
  {
    destroy();
    copy(other);
  }
  return *this;
}

void IntArray::destroy()
{
  delete [] data;
}

void IntArray::copy(const IntArray& other)
{
  // Größe notieren
  size = other.size;

  // Array erzeugen
  data = new int [size];

  // Werte von other kopieren
  for (register int i = 0; i < size; i++)
    data[i] = other.data[i];
}
```

Überladen des Indexoperators `[]`

Implementierung der Klasse `IntArray`:

```
int& IntArray::operator[](int index)
{
  // Liegt eine Bereichsüberschreitung vor?
  if (index < 0 || index >= size)
  {
    cout << "IntArray: "
         << "array index out of range, "
         << "index = "
         << index
         << ", should range from 0 to "
         << size - 1
         << endl;
    exit(1);
  }

  return data[index];
}
```

Überladen von Operatoren

Überladen des Ausgabeoperators <<

cout und cerr sind Objekte der Klasse ostream, die überladene Ausgabeoperatoren für Standard-Datentypen enthält.

```
ostream& operator<<(const char*);
ostream& operator<<(char);
ostream& operator<<(int);
ostream& operator<<(float);
ostream& operator<<(double);
// usw.
```

Wunsch: Überladen des Ausgabeoperators <<, so dass Objekte benutzerdefinierter Klassen ausgegeben werden können.

Beispiel:

```
Person kr("Kernighan and Ritchie",
          "unknown", "unknown");

cout << "Name, address and phone number "
     << "of Person kr:\n"
     << kr
     << endl;
```

Überladen des Ausgabeoperators <<

Die Anweisung

```
cout << kr;
```

kann interpretiert werden entweder als

a) `cout.operator<<(kr);`

oder aber als

b) `operator<<(cout, kr);`

Der Operator kann also keine Elementfunktion der Klasse `Person` sein.

Gemäß a) wäre der Operator `<<` eine Elementfunktion der Klasse `ostream`.

Diese Möglichkeit kommt nicht in Frage. Warum?

Es bleibt nur die Interpretation b), gemäß der der Operator `<<` als globale Funktion mit zwei Argumenten formuliert werden muss.

```
ostream& operator<<(ostream& s,
                    const Person& p);
```

Überladen des Ausgabeoperators <<

<< gibt eine Referenz auf das ostream-Objekt s zurück. Dadurch wird eine Verkettung mehrerer Ausgaben möglich.

```
cout << "K&R:" << kr << endl;
```

ist identisch mit

```
((cout << "K&R:") << kr) << endl;
```

Implementierung des überladenen Ausgabeoperators << für die Klasse Person:

```
// Deklaration (z.B. in Person.h)
ostream& operator<<(ostream& s,
                    const Person& p);

// Definition (z.B. in Person.cpp)
ostream& operator<<(ostream& ausgabe,
                    const Person& p)
{
  return
    ausgabe << "Name:    " << p.getname()
            << "Address: " << p.getaddress()
            << "Phone:   " << p.getphone()
    ;
}
```

Überladen von Operatoren

Überladen des Eingabeoperators >>

Beispiel: Überladen von Ein- und Ausgabeoperatoren für die Klasse `String`

```
#include <iostream>
using namespace std;

class String
{
  public:
    // ...
    void setStr(const char *s);
    const char *getStr() const;
  private:
    char *str;
};

void func()
{
  char buffer[500];
  String s;

  cin >> s.setStr();   // Geht leider nicht!

  cin >> buffer;
  s.setStr(buffer);                  // ok.

  cout << s.getStr() << endl;        // ok.
}
```

Überladen von Operatoren

Überladen des Eingabeoperators >>

Wunsch: Ausgabe von `String`-Objekten

```
int main()
{
  String s("Hello world");

  cout << "The string is: \"" << s << "\""
       << endl;
}
```

Außerdem: Eingabe von `String`-Objekten

```
int main()
{
  String s;

  cout << "Please enter your string: ";

  cin >> s;

  cout << "Got: \"" << s << "\"" << endl;
}
```

Überladen von Operatoren

Überladen des Eingabeoperators >>

Ausgabe von `String`-Objekten

```
ostream& operator<<
  (ostream& ostr, const String& s)
{
  return ostr << s.getStr();
}
```

Eingabe von `String`-Objekten (1. Versuch)
`operator>>()` ist Elementfunktion der Klasse `String`

```
istream& String::operator>>(istream& is)
{
  char buf[500];
  // Wir nehmen an, dass diese Größe ausreicht.

  is >> buf;
  // Eingabe einer Zeichenkette

  delete [] str;
  // "alten" Speicherplatz freigeben

  str = strdupnew(buf);
  // neuen Wert zuweisen

  return is;
  // Referenz auf istream zurückliefern
}
```

Überladen von Operatoren

Überladen des Eingabeoperators >>

Eingabe von `String`-Objekten (1. Versuch)

```
void func()
{
  String s;

  s >> cin;           // (1)
  s.operator>>(cin);  // (1)

  int x;

  s >> (cin >> x);    // (2)

  cin >> x >> s;      // (3)
}
```

- **(1)** Extraktion aus dem `istream`-Objekt `cin` in das `String`-Objekt `s`

- **(2)** `cin >> x` liefert eine Referenz auf ein `istream`-Objekt, das als rechter Operand beim Einlesen von `s` verwendet wird.

- **(3)** Fehlermeldung des Compilers:
  `Call does not match any argument list for "istream::operator>>".`

Überladen von Operatoren

Überladen des Eingabeoperators >>

Eingabe von `String`-Objekten (2. Versuch)
`operator>>()` ist globale Funktion mit zwei Argumenten

```
#include <iostream>
using namespace std;

class String
{
  friend istream& operator>>
    (istream& is, String& s);
  public:
    // ...
  private:
    char *str;
};

istream& operator>>(istream& is, String& s)
{
  char buf [500];

  is >> buf;

  delete [] s.str;

  s.str = strdupnew(buf);

  return is;
}
```

Überladen von Operatoren

Überladen des Eingabeoperators >>

Eingabe von `String`-Objekten (2. Versuch)

```
void func()
{
  String s;

  cin >> s;
  // Anwendung des Eingabe-Operators

  int x;

  cin >> x >> s;
  // Eingabe-Reihenfolge jetzt wie erwartet.
}
```

Überladen von Operatoren

Überladen des Eingabeoperators >>

Eingabe von `String`-Objekten (3. Versuch)
Geht es auch ohne die `friend`-Deklaration?

```
#include <iostream>
using namespace std;

class String
{
  public:
    String& operator=(const char *s);
    // ...
  private:
    char *str;
};

istream& operator>>
  (istream& is, String& s)
{
  char buf[500];

  is >> buf;
  // Einlesen einer Zeichenkette

  s = buf;
  // Zuweisung der Zeichenkette
  // an ein String-Objekt

  return is;
}
```