C++ Teil 12

Sven Groß

18. Jan 2016

Themen der letzten Vorlesung

Wiederholung aller bisherigen Themen:

- Datentypen und Variablen
- Operatoren
- Kontrollstrukturen
- Ein-/Ausgabe
- Funktionen
- Zusammengesetzte Datentypen
- Objektorientierung und Klassen

Neu:

- Funktionen: Default-Parameter
- Assoziative Container: Map, Set

Heutige Themen

1. Klassen (Wdh.)
   - Beispiel: Array-Klasse
2. Konstruktoren: Initialisierungsliste
3. Friends
4. Statische Elemente
5. Projekte mit mehreren Dateien

Beispiel: Array-Klasse

- **Implizite Elemente** jeder Klasse (falls nicht vom Benutzer überschrieben):
  - Default-Konstruktor (falls keine Konstruktoren deklariert werden)
  - Kopierkonstruktor
  - Destruktor
  - Zuweisungsoperator `operator=`

⚠ müssen bei Klassen mit dynam. Speicherverwaltung überschrieben werden!

⇝ Mindestanforderungen an Array-Klasse:

```
class Array
{
  private:
    double* feld; // Daten
    int     L;    // Laenge
  public:
    Array( int laenge);
    Array( const Array& a);
    ~Array();
    Array& operator= (const Array& a);
};
```

Array-Klasse (2)

Schnittstelle soll weitere **Methoden** zur Verfügung stellen:

- Länge abfragen (Getter für `L`):

```
int size() const { return L; }
```

- Länge ändern:

```
void resize( int laenge);
```

- Zugriffsoperatoren:

```
double& operator[] (int i);       // (a) schreiben/lesen
double  operator[] (int i) const; // (b) nur lesen
```

Array-Klasse: Zugriffsoperator

Zugriffsoperatoren als Methoden der Klasse `Array`:

```
double& operator[] (int i);       // (a) schreiben/lesen
double  operator[] (int i) const; // (b) nur lesen
```

- Variante (b) sichert zu, dass Objekt nicht verändert wird
- unterschiedliche Signatur durch `const`!
- **Frage:** Wann wird welche Variante aufgerufen?

```
// Array A;    nicht erlaubt, da kein
//             Default-Konstruktor vorhanden
Array B( 7);        // Array der Laenge 7

B[1]= 1.3;             // Schreibzugriff mit (a)

const Array C( B); // konstante Kopie von B

cout << C[1] << endl; // Lesezugriff mit (b), da C konst.
cout << B[1] << endl; // Lesezugriff mit (a)
```

Initialisierungslisten für Attribute im Konstruktor

- **Beispiel:** Konstruktor der `Sudoku`-Klasse:

```
Sudoku::Sudoku()
// hier wird vector<int> Data
// per Default-Konstruktor erzeugt
{
  // noch ist Data ein leerer Vektor
  Data.resize( 81);
}
```

- besser: statt leerem Vektor direkt Vektor der Länge 81 mit entsprechendem `vector`-Konstruktor erzeugen

⇝ **Initialisierungsliste**

```
Sudoku::Sudoku()
  : Data( 81)    // Init.liste
{}
```

⚠ Reihenfolge der Initialisierungsliste = Reihenfolge der Attribute in der Klasse!

Friends

- *Wdh.*: auf private Elemente dürfen nur Methoden der Klasse zugreifen. Ausnahme: **Friends**, ausgezeichnete externe Funktionen, Methoden oder Klassen
- Klasse erklärt Freundschaft (nicht andersrum)

```
class Student
{
  ...
  friend class ZPA;

  friend void Prof::bewerte( Student& Pruefling);

  friend void Heirat( Student& Married,
                      const string& neuerName);
};
```

- Friend-Funktion ist *keine* Methode, sondern externe Funktion
- weicht Zugriffskontrolle auf, sollte sparsam eingesetzt werden

Statische Elemente

- Statische Elemente ex. 1× pro Klasse, unabhängig vom einzelnen Objekt
- Initialisierung statischer Attribute außerhalb des Konstruktor
- Statische Methoden dürfen nur auf statische Attribute zugreifen

```
class Counter
{ // Diese Klasse zaehlt alle ihre Objekte
  private:
    static int Anzahl;

  public:
    Counter()                    { ++Anzahl; }
    Counter( const Counter&)     { ++Anzahl; }
    ~Counter()                   { --Anzahl; }

    static int Wieviele() { return Anzahl; }
};

int Counter::Anzahl= 0; // Init. bei Programmstart
```

Statische Elemente (2)

```
Counter a;
cout << a.Wieviele() << endl;      // 1
{
    Counter b, c;
    cout << b.Wieviele() << endl; // 3
}
cout << Counter::Wieviele();    // Aufruf ohne Objekt
                                // 1
```

Weil statische Methoden keinem bestimmten Objekt zugeordnet sind,

- können sie auch ohne Objekt aufgerufen werden:
  `Counter::Wieviele();`
- enthalten sie keinen `this`-Zeiger
- dürfen sie nicht auf nicht-statische Elemente zugreifen
- können sie nicht mit `const` qualifiziert werden

Sudoku-Klasse in eigene Datei auslagern

- Bisher: 1 Programm = 1 Datei, z.B. `main.cpp`
- wird für größere Projekte unübersichtlich, deswegen aufteilen in mehrere Dateien
  - **Module**, die bestimmte Aufgaben übernehmen: Klassen, Funktionen, z.B. alles zum Thema Sudoku
  - pro Modul
    - 1 **Header-Datei** (z.B. `sudoku.h`, enthält ***Schnittstellen***)
    - 1 **Source-Datei** (z.B. `sudoku.cpp`, enthält ***Implementierung***).
  - Hauptprogramm, das die benötigten Module **einbindet** (z.B. `#include "sudoku.h"` am Anfang von `main.cpp`)
- Übersetzen aller Module und Hauptprogramme in Objektdateien (z.B. `sudoku.o`, `main.o`)
- Linken aller Objektdateien zu fertigem Programm
- In `Code::Blocks`: Hinzufügen der Header-/Source-Dateien zum Projekt, Compilieren/Linken funktioniert dann automatisch