Vorlesungsmodul Programmieren 3
- VorlMod Prog3 -

Matthias Ansorg

01. Oktober 2002 bis 23. Mai 2003

Zusammenfassung

Studentische Mitschrift zur Vorlesung Programmieren 3 bei Prof. Dr. Henrich (Wintersemester 2002/2003) im Studiengang Informatik an der Fachhochschule Gießen-Friedberg.

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereit: http://www.fh.gecfilm.de.
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieser studentischen Mitschrift liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der verwendeten Quellen zu beachten.
- **Korrekturen:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg, ansis@gmx.de.
- **Format:** Die vorliegende studentische Mitschrift wurde mit dem Programm LyX (graphisches Frontend zu LaTeX) unter Linux erstellt und als pdf-Datei exportiert. Grafiken wurden mit dem Programm xfig unter Linux erstellt und als eps-Datei exportiert. Graphen wurden mit gnuplot erstellt, als xfig exportiert und dort weiterverarbeitet. Die gnuplot-Plotdateien liegen bei.
- **Dozent:** Prof. Dr. Henrich.
- **Verwendete Quellen:** .
- **Klausur:**
  - **Teilnamevoraussetzungen.** Alternativ eine große Hausübung oder regelmäßige (80%) Übungsteilnahme. Es gibt also die Möglichkeit, nicht an Vorlesungen und Übungen teilzunehmen und die Klausur zu schreiben. Das gelingt meist nicht gut, denn in der Klausur müssen bestimmte Sprachmittel verwendet werden, die in der Vorlesung besprochen wurden und die man dann können muss. Dazu gehören z.B. Entwurfsmuster. Die Scheine Prog1 und Prog2 sind keine formale Voraussetzung für die Teilnahme an der Klausur Prog3.
  - **Empfehlungen.** Die Übungsaufgaben der Übungen sollte man in jedem Fall machen und verstehen, um die Klausur bestehen zu können.
  - **Stoff.** In jeder Klausur kommt Kopierkonstruktor, Modellierung mit Vererbung, überladene Konstruktoren und ein Entwurfsmuster vor, außerdem weitere Dinge. In der Klausur kommen keine Wissensfragen vor.
  - **Hilfsmittel.** Alle und beliebig viele Bücher sind erlaubt, jedoch kein eigenes Material oder Skripte. Begründung: die Studenten sollen lernen, mit Büchern zu arbeiten; Nachlesen in Büchern hilft, den Stoff besser zu verstehen; Klausuren können näher an den Übungsaufgaben plaziert werden, weil keine Aufgabensammlung vorhanden ist. Die Klausur wird sich also recht nahe an den Übungsaufgaben orientieren. Besonders für die Entwurfsmuster sollte man auch ein Buch parat haben. Man sollte früh im Semester beginnen, mit einem Buch zu arbeiten, um darin etwas in der Klausur auch zu finden. Man muss jedoch nicht unbedingt viel nachschlagen.

Inhaltsverzeichnis

1 Organisatorisches

Übungsteilnahme Die Anwesenheit wird kontrolliert, es erfolgt jedoch keine Kontrolle des Arbeitsaufwands, den man in die Übungen investiert. Wenn man die Übungsaufgaben beherrscht und evtl. vorher gelöst hat, kann man nach Kontrolle der Anwesenheit und Klärung eventueller Fragen die Übung verlassen. Die Übungsaufgaben veröffentlicht Prof. Henrich auf seiner Homepage `http://homages.fh-giessen.de/~hg6678`, es wird jedoch keine Musterlösungen geben.

In den Übungsterminen Di vor, alternativ nach der Vorlesung wird jeweils derselbe Stoff behandelt. Einteilung der Übungsgruppen: Alle Medizininformtiker dürfen an der Übung (Di) 14:00h teilnehmen; der restliche Platz kann aufgefüllt werden, jeder sollte aber einen eigenen Platz zur Verfügung haben.

Hausübung Es werden Aufgaben im Internet veröffentlicht, die in Gruppen von 2-3 Personen zu bearbeiten sind; eine Literaturangabe wird als Anleitung zur Implementierung gegeben. Hier werden größere Aufgaben, z.B. aus dem Bereich der KI, zu lösen sein; der Aufwand wird ähnlich hoch sein wie bei Übungsbesuch. Außerdem macht es das Lösen der Übungsaufgaben nicht überflüssig!

Konventionen Es ist egal, welchen Compiler man verwendet. Mit `new` belegter Speicher muss immer wieder mit `delete` freigegeben werden. Es werden nur Konsoleanwendungen behandelt. Verwendeter Compiler ist Visual C++ bzw. Visual Studio 6.0.

Förderverein Informatik Es gibt einen Förderverein Informatik, der die »Kontakt 2003« absolviert; es ist eine kleine Messe, wo man Fachvorträge hören kann und BPS-Stellen bekommen kann. Der Förderverein Informatik stiftet der Fachschaft einen Scanner und einen Farblaserdrucker. Man muss nur das Material bezahlen! Studenten können für 5 EUR im Jahr im Verein Mitglied werden.

2 Inhalt

Inhaltsliste zur Veranstaltung, nicht in Reihenfolge der Behandlung, gedacht als Grundlage der Klausurvorbereitung. Der Stoff selbst steht nicht in diesem Skript, man sollte ihn sich aus Büchern aneignen, am besten natürlich aus den Büchern, die man auch in der Klausur verwenden wird. Zusätzlich muss man zur Klausurvorbereitung dann nur noch etliche Übungsaufgaben von Prof. Henrich wiederholen und üben, mit Hilfe des in der Klausur verwendten Buches, denn an diesen Aufgaben orientiert sich die Klausur ja.

- Grundlagen
  - Verkettete Listen.
  - `new` und `delete`
  - `new[]` und `delete[]`
  - Mechanismen und Zeitpunkte der Anforderung und Freigabe von Speicher.
  - Speicherlecks erkennen und beheben. Beispiel: Übung 10, Aufgabe 3[1].

[1] Wichtig für die Klausur 2003-02-23.

- Klassen
  - Initialisierungslisten.
  - pivate, protected und public: Zugriffsrechte.
  - statische Methoden
  - Konstruktoren: Defaultkonstruktor, parametrisierte Konstruktoren.
  - Konvertierungskonstruktoren.
  - Kopierkonstruktor. Ursachen von Speicherzugriffsfehlern in Zusammenhang mit dem Kopierkonstruktor kennen. Kopierkonstruktor mit tiefer Kopie[2].
  - Destruktoren. Iterative und rekursive Destruktion verketteter Listen.
  - `friend`-Funktionen, `friend`-Klassen
- Vererbung
  - einfache `public`-Vererbung.
  - Implementierungsvererbung.
  - Mehrfachvererbung. Ein Anwendungsbeispiel ist hier ein Entwurfsmuster. Beispiel: Übung 6, Aufgabe 3.
  - virtuelle Funktionen und rein virtuelle Funktionen. Beispiel: Übung 7, Aufgabe 3.
- Funktionen und Methoden
  - Überladen von Funktionen.
  - Polymorphismus. Beispiel: Übung 5, Aufgabe 1 und 2.
- Operatoren
  - Überladene Elementoperatoren.
  - Überladene freie Operatoren.
  - Überladen des Ein-/Ausgabeoperators.
  - Besonderheiten beim Überladen von Prä- und Postfixoperatoren.
  - Überladen und doppeltes Überladen des Operators `[]`. Zusammen mit den zweiten kopierten Beispielprogramm wichtig für die Klausur, nach Aussage von Prof. Henrich.
- Templates
  - Funktionstemplates
  - Klassentemplates[3]. Ein Aufgabentyp könnte z.B. sein: Template-Klasse und `main()`-Funktion gegeben, es ist eine Klasse zu definieren, die als Template-Parameter verwendet werden kann.
- Ausnahmebehandlung
- Entwurfsmuster
  - Klassenadapter[4]. Beispiel: Übung 7, Aufgaben 1-2.
- Stoff, der aus Zeitgründen nicht behandelt wurde
  - Konzepte der Wiederverwendung.
  - Laufzeittypinformationen (`dynamic_cast`, RTTI).
  - Managed C++. Auch besitzt die FH Gießen-Friedberg bisher nicht das VisualStudio DotNet.
- Stoff, der aus prinzipiellen Gründen nicht behandelt wurde
  - Die STL. Man darf sie daher auch in der Klausur nicht verwenden!
  - Programmierung grafischer Oberflächen.

[2] Wichtig für die Klausur 2003-02-23.
[3] Wichtig für die Klausur 2003-02-23.
[4] Wichtig für die Klausur 2003-02-23.

3 Einführung

In Zukunft wird man unter Windows mit dem Microsoft DotNet-Framework Oberflächen erstellen. Dies ist wesentlich einfacher als bisher mit den derzeitigen Microsoft Foundation Classes. Microsoft wird dazu die Sprache C++ verändern zu managedC++. Hier sind weitere Schlüsselworte und Funktionen enthalten:

- interface
- garbage collection
- Netzwerktransparenz
- Multithreading

Die meisten Softwareunternehmen verwenden keine CASE-Tools, weil die Fachabteilung, der man die Software vorstellt, kein UML versteht.

3.1 Das Paradigma der objektorientierten Programmierung

3.1.1 Aspekte der Softwareentwicklung

Korrektheit: Aufgaben entsprechend Spezifikation exakt erfüllen.

Erweiterbarkeit: Anpassung an Spezifikationsänderungen. Wenn das Konzept gut, d.h. abstrakt genug war, so ist die Software sehr gut erweiterbar. Erweiterungen sind an allen Softwareprojekten nötig!

Wiederverwendbarkeit: Verwendung in neuen Anwendungen. Schlagwort der Objektorientierung.

Kompatibilität: Verbindung mit anderen Softwareprodukten.

Kosten: Automatisch erfüllt durch Korrektheit, Erweiterbarkeit, Wiederverwendbarkeit.

Termine: Automatisch erfüllt durch Korrektheit, Erweiterbarkeit, Wiederverwendbarkeit.

3.1.2 Funktions/Datenorientiertes Softwareengineering

Funktionsmodellierung structured analysis nach DeMarco. Man ging vom gesamten System in immer feinere Stufen.

Datenmodellierung Die Welt wird in Tabellen zerlegt, d.h. in relationale Datenbanken. Dem entspricht das entity relationship model (ERM). Hier werden also Objekte in Tabellen »zerpflückt«; es gibt auch objektorientierte Datenbanken, die jedoch noch fast nicht eingesetzt werden.

Beide Ansätze boten i.d.R. keine Wiederverwendbarkeit.

3.1.3 Ingenieurmäßige Softwareentwicklung

Wie in der Elektrotechnik sollten Programme aus Bausteinen zusammengesetzt werden; in einem Baustein werden Funktionen und Daten kombiniert. Dies kann durch objektorientierte Softwareentwicklung oder Komponentenentwicklung realisiert werden. Der Inhalt einer Komponente kann (soll) beliebig sein, z.B. ein altes COBOL-Programm. Eine gut entworfene Klasse kann wiederverwendet werden. Die Softwareentwicklung richtet sich immer mehr auf Komponenten aus.

3.1.4 Konzepte der Objektorientierung

Vererbung

Datenabstraktion: Benutzerdefinierte Datentypen inkl. Operationen.

Polymorphismus: Gemeinsame Behandlung von Funktionalität, die zwar im Prinzip gleich ist, in der Detailimplementierung aber total verschieden sein kann. Beispiel: zeichnen() für geometrische Objekte kann sowohl Zeichnen eines Dreiecks als auch eines Kreises sein. Ob es sich um ein Dreieck oder einen Kreis handelt, kann zur Laufzeit festgestellt werden!

3.2 Vorausgesetze Grundkenntnisse

Alle elementaren Grundkenntnisse (»C-Kenntnisse«), die beim Aufbau von Klassen und Methoden benötigt werden, darunter besonders Zeiger und Gültigkeitsbereiche von Variablen.

- Datentypen, Operatoren
  - Systemdefinierte Datentypen, Speicherbedarf
  - Operatoren
  - Variablen und Gültigkeitsbereiche
- Kontrollstrukturen
- Funktionen
  - Argumente
  - Rückgabeparameter
  - Überladen
  - `friend`
  - Statische Funktionen
- Zeiger
  - Zeiger und Adressen
  - Vektoren (gemeint: mit eindimensionalen Feldern umgehen durch Zeigerarithmetik)
  - Zeigerarithmethik
  - Verkettete Listen
- Ein- und Ausgabe
- Klassen
  - Klasse und Instanz (auch: konkretes Objekt)
  - Methoden und Datenelemente
  - Konstruktor und Destruktor
  - dynamische Instanzen (`new` und `delete`)
  - der Zeiger `this`
  - statische Elemente
  - Zugriffsbeschränkungen (ohne protected). Wichtig: das default-Schlüsselwort ist immer private, sowoh bei Vererbung ohne Schlüsselwort als auch bei Attributen und Methoden ohne Schlüsselwort.
  - einfache Operatoren
- Aufbau und Struktur eines C++-Programms

4 Vererbung

4.1 Einführung und Definition

Aufgabenstellung Modellierung von Brüchen: Nenner, Zähler, Multiplikation, Anzeige und Ausgabe in der Form »Zähler/Nenner«.

Lösung Das einfachste softwaretechnische Werkzeug ist die »Textanalyse«, auch »Methode von Abbot«. Dabei extrahiert man Substantive und Verben aus dem Text der Aufgabenstellung - Substantive sind potentielle Klassen, Verben sind potentielle Methoden. Bei der Analyse und dem Entwurf sollte man keinen Schritt überspringen, sondern auch bei einfachen Aufgaben UML-Diagramme usw. zur Hilfe nehmen. Bei der Softwaretechnik geht es darum, komplexe Dinge möglichst einfach zu zerlegen und darzustellen.

```
class Bruch {
  private:
    int zaehler, nenner;
  public:
    Bruch(int z=0, int n=1):zaehler(z),nenner(n) {}
    void show() {cout << zaehler << "/" << nenner << endl;}
    Bruch operator* (Bruch b){
      return Bruch(zaehler*b.zaehler,nenner*b.nenner);
    }
};
void main() {
  Bruch b1(3,4),b2(6,5);
  (b1*b2).show();  //Ausgabe: 18/20
}
```

5 Sonstiges

5.1 Initialisierungslisten

Sie werden in folgenden Fällen zwingend benötigt:

- Initialisierung von Konstanten im Konstruktor einer Klasse. Zuweisungen an Konstanten sind ja nicht erlaubt!
- Basisklassenkonstruktoren.
- Objekte als Datenelemente, denen Werte nur über ihren Konstruktor zugewiesen werden können.

Literatur

[1] Goll, Joachim: »Java als erste Programmiersprache«. Nach diesem Buch richtet sich die Veranstaltung Programmieren 3 bei Prof. Dr. Franzen. Es ist in einigen Exemplaren in der Lerbuchsammlung der Bibliothek der FH Gießen vorhanden, Signatur dav 250 JAV.

[2] Gamma, Helm: »Entwurfsmuster«; Addison Wesley. Standardwerk zu Entwurfsmustern. Es existieren 15 Stück in der Lehrbuchsammlung der Bibliothek der FH Gießen.

[3] Stroustrup: »Die C++ Programmiersprache«. Für den Anfänger jedoch nicht besonders ideal. Es ist ein Standardwerk, daher eine Investition wert.

[4] Prof. Henrich: »Lückenskript C++«. Es muss mit einem Tafelmitschrieb ergänzt werden. Leichte Abweichungen vom Skript sind möglich. Es ist auf der Homepage von Prof. Henrich erhältlich.

[5] Eckel, Bruce: »C++ Inside & Out«; Osborne McGraw-Hill.

[6] Ulrich Breymann: »Objektorientierte Programmierung mit C++«; Hochschule Bremen; Kurs auf Basis des Buchs »C++ Einführung und professionelle Programmierung, 6. Auflage Hanser Verlag München«. Die Datei dieses Kurses datiert vom 21. November 2002 und kann unter http://www.informatik.hs-bremen.de/~brey/ heruntergeladen werden. Dieser Kurs ist leicht und schnell lesbar und sehr empfehlenswert!

[7] »C++ Programmieren III; WS 1996/97; Prof. Henrich«. Studentische Mitschrift aus der Skriptsammlung der Fachschaft MNI der FH Gießen http://www.fh-giessen.de/FACHSCHAFT/Informatik/cgi-bin/navi01.cgi?skripte, direkte Adresse http://www.fh-giessen.de/FACHSCHAFT/Informatik/data/skripte/c%2B%2B.zip. Kann als Orientierungshilfe dienen, was in der Vorlesung behandelt wird. Zum Lernen und Verstehen des Stoffes jedoch unnötig, da ist [6] besser geeignet.

[8] Frank B. Brokken : »C++ Annotations Version 5.2.0b« ftp://ftp.rug.nl/contrib/frank/documents/cplusplus. Ein bekanntes und beliebtes, englisches Dokument in C++. Erhältlich in HTML, PDF, PS. Es enthält u.a. hervorragende Dokumentation zur IO-Stream Library.

[9] David M. Wessels: »C++ Quick Reference«. Quelle: http://csciun1.mala.bc.ca:8080/~wesselsd/csci161/cpp.html. Catchphrase: »Copyright © 1997, David M. Wessels.«.

[10] Charles S. Tritt: »C++ Language Summary«. Quelle:http://people.msoe.edu/~tritt/cpplang.html. Catchphrase: » Portions copyright Charles S. Tritt, Ph.D.«.

[11] »C++ Standard Library Quick Reference«. Quelle: http://www.halpernwightsoftware.com/stdlib-scratch/quickref.html. Catchphrase »readers of book, The C++ Standard Library«. Erstellt für Leser des Buches »The C++ Standard Library from Scratch«.

[12] Matt Mahoney: »C++ Quick Reference«. Quelle: http://sourcepole.ch/sources/programming/cpp/cppqref.html. Catchphrase »// Replace F(1,2) with 1+2«. Besteht aus kommentierten Beispielbefehlen. Englisch.