Programmieren in C++

SS 2010

Vorlesung 11, Mittwoch 7. Juli 2010

(Vererbung und Polymorphismus, this Zeiger)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Erfahrungen mit dem 10. Übungsblatt
  - Nur noch drei Vorlesungen (inklusive dieser hier) ...
  - Anmeldung zur "Prüfung" / Programmierkurs A oder B
  - Treffen mit Ihrem Tutor ... jetzt wird's ernst !!
- Vererbung
  - Neben Templates der andere wichtige Mechanismus zur Wiederverwendung von Code / Vermeidung von code duplication
  - In dem Zusammenhang wichtig: virtual methods
  - Und was wir heute auch zum ersten Mal brauchen: this
  - Übungsblatt: gegeben ein x und eine Menge, finde das Element aus der Menge, das x am ähnlichsten ist. Einmal für Zahlen und einmal für Zeichenketten

Erfahrungen mit dem 10. Übungsblatt

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Die allermeisten fanden die Aufgabe schön
  - Vorlesungsaufzeichnung wurde von vielen vermisst
  - Musterlösung Ü9 wurde von einigen vermisst
  - Bei manchen sehr lange Laufzeit des Programms
  - Ein kleines Test-Wörterbuch benutzt statt dem Großen
  - "Offizielle" Antworten im Forum am Wochenende vermisst
  - Programmieren macht mehr Spaß als Fehler suchen
  - Linter nervt neuerdings wieder
  - Es gibt bei manchen Rückstau bei den Korrekturen

Nur noch drei Vorlesungen ...

UNI FREIBURG

- Und so schaut's aus
  - Heute ein letztes "normales" Übungsblatt
  - Ab nächster Woche beginnt dann schon das Projekt
  - Mehr Informationen dazu in der nächsten Vorlesung und auf dem dazu gehörigen Übungsblatt
  - In den letzten beiden Vorlesungen kommt dann noch diese und jene nützliche Sache und Information

Anmeldung zur "Prüfung"

- Anmeldetermin ist der **24. Juli 2010**
  - Das ist der Samstag nach der letzten Vorlesung
  - Man kann sich aber tatsächlich auch schon früher anmelden!
  - Wie gesagt, es gibt keine Prüfung, es geht darum, ob Sie die ECTS Punkte für diese Veranstaltung haben wollen
  - Das Projekt ist in gewissem Sinne die Prüfung, insbesondere gibt es die Noten erst dann wenn alle Ihr Projekt abgeschlossen haben
  - Wäre aber schön wenn Sie Ihr Projekt schon 2 – 3 Wochen nach Vorlesungsende fertig machen könnten
  - Ansonsten strikte Deadline 15. September 2010

Programmierkurs A oder B

UNI FREIBURG

- Wofür soll ich mich (als Informatiker/in) anmelden?
  - Programmierkurs A = 4 ECTS Punkte
  - Programmierkurs B = 2 ECTS Punkte
  - Der Grund für die Wahl A oder B ist ein rein technischer
    - laut göttlicher Ordnung muss jede/r Studierende am Ende auf **genau** 180 ECTS Punkte kommen können
    - je nach Nebenfach brauchen manche Leute dann aus diesem Kurs hier 4 oder 2 ECTS Punkte
    - wenn Sie sich für die 4 ECTS Punkte anmelden, können Sie nichts falsch machen, Sie haben dann am Ende höchstens 2 ECTS Punkte zuviel (freiwillig darf man)
  - Den ESE Studierenden kann das alles egal sein, die kriegen so oder so 6 ECTS Punkte für den Kurs

Treffen mit Ihrem Tutor

UNI FREIBURG

- Jetzt wird es ernst
  - Wer sich bisher noch keinmal getroffen hat, **unbedingt** bis zur nächsten Woche einen Termin ausmachen
  - Wer trotz wiederholter E-Mail nicht reagiert bekommt **keine** ECTS Punkte für den Kurs
  - Ausgenommen davon sind alle Teilnehmer/innen deren Tutor Felix Ruzzoli oder Yaser Öztürk ist
    - diese Teilnehmer/innen werde diese Woche zum ersten Mal angeschrieben
    - ab übernächster Woche gilt für sie aber ebenfalls die obige Regelung
  - Falls Sie sich treffen wollen, aber es Ihrer Meinung nach nicht an Ihnen liegt, dass es nicht klappt, E-Mail an mich

Vererbung

UNI FREIBURG

- Wofür braucht man das?
  - Möglichst gute Wiederverwendung von Code bzw. Vermeidung von code duplication bei zwei oder mehr Klassen die etwas sehr Ähnliches tun
  - Also im Prinzip derselbe Grund wie bei templates
  - Bei welcher Ähnlichkeit benutzt man templates?
    - wenn der Code zweier Klassen bis auf den Typ identisch ist, wie bei Array<int> und Array<char>
    - und manchmal auch aus Effizienzgründen … das machen wir aber nicht in dieser Vorlesung
  - Bei welcher Ähnlichkeit benutzt man Klassen?
    - wenn es Methoden / Daten gibt, die gemeinsam sind
    - sowie Methoden / Daten, die ganz verschieden sind

Vererbung

- Warum haben wir erst templates gemacht?
  - Wo doch Vererbung ein Grundprinzip des objektorientierten Programmierens ist !?
  - Sowohl templates als auch Vererbung können in fortgeschrittenen Anwendungen beliebig kompliziert und knifflig werden
  - Einfache Anwendungen von templates, wie unser Array<int> und Array<char> kann man aber sehr leicht verstehen, und kommen in der Praxis auch tatsächlich vor
  - Es gibt Spielzeugbeispiele für Vererbung, die man auch leicht verstehen kann, die führen aber eher von dem weg, was man in der Praxis braucht
  - Beispiele für Vererbung mit Praxisbezug sind schon nicht mehr so ganz einfach, deshalb machen wir das erst jetzt

Vererbung

- Beispiel
  - Eine Klasse RealNumber, die eine reelle Zahl darstellt
  - Eine Klasse ComplexNumber, die eine komplexe Zahl darstellt
  - Beide Arten von Zahlen kann man multiplizieren, allerdings beide auf Ihre eigene Weise
    - RealNumber: 0.5 * 4.2 = 2.1
    - ComplexNumber: (1 + 3**i**) * (2 - 2**i**) = 8 + 4**i**
  - Wenn man multiplizieren kann, kann man auch die k-te Potenz berechnen, für eine natürlich Zahl k
    - und zwar $x^k = x \cdot x \cdot \ldots \cdot x$ (k mal)
    - die Formel ist für RealNumber und ComplexNumber gleich
    - also sollte man die Funktion für die Berechnung der k-ten Potenz auch nur einmal implementieren

Vererbung

- Beispiel Fortsetzung
  - Also macht man den gleichen Teil in eine Oberklasse

```
class Number {
  public:
    void computeToThePowerOf(const Number&x, int k);
    virtual void computeProduct(const Number& x, const Number& y) = 0;
};
```

  - Das virtual bedeutet, dass wir diese Funktion in den speziellen Klassen RealNumber und ComplexNumber (kommen gleich) jeweils anders definieren werden
  - Das =0 bedeutet, dass wir für die Klasse Number gar keine Implementierung für computeProduct haben (wie denn auch)
  - So eine Klasse heißt abstrakt
  - Von einer abstrakten Klasse kann man kein Objekt erzeugen
  - Zeiger oder Referenzen auf abstrakte Klassen sind aber erlaubt, wofür das gut ist, sehen wir gleich beim Coden

Verberbung

- Beispiel Fortsetzung 2
  - Die eigenen Teile kommen in eine Unterklasse von Number

```
class RealNumber : public Number {
  public:
    virtual void computeProduct(const Number& x, const Number& y);
  private:
    double _value;
};
class ComplexNumber : public Number {
  public:
    virtual void computeProduct(const Number& x, const Number& y);
  private:
    double _realPart, _imagPart;
}
```

  - das : public Number heißt, dass wir alles von Number übernehmen, und zwar mit denselben public / private Rechten (das heißt, was dort public war bleibt public, was dort private war bleibt private)

Vererbung — Zeiger / Referenzen

- Ein paar Besonderheiten
  - Man kann einen Zeiger auf die Unterklasse als Zeiger auf die Oberklasse übergeben, und dasselbe mit Referenzen

```
void RealNumber::computeProduct(const Number& x, const Number& y);
...
RealNumber r1, r2, r3;
r3.computeProduct(r1, r2);  // This will compile and do the right thing.
```

  - Hat man eine Referenz / einen Zeiger auf die Oberklasse, und die / der eigentlich die Unterklasse meint, braucht man type casting

```
void RealNumber::print(const Number& x)
{
  const RealNumber& r = dynamic_cast<const RealNumber&>(x);
  printf("%d\n", r._value);
}
```

  - Unsert cpplint.py meckert bei dynamic_cast, benutzen Sie dann **static_cast**
  - dynamic_cast ist aber besser, weil es zur Laufzeit die Korrektheit des Typs prüft

Der this Zeiger

- this ist ein Zeiger
  - kann nur innerhalb einer Memberfunktion verwendet werden
  - er zeigt dann zur Laufzeit auf das Objekt, für das diese Memberfunktion aufgerufen wurde
  - Wenn die Klasse Number heißt, ist der Zeiger vom Typ Number*
  - Um ein Objekt vom Typ Number zu bekommen, muss man den Zeiger mit * dereferenzieren (so wie jeden anderen Zeiger auch), das heißt man schreibt *this
  - Beispiel siehe unser Number.cpp
  - Bemerkung: In Java schreibt man bei jedem Zugriff auf eine Membervariable / Memberfunktion this, in C++ nur wo nötig

Literatur / Links

- Vererbung / Inheritance
  - http://www.cplusplus.com/doc/tutorial/inheritance/
- Abstrakte Klassen / Virtuelle Methoden / Polymorphismus
  - http://www.cplusplus.com/doc/tutorial/polymorphism/
- Type casting / dynamic_cast / static_cast
  - http://en.wikipedia.org/wiki/Dynamic_cast
- Der "this" Zeiger
  - http://www.cplusplus.com/doc/tutorial/classes2/

    (Abschnitt "The keyword this")
- Insider FAQ
  - http://www.parashift.com