

# Augmented Reality

## Einführung in C/C++

Übung „Augmented Reality“  
Sebastian Boring, Arnd Vitzthum  
Montag, 23. Oktober 2006

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 1

## Ziele der Übung

- Erlernen von C/C++ und ARToolkit
- Verstehen der Mathematik für 3D-Räume
- Abschluss:
  - Teams á 3-4 Studenten
  - Programmieren einer kleinen Aufgabe

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 2

## Organisatorisches

- Termin:
  - Montag, 16:15 – 17:45, Raum 105
  - Pflichtveranstaltung!
  - Abmeldung bitte vor der Übung telefonisch oder per E-Mail
- Anforderungen:
  - Kenntnisse einer objekt-orientierten Sprache (z.B. Java)
  - Bearbeitung mit C/C++, ARToolkit

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 3

## Organisatorisches

- Bearbeitung der Aufgaben:
  - Abgabe ist jeweils das auf den Blättern genannte Datum um 12:00 Uhr s.t. !
  - Abgabe als ZIP-File (enthält Source-Code und ein Makefile)
  - In der Gruppenphase erfolgt die Abgabe über das SVN
- Scheinvergabe:
  - Alle Aufgaben müssen als korrekt (d.h. sie kompilieren und laufen) bewertet werden

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 4

## Organisatorisches

- Infrastruktur:
  - Betreuer:
    - Dipl.-Medieninf. Sebastian Boring:
      - [sebastian.boring@ifi.lmu.de](mailto:sebastian.boring@ifi.lmu.de)
    - Dipl.-Inf. Arnd Vitzthum:
      - [arnd.vitzthum@ifi.lmu.de](mailto:arnd.vitzthum@ifi.lmu.de)
  - SVN-Repository:
    - <svn://murx.medien.ifi.lmu.de/ws0607/ar>
    - Tutorial: <http://svnbook.red-bean.com>
  - LFE-Wiki:
    - <https://wiki.medien.ifi.lmu.de>

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 5

## Agenda (I)

- Vorstellung der Projektphase
- Einführung in GCC
- Aufbau eines C++-Programms
- Unformatierte und formatierte Ausgabe
- Namespaces
- Benutzereingaben
- Schleifen
- Arrays und Pointer

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 6

## Agenda (II)

- Funktionen
- Header-Dateien
- Klassen
- Vererbung
- Mehrfachvererbung
- Ausnahmen

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 7

## Einführung in GCC

- GCC = “GNU Compiler Collection”
- Sammlung von Kompilern für eine Menge von Programmiersprachen (u.a. C, C++, Java, Fortran, ...)
- Vollständige Liste der Flags auf der Manpage (`man gcc`)
- Kompilieren eines Programms mit einer einzigen Quelldatei:

```
gcc main.cpp
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 8

## Einführung in GCC

- Wichtige Flags:

- o [outfile]: Name der ausführbaren Datei
- c [cppfile]: erzeugt ein Objektfile (.o), das durch späteres Binden (Linken) zu einem ausführbaren Programm wird
- W[Warning]: Bsp. -Wall
- I[directory]: gibt die Verzeichnisse an, die Header-Dateien enthalten, die durch #include eingebunden werden

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 9

## Einführung in GCC

- Wichtige Flags:

- L[directory]: gibt die Verzeichnisse an, in denen Bibliotheken liegen
- l[library]: gibt die Bibliothek an, die eingebunden werden soll

- Weitere Informationen:

<http://www.ijon.de/comp/tutorials/makefile.html>

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 10

## Aufbau eines C++-Programms

- Jedes Programm enthält eine `main`-Methode:
  - `int main()`
  - `int main(int argc, char** argv)`
- Zusätzlich werden `#include` Anweisungen benötigt, um Zugriff auf weitere Funktionen zu erhalten:
  - `#include <iostream>`: Standard-I/O
  - `#include <cmath>`: math. Funktionen
  - Äquivalent in Java: `import ...`

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 11

## Aufbau eines C++-Programms

- Beispielprogramm:

```
#include <iostream>
int main(int argc, char** argv)
{
    std::cout << "AR" << std::endl;
    return 0;
}
```
- Kompilieren:

```
gcc ar_uebung.cpp -o ar_uebung
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 12

## Unformatierte und formatierte Ausgabe

- Unformatierte Ausgabe:
  - Ausgabestrom: `std::cout`
  - Format muss nicht angegeben werden, da sich dies aus dem Datentyp ergibt (ähnlich der Variante `toString()` in Java)
  - Nicht auf Standardtypen beschränkt
  - Alle Teil-Zeichenketten werden mit dem Operator `<<` getrennt

- Beispiel:

```
int i = 7;
std::cout << "#: " << i << std::endl;
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 13

## Unformatierte und formatierte Ausgabe

- Formatierte Ausgabe:
  - Funktion `printf()`
  - Argumente: Eine Zeichenkette plus ALLE vorkommenden Variablen
- Beispiel:

```
printf("Int: %d, Float %4.2f\n", 7, 4.2);
```
- Aufbau der Flags:
  - `%[flags][width][.precision][modifiers]type`

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 14

## Unformatierte und formatierte Ausgabe

- Wichtige Typen:
  - `c`: Character
  - `d` oder `i`: Signed decimal integer
  - `e` oder `E`: Wissenschaftliche Schreibweise (Exponent und Mantisse)
  - `f`: float und double
  - `s`: Kette von Characters
- Weitere Erläuterungen:  
<http://www.cplusplus.com/ref/cstdio/printf.html>

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 15

## Namespaces

- Bisher immer:
  - `std::cout`
- In diesem Fall ist `std` ein Namensraum (Namespace)
- Objekte in Namespaces können im Programm durch `using`-Direktiven global verwendet werden
- Beispiel:

```
using namespace std;
cout << "Blah blah blah!" << endl;
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 16



## Benutzereingaben

- Eingabestrom: `cin`
- Kein `endl` erforderlich
- `fflush(stdin)` leert den Eingabestrom
- Beispiel:

```
int i;  
cin >> i; // Einlesen eines int  
fflush(stdin);
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 17

## Schleifen

- Drei Schleifen (analog zu Java):
  - `for([start];[bedingung];[ende]) {...}`
  - `while([bedingung]) {...}`
  - `do{...}while([bedingung]);`

- Beispiele:

```
- for(int i=0; i<5; i++) {...}  
- while(i < 5) {...}  
- do {...} while(i < 5);
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 18

## Arrays und Pointer

- **Pointer:**
  - Verweise auf Daten, die an einer anderen Stelle stehen
  - Physikalisch: Adressen von Daten
  - Zwei Operatoren:
    - `&`: liefert die Adresse
    - `*`: dereferenziert den Zeiger, d.h. `*` liefert das, was an seiner Adresse steht

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 19

## Arrays und Pointer

- **Pointer (Beispiele):**

```
int x = 42;    //int
int* xp;      //Zeiger auf int
xp = &x;      //xp -> Adr. von x
*xp = 7;      //x = ?
```
- **Pointer (Irreführende Deklaration):**

```
int* p1, p2; //Erzeugt?
int *p1, p2;
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 20

## Arrays und Pointer

- Arrays (Felder):
  - Anlegen eines Arrays: `int werte[10];`
  - Zugriff auf Feldelemente mit `[]` Operator
  - Durchlauf auch mit Pointern möglich
- Beispiel (mit Pointern):

```
int werte[10];
for(int* p=werte; p<werte+10; p++)
{
    cout << *p << endl;
}
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 21

## Arrays und Pointer

- Arrays und Pointer:
  - Weist man einem Pointer ein Array zu, so erhält der Pointer die Adresse des ersten Elements
  - Addiert man zu dem Zeiger einen ganzzahligen Wert  $n$ , so wird der Zeiger um  $n$  Elemente weitersetzt
- Beispiel:

```
int werte[10];
int* wp = werte; //d.h. wp=&werte[0]
*werte = 88;     //d.h. werte[0]=88
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 22

## Arrays und Pointer

- C-Strings:
  - C-Strings sind Felder von Zeichen
  - Endkennung (d.h. letztes Element): `'\0'`
  - In C++ kapselt das Objekt `std::string` dies
  - Auch hier ist eine Iteration mittels Pointer möglich

- Beispiel:

```
const char* s = "hallo";  
for(const char* p=s; *p!='\0'; p++)  
{ ... }
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 23

## Funktionen

- [Rückgabety] [Name] ([Parameter]) {...}
- Wichtig:
  - Funktionen müssen vor dem Aufruf deklariert werden, d.h. sie müssen in den Zeilen über dem Aufruf stehen
- Funktionsprototypen umgehen diese Bedingung und stellen eine Signatur dar
- Beispiel für eine Signatur:  
`int factorial(int);`

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 24

## Header-Dateien

- Dienen zur zentralen Verwaltung von Definitionen, die in verschiedenen Modulen benötigt werden
- Enthalten typischerweise globale Konstanten, Variablen, Funktionen und Klassendefinitionen
- Bedingte Übersetzung:

```
#ifndef HEADER_H
#define HEADER_H
...
#endif
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 25

## Klassen

- Enthalten Attribute und Funktionen
- Werden typischerweise in Header-Dateien deklariert
- Beispiel einer Deklaration:

```
class Person {
private:
    char* name;
    int age;
public:
    void setName(char*);
    void setAge(int);
};
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 26

## Klassen

- Deklaration von Funktionen mit `inline`:
  - Implementierung direkt in der Deklaration
  - Zwei verschiedene Arten:
    - Direkt in der Deklaration (ohne `inline`)
    - Im Header-File nach der Deklaration (mit `inline`)
  - Beispiel (direkt in der Deklaration):

```
... // statt void setAge(int);
void setAge(int age) {
    this->age = age;
}
```

...

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 27

## Klassen

- Deklaration von Funktionen mit `inline`:
  - Beispiel (mit `inline`):

```
class Person {
    ...
    void setAge(int);
    ...
};
inline void Person::setAge(int a) {
    age = a;
}
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 28

## Klassen

- Erzeugen von Objekten:

- Statische Erzeugung:

```
Person peter;
```

```
Person john;
```

- Objekte werden gelöscht, wenn der Gültigkeitsbereich des Objekts verlassen wird

- Dynamische Erzeugung:

```
Person* peter;
```

```
peter = new Person;
```

```
delete peter; // ruft Destruktor auf
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 29

## Klassen

- Konstruktoren:

- Definiert als Methode einer Klasse
  - Name ist identisch zum Namen der Klasse
  - Kein Rückgabewert (auch nicht `void`)
  - Mehrere Konstruktoren durch Überladen
  - Deklaration im `public` Abschnitt
  - Minimaler Default-Konstruktor wird erzeugt, falls kein Konstruktor vorhanden ist

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 30

## Klassen

- Konstruktoren:

- Aufruf bei der Erzeugung durch `new` oder bei der Erzeugung eines statischen Objekts

- Minimaler Konstruktor:

```
Person::Person() { }
```

- Standard-Konstruktor:

```
Person::Person() {  
    name = NULL;  
    age = 0;  
}
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 31

## Klassen

- Konstruktoren:

- Andere Konstruktoren:

```
Person::Person(char* n, int i = 0) {  
    name = n;  
    age = i;  
}
```

```
Person::Person(char* n, int i = 0) :  
    name(n), age(i) {...}
```

- Beschreibt zwei Konstruktoren:

```
Person::Person(char* n) {...}
```

```
Person::Person(char* n, int i) {...}
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 32



## Klassen

- Beispiele für statisches Erzeugen:

```
Person::Person; // Default
Person::Person("Peter"); // age = 0
Person::Person("Peter", 45);
```

- Beispiele für dynamisches Erzeugen:

```
peter* = new Person::Person;
peter* = new Person::Person("Peter");
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 33

## Klassen

- Löschen von Objekten:

- Mittels Destruktor (beginnt mit ~)
- Wird aufgerufen, sobald der Gültigkeitsbereich eines statischen Objekts verlassen wird, oder `delete` aufgerufen wird
- Minimaler Destruktor:

```
Person::~~Person() {}
```
- Besitzt keinen Rückgabetyt (wie Konstruktor)
- Nur ein Destruktor möglich
- Sinn: Löschen von Unterobjekten

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 34

## Klassen

- Zugriffsoperator:
  - Mit dem Punkt-Operator .
  - Bei Zeigern auf ein Objekt mit Pfeil ->

- Beispiel:

```
Person peter;  
Person* john = new Person;  
peter.setName("Peter");  
john->setAge(35);  
delete john;
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 35

## Vererbung

- Syntax:

```
class subclass :  
    [modifier] superClass1,  
    [modifier] superClass2,  
    ...  
    [modifier] superClassN  
{  
    ...  
};
```

- Keine Vererbung für Konstruktoren, Destruktor und Zuweisungs-Operator

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 36

## Vererbung

- Generell gilt:

- Alles was nicht überschrieben wird, wird geerbt
- Zugriff auf eine überschriebene Methode der Superklasse (**super** . [method] in Java):

- Scope-Operator ::
- Beispiel: Employee erbt von Person

```
Employee a;  
a.print();  
a.Person::print();
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 37

## Vererbung

- Beispiel:

```
class Employee : public Person {  
    void print() {  
        // print(); // endlos  
        Person::print();  
        cout << "blah" << endl;  
    }  
};  
Employee a;  
a.print();  
a.Person::print();
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 38

## Vererbung

- Konstruktoren:

- Elemente der Basisklasse werden durch Konstruktor der Basisklasse initialisiert

- Initialisierungsliste:

```
class_name::class_name(parameter_list)  
    : superClass1(parameters),  
      superClass2(parameters), ...
```

- Konstruktoren der Basisklasse werden **vor** dem Konstruktor der abgeleiteten Klasse ausgeführt!

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 39

## Vererbung

- Destruktoren:

- Löschen von Elementen der Basisklasse **muss** in der Basisklasse erfolgen

- Destruktor der Basisklasse wird **automatisch** aufgerufen **nach** dem Destruktor der abgeleiteten Klasse

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 40

## Vererbung

- Zeiger auf Objekte:
  - Zeiger auf ein Objekt einer abgeleiteten Klasse kann einem Zeiger auf eine Basisklasse zugewiesen werden:
    - Unterklasse ist eine Erweiterung
  - Umkehrung funktioniert nicht:
    - Typecasting notwendig
  - Allgemeine Regel:
    - Speziellere Typen (abgel. Klasse) können einem allgemeinerem Typ (Basisklasse) zugewiesen werden

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 41

## Vererbung

- Typecasting:
  - **reinterpret\_cast**<type>(parameter);
    - Keine Überprüfungen, keine Wertänderung
    - Zeigertyp in beliebigen anderen Zeigertyp ändern
  - **static\_cast**<type>(parameter);
    - Implizit mögliche Casts und Umkehrung möglich
    - Basisklasse <-> abgel. Klasse, int <-> char
    - Keine echte Überprüfung auf Typkompatibilität
  - **dynamic\_cast**<type>(parameter);
    - Laufzeittyp wird bei der Überprüfung verwendet
    - Wie das Casting in Java

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 42

## Vererbung

- Virtuelle Methoden:
  - Zugriff auf Elemente einer abgeleiteten Klasse nur nach einem Casting des Zeigers
  - **Ziel:** Ausführung einer geeigneten Methode der Unterklasse (ohne diese explizit zu kennen (bei Java immer so))
  - **Lösung:** Virtuelle Methoden → Semantik wird vom Objekt und nicht vom Zeiger definiert
  - Realisierung mit dem Schlüsselwort `virtual` in der Basisklasse
  - Speicherverbrauch steigt (Objekt enthält Zeiger auf die *vtable* der Klasse)

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 43

## Vererbung

- Beispiel in Java:

```
class Person {...
    public void print();
}
class Employee extends Person {...
    public void print();
}
Person p = new Person();
Person pe = new Employee();
p.print(); // Person.print();
pe.print(); // Employee.print();
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 44

## Vererbung

- Beispiel in C++ (ohne virtuelle Methoden):

```
class Person {...
    public: void print();
};
class Employee : public Person {...
    public: void print();
};
Person* p = new Person;
Person* pe = new Employee;
p.print();      // Person::print();
pe.print();     // Person::print();
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 45

## Vererbung

- Beispiel in C++ (mit virtuellen Methoden):

```
class Person {...
    public: virtual void print();
};
class Employee : public Person {...
    public: void print();
};
Person* p = new Person;
Person* pe = new Employee;
p->print();     // Person::print();
pe->print();    // Employee::print();
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 46

## Vererbung

- Virtuelle Destruktoren:
  - Dynamisch erzeugte Objekte können einem Zeiger der Oberklasse zugewiesen werden
  - Wird das Objekt gelöscht, wird nur der Destruktor der Oberklasse aufgerufen
  - **Lösung:** virtueller Destruktor
  - Das Schlüsselwort `virtual` wird in der Basisklasse vor dem Konstruktor angegeben

```
class class_name {  
    virtual ~class_name() {...}  
};
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 47

## Vererbung

- Abstrakte Klassen:
  - Nicht alle deklarierten Methoden werden implementiert
  - Instanzen dieser Klasse gibt es nicht
  - Unterklassen können Instanzen haben, sofern alle deklarierten Methoden implementiert sind
- Syntax:

```
class class_name {  
    virtual return_type method_name(parameter_list)=0;  
};
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 48



## Mehrfachvererbung

- Unterklasse hat mehrfache Basisklassen
- Unterklasse enthält jede Basisklasse
- Konstruktor der abgel. Klasse kann Konstruktoren von allen Basisklassen aufrufen
- Wird ein Objekt einer abgel. Klasse vernichtet, werden die Destruktoren aller Basisklassen gerufen

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 49

## Mehrfachvererbung

- Beispiel:

```
class Base1 {...
    public: Base1(int, char*);
};
class Base2 {...
    public: Base2(int, float);
};
class Derived : public Base1,
                public Base2 {...
    public: Derived(char* s, int i) :
        Base1(i, s), Base2(i, 4.2f) {...}
};
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 50

## Mehrfachvererbung

- **Problem:** Mehrdeutigkeiten bei Namenskollisionen
- Zwei oder mehr Basisklassen haben gleiches Element:
  - Member-Variablen
  - Methoden (gleicher Name und Parameter)
  - Nutzlos: `private`
- Scope-Operator `::` als Lösung:
  - Methode überschreiben und gewünschte Methode der Basisklasse mit `::` auswählen

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 51

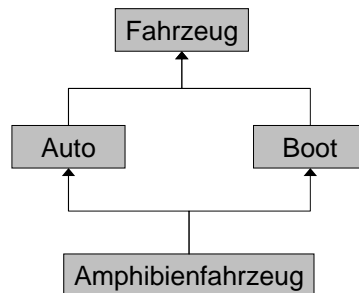
## Mehrfachvererbung

- **Beispiel:**
  - *Amphibienfahrzeug* ist sowohl *Auto* als auch *Boot*
  - Deklarieren beide Basisklassen eine Methode `move(int d)` gibt folgender Aufruf einen Fehler:

```
Amphibienfahrzeug a;  
a.move(10);
```
  - Es ist ein mehrdeutiger Aufruf, da nicht klar ist, ob `move` der Klasse `Boot` oder der Klasse `Auto` aufgerufen werden soll

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 52

## Mehrfachvererbung



- Klasse `Fahrzeug` wird mehrfach eingebunden und damit auch ihre Attribute und Methoden

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 53

## Mehrfachvererbung

- **Lösung:** Erben mit dem Schlüsselwort `virtual` führt dazu, dass die Komponenten nur einmal eingebunden werden
- Zugriff ohne Bereichsoperator eindeutig und deshalb möglich

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 54

## Ausnahmen

- Drei Schlüsselwörter:
  - Versuchsblock: `try`
  - Ausnahme erzeugen: `throw`
  - Abfangen: `catch`
- Es existiert kein `finally`
- Exceptions sind nicht Teil der Methodensignatur, d.h. sie können überall geworfen werden
- Fangen aller Exceptions: `catch(...)`

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 55

## Ausnahmen

- Funktionen können eine Exception-Liste angeben
- Das Schlüsselwort `throw` im Funktionsprototyp:

```
return_type method_name(parameters)
    throw (exception_list) {...}
```
- Funktionen können dennoch weitere Exceptions erzeugen (auch wenn diese nicht in der Liste enthalten sind)

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 56

## Ausnahmen

- Abfangen von Ausnahmen:

```
try
{
    ...
    if(error) throw exception_class(...);
}
catch(exception_class variable)
{
    // exception handling
}
```

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 57

## Literatur

- Nicolai Josuttis, “Objektorientiertes Programmieren in C++”, ISBN 3-8273-1771-1
- <http://www.cplusplus.com/ref/cstdio/printf.html>
- <http://www.ijon.de/comp/tutorials/makefile.html>

LMU München – Medieninformatik – Butz/Boring/Vitzthum – Augmented Reality – WS06/07 – 23.10.2006 – Folie 58