Informatik II

Oliver Jack

Fachhochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2010

Inhalt

Vorlesung 7. Vererbung Teil 1

Vererbung

- Konzept und Definition
- UML-Klassendiagramme
- Beispiele

Zusammenfassung

7. Vererbung Teil 1

Vorige Vorlesung

- Komplexe Zahlen als Klasse
- Friend-Funktionen
- Selbst-definierte Ein- und Ausgabe-Operatoren
- Klassen mit dynamischen mehrdimensionalen Feldern

Heutige Vorlesung

- Konzept der Vererbung
- UML-Klassendiagramme
- Wiederverwendung am Beispiel binärer Suchbaum

Lernziele dieser Vorlesung

- Verständnis der Ableitung von neuen Klassen aus vorhandenen Klassen
- Kenntnis der Darstellung von Klassen in UML-Diagrammen
- Verständnis einer komplexen Datenstruktur und der zugehörigen Algorithmen

Vererbung: Konzept...

- Datenabstraktion ist eine effektive Methode zur Erweiterung des vordefinierten Typsystems, wenn ein einziges, klar definiertes Konzept vorliegt, z. B. komplexe Zahlen
- Oft steht aber nur ein abstrakter Datentyp, eine Klasse, zur Verfügung, die eine bestimmte Situation nur teilweise bzw. annähernd beschreibt
- Es kann auch eine Zusammenstellung von Klassen verfügbar sein, die sich in Bedeutung und Implementierung ähnlich sind, aber nicht identisch
- Hier ist Vererbung eine nützliche Methode zur Ergänzung der Datenabstraktion.

. . .Vererbung: Konzept. . .

- Vererbung bietet die Möglichkeit hierarchischer Klassifizierung von Datentypen (Klassen)
- Eine Klasse kann aus einer anderen Klasse durch
  - Erweiterung oder
  - Spezialisierung

  abgeleitet werden

. . .Vererbung: Konzept. . .

- Die Klasse, aus der abgeleitet wird, heißt Basisklasse, sie vererbt ihre Members (Variablen und Funktionen)
- Die Klasse, die ableitet, heißt abgeleitete Klasse
  - sie erbt die Members ihrer Basisklasse
  - sie kann Leistungsmerkmale an bestimmte Anforderungen anpassen und neue Leistungsmerkmale hinzufügen

. . .Verebung: Konzept

Ableitung von Klassen ermöglicht

- die Implementierung von generischen Datentypen
- durch spezielle Anpassung (Ableitung) die Wiederverwendung bewährten Programm-Codes
- Strukturierung von Datentypen

Definition...

class *klassenbezeichner* : [public|protected|private] *basisklasse* {
[private:]
    private daten und funktionen
[protected:]
    geschützte daten und funktionen
[public:]
    öffentliche daten und funktionen
};

. . .Definition

- Im Kopf der Klassendefinition einer abgeleiteten Klasse kann ein Zugriffsmodus (`public`, `protected` oder `private`) auf die Basisklasse angegeben werden
- Der Zugriffsmodus bestimmt die Möglichkeiten des Zugriffs auf Member-Variablen und Member-Funktionen innerhalb der Vererbungshierarchie

Zugriffsmodi

- `private`: geschützte und öffentliche Members der Basisklasse sind privat in der abgeleiteten Klasse
- `protected`: geschützte und öffentliche Members der Basisklasse sind geschützt in der abgeleiteten Klasse
- `public`: geschützte Members der Basisklasse sind geschützt und öffentliche Members der Basisklasse sind öffentlich in der abgeleiteten Klasse
- Private Members der Basisklasse sind in der abgeleiteten Klasse nicht zugreifbar
- Geschützte Members der Basisklasse sind nur in der Basisklasse und in abgeleiteten Klassen zugreifbar

UML-Klassendiagramme...

- UML (Unified Modelling Language)
- UML-Klassendiagramme beschreiben
  - Objekte, deren Eigenschaften, also Attribute (Member-Variablen) und Methoden (Member-Funktionen),
  - die Abhängigkeiten zwischen Objekten.
- Klassendiagramme sind nur ein Teil der UML, weitere Teile sind z.B. Use-Case Diagramme und Zeit-Sequenzdiagramme (hier nicht behandelt).

. . .UML-Klassendiagramme. . .

- Klassen werden im UML Klassendiagramm als Rechtecke mit einem Namen abgebildet.
- Unterhalb des Klassennamens werden Klassenvariablen angezeigt,
- darunter Methoden mit Parametern.

. . .UML-Klassendiagramme. . .

Variablen und Methoden haben die folgende Syntax:

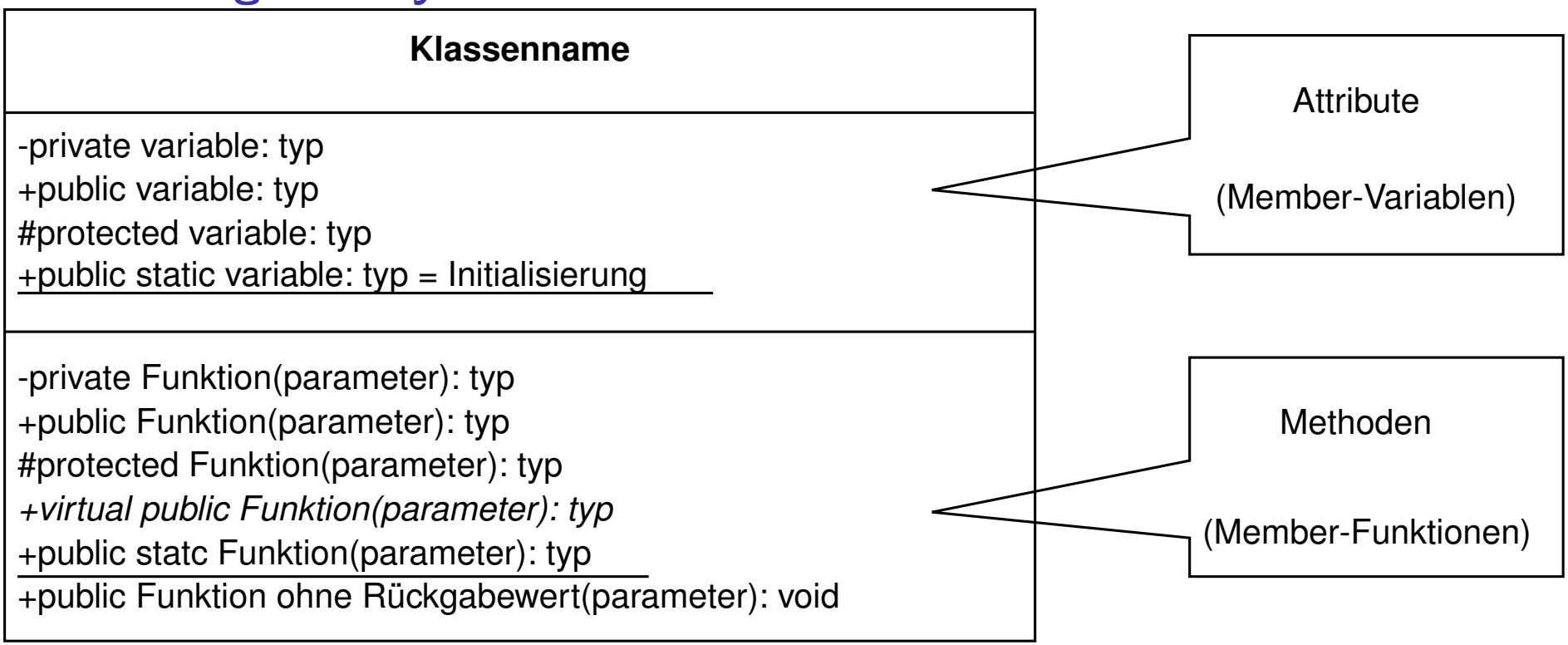
- <Sichtbarkeit> <Name> <Parameter> :<Rückgabewert bzw. Objekttyp>
- [Sichtbarkeit: +(public), nichts(default), -(private), #(protected)]
- static Methoden sind unterstrichen.

. . .UML-Klassendiagramme. . .

- Abstrakte Klassen und Methoden werden *kursiv* geschrieben.
- Wie exakt ein UML- Diagramm spezifiziert ist, hängt vom Anwendungsfall ab.
- Es werden teilweise nur die wesentlichen Attribute und Methoden angegeben.

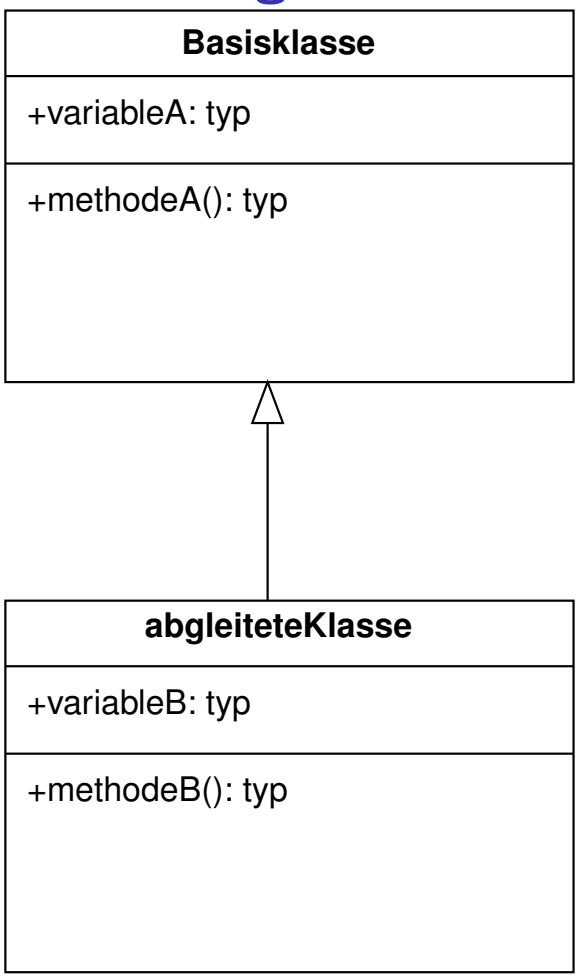
. . .UML-Klassendiagramme. . .

Darstellung und Syntax einer Klasse



. . .UML-Klassendiagramme. . .

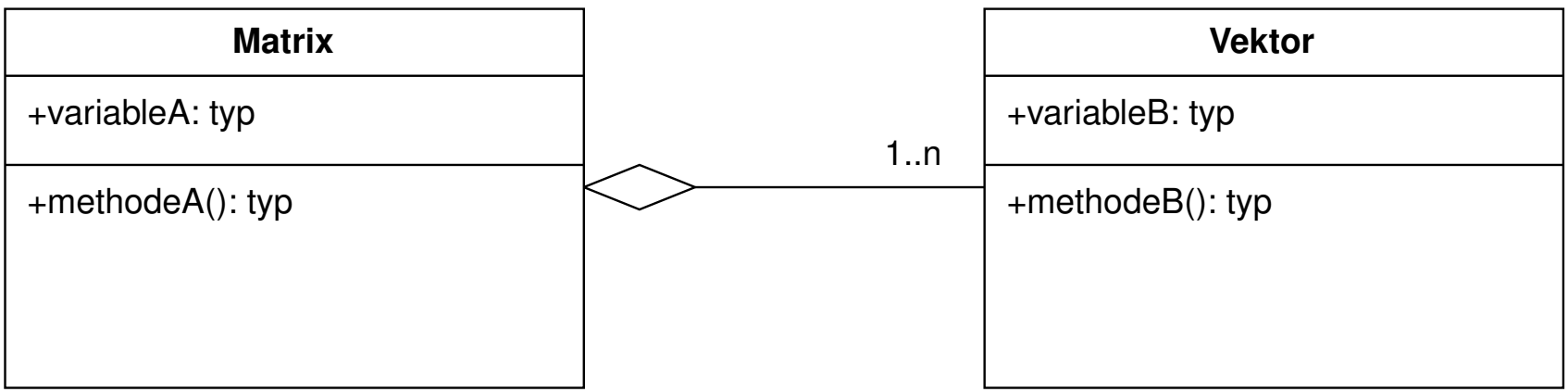
Vererbung



. . .UML-Klassendiagramme. . .

Aggregation

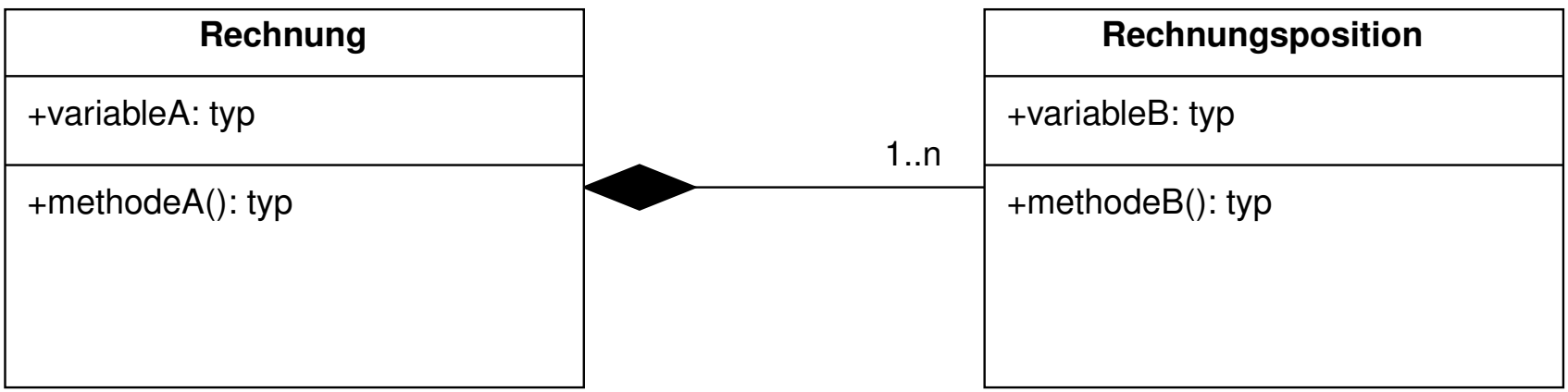
- Aggregation ist eine Form der Assoziation zwischen Klassen.
- Matrix besteht aus 1 bis n Vektoren.
- Bei beliebig vielen Objekten wird statt 1..n der Stern (*) angegeben.



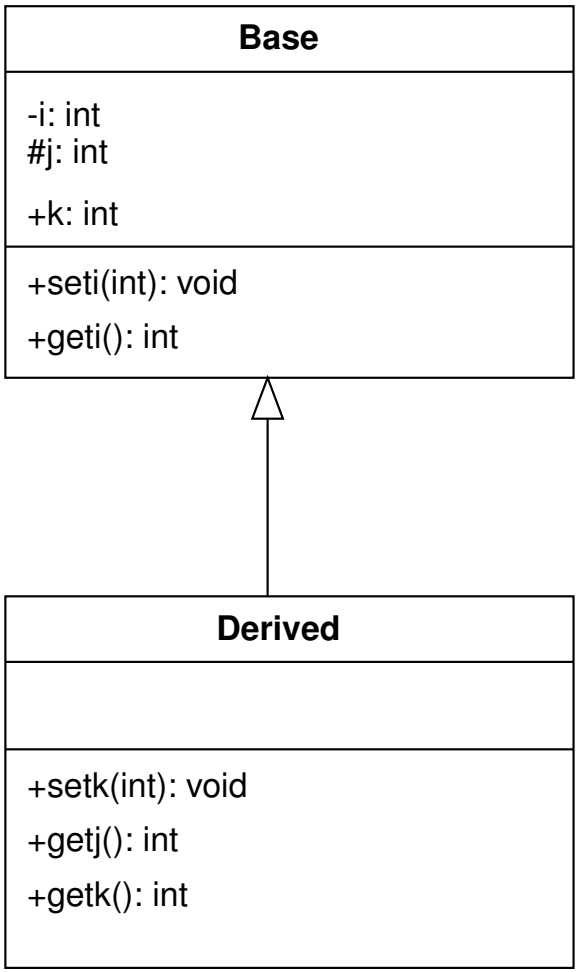
. . .UML-Klassendiagramme

Komposition

- Komposition ist eine Form der Assoziation zwischen Klassen.
- Rechnung besteht aus 1 bis n Rechnungspositionen.
- Im Unterschied zur Aggregation kann die Rechnungsposition nicht ohne das „Ganze“, d. h. die Rechnung existieren.
- Bei beliebig vielen Objekten wird statt 1..n der Stern (*) angegeben.



Beispiel Vererbung: Klassendiagramm



Beispiel Vererbung: Programm

```
class Base {
private:
  int i;
protected:
  int j;
public:
  int k;
  void seti (int a) { i = a; }
  int geti () { return (i); }
};
class Derived: protected Base {
public:
  // j ist protected
  void setj (int a) { j = a; }
  // k ist protected
  void setk(int a) { k = a; }
  int getj () { return (j); }
  int getk () { return (k); }
};
```

```
int main ()
{
  Derived ob;
  // illegal, da protected
  // in derived!
  // ob.seti (10);
  // illegal, da protected
  // in derived!
  // cout << ob.geti ();
  // illegal, da k protected!
  // ob.k = 10;

  ob.setk (10);
  cout << ob.getk () << ' ';
  ob.setj (12);
  cout << ob.getj () << ' ';

  return(0);
}
```

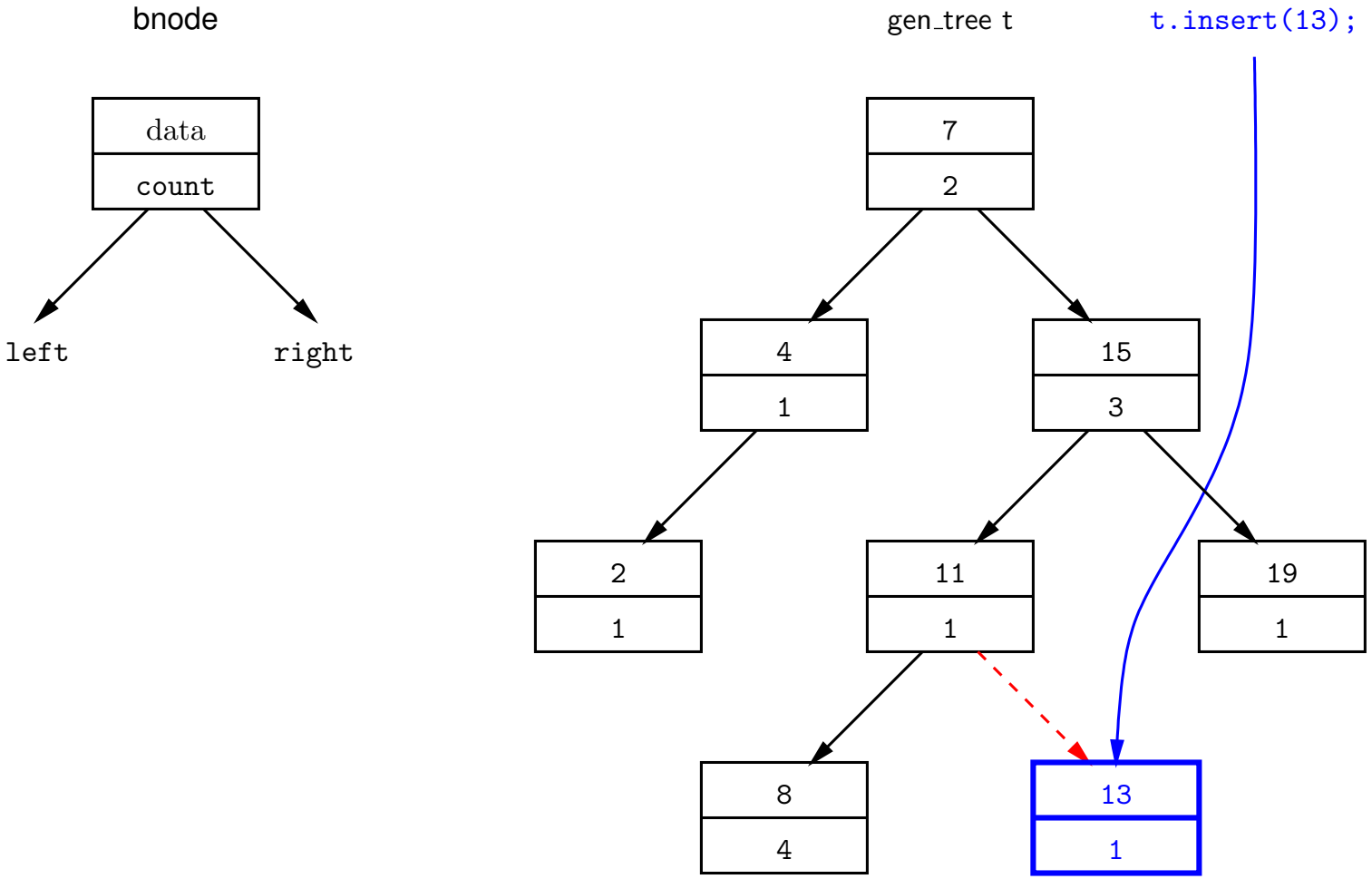
Wiederverwendung

- Klassenhierarchien können verwendet werden, um generische Klassen zu bilden
- Generische Klassen geben nur eine bestimmte Implementierungsstruktur vor, sie sind Basisklassen
- Eine konkrete Implementierung erfolgt in abgeleiteten Klassen für spezifische Ausprägungen der Basisklasse
- Generische Klassen bilden eine Schnittstelle zu bestimmtem Programm-Code, der in abgeleiteten Klassen wiederverwendet wird

Beispiel Binärer Suchbaum...

- Ein binärer Suchbaum ist eine Datenstruktur, die Elemente gemäß einer Sortierung speichert
- Jeder Knoten des Baumes besitzt zwei Nachfolger (`left`, `right`)
- Die Nachfolger sind ebenfalls binäre Suchbäume
- Im Nachfolger `left` befinden sich alle Elemente, die kleiner sind als das Element im Knoten
- Im Nachfolger `right` befinden sich alle Elemente, die größer sind als das Element im Knoten
- Jeder Knoten besitzt zwei Dateneinträge: (1) Element (`data`), (2) Vielflachheit (`count`) des Elements

. . .Beispiel Binärer Suchbaum. . .



...Beispiel Binärer Suchbaum

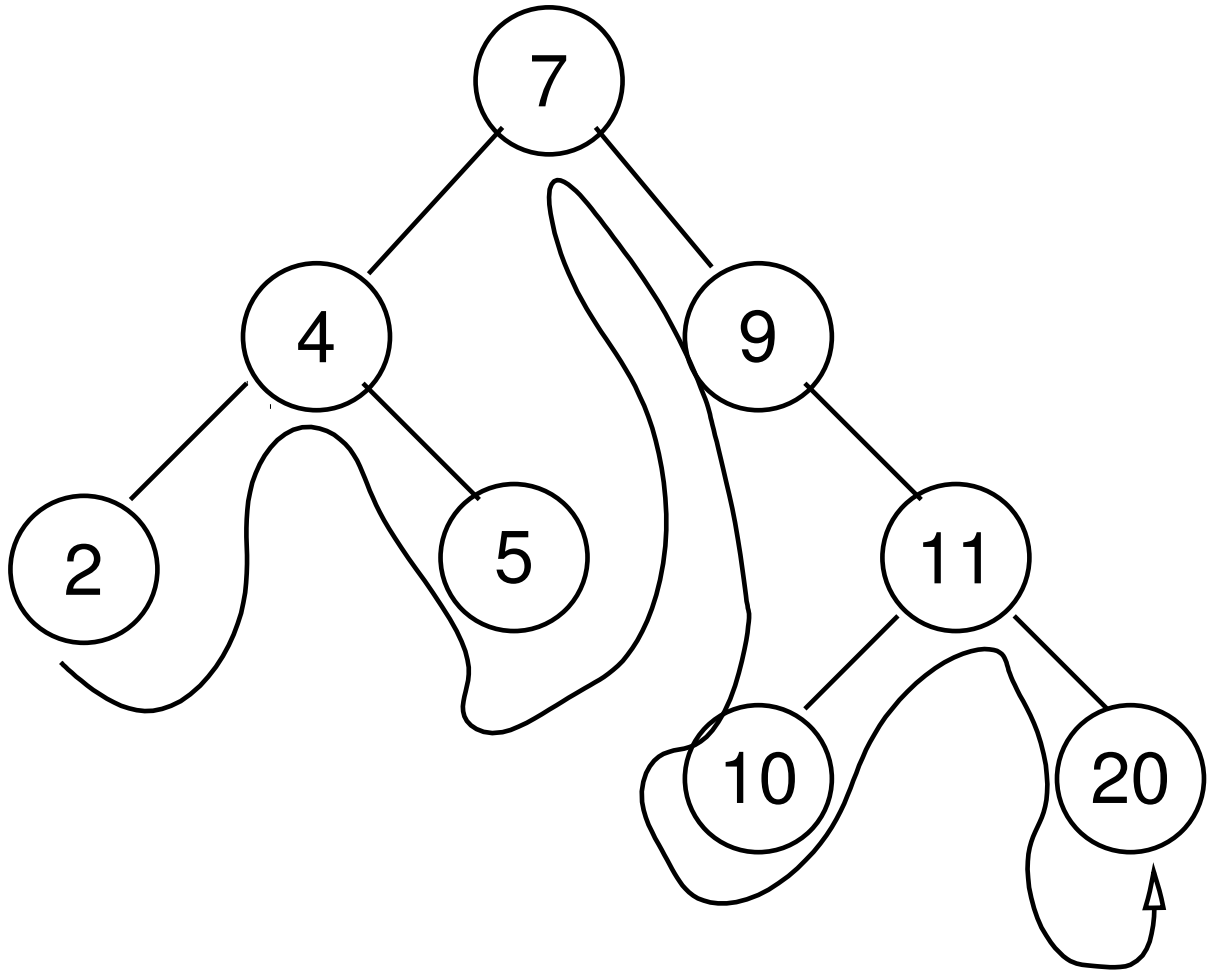
Baumdurchläufe (Traversieren)

- Pre-order (Hauptreihenfolge): Erst der Knoten, dann den linken Teilbaum, dann den rechten Teilbaum
- In-order (symmetrische Reihenfolge): Erst den linken Teilbaum, dann den Knoten, dann den rechten Teilbaum
- Post-order (Nebenreihenfolge): Erst den linken Teilbaum, dann den rechten Teilbaum, dann den Knoten

Eigenschaften

- In-order-Durchlauf: Ausgabe der Daten in aufsteigender Sortierung
- Pre-order-Durchlauf: Finden eines Elementes

Beispiel In-order-Traversieren



Ausgabefolge: 2, 4, 5, 7, 9,10, 11, 20

Beispiel Binärer Suchbaum gentree.h...

```
// Generische binaere Suchbaeume
// generischer Zeiger, verweist in
// abgeleiteter Klasse auf Daten
typedef void *p_gen;

class bnode { // Knoten
private:
  friend class gen_tree;
  bnode *left, *right;
  p_gen data;
  int count; // Vielfachheit eines Datums
  // Konstruktor
  bnode(p_gen d, bnode *l, bnode *r)
    : data (d), left (l), right (r), count (1)
  {}
  // Vergleich von Daten
  friend int comp (p_gen a, p_gen b);
  friend void print (bnode *n);
};
```

...Beispiel Binärer Suchbaum gentree.h

```
class gen_tree { // Baum
protected:
  bnode *root;
  p_gen find (bnode *r, p_gen d);
  void print (bnode *r);
public:
  gen_tree () // Standard-Konstruktor
  {
    root=0; // leerer Baum
  }
  void insert (p_gen d);
  p_gen find (p_gen d)
  {
    return (find (root, d));
  }
  void print ()
  {
    print(root);
  }
};
```

Beispiel Binärer Suchbaum: Klasse...

- Benutzung eines `void`-Zeigers (generischer Zeiger) für die Elemente, damit in abgeleiteten Klassen beliebige Typen deklariert werden können
- Eine Klasse `bnode`, die einen Knoten des Baums beschreibt
- Eine Klasse `gen_tree`, die den binären Suchbaum beschreibt. Member: `bnode`, geschützt, damit nur in abgeleiteten Klassen darauf zugegriffen werden kann
- Alle Members in `bnode` sind privat, `gen_tree` ist `friend`, kann also auf alle Daten von `bnode` zugreifen
- Konstruktor in `bnode` mit Initialisierungsliste

...Beispiel Binärer Suchbaum: Klasse

- Member-Funktionen in `gen_tree`: Konstruktor (erzeugt leeren Baum), Einfügen (`insert()`), Suchen (`find ()`), Ausgeben (`print ()`)
- Vergleichsoperation für Elemente (`comp ()`) ist datentypunabhängig, wird außerhalb der Klasse definiert, (`friend` von `bnode`)
- Gleiches gilt für die Ausgabefunktion `print ()`
- Im geschützten Teil von `gen_tree` gibt es eine Hilfsfunktion `find(bnode *r, p_gen d)` (sucht im Teilbaum `r` nach `d`)
- Im öffentlichen Teil gibt es die Funktion `find (p_gen d)` (sucht im gesamten Baum nach `d`)
- Gleiche Konstruktion für die Ausgabe (`print ()`)

Beispiel Binärer Suchbaum gentree.cpp...

```
#include "gentree.h"
void gen_tree::insert (p_gen d)
{
// Erstes Element im Baum, einfuegen in leeren Baum
  bnode *temp = root;
  bnode *old;
  if (root == 0) {
    root = new bnode (d, 0, 0);
    return;
  }
```

...Beispiel Binärer Suchbaum gentree.cpp...

```
  while (temp != 0) { // zum "richtigen" Knoten bewegen
    old = temp;
    if (comp (temp->data, d) == 0) { // Element vorhanden
      (temp->count)++; // Vielfachheit erhoehen
      return; // fertig
    }
    if (comp (temp->data, d) > 0)
      temp = temp->left; // weiter nach links
    else
      temp = temp->right; // weiter nach rechts
  }
  if (comp (old->data, d) > 0)
    old->left = new bnode (d, 0, 0); // kleineres Element links
  else
    old->right = new bnode (d, 0, 0); // groesseres Elemet rechts
  return;
}
```

...Beispiel Binärer Suchbaum gentree.cpp...

```
// Pre-order Durchlauf, binaere Suche
p_gen gen_tree::find (bnode *r, p_gen d)
{
  if (r == 0)
    return (0);
  else if (comp (r->data, d) == 0)
    return (r->data);
  else if (comp (r->data, d) > 0)
    return (find (r->left, d));
  else
    return (find (r->right, d));
}
```

...Beispiel Binärer Suchbaum gentree.cpp...

```
// In-order Durchlauf, sortierte Ausgabe
void gen_tree::print (bnode *r)
{
  if (r != 0) {
    print(r->left);
    ::print (r); // Nicht-Member-Funktion!
    print (r->right);
  }
}
```

Beispiel Binärer Suchbaum: Eigenschaften

- Die generische Klasse ist nicht zur Erzeugung von Instanzen geeignet
- Es muss eine Klasse abgeleitet werden, in der der Datentyp der Einträge definiert ist
- Die datentypabhängigen Funktionen für den Elementvergleich (`comp()`) und die Ausgabe (`print()`) müssen definiert werden
- Alle Funktionen, die auf Elemente (generischer Datentyp `void *p_gen`) zugreifen, müssen mit einer Typkonversion arbeiten
- Die abgeleitete Klasse leitet mit dem Zugriffsmodus `private` von `gen_tree` ab, denn es soll keine weitere Konkretisierung der Datentypen geben

Beispiel Binärer Suchbaum: Konkretisierung...

```
#include <iostream>
#include <string.h>
#include "gentree.h"
using namespace std;
class s_tree: private gen_tree { // Ableiten
public:
  s_tree () { cout << "Baum erzeugt" << endl; }
  void insert(char *d) //
  { gen_tree::insert (p_gen (d)); }
  char *find (char *d)
  { return ((char*)
     (gen_tree::find (p_gen (d)))); }
  void print () { gen_tree::print (); }
};
```

...Beispiel Binärer Suchbaum: Konkretisierung

```
// C-Stil String-Vergleich
int comp (p_gen i, p_gen j)
{
  return (strcmp ((char*)(i), (char*)(j)));
}

// Daten und Vielfachheit
void print (bnode *n)
{
  cout << (char*)(n->data) << "␣(";
  cout << n->count << ")␣␣";
}
```

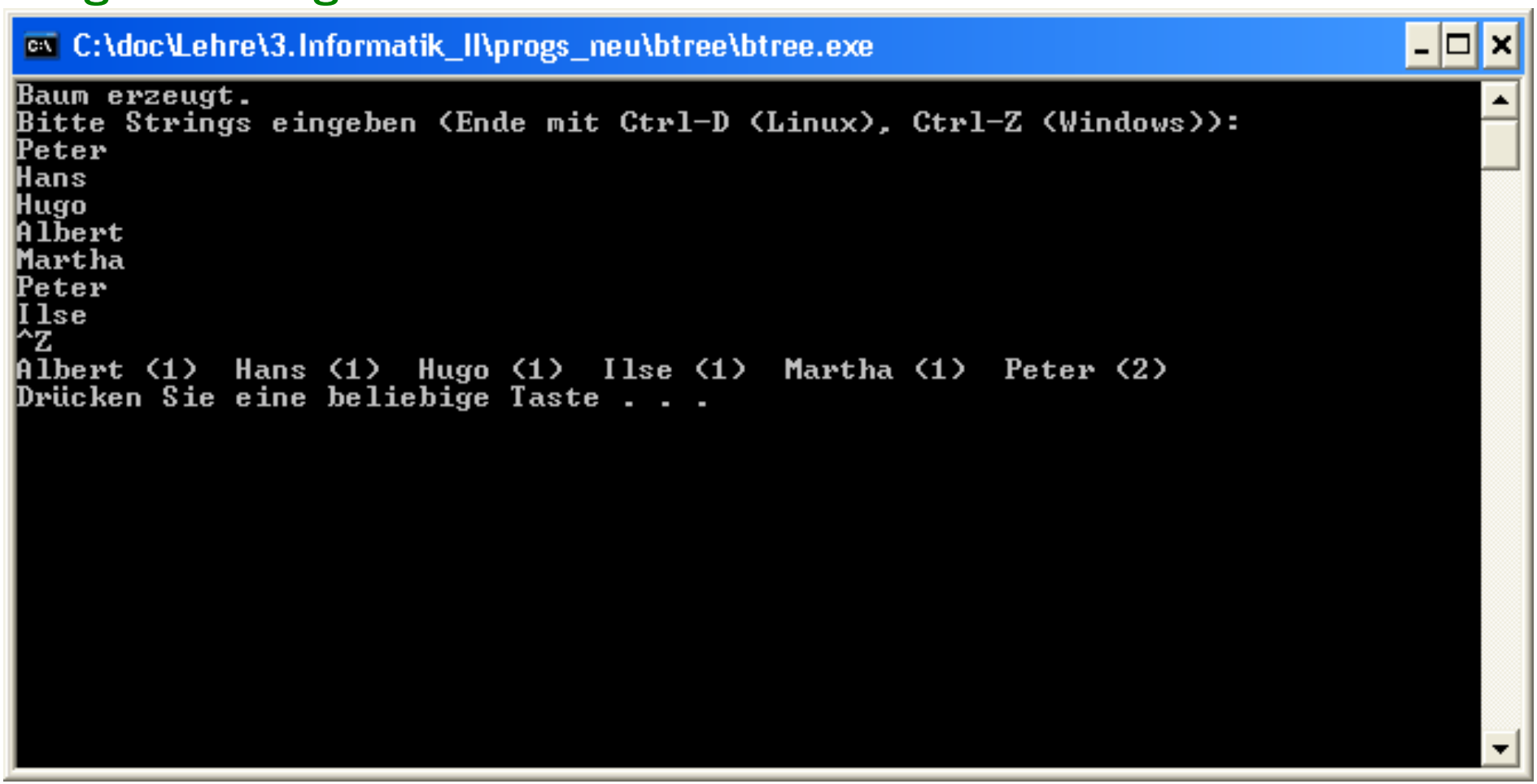
Beispiel Binärer Suchbaum: Benutzung...

```
int main()
{
  s_tree daten; // Suchbaum von Zeichenketten
  char datum[80]; // Zeichenkette
  char *el; // Element des Suchbaums

  cout << "Bitte␣Strings␣eingeben␣";
  cout << "(Ende␣mit␣Ctrl-D␣(Linux),␣Ctrl-Z␣(Windows)):␣"
       << endl;
// cin.good () prueft auf typgerechte Eingabe
  while (cin >> datum && cin.good ()) {
    el = new char[strlen (datum) + 1];
    strcpy (el, datum);
    daten.insert (el);
  }
  daten.print ();
  cout << endl;
  return(0);
}
```

. . .Beispiel Binärer Suchbaum: Benutzung

Programmausgabe



Vererbung Teil 1

- Klassen können durch Vererbung/Ableitung erweitert oder modifiziert werden.
- Durch Vererbung kann ein Konzept und deren Implementierung wiederverwendet werden.
- Die Darstellung von Klassen und deren Beziehungen zueinander kann in UML-Klassendiagrammen übersichtlich dargestellt werden.
- Für spezielle Suchoperationen kann die Klasse des binären Suchbaums eingesetzt werden.