5 Ergänzung zum Klassenbegriff, insbesondere Vererbung

5.1 Einführung und Programmorganistion

5.2 Klassenelemente als Referenzen

5.3 Union

5.4 Verschiebesemantik

5.5 Namensauflösung

5.6 Zugriffsrechte

5.7 Virtuelle Destruktoren

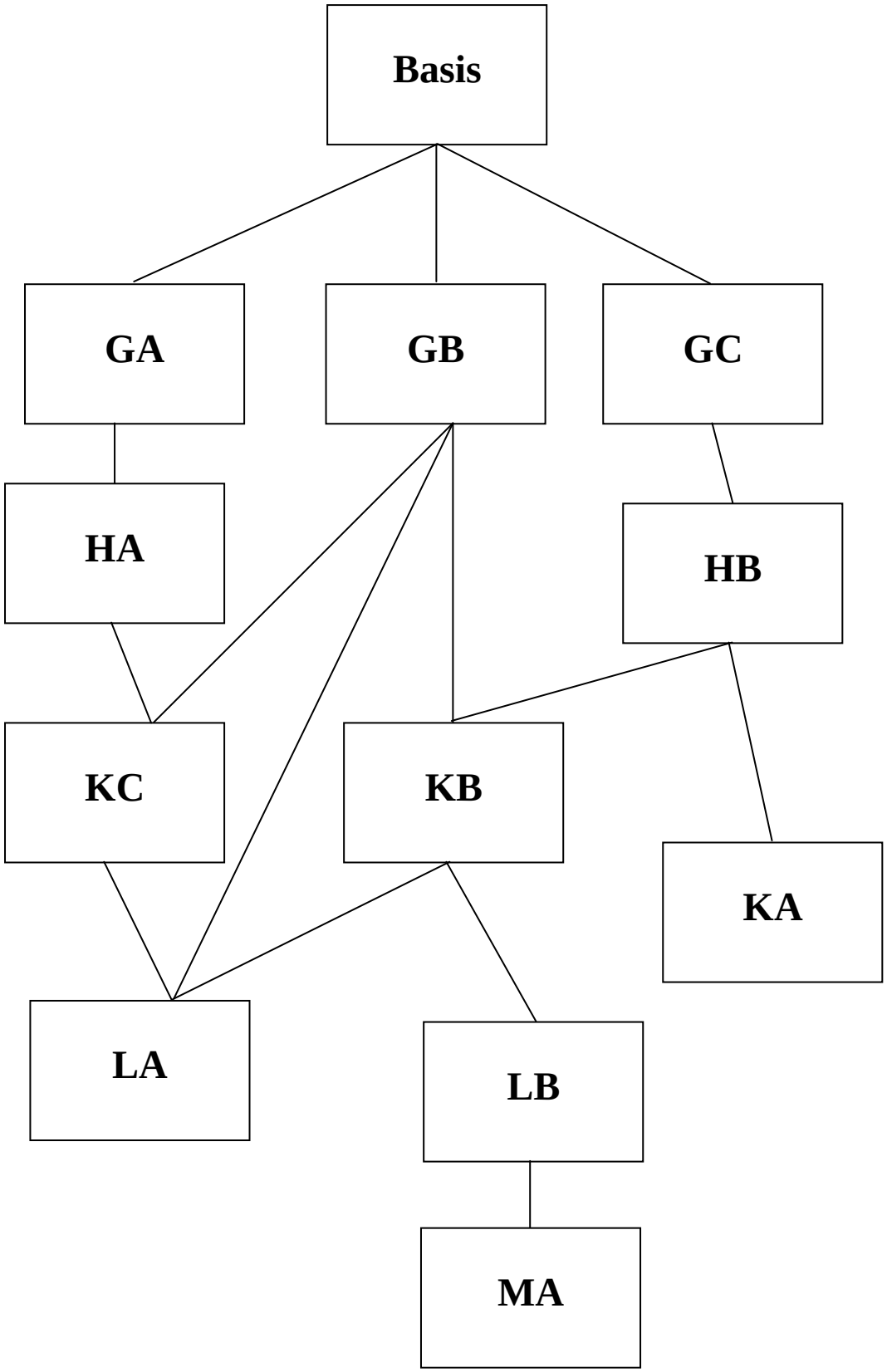
5.8 Virtuelle Funktionen

5.9 Virtuelle Vererbung

5.10 Abstrakte Klassen

5.11 Initialisierung

Ein Konzeptgraph



```
//  Datei  c_time.h

//  Schutzzaun
#ifndef C_TIME_H
#define C_TIME_H

/*
   Eine Klasse zur Beschreibung der Uhrzeit
   (zwischen 00:00:00 und 23:59:60)
*/

class Time {
public:
   Time (int hour,  int min,  int sec);

   Time ();

   int  get_hours () const;

   int  get_minutes () const;

   int  get_seconds () const;

   int  seconds_from (Time t) const;

   void  add_seconds (int s);

protected:
   int  time_in_secs;

};//Time
#endif
```

```
//  c_time.cpp
#include <ctime>
#include <cassert>
#include "c_time.h"
using namespace std;

int  remainder (int a, int b) {
   //  b > 0;
   int r = a % b;
   if (r < 0)  r += b;
   return r;
}
/*  Zeitstruktur aus ctime:
struct tm {
    int tm_sec;       /* Sekunden */
    int tm_min;       /* Minuten */
    int tm_hour;      /* Stunde (0 bis 23) */
    int tm_mday;      /* Tag im Monat (1 bis 31) */
    int tm_mon;       /* Monat (0 bis 11) */
    int tm_year;      /* Jahr (Kalenderjahr minus 1900) */
    int tm_wday;      /* Wochentag (0 bis 6, Sonntag = 0) */
    int tm_yday;      /* Tag im Jahr (0 bis 365) */
    int tm_isdst;     /* Sommerzeit beruecksichtigt */
};

Time::Time () {
   time_t   now     =  time (nullptr);
   //  Microsoft betrachtet localtime als “deprecated”,
   //  eventuell wegen des 2038-Problems.
   struct tm&  t  =  *localtime (&now);
   time_in_secs  =
      60*60*t.tm_hour+60*t.tm_min+t.tm_sec;
}
```

```
Time::Time (int hour,  int min,  int sec) {
   assert (0 <= hour);
   assert (hour < 24);
   assert (0 <= min);
   assert (min < 60);
   assert (0 <= sec);
   assert (sec < 61);
   // ohne Sekundenkorrektur
   time_in_secs =
            60 * 60 * hour + 60 * min + sec;
}


int Time::get_hours () const {
   return  time_in_secs / (60 * 60);
}
int Time::get_minutes () const {
   return  (time_in_secs / 60) % 60;
}
int Time::get_seconds () const {
   return  time_in_secs % 60;
}
int Time::seconds_from (Time t) const {
   return  time_in_secs - t.time_in_secs;
}
void Time::add_seconds (int s){
   int const  SECONDS_PER_DAY
                        = 60 * 60 * 24;
  time_in_secs = remainder
      (time_in_secs + s,  SECONDS_PER_DAY);
}
```

```
//  Nutzer
#include <iostream>
#include <iomanip>
#include <string>
#include "c_time.h"

using namespace std;

class Clock {
public:

   Clock () {}

   string get_location () const;

   int get_hours () const;

   int get_minutes () const;

   int get_seconds () const;

};//Clock


string Clock::get_location () const {
   return "Local";
}

int Clock::get_hours () const {
   Time  now;
    int hours  =  now.get_hours ();
return  hours;
}
```

```
int Clock::get_minutes () const {
   Time  now;
return  now.get_minutes();
}

int Clock::get_seconds() const {
   Time  now;
return  now.get_seconds ();
}

int main() {
   Clock clock1;

   cout << "Time is "
        << clock1.get_hours () << ":"
        << setw (2) << setfill ('0')
        << clock1.get_minutes() << ":"
        << setw (2) << setfill ('0')
        << clock1.get_seconds () << "\n";

}//main


/*  Ausgabe:

Time is 20:44:09

*/
```

```
//  Vererbung
#include <iostream>
#include <iomanip>
#include <string>
#include "c_time.h"

using namespace std;

class Clock {
public:

   Clock () {};

   string get_location() const;

   int get_hours() const;

   int get_minutes () const;

   int get_seconds () const;

};//Clock


string Clock::get_location () const {
   return "Local";
}

int Clock::get_hours() const {
   Time now;
   int hours = now.get_hours ();
return hours;
}
```

```
int Clock::get_minutes() const {
   Time now;
return now.get_minutes ();
}

int Clock::get_seconds() const {
   Time now;
return now.get_seconds ();
}


class TravelClock : public Clock {
public:
    TravelClock (string loc, int diffh, int diffm = 0);
    string get_location () const;
    int get_hours () const;
    int get_minutes () const;
private:
    string location;
    int  time_difference;  // minutes
};

TravelClock::TravelClock (string loc, int diffh, int diffm) :
Clock () {
    location = loc;
    time_difference = diffh * 60 + diffm;
    while (time_difference < 0)
        time_difference = time_difference + 24*60;
}

string TravelClock::get_location () const {
    return location;
}
```

```
int TravelClock::get_hours () const {
   int h = Clock::get_hours ();
   int m = Clock::get_minutes ();
return  ((h*60 + m + time_difference) / 60) % 24;
}

int TravelClock::get_minutes () const {
   int m = Clock::get_minutes ();
return (m + time_difference) % 60;
}


int main() {
   Clock clock1;
   TravelClock clock2 ("Lissabon", -1);
   TravelClock clock3 ("Peking", 6);
   TravelClock clock4 ("Caracas", -6, -30);
   TravelClock clock5 ("Kathmandu", 3, 45);

   cout << clock1.get_location () << " time is "
        << clock1.get_hours () << ":"
        << setw (2) << setfill ('0')
        << clock1.get_minutes() << ":"
        << setw (2) << setfill ('0')
        << clock1.get_seconds () << "\n";
   cout << clock2.get_location() << " time is "
        << clock2.get_hours () << ":"
        << setw (2) << setfill ('0')
        << clock2.get_minutes() << ":"
        << setw (2) << setfill ('0')
        << clock2.get_seconds () << "\n";
```

```
    cout << clock3.get_location() << " time is "
         << clock3.get_hours () << ":"
         << setw (2) << setfill ('0')
         << clock3.get_minutes() << ":"
         << setw (2) << setfill ('0')
         << clock3.get_seconds () << "\n";
    cout << clock4.get_location() << " time is "
         << clock4.get_hours () << ":"
         << setw (2) << setfill ('0')
         << clock4.get_minutes() << ":"
         << setw (2) << setfill ('0')
         << clock4.get_seconds () << "\n";
    cout << clock5.get_location() << " time is "
         << clock5.get_hours () << ":"
         << setw (2) << setfill ('0')
         << clock5.get_minutes() << ":"
         << setw (2) << setfill ('0')
        << clock5.get_seconds () << "\n";
}//main


/*  Ausgabe:

      Local time is          10:40:01
      Lissabon time is        9:40:01
      Peking time is         16:40:01
      Caracas time is         4:10:01
      Kathmandu time is      14:25:01

*/
```

```
//  Initialisierung von Referenzen für Klassenelemente
//  eventueller Schreibschutz
#include<iostream>
using namespace std;

class Zahl {
public:
      Zahl ()
      : nurlesbar (pZahl),     // Initialisierung der Referenz
        pZahl (0) {
      }

      void  aendern (int wert) {
         pZahl = wert;
      }
      int const&  nurlesbar;
private:
      int pZahl;
};//Zahl

int main () {
    Zahl  X;
    //   X.pZahl  =  77;         Fehler! Zugriff nicht möglich!
    //   X.nurlesbar  =  88;    Änderung nicht möglich!
    X.aendern (99);
    cout << "X.nurlesbar  =  "
          << X.nurlesbar << endl;
}//main

/*  Ausgabe

X.nurlesbar  =  99
*/
```

```
//  Ein zweites Beispiel zu Refenzen in Instanzen
#include <iostream>

class C {
//  Die Variable ref erhaelt Wert im Konstruktor.
public:
   C (int& ref) : ref {ref} {}
   int getref () {return ref;}
protected:
   int& ref;
};


int main () {
    int i = 123;
    C cc {i};
    std::cout << "Verweis aus cc = " << cc.getref()
              << std::endl;
    int* pi = new (int);
    *pi = 4567;
    C cp (*pi);
    std::cout << "Inhalt von *pi = "<< cp.getref()
              << std::endl;
   //  Welche Folgen hat
   //  delete pi;
   //   fuer die Instanz cp ?
}


/*
Verweis aus cc = 123
Inhalt von *pi = 4567
*/
```

```
//  Differenz zwischen Links- und Rechtsreferenzen

#include <iostream>
using namespace std;

void incr (int& value) {
   cout << "increment with lvalue reference" << endl;
   ++value;
}

void incr (int&& value) {
   cout << "increment with rvalue reference" << endl;
   ++value;
   cout << "  new value = " << value << endl;
}

int main() {
   int a = 10;
   int b = 20;

   incr (a);      // Will call incr (int& value)
   cout << "  a = " << a << ", b = " << b << endl;

   incr (a + b);  // Increment an expression
   cout << "  a = " << a << ", b = " << b << endl;

   incr (3);      // Increment a literal
   cout << "  a = " << a << ", b = " << b << endl;

   incr (std::move(b));  // Use rvalue reference
   cout << "  a = " << a << ", b = " << b << endl;

}//main
```

```
/*
Ergebnis:

increment with lvalue reference
  a = 11,  b = 20
increment with rvalue reference
  new value = 32
  a = 11,  b = 20
increment with rvalue reference
  new value = 4
  a = 11,  b = 20
increment with rvalue reference
  new value = 21
  a = 11,  b = 21
*/
```

Union

Syntax: union u-bezeichner {u-elemente} var-bezeichner;

Bemerkungen:

(i) **Der u-bezeichner kann leer sein.**

(ii) **Die Elemente einer union belegen den gleichen Speicherplatz, daher kann zu einer Zeit nur ein Element aktiv sein.**

(iii) **Seit C++11 besitzen unions alle sechs speziellen Funktionen: Konstruktoren, Destruktoren, Kopier- und Verschiebekonstruktoren, Kopier- und Verschiebezuweisungen, nur unter sehr eingeschränkten Bedingungen werden diese vom Compiler bereitgestellt.**

Ein Beispiel:

```
#include <iostream>
union U  {
   int i;
   float f;
};
int main () {
   U u;
   u.i = 42;                  // ok,  i ist aktiv
   std::cout << "u.i = " << u.i << std::endl;
   u.f = 3.14f;               // ok,  f ist aktiv
   std::cout << "u.f = " << u.f << std::endl;
}
/*    u.i = 42
      u.f = 3.14   */
```

Ein zweites Beispiel:

```
#include <iostream>
#include <string>
#include <vector>

union U {
    std::string str;
    std::vector<int> vec;
    // Nutzer muss wissen, welches Element aktiv ist.
    ~U() {}
}; // Groesse = max (sizeof(string), sizeof(vector<int>))

int main() {
    U u = {"Hello, world"};
    // aktiv ist str
    std::cout << "u.str = " << u.str << '\n';
    // Speicher freigeben
    u.str.~basic_string<char>();
    // Ueber placement new Speicher zuweisen
    new (&u.vec) std::vector<int>;
    u.vec.push_back(10);
    u.vec.push_back(101);
    u.vec.push_back(201);
    u.vec.push_back(30);
    for (auto i : u.vec)
        std::cout << i << "   ";
    std::cout << '\n';
    u.vec.~vector<int>();
}
/*
u.str = Hello, world
10  101  201  30
*/
```

```
//  Nutzung von Union zur Neuinterpretation von
//  Speicherinhalten
#include <iostream>
#include <iomanip>

void print_d (double d) {
   /*  Beschreibung einer "double"-Zahl auf intel x86.  */
   struct bd {
      unsigned f;
      unsigned c:20;
      unsigned b:11;
      unsigned a:1;
   };

   /*  Ueberlagerung mit Maske bd.  */
   union  {
      double d;
      struct bd x;
   } dba;

   dba.d = d;
   std::cout << "Ausgabe des Parameters:  " << d << '\n';
   std::cout << "Zerlegung der Gleitpunktzahl: \n";
   std::cout << "    Vorzeichen:\t\t" << (int) dba.x.a << '\n';
   std::cout << "    Charakteristik:\t" << std::dec
             << dba.x.b << '\n';
   std::cout << "    Dualbruch:\t\t0x" << std::hex
          << std::setw (5) << std::setfill ('0') << dba.x.c
          << std::setw (8) << std::setfill ('0') << dba.x.f
         << "\n\n";
}
```

```
int main () {
   print_d  (1.75);
   print_d  (3.625);
   print_d  (-5.5625);
}//main


/*
Ausgabe des Parameters:  1.75
Zerlegung der Gleitpunktzahl:
   Vorzeichen:          0
   Charakteristik:      1023
   Dualbruch:           0xc000000000000


Ausgabe des Parameters:  3.625
Zerlegung der Gleitpunktzahl:
   Vorzeichen:          0
   Charakteristik:      1024
   Dualbruch:           0xd000000000000


Ausgabe des Parameters:  -5.5625
Zerlegung der Gleitpunktzahl:
   Vorzeichen:          1
   Charakteristik:      1025
   Dualbruch:           0x6400000000000

*/
```

```
// spt.h
#pragma once
#include "sptcell.h"

class Spt {
public:
   Spt (int breite,  int hoehe);      // Grundkonstruktor
   Spt (const Spt& src);              // Kopierkonstruktor
   Spt (Spt&& src);                   // Verschiebekonstruktor
   ~Spt ();                           // Destruktor

  Spt& operator= (const Spt& rhs);  // Zuweisung
  Spt& operator= (Spt&& rhs);       // Verschiebezuweisung

  void setcellat (int x, int y, const Sptcell& cell);
  Sptcell getcellat (int x, int y) const;

protected:
   bool inrange (int val, int upper) const;
   void copyfrom (const Spt& src);

   int breite;
   int hoehe;
   Sptcell**   mcells;
};
```

```
//  spt.cpp
#include <iostream>
#include <stdexcept>
#include "spt.h"

using namespace std;

bool Spt::inrange (int val, int upper) const {
   return (val >= 0 && val < upper);
}

Spt::Spt (int breite,  int hoehe) :
             breite {breite}, hoehe {hoehe} {
   cout << "Grundkonstruktor" << endl;

    mcells = new Sptcell* [breite];
    for (int i = 0; i < breite; i++) {
       mcells[i] = new Sptcell[hoehe];
   }
}

Spt::Spt (const Spt& src) {
   cout << "Kopierkonstruktor" << endl;
   copyfrom (src);
}
```

```
Spt::Spt(Spt&& src) {
    cout << "Verschiebekonstruktor" << endl;
    // Kopiere src-Daten
    breite = src.breite;
    hoehe = src.hoehe;
    mcells = src.mcells;
    // Ruecksetzen Quelle src
    src.breite = 0;
    src.hoehe = 0;
    src.mcells = nullptr;
}

void Spt::setcellat (int x, int y, const Sptcell& cell) {
    if (!inrange (x, breite) || !inrange(y, hoehe)) {
        throw std::out_of_range ("in setcellat");
    }
    mcells [x] [y]  =  cell;
}

Sptcell Spt::getcellat (int x, int y) const {
    if (!inrange(x, breite) || !inrange(y, hoehe)) {
        throw std::out_of_range ("in getcellat");
    }
return mcells[x][y];
}

Spt::~Spt() {
    for (int i = 0; i < breite; i++) {
        delete [] mcells[i];
    }
    delete [] mcells;
    mcells = nullptr;
}
```

```
void Spt::copyfrom (const Spt& src) {
   breite = src.breite;
   hoehe = src.hoehe;
   mcells = new Sptcell* [breite];
   for (int i = 0; i < breite; i++) {
      mcells[i] = new Sptcell [hoehe];
   }
   for (int i = 0; i < breite; i++) {
      for (int j = 0; j < hoehe; j++) {
         mcells[i][j] = src.mcells[i][j];
      }
   }
}

Spt& Spt::operator= (const Spt& rhs) {
   cout << "Zuweisung" << endl;
   // Pruefung auf Selbstzuweisung
   if  (this == &rhs) {
      return *this;
   }
   // Bereinigung der Daten
   for (int i = 0; i < breite; i++) {
      delete [] mcells [i];
   }
   delete [] mcells;
   mcells = nullptr;

   // Kopiere rhs-Daten
   copyfrom (rhs);
return *this;
}
```

```
Spt& Spt::operator= (Spt&& rhs) {
   cout << "Verschiebung" << endl;

   // Pruefung auf Selbstzuweisung
   if (this == &rhs) {
      return *this;
   }

   // Rueckgabe des eigenen Speichers
   for (int i = 0; i < breite; i++) {
      delete [] mcells[i];
   }
   delete [] mcells;
   mcells = nullptr;   // unnoetig

   // Kopiere rhs-Daten
   breite = rhs.breite;
   hoehe = rhs.hoehe;
   mcells = rhs.mcells;
   // Ruecksetzen Quelle rhs
   rhs.breite = 0;
   rhs.hoehe = 0;
   rhs.mcells = nullptr;

return *this;
}
```

```
//  sptcell.h

#pragma once
#include <string>
using std::string;


class Sptcell {
public:
    Sptcell ();
    Sptcell (double);
    Sptcell (const string&);
    Sptcell (const Sptcell& src);
    Sptcell& operator= (const Sptcell&);
    void setv (double);
    double getv() const;
    void setstring (const string&);
    string getstring() const;

protected:
    string doubletostring (double) const;
    double stringtodouble(const string&) const;

    double value;
    string mstring;
};
```

```
//  sptcell.cpp
#include "sptcell.h"
#include <iostream>
#include <sstream>
using namespace std;

Sptcell::Sptcell() : value{0}, mstring{""} { }

Sptcell::Sptcell (double in) {
   setv (in);
}

Sptcell::Sptcell (const string& in) :
   value {stringtodouble (in)}, mstring {in}{
}

Sptcell::Sptcell (const Sptcell& src) {
   value = src.value;
   mstring  =  src.mstring;
}

Sptcell& Sptcell::operator= (const Sptcell& rhs) {
   if (this == &rhs) {
      return *this;
   }
   value = rhs.value;
   mstring = rhs.mstring;
return *this;
}

void Sptcell::setv (double in) {
  value = in;
   mstring = doubletostring (value);
}
```

```
double Sptcell::getv() const{
   return value;
}

void Sptcell::setstring (const string& instring) {
   mstring = instring;
   value = stringtodouble (mstring);
}

string Sptcell::getstring() const {
   return mstring;
}

string Sptcell::doubletostring (double in) const {
   ostringstream ostr;
   ostr << in;
return ostr.str();
}

double Sptcell::stringtodouble (const string& in) const {
   double temp;

   istringstream istr (in);

   istr >> temp;
   if (istr.fail() || !istr.eof()) {
      return 0;
   }
return temp;
}
```

```
// Testprogramm

#include <iostream>
#include <vector>
#include "spt.h"

using namespace std;

Spt Createspt () {
   return Spt (3, 2);
}

int main() {
   vector<Spt> vec;
   for (int i = 0; i < 2; ++i) {
      cout << "Iteration " << i << endl;
      vec.push_back (Spt (100, 50));
      cout << endl;
   }

   Spt s (2, 3);
   s = Createspt ();

   Spt s2 (5, 6);
   s2 = s;
}//main
```

/*
Bei g++ 4.8.2
Iteration 0
Grundkonstruktor
Verschiebekonstruktor

Iteration 1
Grundkonstruktor
Verschiebekonstruktor
Kopierkonstruktor

Grundkonstruktor
Grundkonstruktor
Verschiebung
Grundkonstruktor
Zuweisung

Bei VS 2012
Iteration 0
Grundkonstruktor
Verschiebekonstruktor

Iteration 1
Grundkonstruktor
Verschiebekonstruktor
Verschiebekonstruktor

Grundkonstruktor
Grundkonstruktor
Verschiebung
Grundkonstruktor
Zuweisung
***/**

```
//  Zur Verschiebesemantik
#include <iostream>
#include <list>
#include <utility>
using namespace std;

class P {
public:
   void settl (const std::list <std::string>& toks) {
      cout << "Kopieren ausgefuert!" << endl;
      tl = toks;
   }
   void settl (std::list< std::string>&& toks) {
      cout << "Verschieben ausgefuehrt!" << endl;
      tl = std::move (toks);
   }
 protected:
   std::list <std::string> tl;
};


auto main () -> int {
   std::list< std::string > tliste {"eins", "zwei",
                                    "drei", "vier"};

   P p;
   p.settl (tliste);
   cout << "Nach Kopieren: \n";
   if (tliste.empty())
      cout << "tliste ist leer!" << endl;
   else
      cout << "tliste.size() = " << tliste.size() << endl;
```

```
 // Verschiebe Zeichen nach pa
    P pa;
    pa.settl (std::move(tliste));
    cout << "Nach Verschieben: \n";
    if (tliste.empty())
       cout << "tliste ist leer!" << endl;
    else
       cout << "tliste.size() = " << tliste.size() << endl;
}



/*

Kopieren ausgefuert!
Nach Kopieren:
tliste.size() = 4
Verschieben ausgefuehrt!
Nach Verschieben:
tliste ist leer!

*/
```

```
// Zur Verschiebesemantik
#include <iostream>
#include <utility>
using namespace std;

class M {
public:
   explicit M (int sz=512):
   size (sz),
   buf (new char [size]) {
      buf [0] = 'H'; buf [1] = 'i';
      buf [2] = 'l'; buf [3] = 'f';
      buf [4] = 'e'; buf [5] = '\0';
     cout << size << "  buf = " << buf << endl;
   }
   ~M () {delete[] buf;}
   M (const M&) = delete;
   M& operator= (const M&) = delete;
   M (M&&);
   M& operator= (M&&);
private:
   size_t size;
   char* buf;
};

M::M (M&& other): size(0), buf(nullptr) {
   size = other.size;    buf = other.buf;
   // reset other
   other.size = 0;    other.buf = nullptr;
   cout << "Im Verschiebekonstruktor\n" << size
        << "   " << buf << endl;
}
```

```
M& M::operator=(M&& other) {
   cout<< "Im operator= \n";
   if (this !=& other) {
      delete[] buf;
      size = other.size;
      buf = other.buf;
      // reset other
      other.size = 0;
      other.buf = nullptr;
   }
return *this;
}


auto main () -> int {
   M  m1 {12};
   M  m2 {34};
   M  m3 (std::move(m1));
}

/*

12  buf = Hilfe
34  buf = Hilfe
Im Verschiebekonstruktor
12  Hilfe

*/
```

```
// Ergaenzung zur Verschiebesemantik
#include <iostream>
#include <utility>
using namespace std;

class M1 {
public:
   explicit M1 (int sz=512):
   size (sz), buf (new char [size]) {
      buf [0] = 'H'; buf [1] = 'i';
      buf [2] = 'l'; buf [3] = 'f';
      buf [4] = 'e'; buf [5] = '\0';
   }
   ~M1 () {delete[] buf;}
   M1 (const M1&);
   M1& operator= (const M1&);
   void ausgeben () {
      cout << "size = " << size;
      if (buf != nullptr) cout << "   buf = " << buf << endl;
   }
   // M1 (M1&&) = delete;                    // wichtig Verbergen
   // M1& operator= (M1&&) = delete; // im Kommentar
private:
   size_t size;
   char* buf;
};

M1::M1 (const M1& other) : size(0), buf (nullptr) {
   size = other.size;
   buf = new char [size];
   if (buf == nullptr) {
      cout << " Abbruch bei Kopie\n";
      exit (99);
   }
```

```
    for (int i = 0;  i < size; ++i)
        buf [i] = other.buf [i];
    cout << "Im Kopierkonstruktor  \n";
}

M1& M1::operator= (const M1& other) {
    if (this != &other) {
        size = other.size;
        delete[] buf;
        buf = new char [size];
        if (buf == nullptr) {
            cout << "Abbruch im operator=\n";
            exit (98);
        }
        for (int i = 0;  i < size; ++i)
            buf [i] = other.buf [i];
        cout << "Im operator=  \n";
     }
return *this;
}

auto main () -> int {
    M1  m1 {12};
    cout << "m1:  "; m1.ausgeben ();
    M1  m2 {34};
    cout << "m2:  "; m2.ausgeben ();
    M1  m3 (std::move(m1)); // Fehler bei explizitem delete
    cout << "m3:  "; m3.ausgeben ();
    cout << "m1:  "; m1.ausgeben ();
    M1  m4 (m2);
    cout << "m4:  "; m4.ausgeben ();
    cout << "m2:  "; m2.ausgeben ();
    m1 = m2;
```

```
   cout << "m1:  "; m1.ausgeben ();
   cout << "m2:  "; m2.ausgeben ();
}

/*

m1:  size = 12   buf = Hilfe
m2:  size = 34   buf = Hilfe
Im Kopierkonstruktor
m3:  size = 12   buf = Hilfe
m1:  size = 12   buf = Hilfe
Im Kopierkonstruktor
m4:  size = 34   buf = Hilfe
m2:  size = 34   buf = Hilfe
Im operator=
m1:  size = 34   buf = Hilfe
m2:  size = 34   buf = Hilfe

*/
```

```
// Beispiel zur Namensauflösung
// Beispiel aus Standard [class.member.lookup]

#include <iostream>


struct A   {int x;};
struct B   {float x;};
struct C : public A, public B { };
struct D : public virtual C { };
struct E : public virtual C {char x;};
struct F : public D, public E { };



int main() {

   F f;
   f.x = 65;                        // OK,  lookup finds E::x
   std::cout << "x = "
             << f.x
             << std::endl;
}//main


/*  Ausgabe:

x = A

*/
```

```
//  Beispiel zur Namensauflösung
//  Beispiel aus Standard [class.member.lookup]

struct B1 {
   void f ();
   static void f (int);
   int i;
};

struct B2 {
   void f (double);
};

struct I1: B1 { };
struct I2: B1 { };

struct D: I1, I2, B2 {
   using B1::f;
   using B2::f;
   void g () {
      f ();              // Ambiguous conversion of this
      f (0);             // Unambiguous (static)
      f (0.0);           // Unambiguous (only one B2)
      int B1::* mpB1 = &D::i; // Unambiguous
            //  Fehlermeldung eines Compilers:
            //  ambiguous access of 'i'
            //  could be the 'i' in base 'B1'
            //  or could be the 'i' in base 'B1'
      int D::* mpD = &D::i;       // Ambiguous conversion
   }
};
```

```
//  Vererbung von Zugriffsrechten

#include <iostream>
using namespace std;


class ober {
public:
   ober () {
      priv = 123;
      prot = 23;
      pub = 3;
   }
private:
   int priv;
protected:
   int prot;
public:
   int pub;
};//ober


class u1 : public ober {
public:
   void fp () {
      // cout << "in u1:  priv = " << priv << endl;
      cout << "in u1:  prot = " << prot << endl;
      cout << "in u1:  pub  = " << pub << endl;
  }
};//u1
```

```
class u2 : protected ober {
//  prot und pub von ober nun protected.
public:
   void fp () {
      // cout << "in u2:  priv = " << priv << endl;
      cout << "in u2:  prot = " << prot << endl;
      cout << "in u2:  pub  = " << pub << endl;
  }
};//u2


class u3 : private ober {
//  prot und pub von ober nun private.
public:
   void fp () {
      // cout << "in u3:  priv = " << priv << endl;
      cout << "in u3:  prot = " << prot << endl;
      cout << "in u3:  pub  = " << pub << endl;
  }
};//u3


/*
class uu : public u3 {
// Kein Zugriff auf Elemente von ober
public:
   void fp () {
      // cout << "in uu:  priv = " << priv << endl;
      cout << "in uu:  prot = " << prot << endl;
      cout << "in uu:  pub  = " << pub << endl;
  }
};//uu
*/
```

```
int main () {
    u1  u;
    cout << "public:" << endl;
    u.fp ();
    cout << "direkt: pub  = " << u.pub << endl;
    u2  ub;
    cout << "protected:" << endl;
    ub.fp ();
    // cout << "direkt: pub  = " << ub.pub << endl;
    u3  uc;
    cout << "private:" << endl;
    uc.fp ();
    // cout << "direkt: pub  = " << ub.pub << endl;

}//main


/*  Ausgabe:

public:
in u1:  prot = 23
in u1:  pub  = 3
direkt: pub  = 3
protected:
in u2:  prot = 23
in u2:  pub  = 3
private:
in u3:  prot = 23
in u3:  pub  = 3
*/
```

```
//  Vererbung von Zugriffsrechten

#include <iostream>
using namespace std;


class ober {
public:
   ober () {
      pubc  =  1234;
      pubb  =  234;
      puba  =  34;
   }
protected:
   int prot;
public:
   int puba;
   int pubb;
   int pubc;
};//ober



class unter  :  private ober {
// Selektive Rechteänderung
public:
   using ober::puba;
   using ober::pubb;
   using ober::prot;
};//unter
```

```
int main () {
   unter  u;
   cout << "public:" << endl;
   cout << "puba  = " << u.puba << endl;
   cout << "pubb  = " << u.pubb << endl;
   u.prot = 1221;
   cout << "prot  = " << u.prot << endl;
}//main



/*  Ausgabe:


public:
puba  = 34
pubb  = 234
prot  = 1221


*/
```

```
//  Umgehung von Zugriffsrechten

#include <iostream>
using namespace std;



class ober {
public:
   ober () {
      pubc = 1357;
      pubb = 357;
      puba = 57;
      prot = 7;
   }
protected:
   int prot;
public:
   int puba;
   int pubb;
   int pubc;
};//ober



class unter : private ober {
private:
   int up;
};//unter
```

```
int main () {
   unter  u;
   unter* up = &u;

   //  Fehler bei ober* op = up;
   //  Fehler bei ober* op = dynamic_cast <ober*> (up);
   //  Fehler bei ober* op = static_cast <ober*> (up);
   ober* op = reinterpret_cast <ober*> (up);
   //  ober*  op = (ober*) (up);   o.k.

   cout << "public:" << endl;
   cout << "puba  = " << op->puba << endl;
   cout << "pubb  = " << op->pubb << endl;
   cout << "pubc  = " << op->pubc << endl;
   //cout << "prot  = " << op->prot << endl;    !Fehler
}//main


/*  Ausgabe:

public:
puba  = 57
pubb  = 357
pubc  = 1357


*/
```

```
//  Virtuelle Destruktoren
#include<iostream>
using namespace std;

#define PRINT(X) cout << (#X) << " = " << (X) << endl


class Basis {
public:

   Basis (int b = 0) : b (b) { }

   virtual ~Basis () {          // virtueller Destruktor!
      cout << "Objekt " << b
           << " Basis-Destruktor aufgerufen!" << endl;
   }
private:
   int  b;
};//Basis


class Derived : public Basis {
public:

   Derived (int b = 0, double a = 0.0)  :  Basis (b), a (a) { }

   ~Derived () {
       cout <<"Objekt " << a
                << " Derived-Destruktor aufgerufen!" << endl;
    }
private:
   double  a;
};//Derived
```

```
int main () {
   Basis* pb = new Basis (1);
   PRINT (sizeof (*pb));
   Derived* pa = new Derived (2,  2.2);
   PRINT (sizeof (*pa));
   Basis* pba = new Derived (3,  3.3);
   PRINT (sizeof (*pba));
   cout << "pb loeschen:" << endl;
   delete pb;
   cout << "pa loeschen:" << endl;
   delete pa;
   cout << "pba loeschen:" << endl;
   delete pba;
}//main



/*  Ausgabe:

sizeof (*pb) = 8
sizeof (*pa) = 16
sizeof (*pba) = 8
pb loeschen:
Objekt 1 Basis-Destruktor aufgerufen!
pa loeschen:
Objekt 2.2 Derived-Destruktor aufgerufen!
Objekt 2 Basis-Destruktor aufgerufen!
pba loeschen:
Objekt 3.3 Derived-Destruktor aufgerufen!
Objekt 3 Basis-Destruktor aufgerufen!

*/
```

```
#include<iostream>
using namespace std;

#define PRINT(X) cout << (#X) << " = " << (X) << endl


class Basis {
public:
   Basis (int b = 0) : b (b) { }

   ~Basis () {          // nicht virtueller Destruktor!
      cout << "Objekt " << b
           << " Basis-Destruktor aufgerufen!" << endl;
   }
private:
   int  b;
};//Basis

class Derived : public Basis {
public:
   Derived (int b = 0, double a = 0.0)
   : Basis (b), a (a) {}

   ~Derived () {
       cout <<"Objekt " << a
            << " Derived-Destruktor aufgerufen" << endl;
    }
private:
   double  a;
};//Derived
```

```
int main () {
   Basis* pb = new Basis (1);
   PRINT (sizeof (*pb));
   Derived* pa = new Derived (2, 2.2);
   PRINT (sizeof (*pa));
   Basis* pba = new Derived (3, 3.3);
   PRINT (sizeof (*pba));
   cout << "pb loeschen:" << endl;
   delete pb;
   cout << "pa loeschen:" << endl;
   delete pa;
   cout << "pba loeschen:" << endl;
   delete pba;
}// main



/*  Ausgabe:

sizeof (*pb) = 4
sizeof (*pa) = 16
sizeof (*pba) = 4
pb loeschen:
Objekt 1 Basis-Destruktor aufgerufen!
pa loeschen:
Objekt 2.2 Derived-Destruktor aufgerufen
Objekt 2 Basis-Destruktor aufgerufen!
pba loeschen:
Objekt 3 Basis-Destruktor aufgerufen!

!!  Es wurde nicht der Derived-Destruktor aufgerufen.  !!
*/
```

```
//  Vererbung:  virtuelle Funktionen
#include <iostream>
using namespace std;


class Ober {
public:
   Ober (int io) : io (io) { }

   virtual int get () const {return io;}

   virtual Ober&  assign (const Ober& rs) {
      std::cout << "Ober::this = " << (unsigned) this
                << "       &rs = " << (unsigned) (&rs)
                << std::endl;
      if (this != &rs) {
         lokal (rs);
      }
   return *this;
   }//assign

Ober& operator= (const Ober& rs) {return assign (rs);}

protected:
   void lokal  (const Ober& rs) {
     io = rs.io;
   }

private:
   int io;
};//Ober
```

```
class Unter : public Ober {
public:
   Unter (int io,  int iu) : Ober (io),  iu (iu)  { }

   virtual int get () const {
       return (1000*Ober::get () + iu);
   }

   virtual Unter& assign (const Ober& rs) {
       std::cout << "Unter::this = " << (unsigned) this
                 << "       &rs = " << (unsigned) (&rs)
                 << std:: endl;
     if (this != &rs) {
         Ober::assign (rs);          // Oberklassensubobjekt
       const Unter& U = dynamic_cast<const Unter&>(rs);
       lokal (U);                  // nur lokale Daten
     }
   return *this;
   }//assign

   Unter& operator=(const Unter& rs) {return assign (rs);}

protected:
   void lokal (const Unter& rs) {
       iu = rs.iu;
   }
private:
   int  iu;
};
```

```
int main() {
    Ober  o1 (11);
    Ober  o2 (22);

    cout << "o1.io = " << o1.get () << endl;
    cout << "o2.io = " << o2.get () << endl;

    Ober  o3  =  o1;
    cout << endl << "o3.Kopierkonstruktor (o1):\n";
    cout << "o3.io = " << o3.get () << endl << endl;
    cout << "o3.operator= (o2):\n";
    o3  =  o2;
    cout << "O3.io = " << o3.get () << endl << endl;
    cout << "Test auf Adressengleichheit"  << endl;
    o3  =  o3;

    cout << endl << endl;
    cout << "Unter erbt von Ober!" << endl << endl;
    Unter  u1 (111, 222);
    Unter  u2 (333, 444);

    cout << "u1.iu = " << u1.get () << endl;
    cout << "u2.iu = " << u2.get () << endl;

    Unter  u3  =  u1;
    cout << endl << "u3.Kopierkonstruktor (u1):" << endl;
    cout << "u3.iu = " << u3.get () << endl;
    cout << "u2.iu = " << u2.get () << endl;
    cout << endl << "u3.operator= (u2):" << endl;
    u3  =  u2;
    cout << "u3.iu = " << u3.get () << endl;
    cout << "u2.iu = " << u2.get () << endl;
    cout << endl << "Test auf Adressengleichheit"<< endl;
    u3 = u3;
```

```
    Ober& oref1 = u1;
    Ober& oref2 = u2;

    cout << "oref1.iu = " << oref1.get () << endl;
    cout << "oref2.iu = " << oref2.get () << endl;

    oref1 = oref2;
    cout << "oref1.iu = " << oref1.get () << endl;

}//main



/*  Ausgabe:

O1.io = 11
O2.io = 22

O3.Kopierkonstruktor (O1):
O3.io = 11

O3.operator= (O2):
Ober::this = 2293424          &rs = 2293432
O3.io = 22

Test auf Adressengleichheit
Ober::this = 2293424          &rs = 2293424


Unter erbt von Ober!

U1.iu = 111222
U2.iu = 333444
```

U3.Kopierkonstruktor (U1):
U3.iu = 111222
U2.iu = 333444

U3.operator= (U2):
Unter::this = 2293388 &rs = 2293400
Ober::this = 2293388 &rs = 2293400
U3.iu = 333444
U2.iu = 333444

Test auf Adressengleichheit
Unter::this = 2293388 &rs = 2293388
Oref1.iu = 111222
Oref2.iu = 333444
Unter::this = 2293412 &rs = 2293400
Ober::this = 2293412 &rs = 2293400
Oref1.iu = 333444

***/**

```
//  Virtuelle Mehrfachvererbung 1
#include <iostream>
#include <string>
using namespace std;

class Basis {
public:
   Basis () {cout << "Basis-Default-Konstruktor\n"; }
   Basis (string a)  {cout << a << endl;}
};//Basis

class Links : virtual public Basis {
public:
    Links (string a) : Basis (a) {}
};

class Rechts : virtual public Basis {
public:
   Rechts (string a) : Basis (a) {}
};

class Unten: public Links, public Rechts {
public:
    Unten (string a) : Links (a), Rechts (a)  {}
};

int main() {
    Links  li   ("Links");
    Rechts re ("Rechts");
    Unten  x   ("Unten");
}//main

/*  Ausgabe:        Links
                    Rechts
                    Basis-Default-Konstruktor
*/
```

```
//  Virtuelle Mehrfachvererbung 2
#include <iostream>
#include <string>
using namespace std;

class Basis {
public:
   Basis () {cout << "Basis-Default-Konstruktor\n";}
   Basis (string a)  {cout << a << endl; }
};//Basis

class Links : virtual public Basis {
public:
    Links (string a) : Basis (a) { }
};

class Rechts : virtual public Basis {
public:
   Rechts (string a) : Basis (a) {}
};

class Unten: public Links, public Rechts {
public:
    Unten (string a) :  Basis (a), Links (a), Rechts (a)  {}
};

int main() {
    Links li ("Links");
    Rechts re ("Rechts");
    Unten x  ("Unten");
}//main

/*  Ausgabe:              Links
                          Rechts
                          Unten*/
```

Abstrakte Klassen:

Bemerkungen:

(i) **Eine Klasse ist abstrakt, falls sie eine „pure virtual function“ enthält.**

(ii) **Man kann keine Objekte einer abstrakten Klasse erzeugen.**

(iii) **Eine abstrakte Klasse kann von einer nicht abstrakten Klasse abgeleitet werden.**

Ein Beispiel:

```
class point {  };
class shape {          // Abstrakte Klasse
private:
        point  center;
        //  . . .
public:
        point  where ()  {
                return center;
        }
        void  move (point p)  {
                center = p;
                draw ();
        }
        virtual void draw () = 0;                  // pure virtual
        virtual void rotate (double) = 0;          // pure virtual
        //  . . .
}
```

```
//  Beispiel zu Abstrakten Klassen
#include<string>
#include<cmath>
#include<iostream>


class Ort {
public:
    Ort (int x = 0, int y = 0)
    : xkoord (x),  ykoord (y)  { }

    int X () const   {return xkoord;}
    int Y () const   {return ykoord;}
    void aendern (int x,  int y) {
        xkoord  =  x;
        ykoord  =  y;
   }
private:
    int   xkoord;
    int   ykoord;
};//Ort


double Entfernung (const Ort& O1, const Ort& O2) {
   double dx = static_cast<double> (O1.X () - O2.X ());
   double dy = static_cast<double> (O1.Y () - O2.Y ());
return std::sqrt (dx*dx + dy*dy);
}//Entfernung


void anzeigen (const Ort& O) {
   std::cout << '(' << O.X () << ", " << O.Y () << ')';
}//anzeigen
```

```
bool operator==(const Ort& O1, const Ort& O2) {
   return  (O1.X () == O2.X ()
              && O1.Y () == O2.Y ());
}


std::ostream& operator<< (std::ostream& os,
                                    const Ort& O) {
     os << " (" << O.X () << ", " << O.Y () << ')';
return os;
}


class Graphobj {
public:
   Graphobj (const Ort& O)
   : Koordinaten (O) {}

   virtual ~Graphobj () {}

   const Ort& Bezugspunkt () const {
      return Koordinaten;
   }//Bezugspunkt

   Ort Bezugspunkt  (const Ort& nO) {
      Ort temp = Koordinaten;
      Koordinaten = nO;
      return temp;
   }

   int X () const  {return Koordinaten.X ();}
   int Y () const  {return Koordinaten.Y ();}
```

```
    virtual double Flaeche () const = 0;
    virtual void zeichnen () const   = 0;

private:
    Ort Koordinaten;
};//Graphobj


inline void Graphobj::zeichnen () const {
    std::cout << "Zeichnen:  ";
}


double Entfernung (const Graphobj& g1,
                           const Graphobj& g2) {
     return Entfernung (g1.Bezugspunkt (),
                              g2.Bezugspunkt ());
}


class Strecke : public Graphobj {
public:
     Strecke (const Ort& Ort1, const Ort& Ort2)
     : Graphobj (Ort1),  Endpunkt (Ort2)  { }

     double Laenge() const {
          return Entfernung (Bezugspunkt (),  Endpunkt);
     }
     virtual double Flaeche () const {
         return 0.0;
     }
```

```
    virtual void zeichnen () const {
        Graphobj::zeichnen ();
        std::cout << "Strecke von ";
        anzeigen (Bezugspunkt ());
        std::cout << " bis ";
        anzeigen (Endpunkt);
        std::cout << std::endl;
    }
private:
    Ort Endpunkt;
};//Strecke


class Rechteck : public Graphobj {
public:
    Rechteck(const Ort& p1, int h, int b)
    : Graphobj (p1),  hoehe (h),  breite (b)  {}

    virtual double Flaeche () const {
       return static_cast<double> (hoehe) *
                 static_cast<double> (breite);
    }
    virtual void zeichnen () const {
        Graphobj::zeichnen ();
        std::cout << "Rechteck "
                    << hoehe << " x " << breite
                     << "  bei  ";
        anzeigen (Bezugspunkt ());
        std::cout << std::endl;
    }
private:
    int hoehe,  breite;
};//Rechteck
```

```
int main() {
   Ort Nullpunkt;
   //Graphobj G0 (Nullpunkt);  //  Nicht erlaubt !
   Ort O1 (2, 20);
   Ort O2 (40, 70);
   Ort O3 (17, 34);

   Strecke  S1 (O1, O2);
   Strecke  S2 (Ort (0, 6), Ort (12, 18));
   S1.zeichnen ();

   Rechteck R1 (Nullpunkt, 30, 50);
   Rechteck R2 (O3, 20, 70);
   R1.zeichnen ();

   std::cout << "Nutzung der Polymorphie mittels Zeiger:"
             << std::endl;
   Graphobj* ga [] = {&S1, &S2, &R1, &R2, 0};
   for (int i =  0; ga [i] != 0; ++i)
      ga[i]->zeichnen ();

   std::cout << "Nutzung der Polymorphie "
                "mittels Referenzen:"
             << std::endl;
   Graphobj&  g1 = S2;
   Graphobj&  g2 = R2;
   std::cout << "Fläche = " << g1.Flaeche () << std::endl;
   std::cout << "Fläche = " << g2.Flaeche () << std::endl;

}//main
```

/* Ausgabe:

Zeichnen: Strecke von (2, 20) bis (40, 70)
Zeichnen: Rechteck 30 x 50 bei (0, 0)

Nutzung der Polymorphie mittels Zeiger:
Zeichnen: Strecke von (2, 20) bis (40, 70)
Zeichnen: Strecke von (0, 6) bis (12, 18)
Zeichnen: Rechteck 30 x 50 bei (0, 0)
Zeichnen: Rechteck 20 x 70 bei (17, 34)

Nutzung der Polymorphie mittels Referenzen:
Fläche = 0
Fläche = 1400

***/**

```
//  Initialisierung
//  Basis-Objekte werden zuerst initialisiert.
#include <iostream>

struct A {
   virtual void f () {};
   A () {std::cout << "Bildung von A\n";}
};

struct B1 : A {       //  nichtvirtuelle Ableitung
   void f () {};
   B1 () {std::cout << "Bildung von B1\n";}
};

struct B2 : A {
   void f () {};
   B2 () {std::cout << "Bildung von B2\n";}
};

struct D : B1, B2 { //     D besitzt zwei A Unterobjekte
   D () {std::cout << "Bildung von D\n";}
};

struct E : B2, B1 {
   E () {std::cout << "Bildung von E\n";}
};


int main () {
   D  d;
   std::cout << "\n\n";
   E  e;
}
```

/* Ausgabe:

Bildung von A
Bildung von B1
Bildung von A
Bildung von B2
Bildung von D

Bildung von A
Bildung von B2
Bildung von A
Bildung von B1
Bildung von E

***/**

```
//  Initialisierung:  Virtuelle und normale Ableitung
//                    können nebeneinander existieren.
#include <iostream>

int z = 0;

struct A {
   int  i;
   A () {
      z++;
      i = z;
      std::cout << "Bildung von A\t\tz = "
                << z << "\n";
   }
   ~A () {std::cout << "A::i = " << i << "\n";}
};

struct B :  virtual A {
   B () {
       z++;
       std::cout << "Bildung von B\t\tz = "
       << z << "\n";}
};

struct C :  B,  A {
   C () {
       z++;
       std::cout << "Bildung von C\t\tz = "
                 << z << "\n";}
       void f () {
            std::cout << "B::A::i  =  " << B::A::i << "\n"
                      << "A::i     =  " << A::i << "\n";
       }
};
```

```
int main () {
    C  c;
    C* cp = &c;
    B* bp = cp;
    std::cout << "B::i = " << bp->i << std::endl;
    c.f ();
}

/*
Warnung bei VS 2012

Fehler bei g++ 4.9.1
In member function 'void C::f()':
error: 'A' is an ambiguous base of 'C'
          std::cout << "B::A::i  =  " << B::A::i << "\n"
error: 'A' is an ambiguous base of 'C'
          << "A::i     =  " << A::i << "\n";
*/

/* Ausgabe:
Bildung von A          z = 1
Bildung von B          z = 2
Bildung von A          z = 3
Bildung von C          z = 4
B::i = 1
B::A::i  =  1
A::i      =  3
A::i = 3
A::i = 1
*/
```

```
//  Initialisierung:  Virtuelle und normale Ableitung
//                    können nebeneinander existieren.
#include <iostream>


int z = 0;

struct A {
   int  i;
   A () {
      z++;
      i = z;
      std::cout << "Bildung von A\t\tz = "
                << z << "\n";
   }
   ~A () {std::cout << "A::i = " << i << "\n";}
};

struct B : virtual A {
   B () {
       z++;
       std::cout << "Bildung von B\t\tz = "
       << z << "\n";}
};

struct C :  A,  B {  // Reihenfolge geändert
   C () {
       z++;
       std::cout << "Bildung von C\t\tz = "
                 << z << "\n";}
};
```

```
int main () {
   C  c;
   C* cp = &c;
   B* bp = cp;
   std::cout << "B::i = " << bp->i << std::endl;
}


/*
Warnung bei g++  4.9.1
warning: direct base 'A' inaccessible in 'C' due to
            ambiguity  struct C :  A,  B {

Fehler bei VS 2012
error C2584: 'C': Kein Zugriff auf direkte Basisklasse 'A',
                     da diese bereits Basisklasse von 'B' ist

*/


/* Ausgabe:
Bildung von A         z = 1
Bildung von A         z = 2
Bildung von B         z = 3
Bildung von C         z = 4
B::i = 1
A::i = 2
A::i = 1
*/
```

```
//  Initialisierung
//  Virtuelle Ableitung wird zuerst berücksichtigt.
#include <iostream>
using namespace std;


class V {
public:
   V () {cout << "Bildung von V ()\n";}
   V (int) {cout << "Bildung von V (int)\n";}
// ...
};

class A : public virtual V {
public:
   A () {cout << "Bildung von A ()\n";}
   A (int);
// ...
};

class B : public virtual V {
public:
   B () {cout << "Bildung von B ()\n";}
   B (int);
// ...
};

class C : public A, public B, private virtual V {
public:
   C () {cout << "Bildung von C ()\n";}
   C (int);
// ...
};
```

```
A::A (int i) : V (i) {cout << "Bildung von A (int)\n";}
B::B (int i) {cout << "Bildung von B (int)\n";}
C::C (int i) {cout << "Bildung von C (int)\n";}

V   v (1);        // use V(int)
A   a (2);        // use V(int)
B   b (3);        // use V()
C   c (4);        // use V()

int main () {}



/*  Ausgabe:

Bildung von V (int)

Bildung von V (int)
Bildung von A (int)

Bildung von V ()
Bildung von B (int)

Bildung von V ()
Bildung von A ()
Bildung von B ()
Bildung von C (int)

*/
```