Clicker

Bitte den Link

https://vc2.sonia.de/b/har-2cy-qhv-bj0

für WS 2025/26 nutzen.

Clicker für HHe

Einloggen vom Handy auf https://vc2.sonia.de/b/har-2cy-qhv-bj0
für WS 2025/26.
Konferenz dort ohne Audio starten.
Auch von Laptop mit anderen Namen dort einloggen und testen, ob man antworten kann
Nun Umfrage erstellen, indem auf Plus geklickt wird

Termine des Semesters

Termin		Vorlesung	Übungen und Feedback
Woche		Freitag ; Block 1+2	
8	14. Nov	Übung Tiefe/Flache Kopie Operatoren;	14.11; Teambildung abgeschlossen
9	21. Nov	Templates, Polymorphie	
10	28. Nov	STL, Iteratoren; ; lineare und assoziative Container; STL Algorithmen	Bis 27.11 Teamaufgabe; erste Demo; Demo per BBB vorher
11	05. Dez	Algorithmus oder Methode Klasse unique_ptr,, shared_ptr, Lambda-Ausdrücke	
12	12. Dez	Verschiebeoperatoren ,	
13	19. Dez	Rest; Vorbereitung Klausur	Finale Abgabe Teamaufgabe Mi. 17.12; Demo per BBB vorher

Nächste Woche Stand zeigen oder per BBB?
In der Vorlesung bevorzugt

Klausur: Mi, 14.1.26; 11:00 – 12:30 Uhr; R. 252
Klausureinsicht: Jan 2026 xxx Uhr

Rückblick

Umfangreiche Wiederholung mit Studenten-Interaktion
zur Objekterzeugung und –zerstörung mit Klasse Hund
Minimale Standard-Schnittstelle am Beispiel der Klasse Auto

Polymorphie

Feedback zur Übungsaufgabe 03
Beginn Templates am Beispiel der Klasse Vektor als Dozenten-Demo und dann selber die Klassen LevenshteinDistance als Template Klasse ausgebaut.

Besuch im DLR

Reihenfolge der Operator-Auswertung

```
int main() {
   fstream datei("datei.txt", ios::out);
   X x(11);   Y y(222);
   datei << x << y << endl;
}
```

Wird erst operator<< von X oder erst von Y aufgerufen?
Wie kann man die Hypothese prüfen?

Reihenfolge der Operator-Auswertung

```
class  X {
public:
   X(int a) : si(a) {}
private:
    int si;
    friend ostream& operator<<
       (ostream&, const X&);
};
```

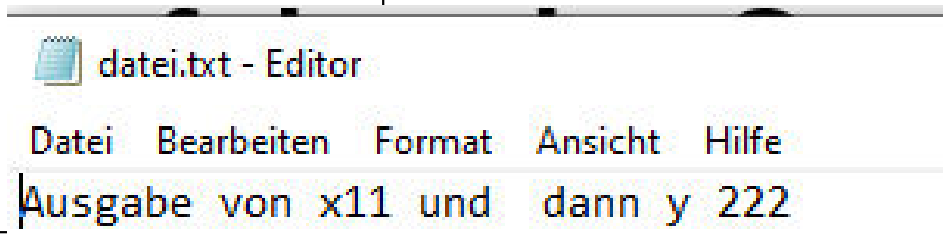
```
class  Y {
public:
   Y(int a) : si(a) {}
private:
   int si;
   friend ostream& operator<<
        (ostream&, const Y&);
};
```

```
ostream& operator<<(ostream&
       ostr, const X& x) {
   cout << " Aufruf von X " <<
       x.si << "; ";
   ostr << x.si;
   return ostr;
}
```

```
ostream& operator<<(ostream&
       ostr, const Y& y) {
   cout << " y wird aufgerufen
       " << y.si << "; ";
   ostr << y.si;
   return ostr;
}
```

Reihenfolge der Operator-Auswertung (2)

```
int main() {
   fstream datei("datei.txt", ios::out);
   X x(11);   Y y(222);
   datei << "Ausgabe von x" << x << " und "
      " dann y " << y << endl;
   cout << endl;
}
```




```
Aufruf von X 11;  y wird aufgerufen 222;
```

Und wie sieht die Aufruf-Reihenfolge nun aus?

```
int main() {
    funkArg(f(3), g(2));
}
void funkArg(int fv, int gv) {
    cout << "\nfunkArg ";
    cout << " fv + gv "
     << fv + gv << endl;
}
```

```
int f(int a) {
   cout << " f(" << a << ") ";
   return a * a;
}
int g(int a) {
    cout << " g(" << a << ") ";
    return a * a * a;
}
```

Funktionsargumente werden von rechts nach links ausgewertet

```
int main() {
    funkArg(f(3), g(2));
}
void funkArg(int fv, int gv) {
    cout << "\nfunkArg ";
    cout << " fv + gv "
     << fv + gv << endl;
}
```

```
int f(int a) {
   cout << " f(" << a << ") ";
   return a * a;
}
int g(int a) {
    cout << " g(" << a << ") ";
    return a * a * a;
}
```

```
 g(2)  f(3)
funkArg  fv + gv 17
```

Clicker: Wie ist die Ausgabe der 2 Print-Aufrufe?

```
class Mitarbeiter {
   public:
   virtual float Gehalt() { return 25; }
   virtual int    Unterg() { return 0; }
};

class Chef: public Mitarbeiter {
   public:
  virtual float Gehalt() { return 400; }
  virtual int  Unterg() { return 20; }
};
```

```
void Print (Mitarbeiter& pers) {
   cout << pers.Gehalt()<<
         "  " << pers.Unterg();
}
...
Mitarbeiter m;
Chef ch;
Print(m);
Print(ch);
```

1. 25 0 25 0
2. 25 0 400 0
3. 25 0 400 20
4. 400 0 400 0

Mit _-_ im Folgenden
immer die Lösung gekennzeichnet

Ergebnis:
__ 1 _ 2 _-_ 3 __4

Clicker: Wie ist die Ausgabe der 2 Print-Aufrufe?

```
class Mitarbeiter {
   public:
   virtual float Gehalt() { return 25; }
   virtual int    Unterg() { return 0; }
};

class Chef: public Mitarbeiter {
   public:
  virtual float Gehalt() { return 400; }
  virtual int  Unterg() { return 20; }
};
```

```
void Print (Mitarbeiter pers) {
   cout << pers.Gehalt()<<
         "  " << pers.Unterg();
}
...
Mitarbeiter m;
Chef ch;
Print(m);
Print(ch);
```

1. 25 0 25 0
2. 25 0 400 0
3. 25 0 400 20
4. 400 0 400 0

Ergebnis:
_- _ 1 _ 2 __ 3 __4

Clicker: Wie ist die Ausgabe der 2 Print-Aufrufe?

```
class Mitarbeiter {
   public:
   virtual float Gehalt() { return 25; }
   int    Unterg() { return 0; }
};

class Chef: public Mitarbeiter {
   public:
  virtual float Gehalt() { return 400; }
  int  Unterg() { return 20; }
};
```

```
void Print (Mitarbeiter& pers) {
   cout << pers.Gehalt()<<
         "  " << pers.Unterg();
}
...
Mitarbeiter m;
Chef ch;
Print(m);
Print(ch);
```

1. 25 0 25 0
2. 25 0 400 20
3. 25 0 400 0
4. 400 0 400 0

Ergebnis:
__ 1 _ 2 _-_ 3 __4

Clicker: Wie ist die Bildschirm-Ausgabe?

```
class Mitarbeiter  {
public:
   Mitarbeiter()     {cout << "+M ";}
   ~Mitarbeiter()   {cout <<"-M ";}
};
class Chef : public Mitarbeiter  {
public:
    Chef()  {cout << "+C ";}
   ~Chef()  {cout << "-C ";}
};
```

```
void Poly() {
        Mitarbeiter m1
        Chef* ph1;  // Zeiger
}
```

1. +M -M
2. +M +M -C
3. +M +M +C
4. +M +M +C –C –M -M

Ergebnis:
_-_ 1 _ 2 __ 3 __4

Clicker: Wie ist die Bildschirm-Ausgabe?

```
class Mitarbeiter  {
public:
   Mitarbeiter()     {cout << "+M ";}
   ~Mitarbeiter()   {cout <<"-M ";}
};
class Chef : public Mitarbeiter  {
public:
    Chef()  {cout << "+C ";}
   ~Chef()  {cout << "-C ";}
};
```

```
void Poly() {
   Mitarbeiter* ph1 = new Chef();
   delete ph1;
}
```

1. +M +C –C -M
2. +M +C -M
3. +M +C
4. +C +M

Ergebnis:
__ 1 _-_ 2 __ 3 __4

Clicker: Wie ist die Bildschirm-Ausgabe?

```
class Mitarbeiter  {
public:
   Mitarbeiter()     {cout << "+M ";}
   virtual ~Mitarbeiter()   {cout <<"-M ";}
};
class Chef : public Mitarbeiter  {
public:
    Chef()  {cout << "+C ";}
   ~Chef()  {cout << "-C ";}
};
```

```
void Poly() {
   Mitarbeiter* ph1 = new Chef();
   delete ph1;
}
```

Ausgabe unabhängig von virtual bei Destruktor von Chef

1. +M +C –C -M
2. +M +C -M
3. +M +C
4. +C +M

Ergebnis:
_-_ 1 _ 2 __ 3 __4

Dynamischer Typ von ph1?

```
class Mitarbeiter  {
public:
   Mitarbeiter()     {cout << "+M ";}
   virtual ~Mitarbeiter()   {cout <<"-M ";}
};
class Chef : public Mitarbeiter  {
public:
    Chef()  {cout << "+C ";}
   ~Chef()  {cout << "-C ";}
};
```

```
void Poly() {
   Mitarbeiter* ph1 = new Chef();
   delete ph1;
}
```

1. Zeiger auf Mitarbeiter
2. Zeiger auf Chef
3. Mitarbeiter
4. Chef

Ergebnis:
__ 1 _-_ 2 __ 3 __4

Statischer Typ von ph1?

```
class Mitarbeiter  {
public:
   Mitarbeiter()     {cout << "+M ";}
   virtual ~Mitarbeiter()   {cout <<"-M ";}
};
class Chef : public Mitarbeiter  {
public:
    Chef()  {cout << "+C ";}
   ~Chef()  {cout << "-C ";}
};
```

```
void Poly() {
   Mitarbeiter* ph1 = new Chef();
   delete ph1;
}
```

1. Zeiger auf Mitarbeiter
2. Zeiger auf Chef
3. Mitarbeiter
4. Chef

Ergebnis:
_-_ 1 _ 2 __ 3 __4

Aufgaben

```
class Saeuger {
public:
   void ton() {datei << " knurr ";}
   virtual int beine() {return 0;}
   };
class Hund: public Saeuger{
public:
   void ton() {datei << " wau ";}
   virtual int beine() {return 4;}
private:
   };
class Katze: public Saeuger{
public:
   void ton() {datei << " miau ";}
private:
   };
```

```
void funk7() {

   Saeuger s1;
   Hund h2;
   Katze k3;

   datei << "\n" << s1.beine();  s1.ton();
   datei << "\n" << h2.beine();  h2.ton();
   datei << "\n" << k3.beine();  k3.ton();
   }
```

0 knurr
4 wau
0 miau

Aufgaben

```
class Saeuger {
public:
   void ton() {datei << " knurr ";}
   virtual int beine() {return 0;}
   };
class Hund: public Saeuger{
public:
   void ton() {datei << " wau ";}
   virtual int beine() {return 4;}
private:
   };
class Katze: public Saeuger{
public:
   void ton() {datei << " miau ";}
private:
   };
```

```
void funk6() {

   Saeuger* s1 = new Hund();
   Hund* h2 = new Hund();
   Saeuger* s3 = new Saeuger();
   Saeuger* s4 = new Katze();

   // Zeigerarray initialisieren
   Saeuger* arr[] = {s1, h2, s3, s4};

   for (int i=0; i<4; ++i) {
      datei << "\n" << i << " "
            << arr[i]->beine();
       arr[i]->ton();
       delete arr[i];
      }
   }
```

```
0 4 knurr
1 4 knurr
2 0 knurr
3 0 knurr
```

Aufgaben

```
class Saeuger {
public:
   void ton() {datei << " knurr ";}
   virtual int beine() {return 0;}
   };
class Hund: public Saeuger{
public:
   void ton() {datei << " wau ";}
   virtual int beine() {return 4;}
private:
   };
class Katze: public Saeuger{
public:
   void ton() {datei << " miau ";}
private:
   };
```

```
void h(Saeuger& s) {
   datei <<  s.beine();  s.ton();
    }
void g(Saeuger* s) {
   datei <<  s->beine();  s->ton();
    }
void f(Saeuger s) {
   datei <<  s.beine();  s.ton();
    }

void funk8() {
   Hund h2;        Katze k3;
   h(h2);   h(k3);
   g(&h2);   g(&k3);
   f(h2);   f(k3);

}
```

4 knurr
0 knurr

4 knurr
0 knurr

0 knurr
0 knurr

Übung 7: Welche Ausgabe ergibt sich? Aufgabe c

```
class Figur   {
   int dummy;
public:
   Figur()  {dummy=0; datei << "+F";}
   ~Figur() {datei << "-F";}
class Ring: public Kreis  {
   int rRad;  // Innenkreis-Radius
public:
      Ring(int r)  {
         rRad=r;
         datei << "+R" << rRad << " ";       }
      Ring(int r, int rr):Kreis(r), rRad(rr) {
         datei << "+R" << rRad << " ";       }
      ~Ring() {
         datei << "-R" << rRad << " ";       }
 /* …*/
};
```

```
class Kreis: public Figur  {
   protected:      enum {PI=3};
      int rad;  // Außenkreis-Radius
   public:
      Kreis(): rad(4)  {
         datei << "+K" << rad << " ";        }
      Kreis(int r) {
         rad=r;
         datei << "+K" << rad << " ";        }
      ~Kreis() {
         datei << "-K" << rad << " ";        }
 /* …* /
};
```

```
void funk3()   {
   Kreis* k1[5];
   datei << " xxx ";
   k1[4]=NULL;
   datei << " yyy ";
   Kreis  k2[2];
   }
```

Übung 7: Welche Ausgabe ergibt sich? Aufgabe d

```
class Figur   {
   int dummy;
public:
   Figur()  {dummy=0; datei << "+F";}
   ~Figur() {datei << "-F";}
class Ring: public Kreis  {
   int rRad;  // Innenkreis-Radius
public:
      Ring(int r)  {
         rRad=r;
         datei << "+R" << rRad << " ";       }
      Ring(int r, int rr):Kreis(r), rRad(rr) {
         datei << "+R" << rRad << " ";       }
      ~Ring() {
         datei << "-R" << rRad << " ";       }
 /* …*/
};
```

```
class Kreis: public Figur  {
   protected:      enum {PI=3};
      int rad;  // Außenkreis-Radius
   public:
      Kreis(): rad(4)  {
         datei << "+K" << rad << " ";         }
      Kreis(int r) {
         rad=r;
         datei << "+K" << rad << " ";         }
      ~Kreis() {
         datei << "-K" << rad << " ";         }
 /* …* /
};
```

```
void funk4()    {
  Kreis* p = new Kreis(12);
  delete p;
   }
```

Übung 7: Welche Ausgabe ergibt sich? Aufgabe e

```
class Figur   {
   int dummy;
public:
   Figur()  {dummy=0; datei << "+F";}
   ~Figur() {datei << "-F";}
class Ring: public Kreis  {
   int rRad;  // Innenkreis-Radius
public:
      Ring(int r)  {
         rRad=r;
         datei << "+R" << rRad << " ";       }
      Ring(int r, int rr):Kreis(r), rRad(rr) {
         datei << "+R" << rRad << " ";       }
      ~Ring() {
         datei << "-R" << rRad << " ";       }
 /* …*/
};
```

```
class Kreis: public Figur  {
   protected:      enum {PI=3};
      int rad;  // Außenkreis-Radius
   public:
      Kreis(): rad(4)  {
         datei << "+K" << rad << " ";         }
      Kreis(int r) {
         rad=r;
         datei << "+K" << rad << " ";         }
      ~Kreis() {
         datei << "-K" << rad << " ";         }
 /* …* /
};
```

```
void funk4()    {
  Figur* p = new Kreis(12);
  delete p;
   }
```

Übung 7: Welche Ausgabe ergibt sich? Aufgabe i

```
class Figur   {
   int dummy;
public:
   Figur()  {dummy=0; datei << "+F";}
   ~Figur() {datei << "-F";}
   virtual double flaeche() = 0;
};

class Ring: public Kreis  {
   int rRad;  // Innenkreis-Radius
public:
      Ring(int r)  {
         rRad=r;
         datei << "+R" << rRad << " ";       }
      Ring(int r, int rr):Kreis(r), rRad(rr) {
         datei << "+R" << rRad << " ";       }
      ~Ring() {
         datei << "-R" << rRad << " ";       }
virtual double flaeche() {return (rad * rad –
    rRad*rRad) * PI - ;]
 /* …*/
};
```

```
class Kreis: public Figur  {
   protected:      enum {PI=3};
      int rad;  // Außenkreis-Radius
   public:
      Kreis(): rad(4)  {
         datei << "+K" << rad << " ";         }
      Kreis(int r) {
         rad=r;
         datei << "+K" << rad << " ";         }
      ~Kreis() {
         datei << "-K" << rad << " ";         }
virtual double flaeche() {return rRad*rRad *
              PI;]

 /* …* /
};
```

Warum kann von den Klassen Figur und Ring kein Array erzeugt werden?

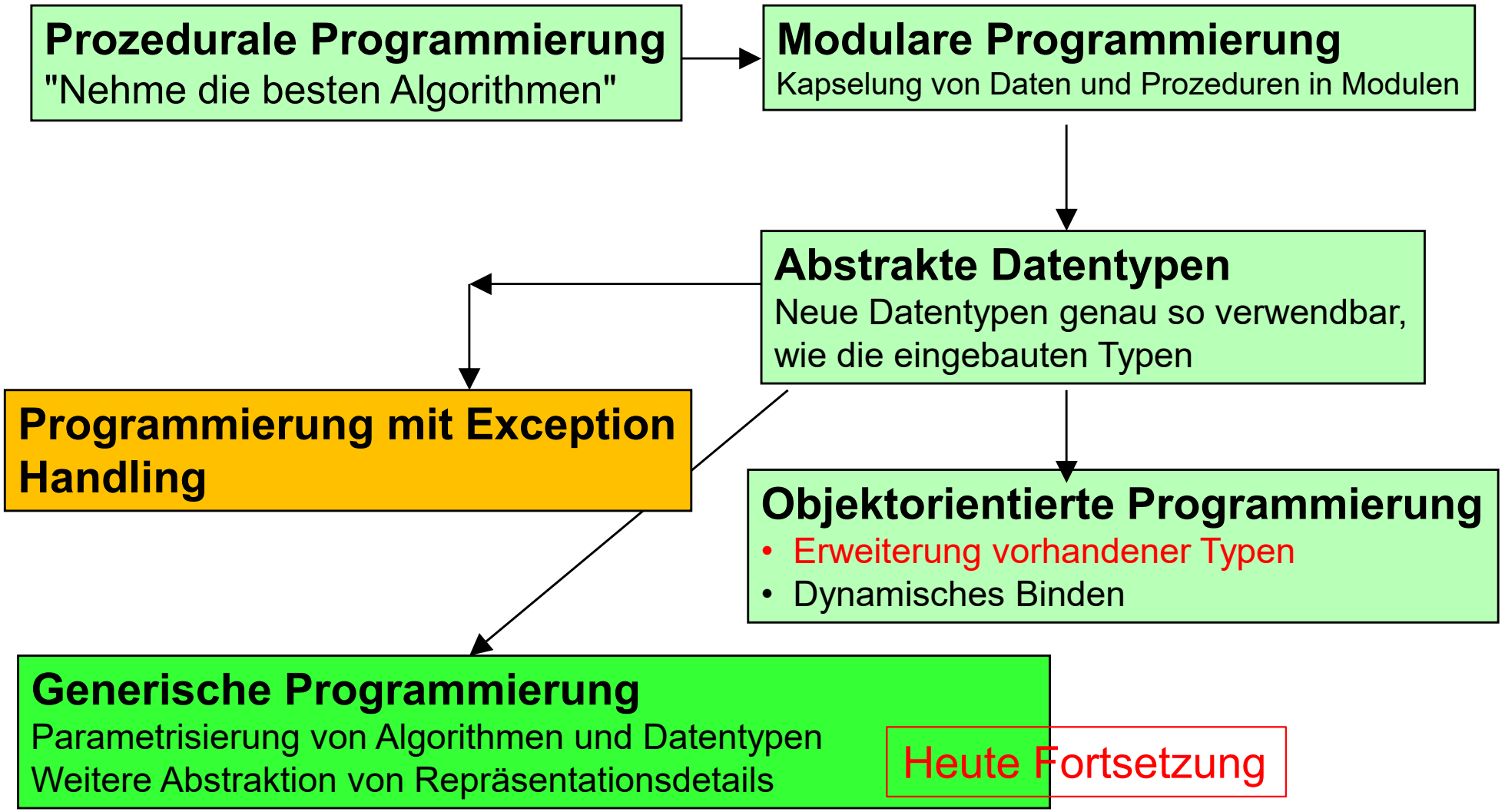
Schlüsselwort „final“: Nachtrag zur letzten Vorlesung

```
class Frucht final
{
public:
   virtual void print() { cout << "Frucht"; }
};

// von Frucht kann nicht abgeleitet werden
class Apfel : public Frucht
{
public:
   virtual void print() { cout << "Apfel"; }
};
```

Final - Methoden

```
class Frucht
{
public:
   virtual void print() final { cout << "Frucht"; }
};

// print kann nicht überschrieben werdne
class Apfel : public Frucht
{
public:
   // virtual void print() { cout << "Apfel"; }
};
```

Programmierparadigmen von C++



Ziele der Veranstaltung (Folie aus Vorlesung 1)

- Testgetriebene Software-Entwicklung (Test first)
- „Erst denken, dann hacken“ (zunächst (mit Worten oder formal) beschreiben)
- Die vier Gesichter von C++:
  - Prozedurale Entwicklung in C++ (Zeigermodell, Werte- und Referenzsemantik)
  - Objektorientierte Software-Entwicklung (Vererbung, Polymorphie)
  - Generische Software-Entwicklung (Templates)
  - Programmierung mit der Standard-Template-Library
- Einstieg in die Programmierung im Großen (Bibliotheken, SW-Entwicklung im Team)

Wie wird man ein guter Software-Ingenieur? Mein Anspruch

Kompetenzen können durch Wissen unterfüttert, aber nicht durch Ausbildung produziert werden können.
Vorbilder können hier helfen.
Codierung ist ein wichtiger Bestandteil der Ausbildung, aber steht nicht im Vordergrund. Jeder Informatiker sollte aber das Handwerk beherrschen, da es die Basis seines Berufs bildet.

Sie lernen einfache, gut lesbare Programme zu erstellen, deren Korrektheit nicht dem Zufall überlassen wird.

Hauptsache „es läuft“ wird schief gehen.
Standards, Stil und Form der Programmierung sind Gegenstand der Lehre und der Kontrolle im Übungsbetrieb.

Probleme der Praxis

- Erhebung von Anforderungen, die alles andere als klar und vorgegeben sind.
- Koordination vieler Projektmitarbeiter.
- Konflikte zwischen Termin-, Qualitäts- und Funktionalitätszielen
- Arbeit mit neuen keinesfalls fehlerfreien Werkzeugen
- Abstimmung mit schwierigen Partnern (mit mir und ggf. mit Kommilitonen)

Vorlesungsplanung heute, 28.11.2025

- Wiederholung zur Polymorphie anhand von Clicker-Fragen
- Team-A, Team-B, Team-C stellen den aktuellen Stand vor
- Systematischer Einstieg in die Programmierung mit Templates (Iterator-Konzept)
- Was bei Lösung von Aufgabe 2 in vorherigen Semestern auffiel
- Ggf. Teil 2 Systematischer Einstieg in die Programmierung mit Templates (Iterator-Konzept)