Clicker

Bitte den Link

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

für WS 2025/26 nutzen.

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

- Wiederholung mit Referenz oder Zeiger oder Werte oder doch was anderes?
- MacheNix als Übung für Sie (Kopierkonstruktor und Zuweisungsoperator, selber „Hand anlegen“)
- Operatoren
- Übung Plus, bei der Vektor-Klasse bzw. bei DynVorgangsArray

Clicker-“Abstimmung“

```
ofstream datei("ausgabe.txt", ios::out);
class Hund
{
public:
   Hund(string f) {farbe=f;
                   datei << "+H " << farbe;};
   ~Hund() {datei << " -H " << farbe;};
private:
  string farbe;
};
```

```
void func1() {
   Hund bello("g ");
   datei << " Ende ";
}
```

Welche Dateiausgabe?

1. +H g Ende
2. Ende
3. Clicker ist anstrengend
4. +H g Ende -H g

Clicker-“Abstimmung“

```
ofstream datei("ausgabe.txt", ios::out);
class Hund
{
public:
   Hund(string f) {farbe=f;
                   datei << "+H " << farbe;};
   ~Hund() {datei << " -H " << farbe;};
private:
  string farbe;
};
```

```
void func2() {
   Hund bello("g ");
   Hund anton("s ");
   datei << " Ende ";
}
```

Welche Dateiausgabe?

1. +H g +H s Ende
2. +H g +H s Ende -H g -H s
3. +H g +H s Ende -H s -H g
4. +H g +H s -H g -H s Ende

„Noch“ sehr komplexe Fragestellung

```
ofstream datei("ausgabe.txt", ios::out);
class Hund   {
public:
   Hund(string f="?") {farbe=f;
              datei << "+H " << farbe;};
   ~Hund() {datei << " -H " << farbe;};
   string GetFarbe()const  {return farbe;}
private:
  string farbe;
};
```

```
void func4_hlp(Hund h) {
     datei << " hlp ";   }

void func4() {
   Hund bello("g");
   func4_hlp(bello);
   datei << " E ";  }
```

Welche Dateiausgabe?

1. +H g hlp -H g E -H g
2. +H g +H g hlp -H g E -H g
3. +H g hlp E -H g
4. +H g hlp E

„Achtung“ Fragestellung

```
ofstream datei("ausgabe.txt", ios::out);
class Hund   {
public:
   Hund(string f="?") {farbe=f;
              datei << "+H " << farbe;};
   ~Hund() {datei << " -H " << farbe;};
   string GetFarbe()const  {return farbe;}
private:
  string farbe;
};
```

```
void func5() {
   Hund bello();
   Hund wau;
   datei << " Ende ";  }
```

Welche Dateiausgabe?

1. +H ? +H ? Ende –H? –H?
2. +H ? Ende –H?
3. Syntaxfehler
4. +H ? +H ? –H? Ende –H?

„Achtung“ Fragestellung

```
double sin(double);
```

Eine Funktion, die „sin“ heißt, ein double akzeptiert und ein double zurückliefert!

```
Hund fritz(double);
```

Eine Funktion, die „fritz“ heißt, ein double akzeptiert und einen Hund zurückliefert!

```
void func5() {
   Hund bello();
   Hund wau;
   datei << " Ende ";  }
```

Eine Funktion, die „**bello**“ heißt, kein Argument besitzt und einen Hund zurückliefert!

Die gleiche Frage noch einmal

```
ofstream datei("ausgabe.txt", ios::out);
class Hund   {
public:
   Hund(string f="?") {farbe=f;
              datei << "+H " << farbe;};
   ~Hund() {datei << " -H " << farbe;};
   string GetFarbe()const  {return farbe;}
private:
  string farbe;
};
```

```
void func5() {
   Hund bello();
   Hund wau;
   datei << " Ende ";  }
```

Welche Dateiausgabe?

1. +H ? +H ? Ende –H? –H?
2. +H ? Ende –H?
3. Syntaxfehler
4. +H ? +H ? –H? Ende –H?

Clicker: Schnittstelle

```
class Baum  {
public:
   Baum(Baum* eltern, int blaetter);
   int GetAnzahlDerBlaetter() const;
   const Baum* GetVater();
   Baum* GetKind();
   Baum* elternTeil;
private:
   void SetAnzahlDerBlaetter(int b);
   int anzahlDerBlaetter;
   Baum* kindTeil;
   Baum* GetKindBaum();
};
void baumTest() {
   Baum b(nullptr, 1008);
   b.elternTeil = nullptr; }
```

Die Codezeile: **b.elternTeil = nullptr;**

1. Ist syntaktisch korrekt und guter Programmierstil
2. Ist syntaktisch falsch
3. Attribute gehören nicht in die Klassen-Schnittstelle.
4. Eine Klasse sollte nicht „Baum“ heißen.

Schnittstelle

```
class Baum  {
public:
   Baum(Baum* eltern, int blaetter);
   int GetAnzahlDerBlaetter() const;
   const Baum* GetVater();
   Baum* GetKind();
   Baum* elternTeil;
private:
   void SetAnzahlDerBlaetter(int b);
   int anzahlDerBlaetter;
   Baum* kindTeil;
   Baum* GetKindBaum();
};
```

```
void baumTest() {
   Baum b(NULL, 1008);
   b.elternTeil = NULL;  // korrekt
```

Das ist aber keine Programmierung mit abstrakten Datentypen.

Attribute sollten nicht direkt zur Schnittstellen gehören.

Minimale Standardschnittstelle

Folgende Elemente sind für viele Klassen eine notwendige Grundausstattung:

- Standardkonstruktor: **X()**
- Kopierkonstruktor: **X(const X& x)**
- Destruktor: **~X()**
- Zuweisungsoperator: **X& operator= (const X& x)**

Die Meinungen zum Thema "Minimale Standardschnittstelle" sind allerdings uneinheitlich.

Empfehlung: Beim Entwurf einer Klasse für jedes dieser vier Elemente prüfen, ob es definiert sein soll. Wenn ja, entscheiden, ob das jeweils vom System erzeugte Default-Element in Ordnung ist, oder ob das Element explizit selbst definiert werden soll. Wenn nein, dann sollte das Element explizit unterbunden werden, siehe dazu die folgende Erläuterung.

Clicker-“Abstimmung“

```
class Auto {
   public:
      Auto(string, float, float, int);
      void SetHubraum(int hub);
      int WieWeitKommeIch();
   private:
      string hersteller;
      float verbrauch;
      float tankinhalt;
      int hubraum;
};
```

Hat Auto eine minimale Standardschnittstelle?

1. Nein, Destruktor fehlt
2. Nein, Kopierkonstruktor fehlt
3. Nein, Standardkonstruktor fehlt
4. Nein, Zuweisungs-Operator fehlt

Clicker-“Abstimmung“

```
class Auto {
   public:
      Auto(int hubraum=14);
     ~Auto();
private:
   int hubraum;
};
```

Hat Auto eine minimale Standardschnittstelle?

1. Nein, Destruktor fehlt
2. Nein, Kopierkonstruktor fehlt
3. Nein, Zuweisungs-Operator fehlt
4. Nein, Standardkonstruktor fehlt

Clicker-“Abstimmung“

```
class Auto {
   public:
      Auto(int hubraum=14);
     ~Auto();
      Auto(const Auto& auto);
      bool operator==(const Auto& a);
private:
   int hubraum;
};
```

Hat Auto eine minimale Standardschnittstelle?

1. Ja
2. Nein, Kopierkonstruktor fehlt
3. Nein, Zuweisungs-Operator fehlt
4. Nein, Standardkonstruktor fehlt

Clicker-“Abstimmung“

```
class Auto {
   public:
 Auto(int hubraum=14);
~Auto();
 Auto(const Auto& auto);
 Auto& operator=(const Auto& auto);
private:
   int hubraum;
};
```

Hat Auto eine minimale Standardschnittstelle?

1. Ja
2. Nein, Kopierkonstruktor fehlt
3. Nein, Zuweisungs-Operator fehlt
4. Nein, Standardkonstruktor fehlt

Vorlesungsplanung

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

Polymorphie

Feedback zur Übungsaufgabe 03
Beginn Templates