# Agenda "C++ Grundlagen"

Grobe zeitliche Einteilung:

- **Tag 1**: Einstieg in Objekte und Klassen
- **Tag 2**: Weitere Vertiefungen
- **Tag 3**: Big-Three (Dynamische Daten)
- **Tag 4**: Vererbung / Templates
- **Tag 5**: STL (Standard Template Library)

**Montag / 1. Tag:**

## Einstieg: Unterschiede zwischen C und C++ / Wiederholung / Parameterübergabe

- Wiederholung von Pointern und Adressen
- Funktion `swap`

---

## Einstieg in Objekte und Klassen

Motivation für das Konzept *Klasse* am Beispiel einer *Uhrzeit*

Erläuterung der Grundkonzepte am Beispiel der Klasse `Time`:

- Klasse und Objekt
- Instanzvariablen, Methoden
- Punkt-Operator für den Zugriff
- Schlüsselwörter `private` versus `public`
- setter- / getter-Methoden
- `this`-Operator (motiviert an Hand von Parameter- und Instanzvariablennamen)
- Überladen von Methoden
- Klassen- vs. Instanzvariablen / Klassen- vs. Instanzmethoden / `static`
- Debugger

---

*Übung*: Die Klassen `Fraction`

---

## Weiterarbeit an den OO-Konzepten mit den Themenschwerpunkten

- Konstruktoren
- Konstruktoren: Verkettung von Konstruktoren
- Konstruktoren: Element-Initialisierungsliste (*member initializer list*)
- Einführung des Konzepts *Referenz* / Vergleich Pointer - Referenzen
- Betrachtung einer Methode `malZwei(int wert)` in den drei Varianten *call-by-value*, *call-by-pointer* und *call-by-reference*.
- Parameterübergabemechanismus: *call-by-value* vs. *call-by-reference*
- Methode add: Diskussion der Schnittstelle (`const Time&`)
- Stolperfalle aufzeigen: `Time t();`
- `const`-Methoden (Einmal `const`, immer `const`)

---

## Prinzipieller Vergleich zwischen prozeduraler und OO-Programmierung

- Code-Snippet für prozedurale Programmierung aufzeigen (Funktionen rufen Funktionen)
- Code-Snippet für objektorientierte Programmierung aufzeigen (Methoden an Objekten aufrufen)
- In Objekten können benachbarte Methoden aufgerufen werden
- Wie wird der Aufruf einer Methode (OO-Programmierung) auf den Aufruf einer C-Funktion abgebildet?

---

*Übung*: Weiterarbeit an der Klasse `Fraction`

---

**Dienstag / 2. Tag:**

## Operator-Overloading

- Operator-Overloading: Innerhalb und außerhalb der Klasse
- Operatoren: Stelligkeit und Priorität
- Operatoren: Assoziativität
- Operator-Overloading: Vergleichsoperatoren <=, >=, == und Addition +
- Increment ++ / Decrement --

---

*Übung*: Weiterarbeit an der Klasse `Fraction`

---

## Ein- und Ausgabe mit IOStream

Klasse `Time` um Ausgabe auf der Konsole ergänzen

---

## OO-Programmierung / Entwurf von Methoden

Bei Methoden (Operatoren) gibt es zwei Möglichkeiten der Realisierung:

- Das Ergebnis des Methodenaufrufs (der Operation) wird in den Instanzvariablen des gerufenen Objekts abgelegt.
- Das Ergebnis des Methodenaufrufs (der Operation) wird als Resultat zurückgegeben, die Instanzvariablen des gerufenen Objekts bleiben unverändert.

Beide Varianten erläutern, die Begriffe *immutable behaviour* (unveränderbares Verhalten) und *mutable behaviour* (veränderbares Verhalten) einführen.

---

## Namensräume

- Schlüsselwort `namespace`
- Namensräume definieren und verwenden

---

## OO-Programmierung: Abrundung

- Überladen von unären Operatoren (invertieren, negieren)
- Konversionsoperator einführen: `int` --> `Time` / `Time` --> `int` (Sekunden)

Hinweis: Folgendes Szenario ist zu betrachten:

```
Fraction f(1, 7);
Fraction f1;
f1 = f + 1;
```

Nun einen Konstruktor mit der Signatur `(int)` einführen und erläutern, was passiert.

---

## Exception-Handling

In der Klasse `String` zum Beispiel eine `setAt`-Methode mit Exception-Handlung ausstatten.

---

## Daten dynamisch allokieren und freigeben

- Lebensdauer von Variablen (lokal, global, dynamisch)
- Pointer auf dynamische Daten: `new` und `delete`-Operator
- Betrachtung von Konstruktor und Destruktor
- Arrays allokieren: `new[]` und `delete[]`

---

**Mittwoch / 3. Tag:**

## “Big-Three”

Beobachtungen:

- Wertzuweisung: Geht! (ohne Unterstützung des Programmierers !!!)
- Kopier-Konstruktor: Geht! (ohne Unterstützung des Programmierers !!!)

---

Am Beispiel der Klasse `BigData` ist das Konzept von “Big-Three” zu erläutern.

- Zuerst mit dem Kopier-Konstruktor beginnen (keine Freigabe der linken Seite)
- Danach `operator=`
- Danach: Eine Funktion `createBigData` mit einer `return`-Anweisung diskutieren

---

*Übung*: Klasse `String` oder Klasse `IntegerSet`

---

**Donnerstag / 4. Tag:**

## Vererbung / Templates

- Prinzip der beiden Beziehungen erläutern: "is-a" und "has-a".
- Am Beispiel der beiden Klassen `Rectangle` und `ColoredRectangle` sowie `Point` und `Line` diskutieren (eine Linie besteht aus zwei Punkten, ist aber kein Punkt).
- Dann das Beispiel `ColoredRectangle` und `BlackWhiteRectangle` erläutern
- Basisklasse `Rectangle`: `eraseBackground` und `drawBorder`

Der Reihe nach besprechen:

- Zugriff auf Membervariablen der Vaterklasse von der Kindklasse aus
- Dto. für Methoden
- Überschreiben von Methoden
- Verdecken von Elementen
- `virtual` vs non-`virtual`

*Beachte*: Die Vererbung von Methoden kann unter drei Gesichtspunkten betrachtet werden:

- *Wiederverwendung*: Die Methode wird in der abgeleiteten Klasse nicht überschrieben
- *Erweiterung*: Die Methode wird in der abgeleiteten Klasse überschrieben, es wird ferner die Methode bei der Basisklasse aufgerufen.
- *Ersatz*: Die Methode wird in der abgeleiteten Klasse überschrieben, die Methode der Basisklasse wird *nicht* aufgerufen.

---

## Polymorphismus

Am Beispiel von Figuren, Rechtecken und Kreisen aufzeigen, wie Methoden mit unterschiedlichem Code (also "polymorpher Code") mit Hilfe der Reduktion auf gemeinsame Basisklassen in einem homogenen Umfeld (z. B. ein Array von Figuren) aufgerufen werden können.

- Polymorphismus
- Abstrakte Klasse
- Interface
- Virtueller Basisklassendestruktor

---

*Übung*: Klasse `Account` / Bankkonten

*Übung*: `transfer`-Methode diskutieren (Design Pattern *Template Method*)

---

## Templates

- Templates
  - Klassentemplate: `Calculator`
  - Funktionstemplate: `min`, `max`

---

###### Freitag / 5. Tag

## STL (Standard Template Library)

- Basiskonzepte:
  - Container
  - Iteratoren
  - Algorithmen
  - Funktoren
- Weitere Beispiele:
  - Container demonstrieren: `std::vector`
  - Container demonstrieren: `std::map`
  - Auf `std::pair` eingehen
  - Container demonstrieren: `std::set`
- Algorithmus: `std::for_each`
  - Iteratoren-Objekte
  - Globale Funktion für die Ausgabe
  - Funktor-Objekt für die Ausgabe (Instanzvariable für Index-Variable)
- Weitere Algorithmen:
  - `std::fill`, `std::generate`
  - `std::find`, `std::find_if` (Rückgabewerte sind Iteratoren)
  - `std::sort` (Sortierkriterium)
  - `std::transform`, `std::copy_if`, `std::accumulate` (funktionale Programmierung)
- Wenn die Zeit reicht:
  - `std::transform` oder `std::copy_if` mit `std::back_inserter`
  - `emplace_back`

---

*Übung*: Klasse `Phonebook`

oder:

*Übung*: Klasse `Account` mit Bankinstitut (`std::vector` oder `std::map`)

---

## Literatur

- Github
- peterloos.de
- Siehe bei *Modern C++*

---