

Hydroinformatik - SoSe 2025

UW-BHW-414-D: Objekt-Orientierte Programmierung

Olaf Kolditz

¹Helmholtz Centre for Environmental Research – UFZ, Leipzig

²Technische Universität Dresden – TUD, Dresden

³Center for Advanced Water Research – CAWR

⁴TUBAF-UFZ Center for Environmental Geosciences – C-EGS, Freiberg / Leipzig

Dresden, 09.05.2025

<https://www.ufz.de/index.php?de=40416>

<https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/32518209537?10>

Zeitplan: Hydroinformatik I+II

Sommersemester 2025

Stand: 08.05.2025

Nr.	KW	Datum	ID	Thema
01	15	11.04.2025	UW-BHW-414-A	Einführung, Werkzeuge#1, Hello World
03	17	25.04.2025	UW-BHW-414-B	Umweltinformatik, Werkzeuge#2 (git), Datentypen
05	18	02.05.2025	UW-BHW-414-C	Selbststudium
07	19	09.05.2025	UW-BHW-414-D	Objekt-Orientierte Programmierung: C++, Klassen
09	20	16.05.2025	UW-BHW-414-E	Python
11	21	23.05.2025	UW-BHW-414-F	Modellierung, Digitalisierung, Wasser 4.0
11	22	30.05.2025	UW-BHW-414-G	KI, Maschinelles Lernen, Neuronale Netzwerke
13	23	06.06.2025	UW-BHW-414-H	Kontinuumsmechanik, Hydromechanik
05	18	13.06.2025		Vorlesungsfreie Woche
15	25	20.06.2025	UW-BHW-414-I	Differentialgleichungen, Näherungsverfahren
17	26	27.06.2025	UW-BHW-414-J	Finite-Differenzen, explizite Verfahren
19	27	04.07.2025	UW-BHW-414-K	Finite-Differenzen, implizite Verfahren
21	28	11.07.2025	UW-BHW-414-L	Gerinnehydraulik, Grundwasserhydraulik
23	29	18.07.2025	UW-BHW-414-M	Zusammenfassung, Klausurvorbereitung

Fahrplan für heute ...

0. Rückblick letzte Veranstaltung: Umweltinformatik, Werkzeuge (git), Datentypen ...
-

1. Objekt-Orientierung: Geschichte und Konzept
 2. Objekt-Orientierte Programmierung (OOP): Daten-Abstraktion, Klassen, Instanzen, ...
 3. OpenGeoSys: ein OOP Software-Projekt in der Umweltforschung
 4. Übung: Datentypen
 5. Übung: Instanzen einer Klasse erzeugen und anwenden
-

6. Ausblick auf die nächste Veranstaltung: Python

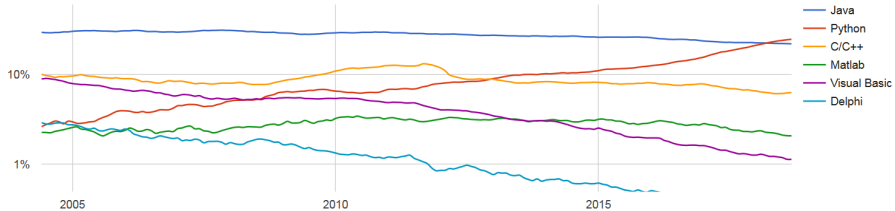
Objekt-Orientierung

Warum C++?

- Universalsprache
- objektorientiert
- performant
- sehr weit verbreitet
- Seit über 30 Jahren etabliert

Rank	Change	Language	Share	Trend
1	↑	Python	24.72 %	+5.4 %
2	↓	Java	22.01 %	-0.7 %
3	↑	Javascript	8.4 %	+0.1 %
4	↑	C#	7.71 %	-0.4 %
5	↓↓	PHP	7.42 %	-1.6 %
6		C/C++	6.32 %	-0.5 %
7		R	4.11 %	-0.1 %
8		Objective-C	3.29 %	-0.9 %
9		Swift	2.69 %	-0.8 %
10		Matlab	2.08 %	-0.3 %

PYPL Popularity of Programming Language



Objekt-Orientierung und C/C++: Geschichte



- C Entwickelt 1972 von Dennis Ritchie
 - ▶ Prozedurale Sprache
 - ▶ Universalsprache: Anwendbar zur Bearbeitung aller Arten von Problemen, d.h. nicht nur für bestimmte Gruppe von Anwendungen konzipiert
 - ▶ Compiler übersetzt Quelltext in Maschinencode, d.h. Programm ist auf allen Systemen lauffähig, für die es einen Compiler gibt
- C++ Entwickelt 1985 von Bjarne Stroustrup (<<)
 - ▶ Objekt-orientierte Sprache
 - ▶ 100% C-kompatibel (C++ ist ein Superset von C) d.h. man kann insbesondere auch C-Bibliotheken in C++ verwenden
 - ▶ Aktive Entwicklung: letztes Update war C/C++ 23

Objektorientierte Programmierung

- Fokus auf Daten statt auf Algorithmen
- Entspricht eher der menschlichen Sicht auf die Welt als prozedurale Programmierung
- Kapselung von Daten und darauf definierten Methoden in Klassen, d.h.
 - Daten und Funktionalität, die zusammengehören, sind auch im gleichen Objekt definiert/implementiert
 - Nach außen ist die Sicht auf die Daten abstrakt
 - Die Daten in Klassen können selbst wiederum Instanzen von Klassen sein
- Klassen können Eigenschaften und Funktionalität „vererben“, d.h. an spezialisierte Objekte weitergeben

⇒ Erleichtert das Erstellen von komplexen Programmen

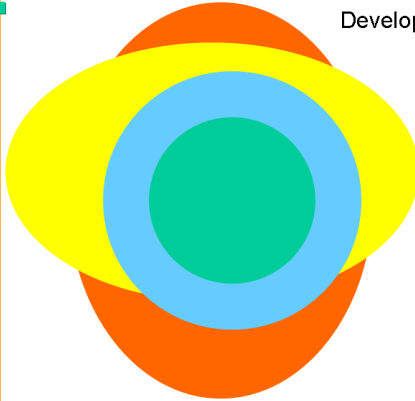
Objekt-Orientierung: C-Version



Contents

Definition

Object-Oriented Programming



Development of software:

- ever increasing functionality
- team work
- user interfaces
- ...

► Prozedurales Programmieren

► ...



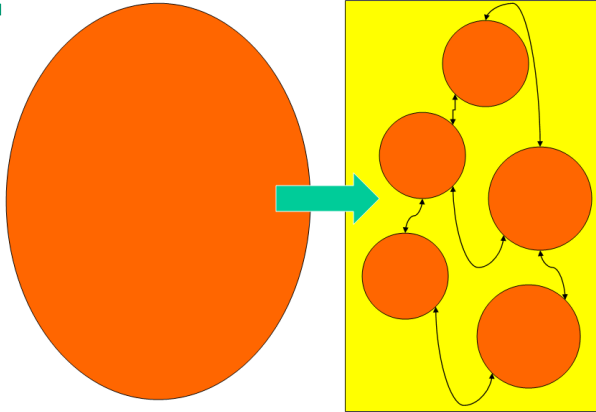
Objekt-Orientierung: C++ Version



Contents

Definition

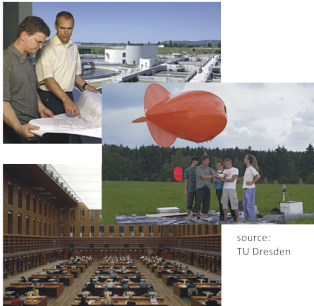
Object-Oriented Programming



- ▶ Zerlegen in logische Objekte
- ▶ Bilden von "Klassen"
- ▶ schmale Schnittstellen
- ▶ ...



Klassen::Konzept



Data abstraction ↓

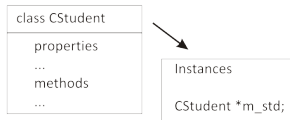


Figure: Das Klassen-Konzept - CStudent

Das Sprachelement der **Klassen** sind das entscheidende Kriterium von objekt-orientierten Konzepten und die objekt-orientierte Programmierung (OOP). Klassen sind eine Art Schablone für einen benutzerdefinierten Datentypen. Darüber hinaus enthält die Klasse neben den Daten auch alle Methoden (Funktionen), um mit den Daten der Klasse operieren zu können. Unser Beispiel für Klassen, das uns im Verlaufe der Vorlesung beschäftigen wird, ist - wie könnte es anders sein - CStudent (Abbildung). Für die Konzipierung von Klassen spielt die Abstraktion der Daten einer Klasse eine besonders wichtige Rolle.

Klassen::Daten-Abstraktion



Data abstraction ↓

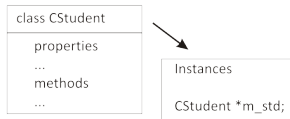


Figure: Das Klassen-Konzept - CStudent

Die Abbildung illustriert uns, dass eine Abstraktion von Daten (d.h. Eigenschaften) der Klasse Studenten eine durchaus vielschichtige Angelegenheit sein kann. Eine Aufstellung von Daten / Eigenschaften, die es aus ihrer Sicht zu berücksichtigen gilt, ist ihre nächste Hausaufgabe.

```

1  class CStudent
2  {
3      //data:
4      ...
5      //methods:
6      ...
7  };

```

Listing: C++ Daten und Methoden

Klassen::Daten-Schutz

Ein weiterer Vorzug von OO-Sprachen ist z.B. die Sichtbarkeit / Zugreifbarkeit von Daten zu regeln. Der nachfolgende Block zeigt das Datenschutz-Konzept von C++ (Sicherheitsstufen): Daten können öffentlich sein (public) oder gezielt für 'Freunde' verfügbar gemacht werden (protected) oder nur exklusiv für die eigene Klasse sichtbar zu sein (private).

```
1 class CStudent
2 {
3     private:
4         ...
5     protected:
6         ...
7     public:
8         ...
9 };
```

Listing: C++ Datenschutz

Klassen::Deklaration und Definition

Im vorangegangenen Abschnitt haben wir uns mit der Datenabstraktion mittels Klassen beschäftigt. So sollte konsequenterweise jede Klasse auch ihre eigenen Quelldateien besitzen. Die Deklaration von Klassen erfolgt üblicherweise in einer sogenannten Header-Datei *.h. Für die Methoden / Funktionen der Klasse ist eine *.cpp Datei reserviert. Für uns bedeutet dies, zwei Dateien anlegen:

- ▶ student.h - die Deklaration der Klasse CStudent
- ▶ student.cpp - die Methoden der Klasse CStudent

Klassen::Deklaration

Um mit der Klasse arbeiten zu können, müssen wir das entsprechende Header-File inkludieren. Dies erfolgt mit der Anweisung `#include "student.h"` am Anfang unseres Main-Files.

```
1 #include "student.h"
2 int main
3 {
4     return 0;
5 }
```

Listing: C++ - Declararion

Klasse::Instanzen



Data abstraction ↓

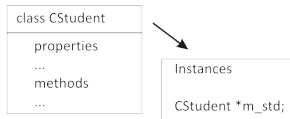


Figure: Das Klassen-Konzept - CStudent

An dieser Stelle erinnern wir uns an die Eingangsgraphik: Instanzen sind Kopien einer Klasse mit denen wir arbeiten können, dass heißt diese bekommen echten Speicher für ihre Daten (die natürlich für jede Instanz einer Klasse unterschiedlich sein können).

Klasse::Instanzen

Es gibt zwei Möglichkeiten, Instanzen einer Klasse zu erzeugen:

```
1 #include "student.h"
2 void main()
3 {
4     // Creating an instances of a class - 1
5     CStudent m_std_A;
6     // Creating an instances of a class - 2
7     CStudent *m_std_B;
8 }
```

Listing: Creating instances

Der direkte und der mittels eines sogenannten Zeigers (hierfür gibt ein Extra-Kapitel). Wir werden sehen, dass der zweite Weg oft der bessere ist, da wir z.B. die Initialisierung und das Speichermanagement für unsere Daten selber in die Hand nehmen können. Dies können wir mittels sogenannter Konstruktoren und Destruktoren erledigen. Damit beschäftigen wir uns im nächsten Abschnitt.

Objekt-Orientierung: Internet-Quellen

Links:

<https://www.youtube.com/watch?v=JBjjnqG0BP8>

<http://www.stroustrup.com/>

https://en.wikipedia.org/wiki/Bjarne_Stroustrup

Beispiele

- ▶ Datenlogger
- ▶ OpenGeoSys
- ▶ ...

Objektorientierte Programmierung: Beispiel

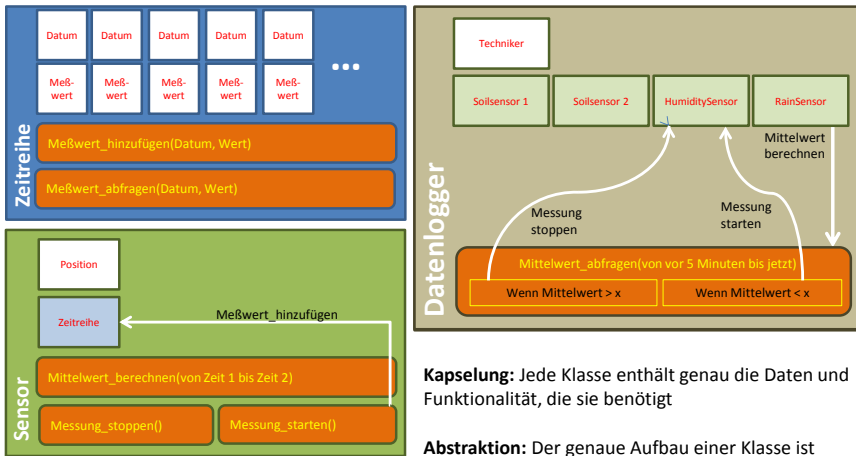
- Sie haben einige Datenlogger in einem Versuchsgelände installiert. Die angeschlossenen Sensoren messen Bodenfeuchte, relative Luftfeuchtigkeit und Niederschlag in festen Abständen. Sie möchten die Daten mit einem C++ Programm verwalten.



Objektorientierte Programmierung: Beispiel (2)

- Definition einer Klasse „**Datenlogger**“
 - An jeden Datenlogger sind ein oder mehrere **Sensoren** angeschlossen.
 - Jedem Datenlogger ist ein verantwortlicher **Techniker** zugeordnet
 - Wenn es zu stark regnet, soll **keine** Luftfeuchtigkeit gemessen werden, weil die Messwerte dann verfälscht sind
- Definition von Klasse „**Sensor**“ mit den davon abgeleiteten, spezialisierten Klassen „**SoilSensor**“, „**HumiditySensor**“ und „**RainSensor**“
 - Jeder Sensor kennt seine genaue **Position**
 - Jeder Sensor speichert eine **Zeitreihe**
 - Es soll möglich sein, den **Mittelwert** der Messwerte zwischen zwei Zeitpunkten zu berechnen
- Definition einer Klasse „**Zeitreihe**“
 - Jede Zeitreihe speichert eine Reihe von **Datumsangaben** und **Messwerten**
 - Man kann neue Messwerte **hinzufügen** oder alte Messwerte **abfragen**

Objektorientierte Programmierung: Beispiel (3)



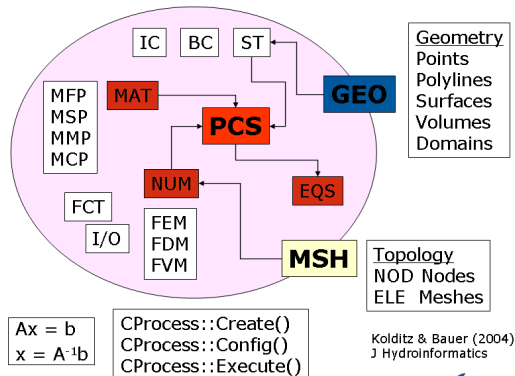
Kapselung: Jede Klasse enthält genau die Daten und Funktionalität, die sie benötigt

Abstraktion: Der genaue Aufbau einer Klasse ist nach aussen verborgen. Der Datenlogger weiss z.B. nicht, wie eine Zeitreihe aufgebaut ist.

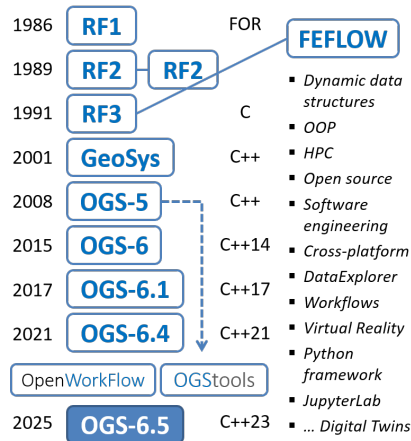
Objekt-Orientierung - OpenGeoSys (OGS)



V4: Object-Orientation: Multifield Problems



Kolditz & Bauer (2004)
J Hydroinformatics

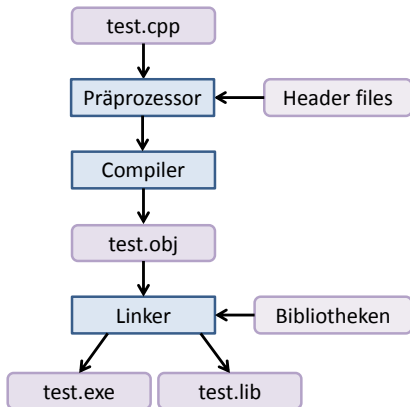


Übungen

- ▶ Compilieren und Linken
- ▶ die `main` function

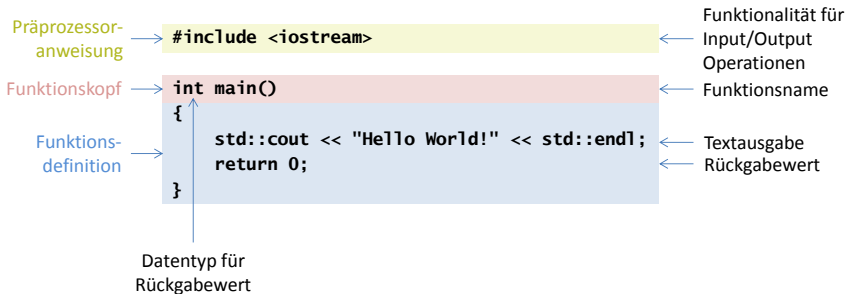
EX04 Datentypen

Compilieren und Linken



- **Präprozessor:** In allen C++-Files werden nacheinander `#include`-Anweisungen durch den Inhalt der Dateien ersetzt, Makros ersetzt (`#define`) und Teile des Quelltexts anhand definierter Bedingungen ausgewählt (`#if/#ifdef`)
- **Compiler:** Übersetzt den C++-Quellcode in maschinenspezifischen Binärcode und schreibt diesen in Objekt-Dateien
- **Linker:** Verknüpft aller Objektdateien, ersetzt Referenzen werden durch korrekte Speicheradressen und schreibt das Ergebnis entweder als eine shared library (z.b. unter Windows eine `*.dll`-Datei oder `*.dylib` unter macOS) oder als eine ausführbare Datei.

Ein erstes Beispielprogramm



Der tatsächliche Rückgabewert muss den Datentyp haben, der im Funktionskopf definiert ist. Der Datentyp `void` definiert, dass es keinen Rückgabewert gibt.

Dateinamen

- Die Datei, in der `main()` definiert ist, ist der Programmeinstiegspunkt `[Programmname].cpp`
- Bei größeren Projekten oder der Verwendung von Bibliotheken, werden weitere Dateien mittels `#include` eingebunden
 - Systembibliotheken: `#include <iostream>`
 - Benutzerdefiniert: `#include "hydro.h"`
- Alle Programmteile ausser dem Programmeinstiegspunkt bestehen aus
 - Headerdatei: `"hydro.h"` (Deklaration von Methoden)
 - Implementationsdatei: `"hydro.cpp"` (Implementation von Methoden)
- Durch das Einbinden der Headerdateien, werden existierende Funktionen dem Programm bekannt gemacht, ohne die Implementation zu kennen

EX04: Datentypen

Datentypen

- Alle Daten in einem Programm werden in Variablen gespeichert, z.B.:

```
string Vorlesungsname = "C++Datentypen";  
int    Dauer_in_Minuten = 90;  
float  Aufmerksame_Studenten = 27.3;  
bool   Ist_Vertretungsveranstaltung = true;
```

- Dafür muss das Programm folgende Dinge wissen:
 1. Wo ist die Information gespeichert? $\Rightarrow$ Speichermanagement
 2. Welche Art von Information ist gespeichert? $\Rightarrow$ Datentyp
 3. Welche Information ist gespeichert? $\Rightarrow$ Wert
- Der einfachste Datentyp: **bool**
 - Speichert Wahrheitswerte, d.h. **true** (1) oder **false** (0)
 - Benötigt 1 *byte* Speicherplatz

```
bool x = 17 > 3;  
std::cout << "Ist 17 grösser als 3? " << x << std::endl;
```

```
>Ist 17 grösser als 3? true
```

Ganzzahlige Datentypen in C++

Vorbemerkung:

- Daten werden in *bytes* gespeichert
- Ein *byte* ist eine digitale Informationseinheit und besteht aus 8 *bit*

$$1 \text{ byte} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array} = 2^6 + 2^3 + 2^1 + 2^0 = 75$$

- Jedes *bit* speichert eine binäre Zahl, d.h. entweder **0** oder **1**
- Das erste *bit* einer Zahl codiert das Vorzeichen (0 = positiv, 1 = negativ)

Datentypen für ganze Zahlen:

Datentypname	Speicherbedarf	Wertebereich
<i>short</i>	2 byte	[-32767, 32768], d.h. $-2^{15}-1$ bis 2^{15}
<i>int</i>	2-4 byte	Min. so groß wie <i>short</i> , höchstens so groß wie <i>long</i>
<i>long</i>	min. 4 byte	[-2147483647, 2147483648], d.h. $-2^{31}-1$ bis 2^{31}

Anzahl der bytes ist systemabhängig

Gleitkommazahlen (Reelle Zahlen)

- Reelle Zahlen werden als Kombination von **Mantisse** und **Exponent** gespeichert,

z.B.	1741.67	1.75167E+3	$1.75167 \cdot 10^3$
	0.0833	8.33E-2	$8.33 \cdot 10^{-2}$
	-25.876	-2.5876E+1	$-2.5866 \cdot 10^1$

- Durch die Speicherung von Zahlen in Binärcode und die begrenzte Anzahl von *bits* können Zahlen ggf. nicht exakt gespeichert werden und deshalb gerundet, z.b.

```
double x = 0.1;
std::cout << "x = " << x << std::endl;
```

```
>x = 0.10000000000000001
```

Datentypen für Gleitkommazahlen:

Datentypname	Speicherbedarf	Wertebereich
<i>float</i>	4 byte	$\pm 3.4E+38$, 6 Stellen Genauigkeit
<i>double</i>	8 byte	$\pm 1.7E+308$, 15 Stellen Genauigkeit

Zeichen/Text

char

- Zahlen von 0 bis 255 *oder* entsprechendes Zeichen in der ASCII-Tabelle
- Größe: 1 byte

```
char x = 'G';
std::cout << "Eingegebenes Zeichen: " << x << std::endl;
int y = x;
std::cout << "Der ASCII-Code für " << x << " ist " << y << std::endl;
```

```
> Eingegebenes Zeichen: G
> Der ASCII-Code für G ist 71
```

string

- Zeichenketten
- Größe: abhängig von der Textlänge, 1 + ein *byte* pro Zeichen

```
std::string text = "Learn C++ at http://www.learncpp.com";
```

ASCII	Char	ASCII	Char	ASCII	Char
0	(Null)	46	.	92	\
1	☺	47	/	93	]
2	☹	48	0	94	^
3	♥	49	1	95	_
4	♦	50	2	96	`
5	♣	51	3	97	a
6	♠	52	4	98	b
7		53	5	99	c
8	■	54	6	100	d
9	○	55	7	101	e
10	■	56	8	102	f
11	☞	57	9	103	g
12	☜	58	:	104	h
13	♪	59	;	105	i
14	♪	60	<	106	j
15	☼	61	=	107	k
16	▶	62	>	108	l
17	◀	63	?	109	m
18	↕	64	@	110	n
19	!!	65	A	111	o
20	¶	66	B	112	p
21		67	C	113	q
22	—	68	D	114	r
23	↓	69	E	115	s
24	↑	70	F	116	t
25	↓	71	G	117	u
26	→	72	H	118	v
27	←	73	I	119	w
28	⌞	74	J	120	x
29	↔	75	K	121	y
30	▲	76	L	122	z
31	▼	77	M	123	{
32	(Leerstelle)	78	N	124	
33	!	79	O	125	}
34	"	80	P	126	~
35	#	81	Q	127	☐
36	\$	82	R	128	Ç
37	%	83	S	129	ü
38	&	84	T	130	é
39		85	U	131	â
40	(	86	V	132	ä
41	)	87	W	133	à
42	*	88	X	134	â
43	+	89	Y	135	ç
44		90	Z	136	è
45	☞	91	[	137	ë

Datentypen: EX04-data-types

```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\okolditz>cd C:\User\15_REP\HYDROINFORMATIK-I
C:\User\15_REP\HYDROINFORMATIK-I>g++ EX04-data-types.cpp

C:\User\15_REP\HYDROINFORMATIK-I>a
EX04-data-types: Size of data types
Type      Number of bytes
-----
bool       1
char       1
short      2
int        4
long       4
float      4
double     8
long double 12

C:\User\15_REP\HYDROINFORMATIK-I>EX04-data-types.py
True
<class 'bool'>
28
3
<class 'int'>
28
3.1415
<class 'float'>
24
Hydroinformatik
<class 'str'>
2823712757680
64

C:\User\15_REP\HYDROINFORMATIK-I>
```

- cmd: Consolenanwendung starten
- cd: ins Verzeichnis mit den Übungen wechseln
- git: Aktualisieren der Übungen aus dem Repo
 - git fetch --all: Infos zu Änderungen
 - git pull: Änderungen aktualisieren
- g++ file.cpp: C++ compilieren
- a.exe: Programm ausführen
- file.py: Python-Programm ausführen

Voraussetzungen (Software-Installationen):

- git: Versionskontrolle
- MinGW: Compiler
- Python: Python

Datentypen: EX04-data-types.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     cout << "E04-data-types: Size of data types\n";
6     // system("pwd");
7     cout << "Type\tNumber of bytes\n";
8     cout << "-----\n";
9     cout << "bool\t\t" << sizeof(bool) << endl;
10    cout << "char\t\t" << sizeof(char) << "\n";
11    cout << "short\t\t" << sizeof(short) << endl;
12    cout << "int\t\t" << sizeof(int) << endl;
13    cout << "long\t\t" << sizeof(long) << endl;
14    cout << "float\t\t" << sizeof(float) << endl;
15    cout << "double\t\t" << sizeof(double) << endl;
16    cout << "long double\t" << sizeof(long double) << endl;
17    return 0;
18 }
```

Listing: Exercise: Data types

Datentypen: EX04a-data-types-bool.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     cout << "E04a-data-types: bool\n";
6     bool x = 17 > 3;
7     //std::cout << "Ist 17 groesser als 3? " << x << std::endl;
8     cout << "Ist 17 groesser als 3? " << x << std::endl;
9     return 0;
10 }
```

Listing: Exercise: Data types

```
1 using namespace std;
```

Listing: Namespaces

Datentypen: EX04b-data-types-ascii.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     char x = 'G';
6     std::cout << "Eingegebenes Zeichen: " << x << std::endl;
7     int y = x;
8     std::cout << "Der ASCII-Code fuer " << x << " ist " << y << std::endl;
9     return 0;
10 }
```

Listing: Exercise: Data types

```
1 using namespace std;
```

Listing: Namespaces

Datentypen: EX04-data-types.py (kommt in der nächsten Vorlesung)

```
1 #boolean
2 d = True
3 print(d)
4 type(d)
5 print(type(d))
6 print(d.__sizeof__())
7 #integer
8 a = 3
9 print(a)
10 type(a)
11 print(type(a))
12 print(a.__sizeof__())
13 #float
14 b = 3.1415
15 print(b)
16 type(b)
17 print(type(b))
18 print(b.__sizeof__())
19 #string
20 c = 'Hydroinformatik'
21 print(c)
22 type(c)
23 print(type(c))
24 id(c)
25 print(id(c))
26 print(c.__sizeof__())
```

Listing: Exercise: Data types

EX05: Klassen

Deklaration einer Klasse: student.h

```
1  #include <string>
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5  class CStudent
6  {
7      private:
8          int year;
9          long bank_account;
10     public:
11         string name_first;
12         string name_last;
13         string course;
14         string birthday;
15         string nation;
16         string gender;
17         long id;
18         string email;
19         string city;
20         string street;
21         int bmp_id;
22     public:
23         CStudent(); // constructor
24         ~CStudent(); // destructor
25         ios::pos_type Read(ifstream&);
26 };
```

Listing: Exercise: Class declaration

Definition einer Klasse: student.cpp

```
1  #include "student.h"
2
3  CStudent::CStudent()
4  {
5      name_first = "Maxi";
6      name_last  = "Musterfrau";
7  }
8
9  CStudent::~~CStudent()
10 {
11 }
12
13 /*****
14 STD read function
15 04/2009 OK Implementation
16 *****/
17 ios::pos_type CStudent::Read(istream& input_file)
18 {
19 ...
20 }
```

Listing: Exercise: Class definition

Instanzen einer Klasse: EX05a

```
1 #include "student.h"
2 #include <iostream>
3 using namespace std;
4 int main()
5 {
6     cout << "E05-classes: Klassen\n";
7     CStudent *m_std_cpp; // pointer to an instance
8     cout << "Instances of classes" << endl;
9     cout << "What have we created?\t : " << m_std << endl;
10    cout << "What size has it?\t : " << sizeof(m_std) << endl;
11    return 0;
12 }
```

Listing: Exercise: Creating instances

Zeiger oder direkter Zugriff auf eine Instanz einer Klasse: EX05b

```
1 #include <iostream>
2 using namespace std;
3 #include "student.h"
4 int main()
5 {
6     cout << "EX05b: Access to instances" << endl;
7     CStudent *m_std = new CStudent(); // instance
8     cout << "CStudent *m_std = new CStudent();" << endl;
9     cout << "*m_std: What have we created?\t\t : *m_std" << endl;
10    cout << "*m_std: What size has it?\t\t : " << sizeof(*m_std) << endl;
11    cout << "&m_std: What have we created?\t\t : " << &m_std << endl;
12    cout << "&m_std: What size has it?\t\t : " << sizeof(&m_std) << endl;
13    cout << " m_std: What have we created?\t\t : " << m_std << endl;
14    cout << " m_std: What size has it?\t\t : " << sizeof(m_std) << endl;
15    //delete &m_std;
16    return 0;
17 }
```

Listing: Exercise: Accessing instances

Übungen: Übersicht

- ▶ **EX05a: Instanzen einer Klasse - Kreieren**
- ▶ **EX05b: Instanzen einer Klasse - Speicher und Zugriff**
- ▶ EX05c: Eigenschaften einer Klasse
- ▶ EX05d: Weitere Eigenschaften einer Klasse
- ▶ EX05e: Aktion "Gummibärchen" (Datenbank häcken) ...

EX08: Komplexere OO Programme

Lesen und Sortieren einer Datenbank: EX08b

```
1  #include <iostream>    // for using cout
2  #include <fstream>     // for using ifstream / ofstream
3  #include <string>       // for using string
4  #include <vector>       // for using vectors
5  #include <list>         // for using lists
6  #include "student.h"   // for using CStudents
7  bool STDRead(ifstream&);
8  void STDSort();
9  vector<CStudent*>std_vector;
10
11 int main()
12 {
13     cout << "EX08b data base: an object-oriented DB read function" << endl;
14     //-- 1 File handling
15     ifstream input_file; // ifstream instance
16     input_file.open("data_base.txt");
17     if(!input_file.good())           // Check is file existing
18     {
19         cout << "! Error: input file could not be opened" << endl;
20         return 0;
21     }
22     //-- 2 Read data base
23     STDRead(input_file);
24     //-- 3 Sort data base
25     STDSort();
26     return 0;
27 }
```

Listing: Exercise: Accessing instances