Hydroinformatik - SoSe 2026

UW-BHW-414-05: Objekt-Orientierte Programmierung

Prof. Dr.-Ing. habil. Olaf Kolditz

[1]Helmholtz Centre for Environmental Research – UFZ, Leipzig

[2]Technische Universität Dresden – TUD, Dresden

[3]Center for Advanced Water Research – CAWR

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

Dresden, 08.05.2026

Technische Universität Dresden

Zeitplan: Hydroinformatik I+II

Sommersemester 2026: Stand: 06.04.2026

Nr.	KW	Datum	ID	Thema
01+02	16	17.04.2026	UW-BHW-414-01/02	Einführung in die Vorlesung, Umweltinformatik
03	16	17.04.2026	UW-BHW-414-03	Werkzeuge, Hello World (in C++)
05	17	24.04.2026	UW-BHW-414-04	Selbststudium: Software-Installationen
07	19	08.05.2026	UW-BHW-414-D	Objekt-Orientierte Programmierung: C++, Klassen
09	20	15.05.2026	UW-BHW-414-E	Python
11	21	22.05.2026	UW-BHW-414-F	Modellierung, Digitalisierung, Wasser 4.0
00	22	29.05.2026		Vorlesungsfreie Woche
13	23	05.06.2026	UW-BHW-414-G	KI, Maschinelles Lernen, Neuronale Netzwerke
15	24	12.06.2026	UW-BHW-414-H	Kontinuumsmechanik, Hydromechanik
17	25	19.06.2026	UW-BHW-414-I	Differentialgleichungen, Näherungsverfahren
19	26	26.06.2026	UW-BHW-414-J	Finite-Differenzen, explizite Verfahren
21	27	03.07.2026	UW-BHW-414-K	Finite-Differenzen, implizite Verfahren
23	28	10.07.2026	UW-BHW-414-L	Gerinnehydraulik, Grundwasserhydraulik
25	29	17.07.2026	UW-BHW-414-M	Grundwasserhydraulik
27	30	24.07.2026	UW-BHW-414-N	Zusammenfassung, Klausurvorbereitung

Technische Universität Dresden

Fahrplan für heute ...

UW-BHW-414-05: Objekt-Orientierte Programmierung

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

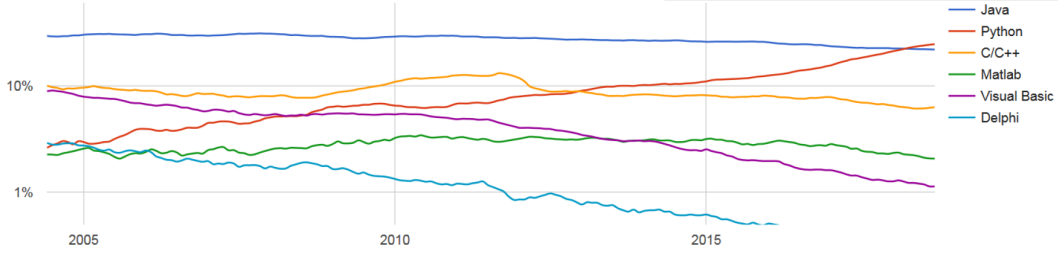
Technische Universität Dresden

Objekt-Orientierung

Warum C++

- Universalsprache
- objekt-orientiert
- performant
- sehr weit verbreitet
- seit über 40 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 %	-0.6 %
10		Matlab	2.09 %	-0.3 %



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 bestimme 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

Technische Universität Dresden

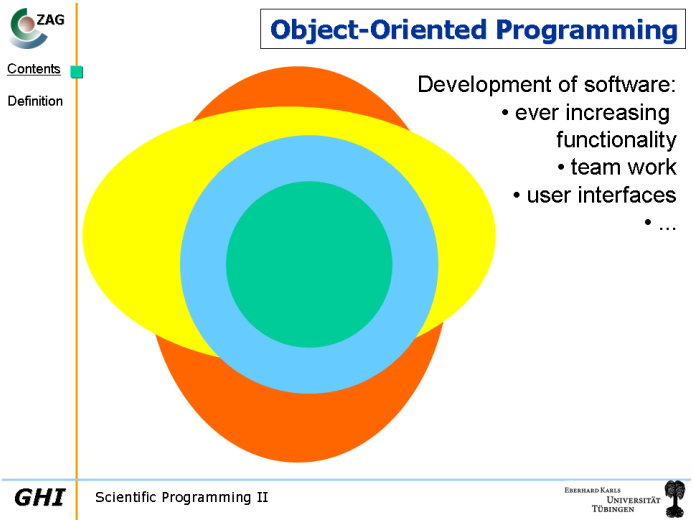
Objekt-Orientierte Programmierung

- Fokus auf Methoden und 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

$\Rightarrow$ Erleichtert das Erstellen von komplexen Programmen

Technische Universität Dresden

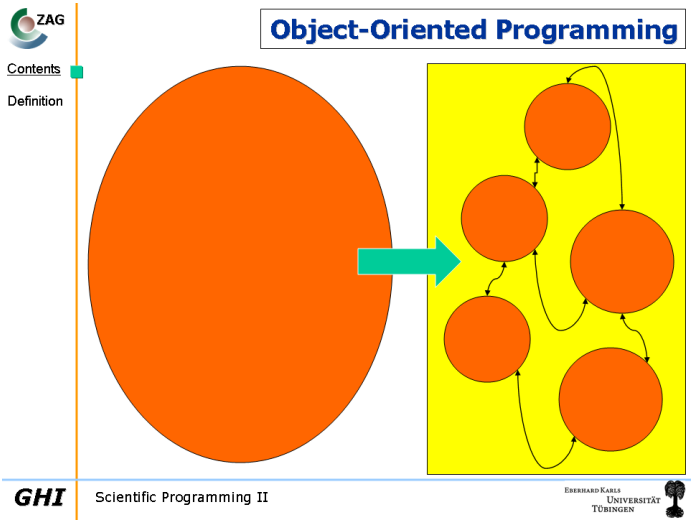
Objekt-Orientierung: C-Version



- Prozedurales Programmieren
- ...

Technische Universität Dresden

Objekt-Orientierung: C++ Version



- Zerlegen in logische Objekte
- Bilden von ”Klassen”
- schmale Schnittstellen
- ...

Technische Universität Dresden

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? ⇒ Speichermanagement
  2. Welche Art von Information ist gespeichert? ⇒ Datentyp
  3. Welche Information ist gespeichert? ⇒ 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
```

Technische Universität Dresden

Vorbemerkung:

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

1 byte = | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | = $2^6+2^3+2^1+2^0 = 75$

- 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
short	2 byte	[-32767, 32768], d.h. -2^{15}-1 bis 2^{15}
int	2-4 byte	Min. so groß wie *short*, höchstens so groß wie *long*
long	min. 4 byte	[-2147483647, 2147483648], d.h. -2^{31}-1 bis 2^{31}

Anzahl der bytes ist systemabhängig

Technische Universität Dresden

Gleitkommazahlen (Reelle Zahlen)

Datentypen

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

z.B.

1741.67	1.75167E+3	$1.75167 * 10^3$
0.0833	8.33E-2	$8.33 * 10^{-2}$
-25.876	-2.5876E+1	$-2.5866 * 10^1$

- 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
float	4 byte	±3.4E+38, 6 Stellen Genauigkeit
double	8 byte	±1.7E+308, 15 Stellen Genauigkeit

Technische Universität Dresden

Übungen:

- EX02: Data-Types

Klassen

Klassen::Konzept

Fig.: 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.

Technische Universität Dresden

Klassen::Daten-Abstraktion

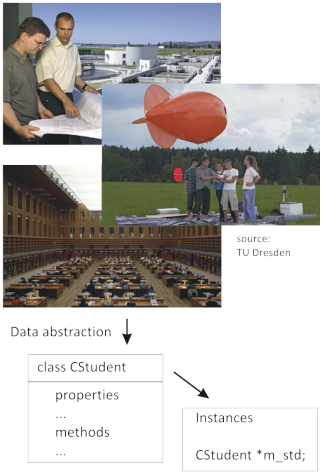


Fig.: Das Klassen-Konzept - `CStudent`

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

Technische Universität Dresden

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

Technische Universität Dresden

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

Technische Universität Dresden

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

Technische Universität Dresden

Klasse::Instanzen

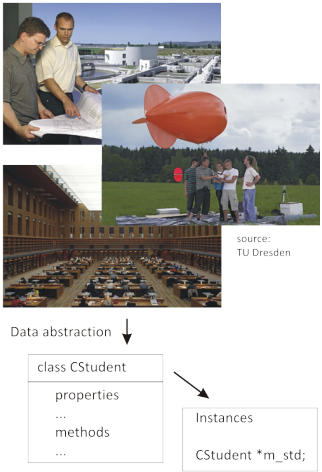


Fig.: Das Klassen-Konzept - `CStudent`

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:

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

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.

Technische Universität Dresden

Objekt-Orientierung: Internet-Quellen

Links:

https://www.youtube.com/watch?v=JBjjnqG0BP8
http://www.stroustrup.com/
https://en.wikipedia.org/wiki/Bjarne_Stroustrup

Technische Universität Dresden

Beispiele:

- Datenlogger
- OpenGeoSys

Technische Universität Dresden

Objekt-Orientierte Programmierung: Beispiel

Datenlogger

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.

Technische Universität Dresden

Objekt-Orientierte Programmierung: Beispiel

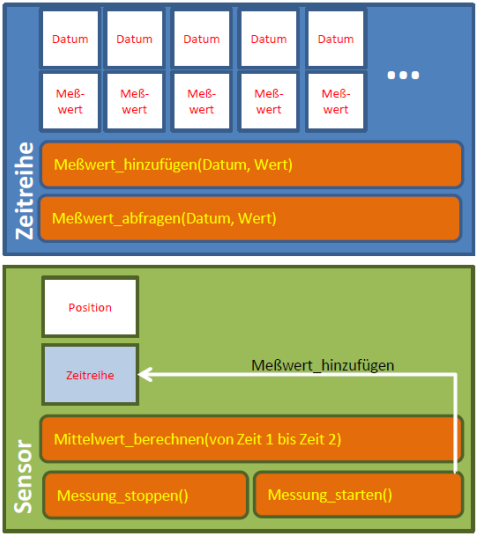
Datenlogger

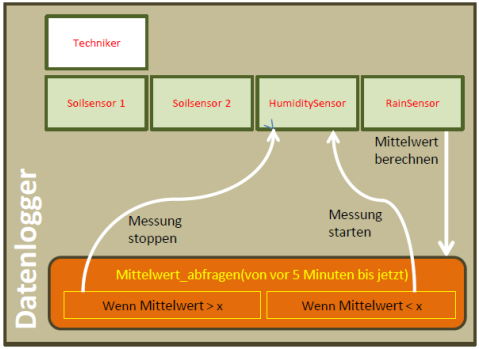
- Definition einer Klasse „Datenlogger“
  - An jeden Datenlogger sind ein oder mehrere Sensoren angeschlossen.
  - Jedem Datenlogger ist ein verantwortlicher Techniker zugeordnet
  - Wenn es zu stark regnet, soll keine Luftfeuchtigkeit gemessen werden, weil die Messwerte dann verfälscht sind
- Definition 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

Technische Universität Dresden

Objekt-Orientierte Programmierung: Beispiel

Datenlogger



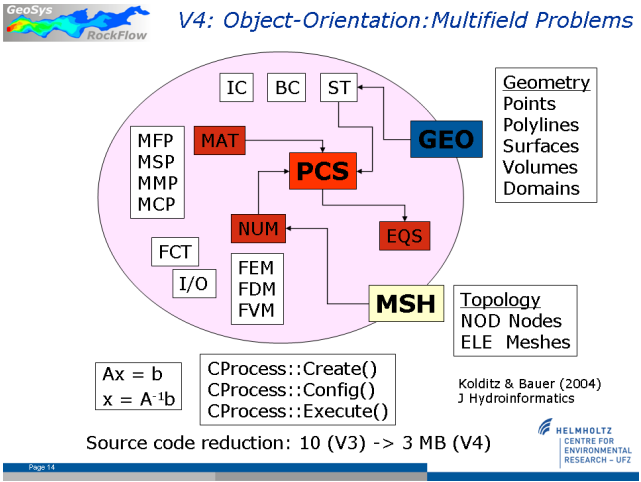


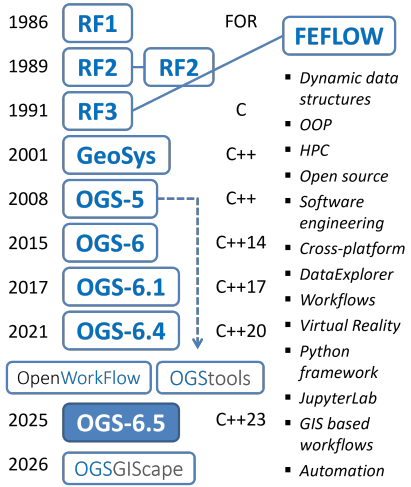
Kapselung: Jede Klasse enthält genau die Daten und Funktionalität, die sie benötigt

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

Technische Universität Dresden

Objekt-Orientierung - OpenGeoSys (OGS)





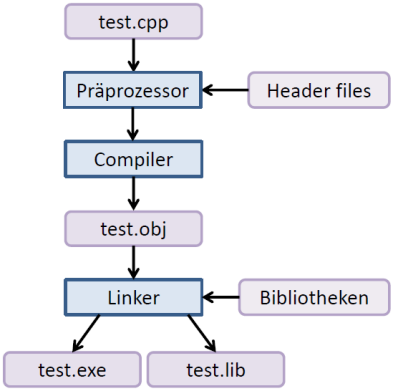
Technische Universität Dresden

Übungen:

- EX01: ”Hello World”
- EX02: Data Types

Technische Universität Dresden

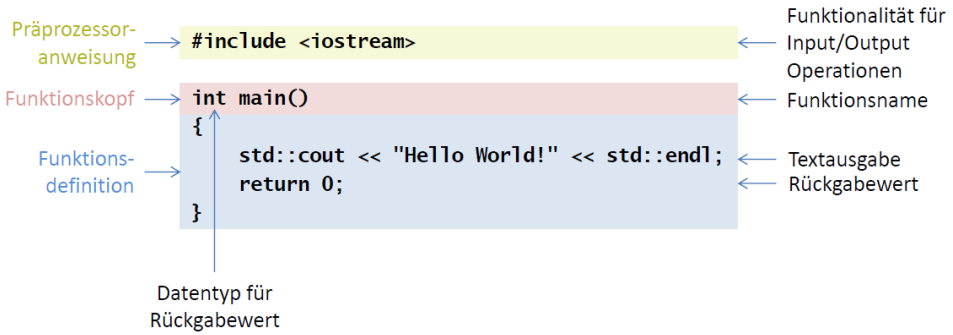
Compilieren und Linken



- 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 maschienenspezifischen 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.

Technische Universität Dresden

Ein erstes Beispielprogramm



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

Dateinamen

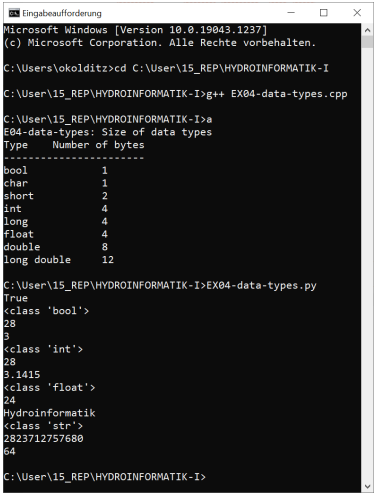
- Die Datei, in der main() definiert ist, ist der Programmeinstiegspunkt [Programmname].cpp
- Bei größeren Projekten oder der Verwendung von Bibliotheken, werden weitere Dateien mittels #include eingebunden
  - Systembibliotheken: #include <iostream>
  - Benutzerdefiniert: #include "hydro.h"
- Alle Programmteile 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

Technische Universität Dresden

Übungen:

- EX02: Data-Types

Datentypen: EX02-data-types



- `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

Technische Universität Dresden

Datentypen: EX02-data-types.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
  cout << "E04-data-types: Size of data types\n";
//  system("pwd");
  cout << "Type\tNumber of bytes\n";
  cout << "-----------------------\n";
  cout << "bool\t\t" << sizeof(bool) << endl;
  cout << "char\t\t" << sizeof(char) << "\n";
  cout << "short\t\t" << sizeof(short) << endl;
  cout << "int\t\t" << sizeof(int) << endl;
  cout << "long\t\t" << sizeof(long) << endl;
  cout << "float\t\t" << sizeof(float) << endl;
  cout << "double\t\t" << sizeof(double) << endl;
  cout << "long double\t" << sizeof(long double) << endl;
  return 0;
}
```

Listing: Exercise: Data types

Technische Universität Dresden

Datentypen: EX02a-data-types-bool.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
  cout << "E04a-data-types: bool\n";
  bool x = 17 > 3;
  //std::cout << "Ist 17 groesser als 3? " << x << std::endl;
  cout << "Ist 17 groesser als 3? " << x << std::endl;
  return 0;
}
```

Listing: Exercise: Data types

```cpp
using namespace std;
```

Listing: Namesspaces

Technische Universität Dresden

Datentypen: EX02b-data-types-ascii.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
  char x = 'G';
  std::cout << "Eingegebenes Zeichen: " << x << std::endl;
  int y = x;
  std::cout << "Der ASCII-Code fuer " << x << " ist " << y << std::endl;
  return 0;
}
```

Listing: Exercise: Data types

```cpp
using namespace std;
```

Listing: Namesspaces

Technische Universität Dresden

Datentypen: EX02-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

Technische Universität Dresden

Übungen:

- EX03: 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

Technische Universität Dresden

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(ifstream& input_file)
18 {
19 ...
20 }
```

Listing: Exercise: Class definition

Technische Universität Dresden

Instanzen einer Klasse: EX03

```cpp
#include <iostream>
using namespace std;
#include "student.h"
int main()
{
  cout << "EX03: Access to instances" << endl;
  CStudent *m_std = new CStudent(); // instance
  cout << "CStudent *m_std = new CStudent();" << endl;
  cout << "*m_std: What have we created?\t\t : *m_std" << endl;
  cout << "*m_std: What size has it?\t\t : " << sizeof(*m_std) << endl;
  cout << "&m_std: What have we created?\t\t : " << &m_std << endl;
  cout << "&m_std: What size has it?\t\t : " << sizeof(&m_std) << endl;
  cout << " m_std: What have we created?\t\t : " << m_std << endl;
  cout << " m_std: What size has it?\t\t : " << sizeof(m_std) << endl;
  //delete &m_std;
  return 0;
}
```

Listing: Exercise: Accessing instances

Technische Universität Dresden

Übungen:

- EX04: Komplexere OO Programme
- … Finite-Differenzen-Methode (FDM)

Technische Universität Dresden

Lesen und Sortieren einer Datenbank: EX04

```cpp
#include <iostream>  // for using cout
#include <fstream>   // for using ifstream / ofstream
#include <string>    // for using string
#include <vector>    // for using vectors
#include <list>      // for using lists
#include "student.h" // for using CStudents
bool STDRead(ifstream&);
void STDSort();
vector<CStudent*>std_vector;

int main()
{
  cout << "EX04 data base: an object-oriented DB read function" << endl;
  //--- 1 File handling
  ifstream input_file; // ifstream instance
  input_file.open("data_base.txt");
  if(!input_file.good())               // Check is file existing
  {
    cout << "! Error: input file could not be opened" << endl;
    return 0;
  }
  //--- 2 Read data base
  STDRead(input_file);
  //--- 3 Sort data base
  STDSort();
  return 0;
}
```

Listing: Exercise: Sorting Data Base

Technische Universität Dresden