Dennis M. Junger
jungerdm@htw-berlin.de

Vertiefung in der Programmierung

WS 2019/20

10.10.2019

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Agenda der 1. Vorlesung

Organisatorisches
Semesterplanung
Nachholtermin-Findung
Prüfungsmodalitäten
Weshalb C++
IDE Visual Studios
Einstieg

Konzept der Lehrveranstaltung

Lernziel laut Modulhandbuch:
„Die Studierenden erlernen neben der bisher vermittelten eine weitere Programmiersprache wie php oder C++ und vertiefen damit die bisher erworbenen Programmierkenntnisse. Dabei werden auch grundsätzlich neue Anwendungsgebete wie mobile Computing, Web-Programmierung oder Anwendungsprogrammierung von Ingenieurwissenschaftlichen Softwaresystemen erlernt."

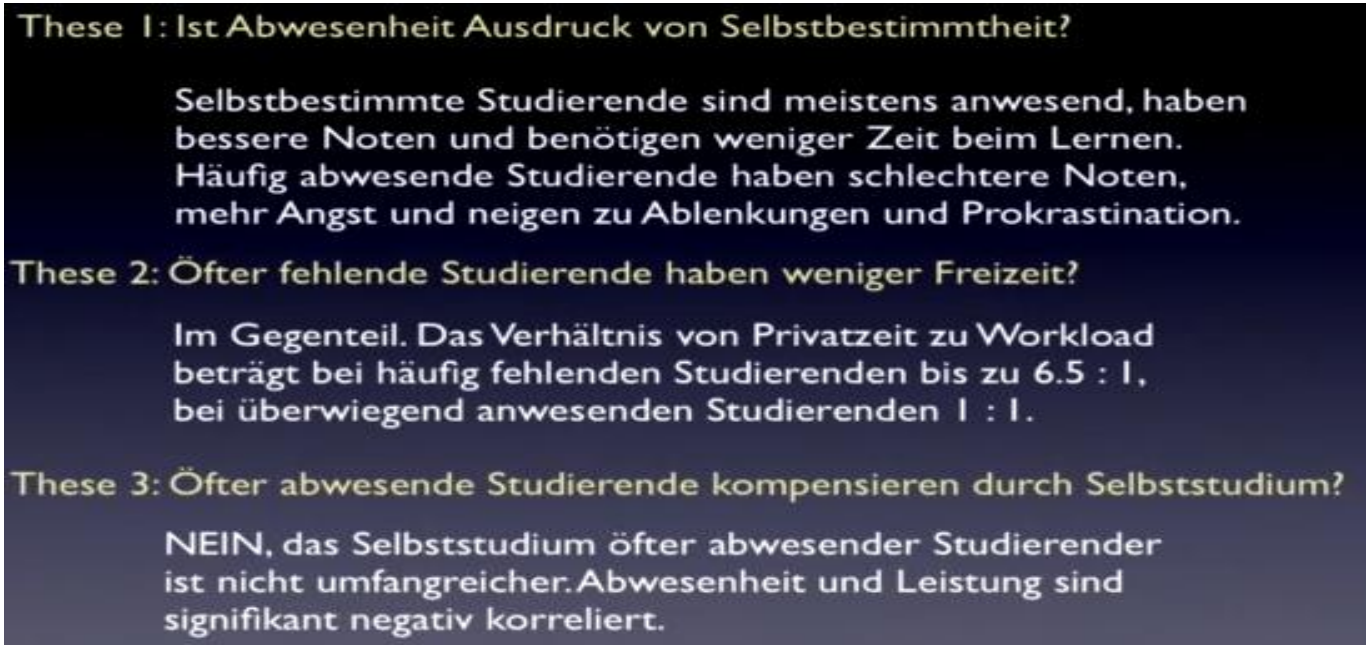
- Die Lehrveranstaltung setzt sich aus 3 Teilen zusammen:
  - Vorlesung
  - Präsentation praktischer Übungen
  - Entwicklung eines Abschlussprojektes

Konzept der Lehrveranstaltung

- Sie erlangen Kenntnisse in folgenden Richtungen:
  - C++ als industrieller Standard, Anwendungsgebiete und Vorteile
  - Aufbau von C++ Projekten
  - Verwenden von C++ in Programmierprojekten
- Verwenden der C Sprachen mit Microcontrollern
- Erstellen eines Microcontroller Projektes oder eine Desktopapplikation mit C/C++

Warum ist Anwesenheit wichtig?

Ergebnisse der ZEITLast-Studie von Metzger und Schulmeister (2010):



Vorlesungs- und Übungstermine:

- WS 2019/20: Insgesamt 15 Termine

Zulassung zum Abschlussprojekt:

- Erfolgreiche Teilnahme an Übungen – Kriterien:
  - Abgabe der Übungsaufgaben vor der Veranstaltung via moodle (gezippt)
  - Kulanz: 2 Übungsaufgaben

Prüfungsleistungen:

- Projekt zur Microcontroller-Entwicklung im Bereich der Ingenieurinformatik im Januar2018 (keine Überschneidung mit der Abgabe für das fachübergreifende Projekt)
- Festlegung des Projektes bis zum 0512.2019
- Präsentation und Abgabe der Projekte am 23.01.2020
- Bewertung im Verhältnis 20% (Vorstellung von mindestens zwei Laborübungen) zu 80% (Abschlussprojekt)

Zugriff auf Vorlesungsskripte:

- Die Vorlesungsskripte sind im Moodle verfügbar

Termine (1): Übersicht

- **Vorlesung: Donnerstag 17:30 bis 19:00 Uhr (FZ01)**
- **Übungen: Donnerstag 19:00 bis 20:30 Uhr (FZ01)**

- Beginn der Veranstaltungen am 10.10.2019

- Detaillierte Zeitplanung: Siehe nächste Seiten

Organisatorisches: Grober Fahrplan

Datum	Inhalt
1. Vorlesung	Organisation, Prüfungsform festlegen, Ausblick, Nachholtermin abstimmen, Einführung in C++, IDE, Entwicklungsumgebung, Aufbau eines Projektes C++, .h .cpp , Syntax, Typen, Eingabe/Ausgabe, using Namespaces, Präprozesordirektiven
2. Vorlesung	Einführung, Syntax, Typen, Bedingungen, Schleifen, Switch, Funktionen, Überladene Funktionen
3. Vorlesung	C-Array, STL-Array, STL-Vektor, Strukturen ,Unions, Typedef -/- Defines, Iteratoren, Operator Overloading
4. Vorlesung	Klassen, Attribute, Member, Zugriffsspezifizierer, Initialisierungsliste, Initialisierung im Funktions/Klassen-Prototyp, Enumerationen, Strukturierung von Code
5. Vorlesung	Statische Elemente, Static_Cast, Inline Funktionen,Pointer , Null-Pointer, Pointer Arithmetic, Pointer Correctness, Referenzen, Call By Reference, Call By Value, Const bei Pointern,
6. Vorlesung	Memory Management, Stack & Heap Problematik, Function Pointer, Exception Handling, List, Forward_List, Deque
7. Vorlesung	GUI - Qt
8. Vorlesung	Vererbung, Vererbung mit Zugriffspezifizierer, Virtual, Pure Virtual, Abstrakte Klassen, Virtual Destructor, Mehrfach Vererbung, Deadly Diamond, Friends, Weitere STL: Deque, Stack, Queue, Priority Queue, Set, Multiset, Pair, Map, Multimap, Container-Methoden, Container-Schaubild

Organisatorisches: Grober Fahrplan

Datum	Inhalt
9. Vorlesung	Final ,Templates Lizenzen in der Softwareentwicklung: Workshop Lizenzbestimmung für das Abschlussprojekt, Kurzpräsentation Lizenzen, Lizenzwahl Hardwarevorstellung –Hardwarewahl: Arduino, Raspberry Pi3, Softwareanwendung **Hausaufgabe**: Abschlussprojekt-Wahl, Team-Wahl, Hardwareliste, Präsentation des geplanten Projektes
10. Vorlesung 05.12.2020	**Präsentationen der Abschlussprojekt- Ideen** – Festlegung der Abschlussprojekte, Wahl der Entwicklerboards
11. Vorlesung	Bearbeitungszeit und OpenLab, Ausgabe der Hardware
12. Vorlesung	Bearbeitungszeit und OpenLab
13. Vorlesung	Bearbeitungszeit und OpenLab
14. Vorlesung	Bearbeitungszeit und OpenLab
15. Vorlesung	Bearbeitungszeit und OpenLab
16. Vorlesung 23.01.2020	**Abgabe der Projekte** Präsentationen der Projekte

Nachholtermin Vereinbaren

Ein Termin wird vorgeschoben, sodass wir uns in anderen Fächern auf die Klausuren vorbereiten können.

Der einmalige Extra-Termin wird Ende Oktober stattfinden.

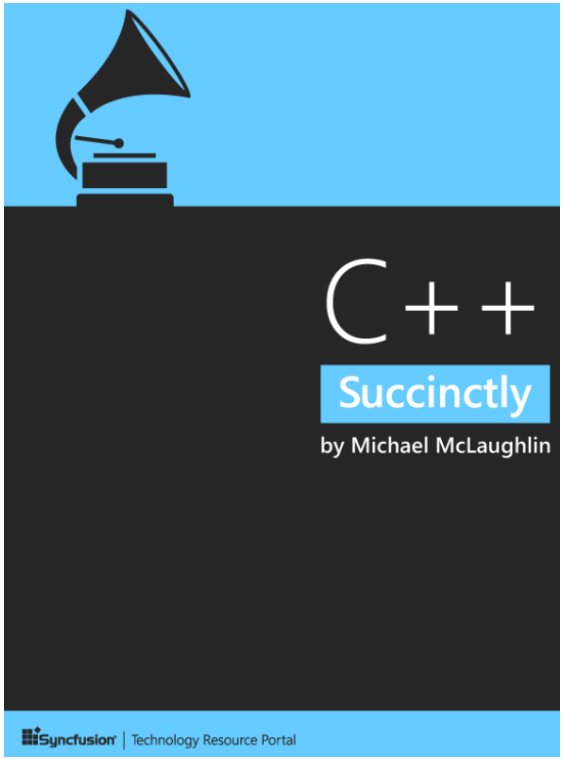
- **Montag Nachmittag?**
- **Mittwoch Nachmittag?**

Ist es erwünscht im ersten Monat mehrere Termine pro Woche stattfinden zu lassen?

2 Termine: Projektabgabe und Präsi am 16. Januar 2020

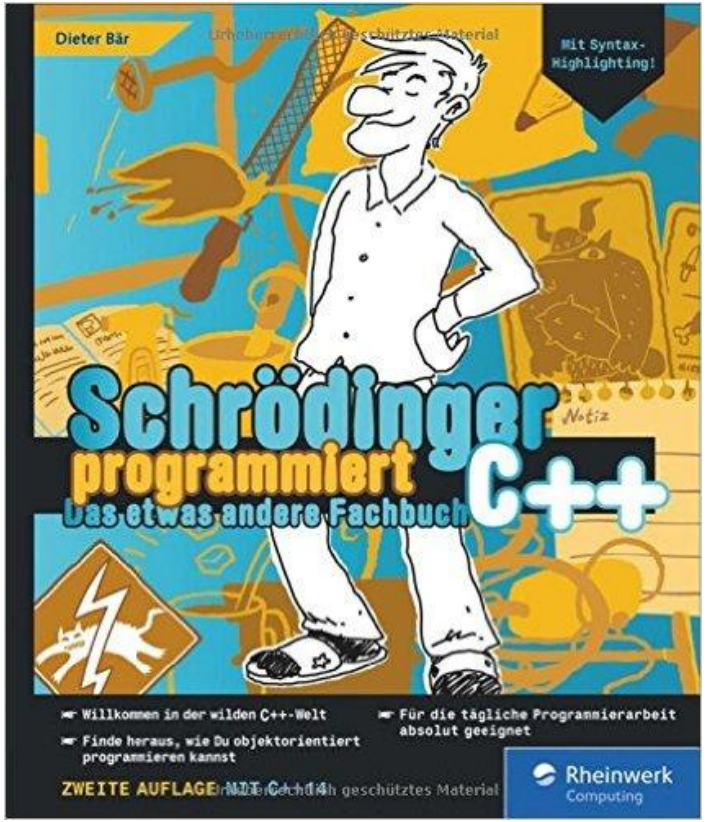
Literaturempfehlungen

- Michael McLaughlin: C++ Succinctly
- Free Download: *https://www.syncfusion.com/resources/techportal/details/ebooks/cplusplus*



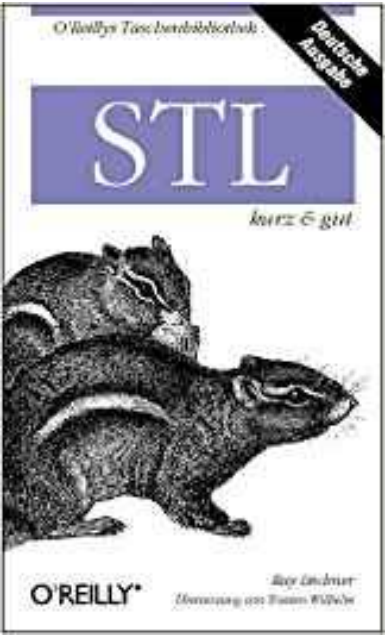
Literaturempfehlungen

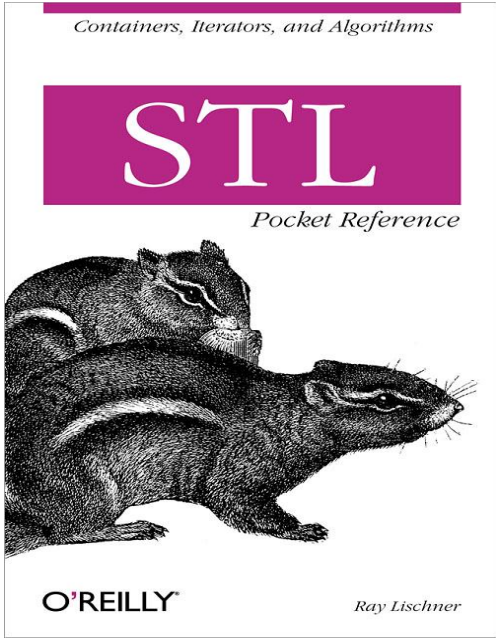
Rheinwerk Computing –
Schrödinger programmiert C++



Literaturempfehlungen

- O‘Reilly : Ray Lischner – STL Pocket Reference
- O‘Reilly : Ray Lischner – STL Kurz & Gut





Literaturempfehlungen

- https://www.codeproject.com/
- *https://stackoverflow.com/*
- *https://www.c-plusplus.net/forum/*
- *http://www.cplusplus.com*

Einführung

- C++ wurde 1979 von Bjarne Stroustrup bei AT&T als Erweiterung der Programmiersprache C entwickelt.
- Hohes Abstraktionsniveau
- Wird eingesetzt zur
  - Systemprogrammierung:
    - Betriebssysteme
    - Embedded Systeme
    - Virtuelle Maschinen
    - Treiber
    - Hat nach und nach den Platz von Assembler und C eingenommen

Einführung

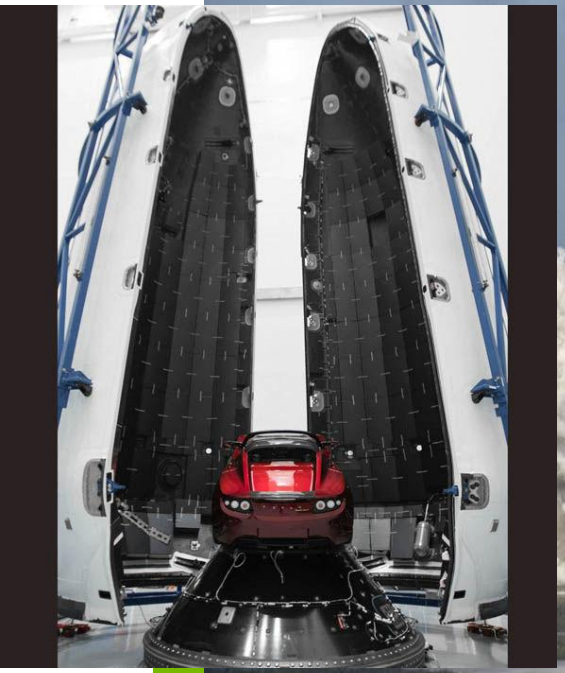
Bjarne Stroustrup: Why I Created C++:
https://www.youtube.com/watch?v=JBjjnqG0BP8

Einführung

https://giphy.com/gifs/java-iedFwEvN4OZvW

- Anwendungsprogrammierung
  - Erfüllt hohe Effizienzansprüche
  - Optimal um Programme mit strikten technischen Rahmenbedingungen zu realisieren
  - Kann (mit etwas Mehraufwand) für jedes Betriebssystem genutzt werden
  - Wird in den meisten Firmen die vor 2000 gegründet wurden noch verwendet
  - Wurde jedoch durch die einfachere Usability 2000 von C# und Java in der Anwendungsprogrammierung als Standard abgelöst

Falcon Heavy 9



SPACEX
Space Exploration Technologies

FALCON

TESLA

htw.

[-] **phil_jp1** 122 points 2 hours ago

What kind of operating systems you use in your navigation/propulsion control systems?

permalink report give gold **reply**

[-] **spacexdevtty** SpaceX [S] 224 points 2 hours ago

Dragon and Falcon 9 use a version of Linux.

permalink parent report give gold **reply**

[-] **ken27238** 366 points 2 hours ago

I'm imagining a Raspberry Pi taped to the inside of a Falcon 9.

permalink parent report give gold **reply**

[-] spacexdevtty SpaceX [S] 130 points 1 hour ago

For all software engineers (not space related) C is a great language to get started. It forces you to learn about how the CPU works, how memory works, etc., but high enough you're not writing assembly. Once you've mastered C take a look at C++.

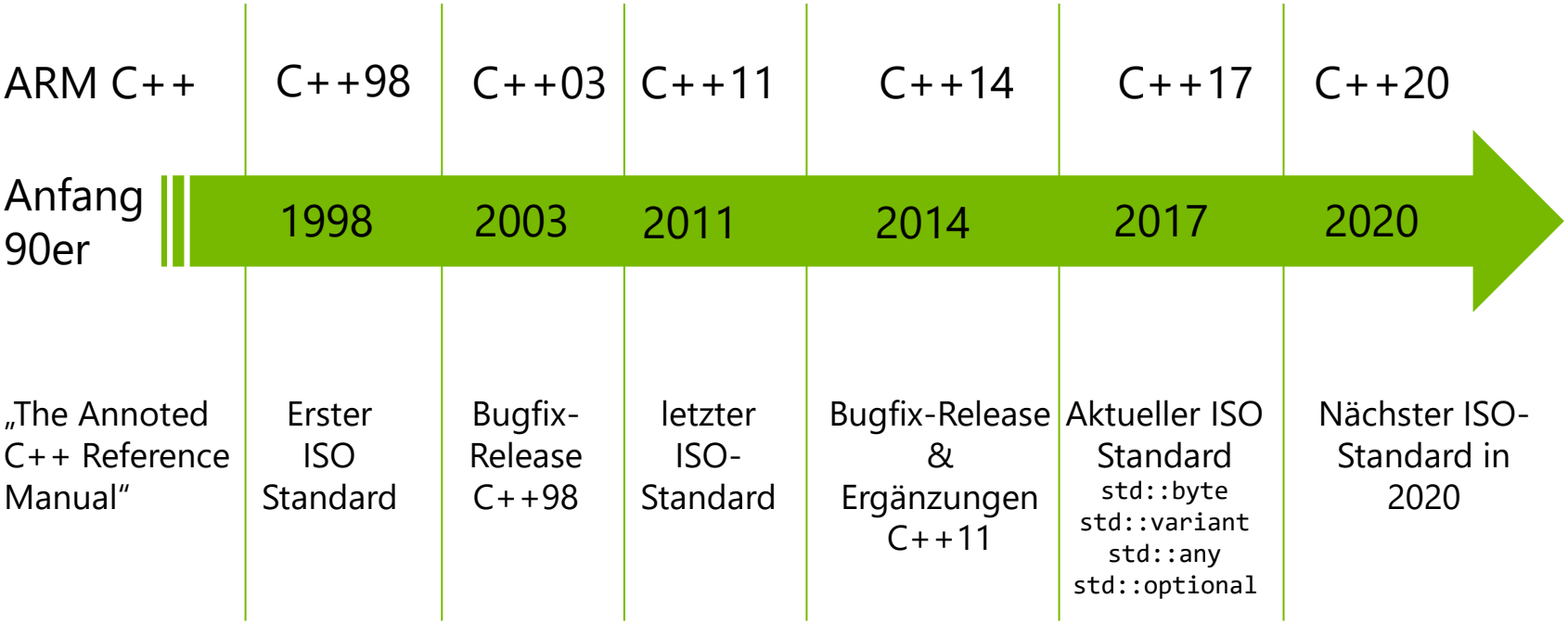
We have so much going on right now--one of the coolest project is probably Grasshopper. This is a development vehicle we're using to test the engineering necessary to not just send rockets up, but to bring them back--something that's never been done before. You can see some of those tests on YouTube http://www.youtube.com/watch?v=B4PEXLODw9c and we've got more tests coming up later this year.

We're also working on getting Dragon ready to take people to station. Right now Dragon just carries cargo, but it was always designed to take people. We're working with NASA to make the final mods necessary--basically seats, environments and pretty bad-ass launch escape system--and we're looking to start crew trials as early as 2015.

All of us came from a pretty diverse set of backgrounds, there's no one particular path we all followed to get here. Speaking for me personally the best experience I got in school was working on my own personal programming projects on the side. There's a lot of opportunity these days on the web and the app stores to complete a product and get your work out there to be seen.

[Spaceman Live]

Chronologie



Generelle Unterschiede von C++ zu C#

- Mit C++ kann man für ziemlich jedes Betriebssystem Programmieren
- Es gibt keinen Garbage-Collector: In C++ muss der Entwickler selbst den Arbeitsspeicher verwalten.
  - Vorteil: Es wird exakt der Speicher verwendet wie erwünscht
  - Es kann auf exakte Speicheradressen zugegriffen werden (Pointer)
  - Nachteil: Anfällig für Fehler durch den Entwickler
- Viele Bibliotheken müssen selbst ausgewählt und eingebunden werden ./. Dot Net
- Es muss sehr akribisch gearbeitet werden, bis zum vollständigen erlernen können Jahre vergehen

Bjarne Stroustrup: „In C++ it's harder to shoot yourself in the foot,
but when you do, you blow of your whole leg!"

Aufbau eines C++ Programmes

In C++ ist es vorgesehen, dass eine der Quellcode in .cpp und .h Dateien untergliedert wird.

•.h Dateien sind die Header Dateien, dienen beispielsweise als Schnittstellen zum einbinden ganzer Bibliotheken oder von Quelldateien

- In der .h Datei werden die Klassendefinitionen festgelegt und bekannt gemacht

•.cpp Dateien sind die Quell-Dateien, binden typischerweise ihre gleichnamige Header-Datei ein

Vom C++ Code zur Ausführbaren Datei

In folgenden Schritten wird ein C++ Programm gebaut:

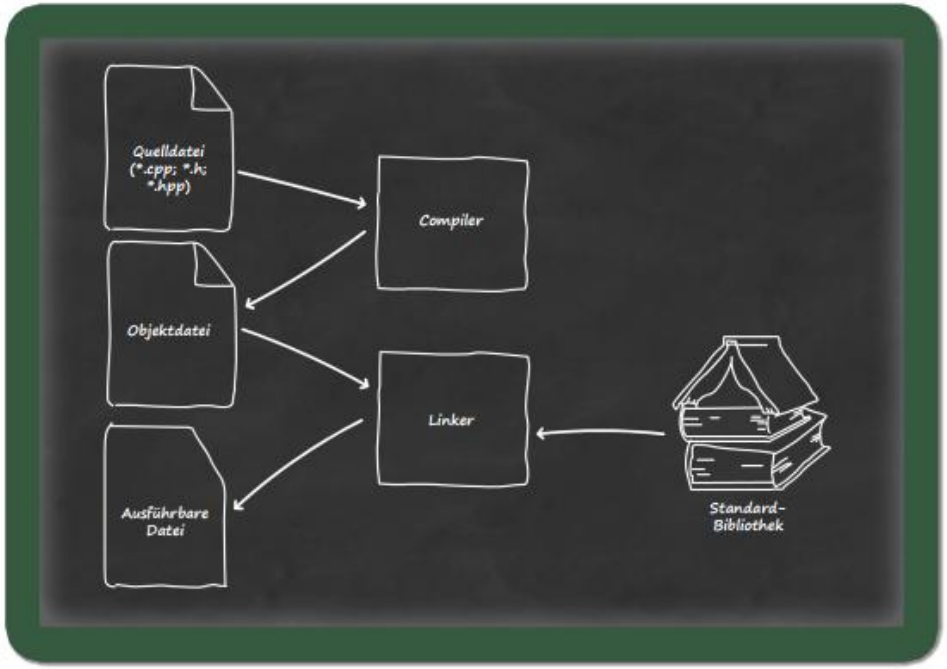
Quelldateien (*.cpp/*.h)-> Compiler

Objektdateien werden erzeugt

(*.obj / .*o)

Diese werden mit dem Linker an die Verweise (z.B. Bibliotheken geknüpft

->bau der .exe



Schrödinger Programmiert C++, S26

Vom C++ Code zur Ausführbaren Datei

Mehrere Methoden

- ASCII-Editor, manuelles verwenden von Compiler und Linker
- Verwenden einer IDE
  - z.B. Visual Studios unter Windows, komfortabel da z.B. Debugger, Projektverwaltung, Profiler, etc. schon integriert sind
  - z.B. Eclipse und NetBeans für alle Entwicklungssysteme

Vom C++ Code zur Ausführbaren Datei

- Gängige Compiler sind:
  - GCC-Paket (GNU Compiler Collection)
  - Ursprünglich von Linux-Systemen
    - Enthält unter anderem
      - gcc C-Compiler
      - G++ C++ Compiler
  - MinGW-Compiler (Commandozeilen portierung von GCC auf Windows)
  - Auf Mac wird das LLVM-Frontend namens Clang eingesetzt

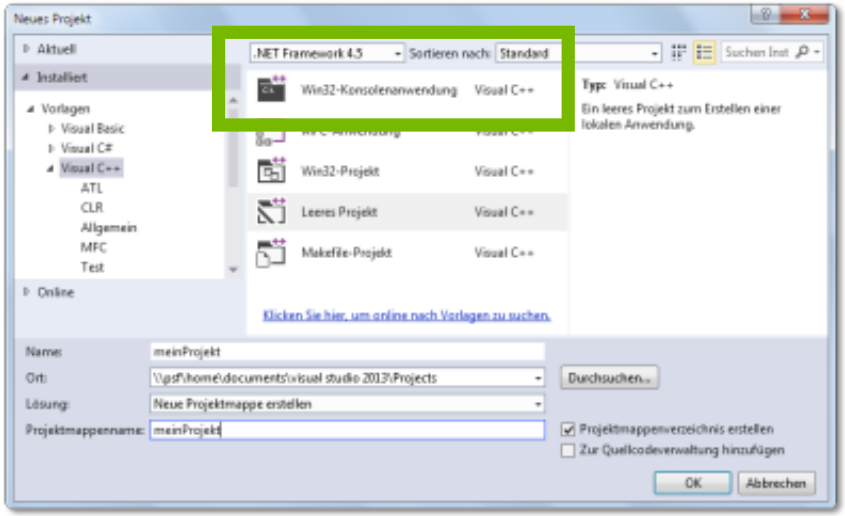
Übersicht der Compiler bzw IDEs für C++

Compiler	Kostenlos	Link	System	IDE
GCC	ja	http://gcc.gnu.org/	Win, Linux, Mac, Unix-like	nein
Clang	ja	http://clang.llvm.org/	Unix-like, Mac	nein
Microsoft Visual Studio	nein/ja	http://www.visualstudio.com/de-de/downloads/download-visual-studio-vs.aspx	Windows	ja
Microsoft Visual Studio Community	ja	http://www.visualstudio.com/de-de/downloads/download-visual-studio-vs.aspx	Windows	ja
Microsoft Visual Studio Express	ja	http://www.visualstudio.com/de-de/downloads/download-visual-studio-vs.aspx	Windows	ja
NetBeans	ja	http://netbeans.org/	plattformunabhängig	ja
Eclipse	ja	http://www.eclipse.org/	plattformunabhängig	ja
Qt Creator IDE	ja	http://www.qt.io/download/	Win, Linux, Mac	ja
Code::Blocks	ja	http://www.codeblocks.org/	Win, Linux, Mac	ja
C++Builder XE2	nein	http://www.embarcadero.com/products/cbuilder	Win, Mac	ja
KDevelop	ja	http://kdevelop.org/	Linux/Unix-like, Mac, Win	ja
Anjuta	ja	http://www.anjuta.org/	Linux, BSD	ja
Xcode	nein	http://developer.apple.com/technologies/tools/	Mac OS X	ja
MinGW	ja	http://www.mingw.org/	Windows	nein
Orwell Dev-C++	ja	http://sourceforge.net/projects/orwelldevcpp/	Windows	ja
Intel-C++	nein	http://software.intel.com/en-us/articles/intel-compilers/	Linux, Win, Mac	nein

Schrödinger Programmiert C++, S37

Vom C++ Code zur Ausführbaren Datei

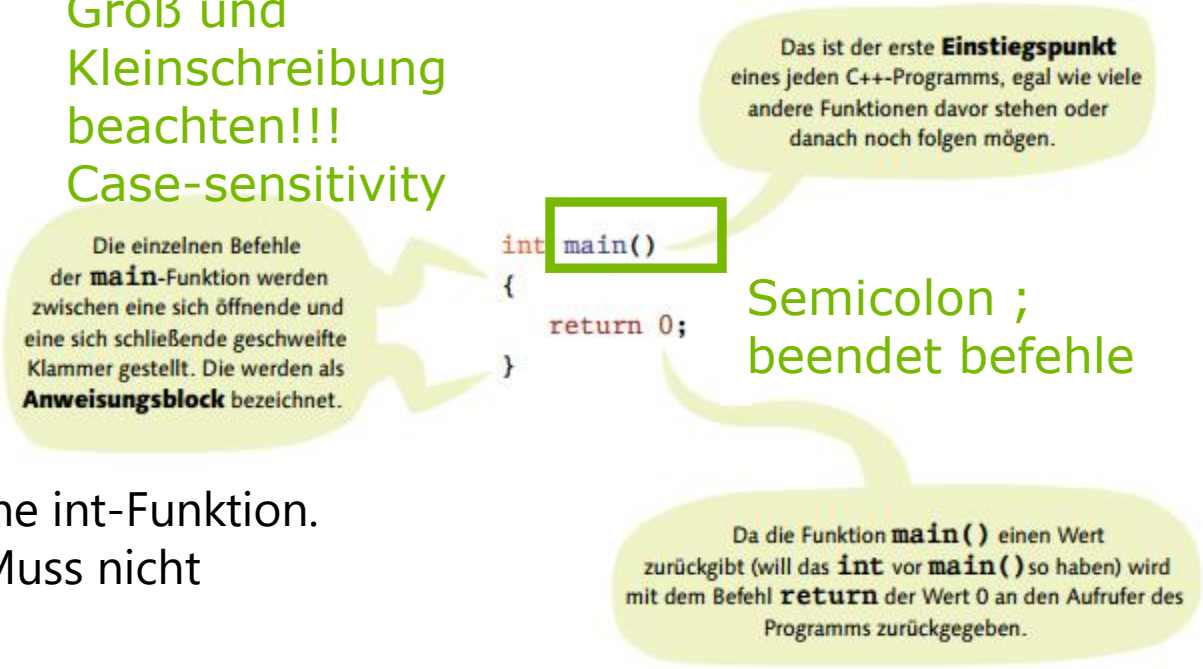
Erstellen einer Konsolenanwendung mithilfe von Visual Studios



Start eines C++ Programmes

Auch in C++ ist die erste vom Linker aufgerufene Funktion die main() Funktion

Groß und Kleinschreibung beachten!!! Case-sensitivity



Die main ist standarisiert eine int-Funktion. Standard Returnwert ist 0. Muss nicht explizit „returniert“ werden

Schrödinger Programmiert C++, S41

Includes in C++

Bibliotheken werden mit

`# include <abc>` oder

`#include “abc.h“` eingebunden (Header-Datei aus eigenem Projekt)

Präprozessorbefehle in C++

Aufgaben:

`#include “xxxx“` – Einbinden von Schnittstellen

`#include <yyyy>` -Einbinden von Systembibliotheken

Sicherstellen das eine Schnittstelle nur einmal eingebunden ist:

```
#ifndef MEINDEFINE_H_    //wenn MEINDEFINE_H_ noch nicht definiert
#define MEINDEFINE_H_    //definiere ich sie hier und
//hier code-block oder das #include
#endif /*MEINDEFINE_H_*/ //ende des blocks
```

HeaderGuards

Präprozessorbefehle in C++

Dies kann auch mit folgendem Befehl sichergestellt werden
#pragma once

Ein erstes Programm

```
#include <iostream>
int main()
{
std::cout << “Hello World“ <<std::endl;
}
```

Ein erstes Programm

```
#include <iostream>
int main( int argc, char* argv[] )
{
std::cout << “Hello World“ <<std::endl;
}
```

Kommentieren in C++

```
// ich bin ein Kommentar

/* ich bin
 ein mehrzeiliges
Kommentar
*/
```

Sind Grundlage für automatische Dokumentations-Tools

Bei C++ ist Doxygen weit verbreitet

Namespaces in C++

Mithilfe der Direktive kann „using namespace“ kann ein Namensraum zur Verfügung gestellt werden. Dies erleichtert das folgende Verwenden von beispielsweise in Bibliotheken eingebundenen Namespaces

Ein erstes Programm mit Namespaces

```
#include <iostream>
using namespace std;
int main()
{
std::cout << “Hello World“ <<std::endl;
cout << “Hello World“ << ‘/n‘;

}
```

Consolenfenster verschwindet: Abhilfe

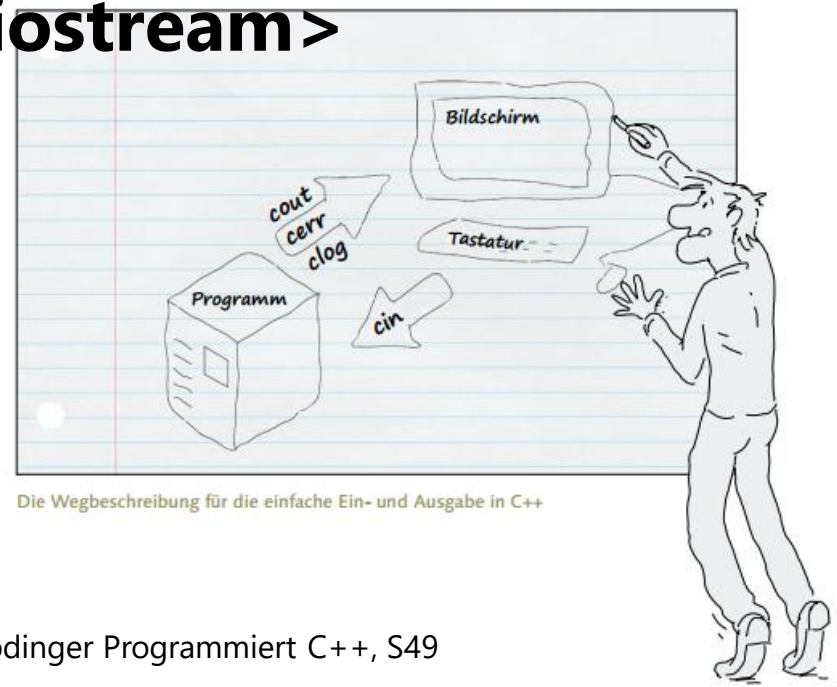
```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
cout << “Hello World“ <<endl;
system(“PAUSE“);
}
```

•Oder Breakpoint im Debug-Modus

Ein- & Ausgabe mit<iostream>

Objekt	Bedeutung	Wohin/Woher
cout	Standardausgabe	Bildschirm
cerr	Standardfehlerausgabe	Bildschirm
clog	Standardfehlerausgabe (gepuffert)	Bildschirm
cin	Standardeingabe	Tastatur

Gepuffert



Die Wegbeschreibung für die einfache Ein- und Ausgabe in C++

Schrödinger Programmiert C++, S49

Datentypen in C++ : Überblick

Typ	Speicher	Wertebereich
`bool`	1 Byte	0 oder 1 bzw. `false` oder `true`
`char`	1 Byte	-128 bis +127 bzw. 0 bis 255
`signed char`	1 Byte	-128 bis +127
`unsigned char`	1 Byte	0 bis 255
`wchar_t`	2 oder 4 Bytes	abhängig von der Implementierung
`short`	2 Bytes	-32.768 bis +32.767
`unsigned short`	2 Bytes	0 bis 65.535
`int`	4 Bytes	-2.147.483.648 bis +2.147.483.647
`unsigned int`	4 Bytes	0 bis 4.294.967.295
`long`	4 oder 8 Bytes	wie `int` oder -9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807
`unsigned long`	4 oder 8 Bytes	wie `unsigned int` oder 0 bis 18.446.744.073.709.551.615

Schrödinger Programmiert C++ S.83

Datentypen in C++ : Teuerster Softwarefehler der Geschichte: Erstflug Ariane 5

Die Ariane 5 startete am 4. Juni 1996 zu ihrem Erstflug V88 mit den vier Cluster-Satelliten als Nutzlast. Nach 37 Sekunden stellte sich die Rakete plötzlich quer, brach durch die Luftkräfte auseinander und sprengte sich selbst

370 Millionen US-Dollar

Code wurde von Ariane 4 zu Ariane 5 kopiert

16 Bit signed Integer (-32768 bis 32768) wurden „Overflowd" (>32,767)

Datentyp-Overflow ohne gescheitem Exception Handling

Absturz der Steuerungssoftware->Selbstzerstörung

https://www.youtube.com/watch?v=gp_D8r-2hwk

https://www.raumfahrer.net/news/images/va207120705msglaunch.jpg

Built_in Literale in C++

Bezeichnung	Literal	Code-Bsp	Ausgabe
char	‘‘	‘a‘	a
char16_t	u‘‘	u‘a‘	a
char32_t	U‘‘	U‘a‘	a
wchar_t	L‘‘	L‘a‘	a
int	KEINES	2017	2017
octal	0	03741	2017
hexadecimal	0x	0x7E1	2017
Binär	0b	0b01111101 1011	2017

Built_in Literale in C++

Bezeichnung	Literal	Code-Bsp	Ausgabe
long	L	2017L	2017
unsigned long	UL	2017UL	2017
double	.	1.234	1.234
float	. F	1.234F	1.234
long double	. L	1.234L	1.234
exponential	e+	1.234e+2	123.4
Longlong	LL	1.234LL	1.234

Built_in Literale in C++ : Bool & un-signed

Boolean-Variablen (**bool**) werden mit den Werten **true** oder **false** belegt

Dies kann auch mit den Zahlen n (true) und 0(false) geschehen, wobei n jede natürliche Zahl (int) sein kann.

In C wird true und false : TRUE und FALSE geschrieben, dies ist jedoch ein define und in C++ zu vermeiden.

Jeder Integer wert der nicht näher definiert wird ist ein signed Integer. D.h. er kann mit positive und negative Werte belegt werden.

Wenn eine Integer Variable lediglich POSITIVE Werte abbilden soll wird mittels des Literals **unsigned** ein Vorzeichen ausgeschlossen. Nebenbei verdoppelt sich dadurch auch die Range

Limits und Speicherverbrauch

Mithilfe der <limits> Bibliothek können die Grenzen von Datentypen abgefragt werden:

numeric_limits<TYP>::max()		liefert den maximalen Wert von TYP
	::min()	kleinster Wert
	::digits	Anzahl der Bits
	::is_signed	bool Ausgabe: true= signed

sizeof(TYP) Liefert den Speicherverbrauch in Bytes

Built_in Literale in C++

Abfrage von Literalen mit Bibliothek typinfo

```
#include <typeinfo>

...
```

Befehl	Ausgabe	Typ
typeid(1.234).name()	d	double
typeid(1.234F).name()	f	Float
typeid(1.234L).name()	e	Extended (LongDouble)
…	…	…

String & Zeichenliterale

```
#include <iostream>

using namespace std;


int main()

{

  cout << "Der Manipulator" << endl; *1

  cout << "Das Zeichenliteral" << '\n'; *2

  cout << "Das Stringliteral" << "\n"; *3

  cout << "Oder gleich im Stringliteral\n"; *4

  return 0;

}
```

*1 Der Manipulator **endl** stellt die langsamste Version da, weil noch eine **Extra-Synchronisation** ausgeführt wird. Und zwar werden hier alle sich noch im Puffer befindlichen Daten sofort an die Ausgabe geschickt.

*2 Die wohl **sparsamste Lösung**, weil nur ein einzelnes Zeichen verwendet wird.

*3 Hier wird neben dem Zeilenendezeichen noch ein zusätzliches (aber nicht sichtbares) **Terminierungszeichen** verwendet, welches das Ende des Strings kennzeichnet (Hasta la vista, baby!).

*4 Die wohl für den Prozessor **günstigste Lösung**, womit du den Ausgabeoperator << nicht nochmals extra belasten musst.

Schrödinger Programmiert C++, S.53

Der Char-Typ

- Zum Speichern **einzelner Zeichen** kannst du `char` verwenden.
- `char` selbst speichert **keine Zeichen** im eigentlichen Sinne, sondern sucht das passende Zeichen aus einem Zeichensatz auf dem Rechner anhand eines dezimalen Wertes aus.
- Einzelne Zeichen können zwischen **einzelnen Anführungszeichen** oder als **Ganzzahl** mit dem Wert des Zeichens aus dem Zeichensatz verwendet werden.
- Die Verwendung von einzelnen Zeichen wie `'T'` ist im Grunde nur eine **symbolische Konstante** für den ganzzahligen Wert im Zeichensatz des Rechners.
- C++ schreibt nicht vor, welcher **Zeichensatz** verwendet werden soll. Zwar kannst du fast sicher sein, dass der **ASCII-Zeichensatz** auf deinem Rechner unterstützt wird, aber bei der Verwendung von Zeichen darüber hinaus (bspw. Umlauten) wird es dir oft passieren, dass auf dem einen Rechner alles glattgeht, während auf einem anderen Rechner **Zeichensalat** ausgegeben wird.
- Solltest du breitere Zeichen als `char` benötigen, findest du mit `wchar_t` einen Typ dafür. Anlog musst du dann auch die Streams für breite Zeichen verwenden (`wcout`, `wcerr`, `wcin`).
- C++11 unterstützt mit UTF-8, UTF-16 und UTF-32 drei Unicode-Kodierungen. Für UTF-16 wurde der Typ `char16_t` und für UTF-32 der Typ `char32_t` eingeführt. Die neue Unicode-Kodierung ist natürlich `wchar_t` vorzuziehen.

Schrödinger Programmiert C++, S.76

Benutzerdefinierte Literale (später)

Es können benutzerdefinierte Literale erstellt werden.
Hierzu wird in der gewünschten Klasse ein Operator definiert

Schrödinger Programmiert C++, S.53

Operatoren

1 Operand: Unäres Minus (i = -1338 ->i=-(1338))
2 Operanden: Arithmetische Operatoren (sum = a+b)
3 Operanden: Bedingeter Ausruck zB ternärer Operator

Ternärer Operator: a=<20 ?a+10:a-10 -> if bla else bla

Schrödinger Programmiert C++, S.53

Arithmetische Operatoren

Operator	Bedeutung	
+	Addition	*1 *2
-	Subtraktion	*1 *2
*	Multiplikation	*1
/	Division	*1
%	Modulo	*3

*1 sind auf ganzzahlige und reelle Operanden anwendbar
*2 können außerdem für Adresswerte benutzt werden
*3 ist nur für ganzzahlige Operanden einsetzbar

Zuweisungs Operatoren (binäre Operatoren)

Operator	Bedeutung
=	Zusweisung
+=	Addition
-=	Subtraktion
*=	Multiplikation
/=	Division
%=	Modulo
<<=	Zuweisung nach bitweisem Linksschieben (shift left)
>>=	Zuweisung nach bitweisem Rechtsschieben (shft right)
&=	Bitweises AND (Und)
|=	Bitweises OR (Oder
^=	Bitweises XOR (exklusives OR)

Relationale Operatoren (binäre Operatoren)

Operator	Bedeutung
<	Kleiner als
<=	Kleiner oder gleich
>	Größer als
>=	Größer oder gleich
==	Gleich
!=	ungleich

Logische Operatoren (binäre Operatoren)

Operator	Bedeutung
&&	AND (Und)
||	OR (Oder)
!	NOT (Negation, Nicht)

Bitmaniupulation

Operator	Bedeutung
&	Bitweises AND
|	Bitweises OR
^	Bitweises XOR
<<	Bitweises Linksschieben (shift left)
>>	Bitweises Rechtsschieben (shift right)

Inkrement- und Dekrement-Operatoren – (unäre O.)

Operator	Bedeutung
++	Erhöhung (increment)
--	Erniedrigung (decrement)

Die Operatoren dienen der Erhöhung bzw. der Verringerung des Wertes eines ganzzahligen Operandens um 1.
Beide Operatoren können sowohl als **Prefix-** als auch als **Postfix-Operatoren** eingesetzt werden.

Unterschied Prefix- & Postfix-Operatoren

Prefix-	Postfix
Der aktuelle Wert der zu in-/dekrementierenden Variablen wird verändert, danach wird der neue Wert in den Ausdruck eingebracht.	Der aktuelle Wert der zu in-/dekrementierenden Variablen wird in den Ausdruck eingebracht, danach erfolgt die Veränderung des Wertes.

`int i1=1,  i2=2;`
`int iRes;`
`iRes= ++i1 * i2  //iRes=4`
	`int i1=1,  i2=2;`
	`int iRes;`
	`iRes= i1++ * i2  //iRes=2`

www.htw-berlin.de