Einführung in die Programmierung für Studierende der Physik

JOHANN WOLFGANG GOETHE UNIVERSITÄT
19.04.2022

MATTHIAS HANAUSKE

FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

2. Vorlesung

Plan für die heutige Vorlesung

- Kurze Wiederholung der Vorlesung 1
- Das erste C++ Programm (Hello World)
- Der numerische Zahlenraum eines C++ Programms
- Datentypen und Variablen
- Die Standardbibliothek <cmath>
- Arithmetik und Operatoren
- Die Ein- und Ausgabe
- Übungsaufgabe Nr.2

http://th.physik.uni-frankfurt.de/~hanauske/VPROG/

Einführung in die Programmierung für Studierende der Physik

von Dr.phil.nat.Dr.rer.pol. Matthias Hanauske

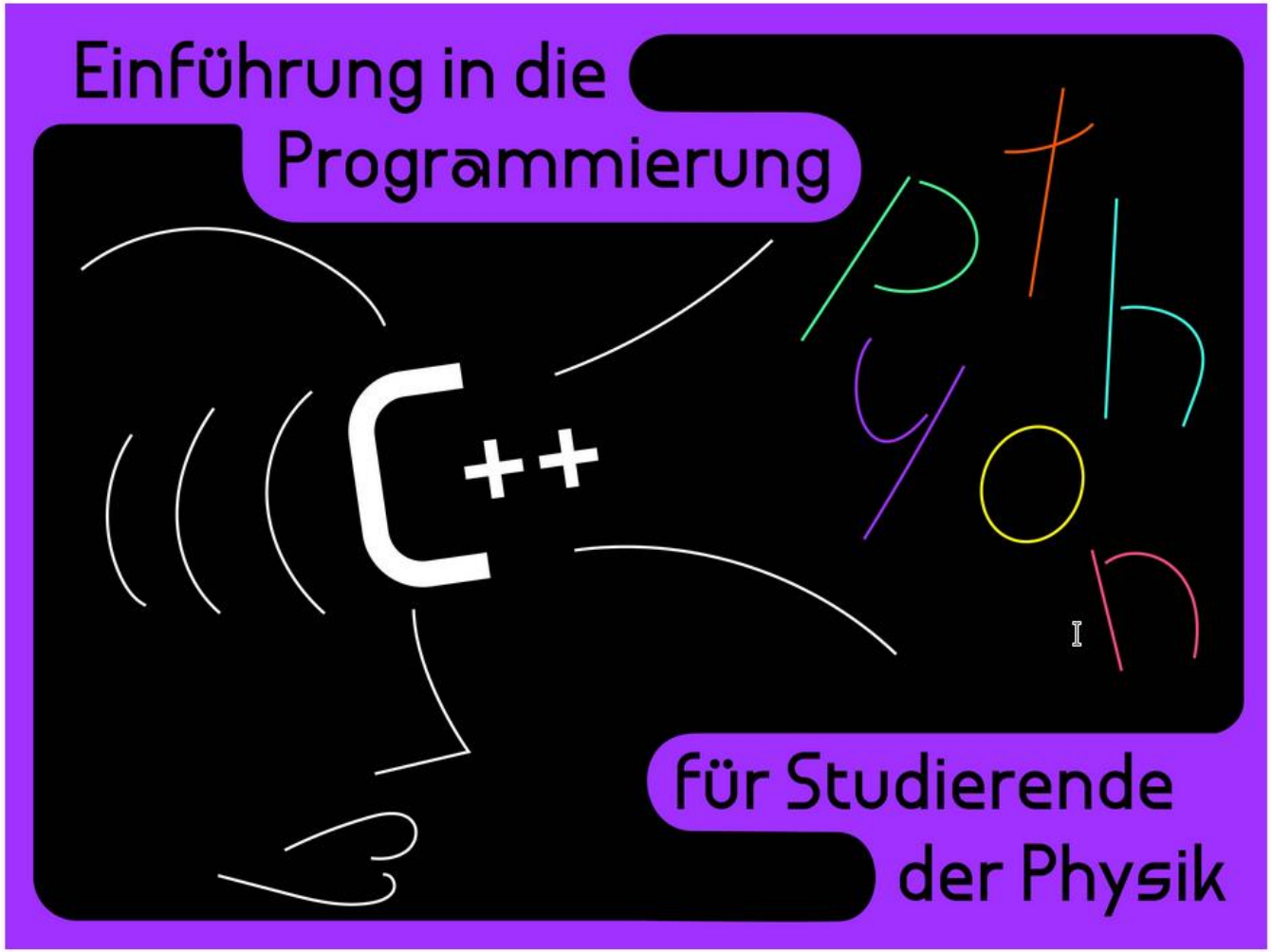


Illustration: Deborah Moldawski

Nächster Zoom Link am 12.04.2022, 15:00-16:00 Uhr: ID: 794 847 5614, PWD: 785453

Einführung in die Programmierung für Studierende der Physik
(Introduction to Programming for Physicists)
Vorlesung SS 2022

Diese Internetseite fasst die Online-Angebote der Vorlesung *Einführung in die Programmierung für Studierende der Physik* zusammen. Die Vorlesungstermine finden jeweils dienstags von 15.00-16.00 Uhr und donnerstags von 14.00-16.00 Uhr im Raum Phys-0.111 statt. Die Termine der Übungen/Praktika finden Sie auf der Online-Lernplatform OLAT und die Übungsaufgaben werden im linken Panel unter der jeweiligen Vorlesung und zusätzlich auf OLAT bereitgestellt.

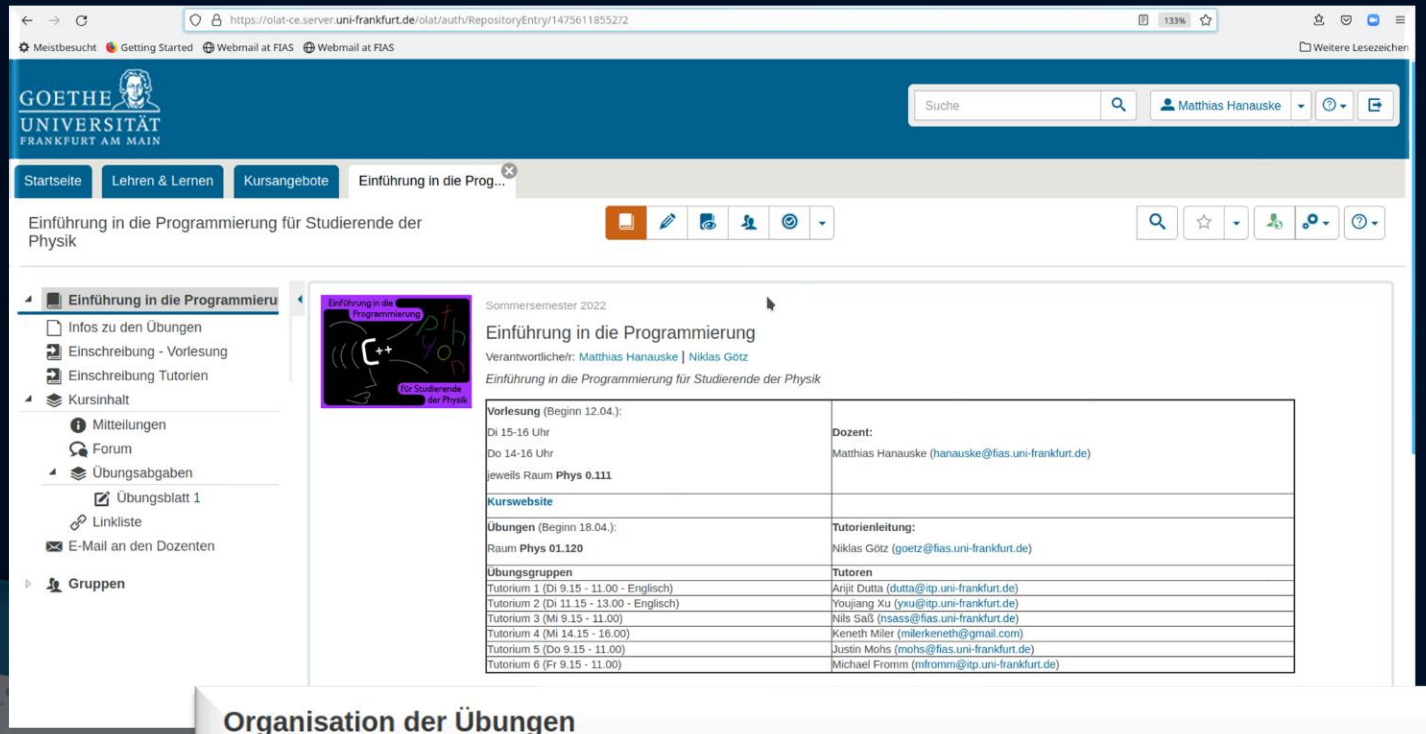
Die Vorlesung gibt einerseits eine Einführung in die Objekt-orientierte Programmiersprache C++ und vermittelt andererseits einige wesentliche Grundlagen der numerischen Mathematik. Es werden die grundlegenden Elemente der Programmiersprache, das Programmierparadigma der Objektorientierung und Simulationen von komplexen physikalischen Problemen behandelt. Das Hauptanliegen der Vorlesung besteht darin, dass die Studierenden die numerische Lösung eines komplexen physikalischen Problems auf dem Computer erstellen können. Der Schwerpunkt wird hierbei auf der Programmiersprache C++ liegen, wobei für die Visualisierung der berechneten Daten die Skriptsprache Python benutzt wird. Zusätzlich werden die berechneten mathematisch/physikalischen Gleichungen mittels Python Jupyter Notebooks analysiert und illustriert.

Literatur zu C++

- Prof. Dr. Marc Wagner, Vorlesung im WS 2019/20: Einführung in die Programmierung für Physiker
- B. W. Kernighan, D. M. Ritchie, Hanser: Programmieren in C
- Bjarne Stroustrup 2015: Die C++ Programmiersprache
- Bjarne Stroustrup 2009: Programming: Principles and Practice Using C++
- Prof. Dr. Claudius Gros, Vorlesung im WS 2021/22: Advanced

Die OLAT Seite

http://olat.server.uni-frankfurt.de



Auf der OLAT Seite der Vorlesung finden Sie Informationen zur Organisation der Übungen

Organisation der Übungen

Hier findet ihr alle wichtigen Informationen zu den Übungen

Aus- und Abgabe der Übungen

- In jeder Woche des Semesters wird auf der Kurswebsite das Skript zur jeweiligen Woche sowie das zugehörige Übungsblatt veröffentlicht.
- Für jedes Übungsblatt habt ihr bis Freitag 23:59 Uhr der darauf folgenden Woche Zeit, die Lösung hier in OLAT hochzuladen. Nutzt dazu das ensprechenden Aufgabenfeld unter "Kursinhalt"
- Sofern nicht anders von eurer*em Tutor*in gewünscht, ladet ihr dort eine .zip-File hoch, welche für jede Aufgabe einen einzelnen Ordner enthält. Für Programmieraufgaben werden dort die Quellcode-Files (KEINE kompilierten Programme) und eine .txt-File mit dem Befehl zum Kompilieren erwartet. Nutzt Kommentare, um auf weiterführende Fragen in der Aufgabe einzugehen und euren Lösungsweg zu erläutern. Für Aufgaben, bei denen ihr Bugs sucht, wird eine verbesserter Code und Erläuterungen der Fehler erwartet.
- Stellt sicher, dass sich eure Abgabe komplieren lässt, da die*der Tutor*in ansonsten auf diesen Teil der Abgabe keine Punkte geben muss.
- Die Musterlösungen der Übungen werden auf der Kurswebsite in der Woche nach der Korrektur veröffentlicht. Sie stellen dabei nur einen Lösungsweg dar.
- Über OLAT informieren euch die Tutor*innen in der Woche während der Besprechung des Übungsblatts über eure erreichte Punktzahl und die Verbesserungsvorschläge.

Format der Übungen

- In den Übungen herrscht Anwesenheitspflicht. Solltet ihr verhindert sein, gebt bitte eurer*em Tutor*in im Vorfeld Bescheid.
- Kerninhalt der Übungen ist die Vorstellung eurer Lösungsverschläge. Jeder von euch sollte mehrmals im Semester seine Lösung präsentieren. Hierzu stellt ihr zu einer Aufgabe eure Abgabe in OLAT euren Mitstudierenden vor, erklärt euren Ansatz und diskutiert diesen mit der*dem Tutor*in.
- Gegebenenfalls werden in den Übungen auch Inhalte der Vorlesungen vertieft oder auf häufige Probleme in den Abgaben eingegangen.
- In zweiten Woche des Semesters, wenn es noch keine Abgaben zu Besprechen gibt, helfen euch die Tutor*innen beim Aufsetzen eurer Entwicklungsumgebung.

Zulassung zur Klausur

- Um an der Klausur teilzunehmen, müssen mindestens 50% der Punkte in den Übungsaufgaben erreicht werden. Daneben müsst ihr regelmäßig zu den Tutorien erschienen sein und mindestens einmal eure Lösung vorgestellt haben.
- Solltet ihr bereits einen Schein von einem vorherigen Semester besitzen, gebt bitte Niklas Bescheid. Aktive Teilnahme an den Übungen und der Vorlesung ist dennoch dringend empfohlen.

Entwicklungsumgebung

- Am einfachsten ist es, wenn ihr von eurem eigenen Computer/Lapotp entwickelt. Solltet ihr nur ein Windows oder macOS Betriebssystem zur Verfügung haben, müsst ihr zur Nutzung des Terminals ggf. einige Anpassungen vornehmen. Hierzu gibt es sehr viele Tutorials im Internet, eure*euer Tutor*in kann euch hierbei auch behelflich sein.
- Steht euch kein Computer zur Verfügung, oder ihr möchtet mit einem bereits fertig eingerichteten System arbeiten, so gibt es die Möglichkeit an den PCs im Übungsraum zu arbeiten. Diese haben ein Ubuntu-Betriebssystem und alle notwendigen Progamme bereits installiert. Wir stellen euch hierfür die Login-Daten zur Verfügung. Ihr könnt dann von zu Hause aus mittels ssh auf die PCs zugreifen (mehr Details dazu in der Vorlesung)
- Welche Tools ihr zum Programmieren verwenden wollt, ist euch freigestellt. Theoretisch reichen das Terminal, ein Compiler, ein Texteditor und Jupyter für die Vorlesung aus. Moderne integrierte Entwicklungsumgebungen (IDE) können euch jedoch helfen, Probleme in eurem Code zu erkennen und schneller zu programmieren. Ein Beispiel für eine anfängerfreundliche IDE ist Visual Studio Code. Solche IDEs werden in allen Bereichen von Forschung und Wirtschaft beim Programmieren verwendet, weshalb es sinnvoll ist, sich damit vertraut zu machen. Solltet ihr beim Installieren oder bei der Nutzung auf Probleme stoßen, könnt ihr euch an eure*n Tutor*in wenden.

Warum und wozu Programmierung

C++

python

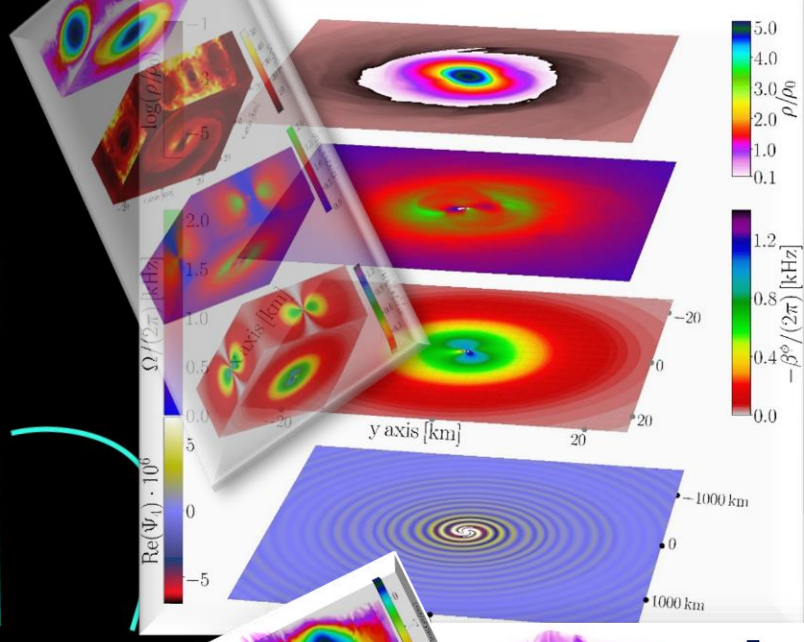
Keine analytische Lösung des betrachteten Systems möglich

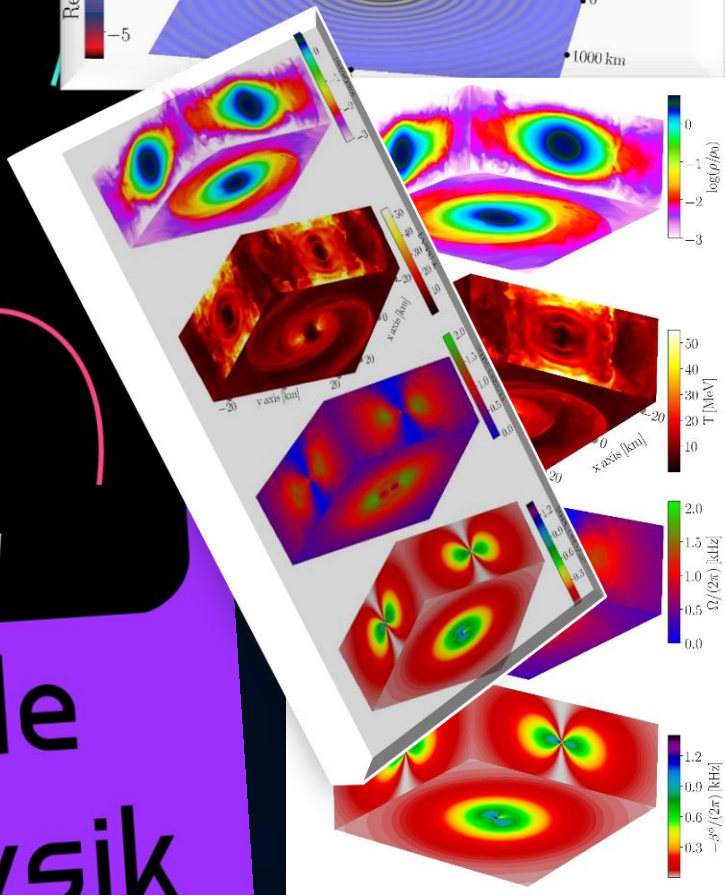
Computer Simulation

einstein toolkit

Visualisierung

$$R_{\mu\nu} - \frac{1}{2} R\, g_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}$$





für Studierende der Physik

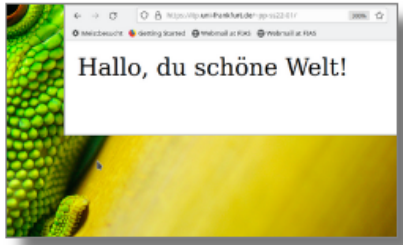
Programmiersprachen

Höhere Programmiersprachen gibt es schon seit den 1950er Jahren und die älteste noch in weitem Gebrauch befindliche Programmiersprache ist *Fortran* (FORmula TRANslator). Die Programmiersprache *BASIC* (Beginner's All-purpose Symbolic Instruction Code) wurde Ende der 1970er Jahre, aufgrund der für Jedermann erschwinglichen Heimcomputern (z.B. der Commodore C-64) populär. Die Programmiersprache C entstand 1972 und die Objekt-orientierte Variante von C (*C++*) wurde im Jahre 1983 von Bjarne Stroustrup vorgestellt. In der Vorlesung wird der Schwerpunkt auf der Programmiersprache *C++* liegen und für die Visualisierung der berechneten Daten wird die Skriptsprache *Python* benutzt, die Anfang der 1990er Jahre entwickelt wurde (siehe Zeittafel der Programmiersprachen). Die Installation von *C++* und *Python* ist auf allen gängigen Betriebssystemen möglich und wird am Beispiel des Betriebssystems Linux unter dem folgenden Link kurz beschrieben: Programmiersprachen.

Eine kleine Einführung in Linux

Damit man ein Computerprogramm erstellen und ausführen kann, benötigt man zunächst einen Computer (PC oder Laptop) und ein Betriebssystem, welches die Systemkomponenten des Computers (z.B. Arbeitsspeicher, Festplatten) verwaltet. Betriebssysteme, wie Windows von Microsoft, macOS von Apple oder Linux bilden die Schnittstelle zwischen den Hardware-Komponenten und der Anwendungssoftware des Benutzers des Computers. Das frei erhältliche Betriebssystem Linux wird weltweit auf vielen Großrechnern, Computerclustern und Supercomputern eingesetzt und bildet auch, durch seine führende Rolle als Internet-Server-System, die grundlegende Struktur des Internets. Man kann mittlerweile Linux relativ einfach auf seinem eigenen Computer installieren (siehe Übungsaufgabe 1.1) und mittels einiger Linux-Shell-Befehle ist eine kommandobasierte Benutzung des Computersystems möglich. Zusätzlich wird in diesem Unterpunkt auch vorgestellt, wie man sich mittels des kryptografischen Netzwerkprotokolls SSH auf die Rechner des Instituts für Theoretische Physik der Goethe-Universität einloggt. Die dafür nötigen Login-Accounts werden in den ersten beiden Vorlesungen an die Studierenden verteilt. Beim Klicken auf das nebenstehende Bild erfahren Sie mehr.

Anwendungsbeispiel: Erstellen einer eigenen Internet-Homepage



In diesem Unterpunkt werden einige der besprochenen Linux Shell-Befehle angewendet, um die Erstellung einer eigenen Internet-Homepage zu realisieren. Zusätzlich wird das *Secure File Transfer Protocol (SFTP)* vorgestellt, mit dem man in verschlüsselter Form von einem Rechner auf einen anderen Rechner Daten übertragen kann (näheres siehe Anwendungsbeispiel: Erstellen einer eigenen Internet-Homepage).

Kompilierung und Ausführung des C++ Programms

Quelltext des C++ Programms „HelloWorld.cpp“

HelloWorld.cpp

```cpp
#include <iostream>                                  // Ein- und Ausgabebibliothek
using namespace std;                                 // Benutze den Namensraum std

int main(){                                          // Hauptfunktion
    cout << "Hallo, du schöne Welt!" << endl; // Ausgabe eines Textes
}
```

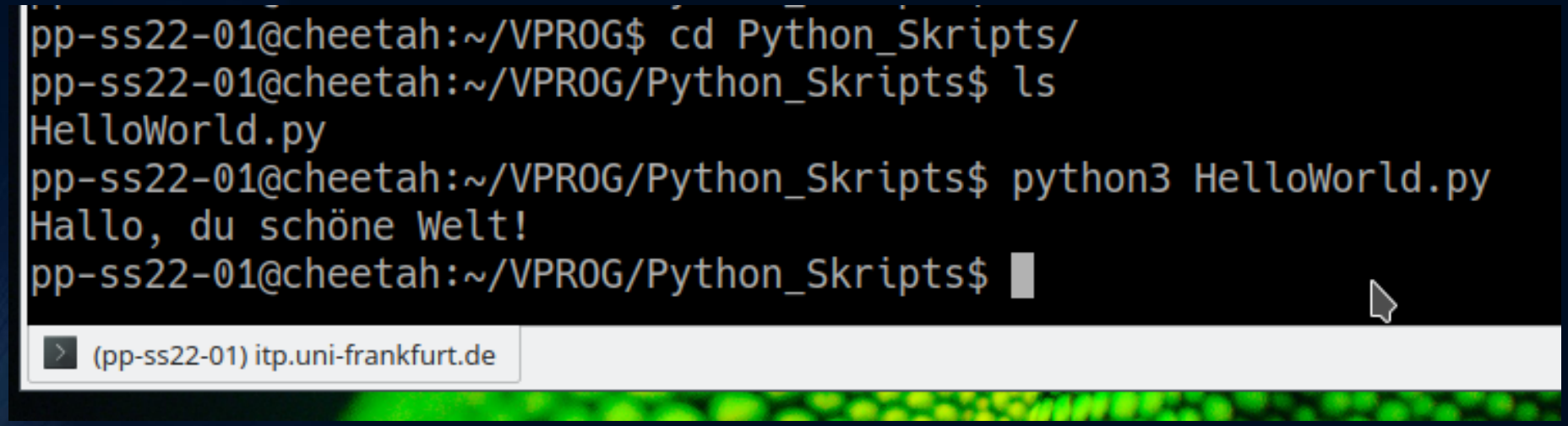
g++ Helloworld.cpp


```
pp-ss22-01@cheetah:~/VPROG/C++$ ls
HelloWorld.cpp
pp-ss22-01@cheetah:~/VPROG/C++$ g++ HelloWorld.cpp
pp-ss22-01@cheetah:~/VPROG/C++$ ls
a.out  HelloWorld.cpp
pp-ss22-01@cheetah:~/VPROG/C++$ ./a.out
Hallo, du schöne Welt!
pp-ss22-01@cheetah:~/VPROG/C++$ ./a.out > textfile.txt
pp-ss22-01@cheetah:~/VPROG/C++$ ls
a.out  HelloWorld.cpp  textfile.txt
pp-ss22-01@cheetah:~/VPROG/C++$
(pp-ss22-01) itp.uni-frankfurt.de
```

Um zu sehen, was das C++ Programm macht, muss man es zunächst kompilieren; dies funktioniert mit dem Befehl **g++ HelloWorld.cpp** Während des Kompilierung-Prozesses wurde eine neue Datei mit dem Namen "a.out" erzeugt, welche das ausführbare Programm darstellt. Führt man das Programm schließlich mit dem Befehl **./a.out** aus, so erscheint die folgende Ausgabe auf der Shell-Konsole: "Hallo, du schöne Welt!" Man kann sich die Terminalausgabe auch in eine Textdatei (z.B. in "textfile.txt") umleiten lassen, wenn man den folgenden Befehl ausführt: **./a.out > textfile.txt**

Ausführen eines Python Programms

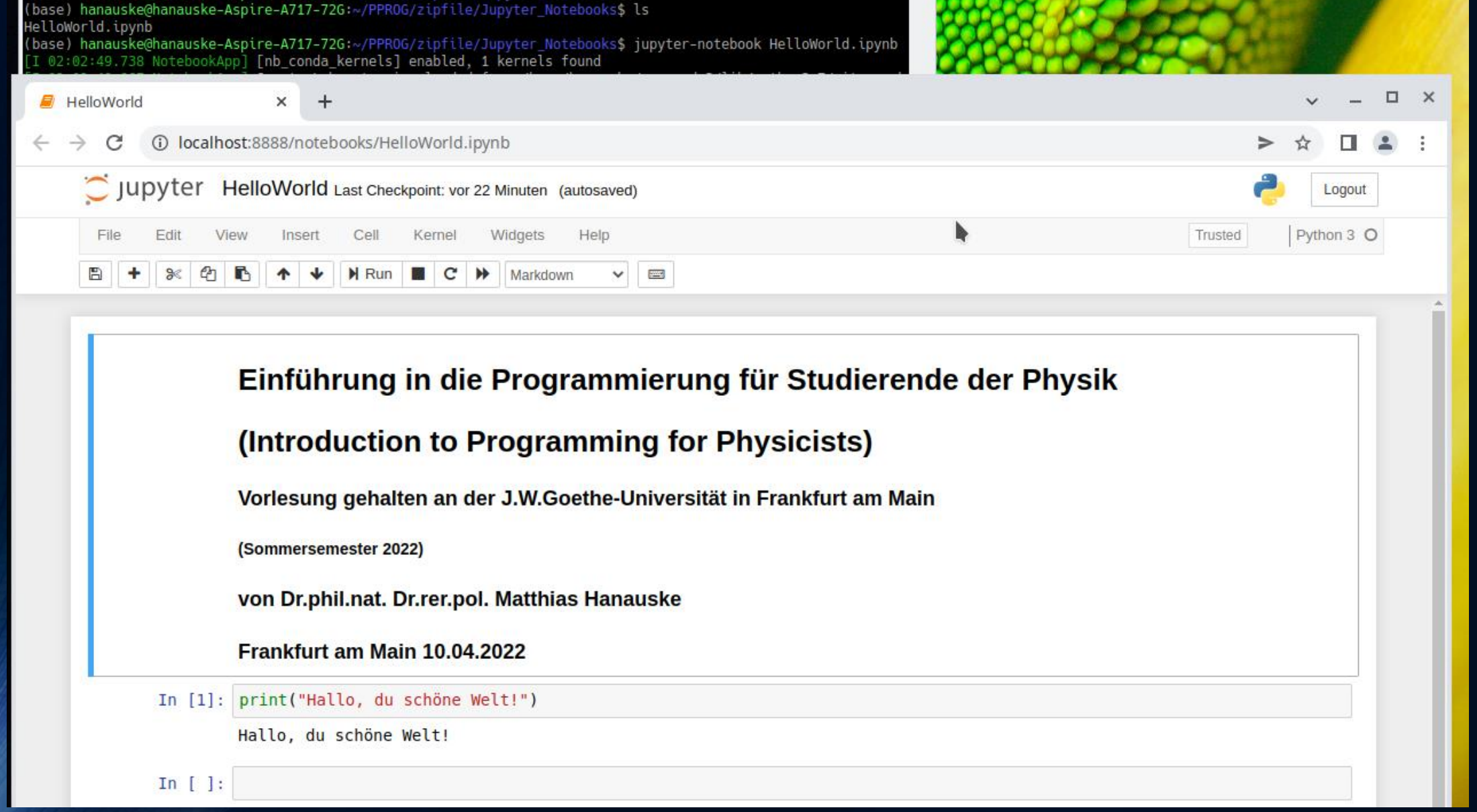
Gehen Sie nun mit **cd ~/VPROG/Python_Skripts/** in das Verzeichnis "Python_Skripts" und listen mit **ls** die im Ordner befindlichen Dateien (siehe untere linke Abbildung). Man erkennt, dass sich eine Datei HelloWorld.py darin befindet. Es handelt sich hierbei um den Quelltext eines Python Skripts, welches in der linken unteren Abbildung dargestellt ist. Um zu sehen was das Python Skript macht, gibt man den Befehl **python3 HelloWorld.py** ein und es erscheint die folgende Ausgabe auf der Shell-Konsole: "Hallo, du schöne Welt!"



HelloWorld.py

```
print("Hallo, du schöne Welt!")
```

Quelltext des Python Programms „HelloWorld.py“



Python Jupyter Notebooks
eine web-basierte interaktive Programmierumgebung für Python

Jupyter Notebooks

Nutzung ähnlich wie bei *Maple Worksheets* oder *Mathematica Notebooks*

Übungsblatt Nr. 1

Aufgabe 1 *(10 Punkte)*

In den ersten beiden Vorlesungen haben Sie Ihren Login-Account für die Rechner des Instituts für Theoretische Physik (ITP) der Goethe-Universität erhalten. Sollte dies jedoch nicht so sein, dann wenden Sie sich bitte direkt per E-Mail an Ihren Tutor (siehe Informationen zu den Übungen auf der Online-Lernplatform OLAT). Bitte lesen Sie die Allgemeine Nutzungsordnung für die Informationsverarbeitungs- und Kommunikations-Infrastruktur der Johann Wolfgang Goethe-Universität, bevor Sie den Login-Account das erste Mal verwenden.

Führen Sie die folgenden Teilaufgaben aus und halten diese mittels eines Bildschirmfotos (Screenshots) fest:

- Bauen Sie eine SSH-Verbindung mit den Rechnern des ITP auf.
- Ändern Sie das Passwort Ihres ITP-Accounts.
- Erstellen Sie einen neuen Ordner "VPROG" und entzippen dort den folgenden zip-File: Download zip-File.
- Erstellen Sie Ihre eigene Homepage. Es reicht hierbei aus, wenn Sie den im zip-File befindlichen HTML-File "index.html" in Ihr "public_html" Verzeichnis kopieren und überprüfen, dass die Zugriffsrechte für alle lesbar sind.
- Zeigen Sie durch ein Bildschirmfoto eines Internet-Browsers, dass ihre Homepage Online ist.
- Ändern Sie nun die Rechte der HTML-Datei "index.html", sodass nur noch Sie und Gruppenmitglieder die Datei lesen können und zeigen Sie dann mittels eines Bildschirmfotos eines Internet-Browsers, dass ihre Homepage nicht mehr zugänglich ist.
- Kompilieren Sie den C++ Quelltext "HelloWorld.cpp" und führen das Programm aus.
- Führen Sie das Python Programm "HelloWorld.py" aus.

Aufgabe 2 *(5 Punkte)*

Erstellen Sie einen bootfähigen USB-Stick mit einer Linux-Distribution Ihrer Wahl und starten Sie Linux im Live USB Modus. Machen Sie ein Bildschirmfoto, wenn das Linux Betriebssystem hochgefahren ist. Falls Sie Linux schon installiert haben, machen Sie ein Bildschirmfoto von Ihrem Heimrechner.

Aufgabe 3 *(5 Punkte)*

Installieren Sie Python und Jupyter auf Ihrem Heimrechner und öffnen Sie die das Jupyter Notebook "HelloWorld.ipynb" auf ihrem eigenen System. Alternativ können Sie auch mittels einer grafischen Remote-Verbindung das Jupyter Notebook auf den Rechnern des ITP öffnen. Machen Sie ein Bildschirmfoto.

Bitte laden Sie die einzelnen Bildschirmfotos der Aufgaben auf der Online-Lernplatform OLAT hoch; gerne auch als gebündeltes, komprimiertes .zip-File.

Übungsblatt Nr. 1

Aufgabe 1 *(10 Punkte)*

In den ersten beiden Vorlesungen haben Sie Ihren Login-Account für die Rechner des Instituts für Theoretische Physik (ITP) der Goethe-Universität erhalten. Sollte dies jedoch nicht so sein, dann wenden Sie sich bitte direkt per E-Mail an Ihren Tutor (siehe Informationen zu den Übungen auf der Online-Lernplatform OLAT). Bitte lesen Sie die Allgemeine Nutzungsordnung für die Informationsverarbeitungs- und Kommunikations-Infrastruktur der Johann Wolfgang Goethe-Universität, bevor Sie den Login-Account das erste Mal verwenden.

Führen Sie die folgenden Teilaufgaben aus und halten diese mittels eines Bildschirmfotos (Screenshots) fest:

- Bauen Sie eine SSH-Verbindung mit den Rechnern des ITP auf.
- Ändern Sie das Passwort Ihres ITP-Accounts.
- Erstellen Sie einen neuen Ordner "VPROG" und entzippen dort den folgenden zip-File: Download zip-File.
- Erstellen Sie Ihre eigene Homepage. Es reicht hierbei aus, wenn Sie den im zip-File befindlichen HTML-File "index.html" in Ihr "public_html" Verzeichnis kopieren und überprüfen, dass die Zugriffsrechte für alle lesbar sind.
- Zeigen Sie durch ein Bildschirmfoto eines Internet-Browsers, dass ihre Homepage Online ist.
- Ändern Sie nun die Rechte der HTML-Datei "index.html", sodass nur noch Sie und Gruppenmitglieder die Datei lesen können und zeigen Sie dann mittels eines Bildschirmfotos eines Internet-Browsers, dass ihre Homepage nicht mehr zugänglich ist.
- Kompilieren Sie den C++ Quelltext "HelloWorld.cpp" und führen das Programm aus.
- Führen Sie das Python Programm "HelloWorld.py" aus.

Bemerkung für Windows 10 und 11 User: Die Aufgabe ist auch erfüllt, falls Sie sich für eine Linux Installation mittels Windows Subsystem for Linux entscheiden.

Aufgabe 2 *(5 Punkte)*

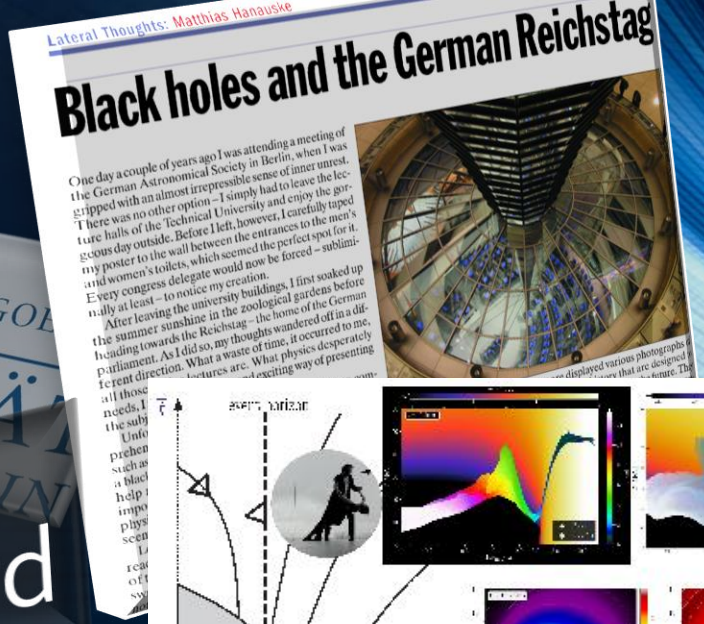
Erstellen Sie einen bootfähigen USB-Stick mit einer Linux-Distribution Ihrer Wahl und starten Sie Linux im Live USB Modus. Machen Sie ein Bildschirmfoto, wenn das Linux Betriebssystem hochgefahren ist. Falls Sie Linux schon installiert haben, machen Sie ein Bildschirmfoto von Ihrem Heimrechner.

Aufgabe 3 *(5 Punkte)*

Installieren Sie Python und Jupyter auf Ihrem Heimrechner und öffnen Sie die das Jupyter Notebook "HelloWorld.ipynb" auf ihrem eigenen System. Alternativ können Sie auch mittels einer grafischen Remote-Verbindung das Jupyter Notebook auf den Rechnern des ITP öffnen. Machen Sie ein Bildschirmfoto.

Bitte laden Sie die einzelnen Bildschirmfotos der Aufgaben auf der Online-Lernplatform OLAT hoch; gerne auch als gebündeltes, komprimiertes .zip-File.

MATTHIAS HANAUSKE
FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN

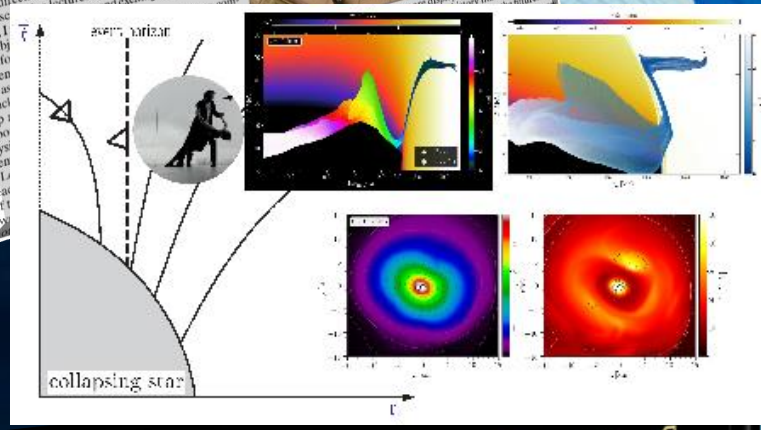

Lateral Thoughts: Matthias Hanauske

Black holes and the German Reichstag

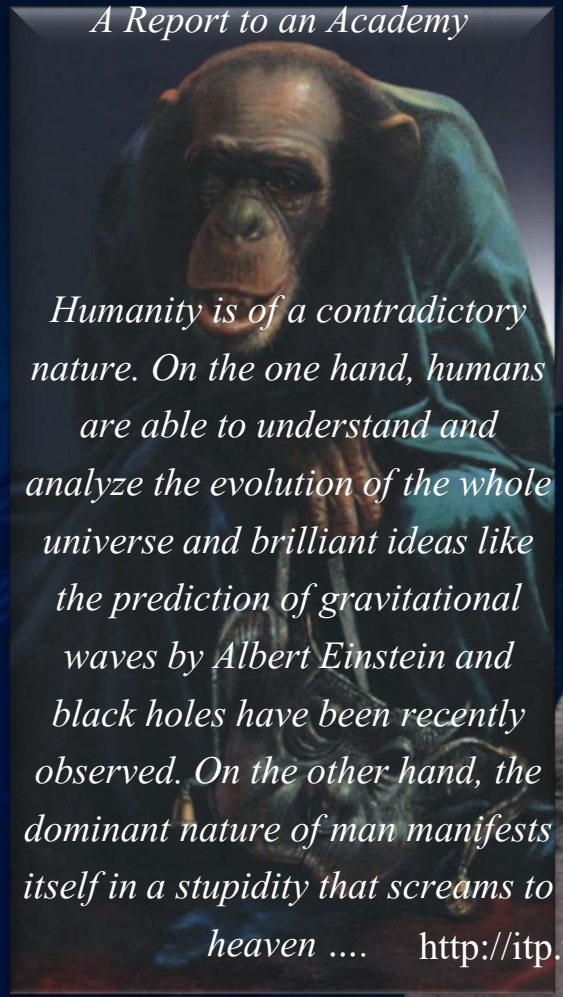
One day a couple of years ago I was attending a meeting of the German Astronomical Society in Berlin, when I was gripped with an almost irrepressible sense of inner unrest. There was no other option – I simply had to leave the lecture halls of the Technical University and enjoy the gorgeous day outside. Before I left, however, I carefully taped my poster to the wall between the entrances to the men's and women's toilets, which seemed the perfect spot for it. Every congress delegate would now be forced – sublimi-nally at least – to notice my creation.

After leaving the university buildings, I first soaked up the summer sunshine in the zoological gardens before heading towards the Reichstag – the home of the German parliament. As I did so, my thoughts wandered off in a different direction. What a waste of time, it occurred to me, all those lectures are. What physics desperately needs, I thought, is a more exciting way of presenting the subject.

General Relativity in the Theater of the Absurd



A Report to an Academy



Humanity is of a contradictory nature. On the one hand, humans are able to understand and analyze the evolution of the whole universe and brilliant ideas like the prediction of gravitational waves by Albert Einstein and black holes have been recently observed. On the other hand, the dominant nature of man manifests itself in a stupidity that screams to heaven ....

Parallel session: Education

Teaching Einsteinian Physics to School Students

08.07.2021, 17:20

MG16 5-10 JULY 2021
SIXTEENTH MARCEL GROSSMANN MEETING
ON RECENT DEVELOPMENTS IN THEORETICAL AND EXPERIMENTAL GENERAL RELATIVITY, ASTROPHYSICS AND RELATIVISTIC FIELD THEORIES

VIRTUAL MEETING
websites: http://www.icra.it/mg/mg16/ https://indico.icranet.org/event/1/
email: mg16@icranet.org
6:30-19:30 CENTRAL EUROPEAN SUMMER TIME

50TH ANNIVERSARY OF "INTRODUCING THE BLACK HOLE"

"Peace cannot be kept by force. It can only be achieved by understanding."
Albert Einstein

http://itp.uni-frankfurt.de/~hanauske/new/etc/pdf/MG16-Hanauske.pdf

Die Quanten-Spieltheorie
und die eichtheoretische Beschreibung der elementaren Materie

MATTHIAS HANAUSKE
FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN

Talk at the

Frankfurt Institute for Advanced Studies

Organized by the group of

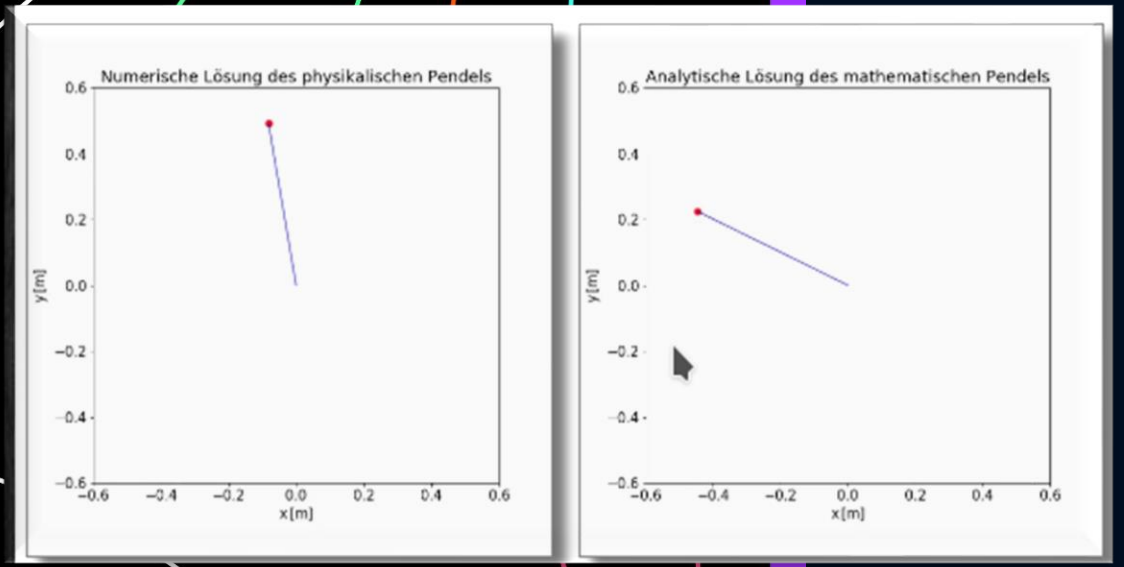
Kanonische Gravitationstheorie

21.04.2022, 11.00 Uhr

Join CCGG Zoom Meeting:
https://zoom.us/j/97186077843?pwd=eTRhUC91ajBodGpIOENxV09RdEl2dz09
Meeting ID: 971 8607 7843
Passcode: 889205

Einführung in die Programmierung

C++



Computer Simulation

Visualisierung

Keine analytische Lösung des betrachteten Systems möglich

für Studierende der Physik

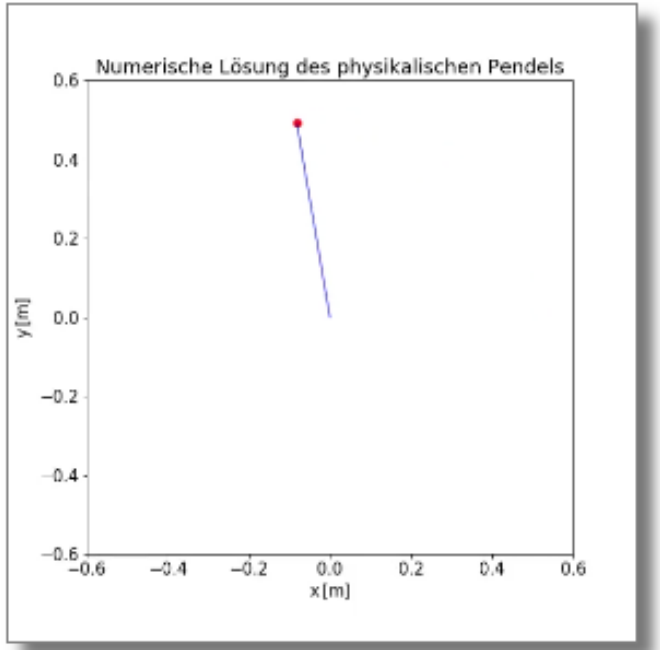
Das physikalische Pendel

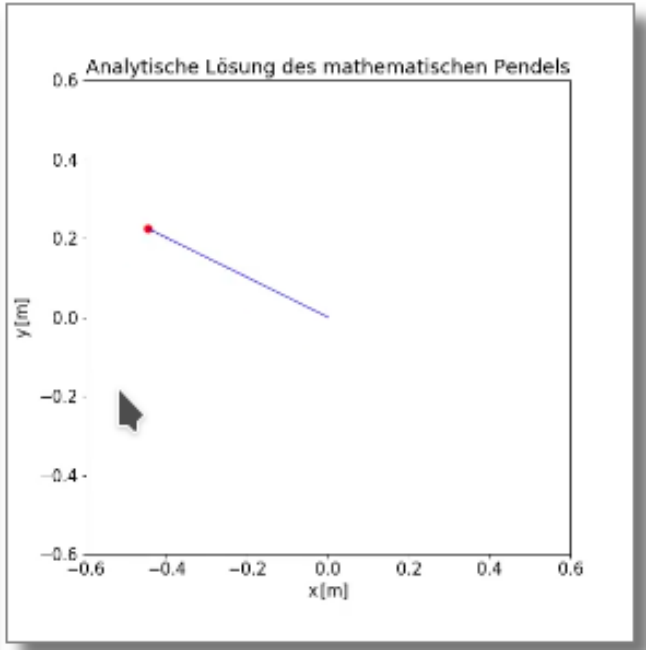
als Beispiel für ein nicht analytisch lösbares System

W.Greiner, Klassische Mechanik I). Die zugrundeliegende Differentialgleichung (DGL) des Problems lautet

$$\frac{d^2\phi(t)}{dt^2} = \frac{g}{l} \cdot \sin\left(\phi(t)\right) \quad ,$$

wobei g die Erdbeschleunigung, l die Länge des Pendels und $\phi(t)$ die zeitliche Entwicklung des Pendelwinkels beschreibt.





Vorlesung 2

Im Folgenden wird vorausgesetzt, dass Sie auf Ihrem eigenen Computer einen lauffähigen *C++ Compiler* und *Python 3* installiert haben. Wir beginnen, wie viele andere Einführungen in die Programmierung, mit der C++ Version des klassischen Hello World Programms. Es wird die Textausgabe im Linuxterminal und Eingabe von Zahlenwerten über die Tastatur vorgestellt. Will man numerische Berechnungen in der Programmiersprache C++ durchführen, muss man zunächst definieren, in welchem Zahlenraum die numerischen Variablen sich aufhalten und einen speziellen Typ den Variable zuordnen. Die grundlegenden, integrierten Datentypen von C++ und die mittels Operatoren integrierte Arithmetik werden in dieser Vorlesung vorgestellt. Am Ende der Vorlesung werden die Benutzereingabe und die formatierte Ausgabe von Zahlenwerten behandelt.

Das erste C++ Programm (Hello World)

In diesem Unterpunkt wird die Programmiersprache C++ am Beispiel des klassischen *Hello World Programms* vorgestellt. C++ ist eine kompilierte Programmiersprache, d.h. der vom Programmierer geschriebene *Quelltext* des Computerprogramms muss zunächst mittels eines *Compilers* in ein ausführbares Programm umgewandelt werden, damit es genutzt werden kann. Dieser Kompilierungsprozess und die Textausgabe des ausführbaren *Hello World Programms* im Linuxterminal (mittels cout bzw. printf) wird unter folgendem Link vorgestellt: Das erste C++ Programm (Hello World).

Datentypen und Variablen

In gleicher Weise, wie es in der Mathematik die unterschiedlichen Zahlenmengen (z.B. $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{R}$, $\mathbb{C}$) gibt und man bei mathematischen Berechnungen den Wertebereich einer Variable oder Funktion vorher definieren muss, ist dies auch beim Programmieren in C++ nötig. In diesem Unterpunkt werden die Begriffe *Deklaration einer Variable*, *Datentyp*, *Wert einer Variable* und die *Initialisierung einer Variable* diskutiert. Die wichtigsten integrierten Datentypen von C++ sind bool, char, int, float und double. Näheres siehe unter folgendem Link: Datentypen und Variablen

Arithmetik und Operatoren

Möchte man mit Datentypen und Variablen mathematische Berechnungen durchführen oder die Datenwerte miteinander vergleichen, ist es zunächst nötig eine Arithmetik (die zum Zählen oder Rechnen gehörige Kunst) zu definieren. Hierzu wurden in C++ geeignete Operatoren definiert, die dem Programmierer neben den arithmetischen Grundrechenarten (Multiplikation, Addition, ...) noch diverse weitere integrierte Operatoren zur Verfügung stellen. Man unterscheidet hierbei die *arithmetischen Operatoren*, die *logischen Vergleichsoperatoren* und weitere *spezifische Operatoren*. Zusätzlich sind einige wichtige mathematische Funktionen (Sinus, Cosinus, ..) in der Standardbibliothek <cmath> vordefiniert (näheres siehe Arithmetik und Operatoren).

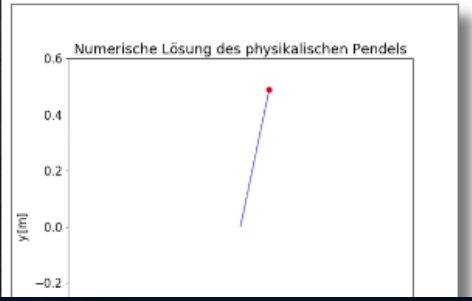
Die Ein- und Ausgabe

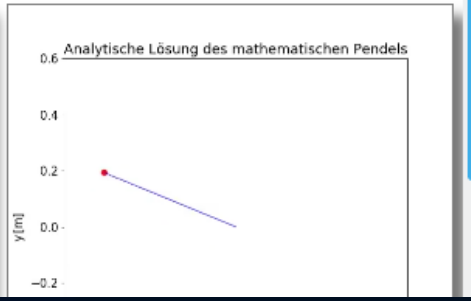
Vorlesung 2

Es geschieht einem theoretischen Physiker leider nur zu oft, dass er ein System beschreiben möchte, welches er auf analytischem, mathematischem Weg nicht lösen kann. Betrachten wir z.B. das einfache physikalische Pendel aus dem Themenbereich der Mechanik (siehe z.B. W.Greiner, Klassische Mechanik I). Die zugrundeliegende Differentialgleichung (DGL) des Problems lautet

$$\frac{d^2\phi(t)}{dt^2} = \frac{g}{l} \cdot \sin\left(\phi(t)\right) \quad ,$$

wobei g die Erdbeschleunigung, l die Länge des Pendels und $\phi(t)$ die zeitliche Entwicklung des Pendelwinkels beschreibt. Ungünstigerweise besitzt diese DGL keine analytisch darstellbare Lösung und gewöhnlich vereinfacht man dann das zu beschreibende System (z.B. durch Approximation zum mathematischen Pendel) und gelangt auf diesem Wege doch zu analytischen Lösungen, deren Verhalten man studieren kann. Vergleicht man die Lösung dann mit der Realität, stimmt diese oft nur unter gewissen Randbedingungen. Nimmt man jedoch Anfangswerte des Pendels, die außerhalb des Gültigkeitsbereiches der approximierten Lösung liegen, weichen die Vorhersagen der Pendelbewegung deutlich von der wirklichen Lösung ab. Stoßen wir z.B. das Pendel mit einem sehr starken "Schubs" aus seiner Ruhelage, so stimmt die analytische Lösung des mathematischen Pendels nicht und das wirkliche Pendel verhält sich vollkommen anders. Die folgenden Animationen entstammen einem Python Jupyter Notebook und sie vergleichen die numerische Lösung des physikalischen Pendels mit der analytischen Lösung des mathematischen Pendels, bei der ein Pendelüberschlag nicht möglich ist.





Das erste C++ Programm (Hello World)

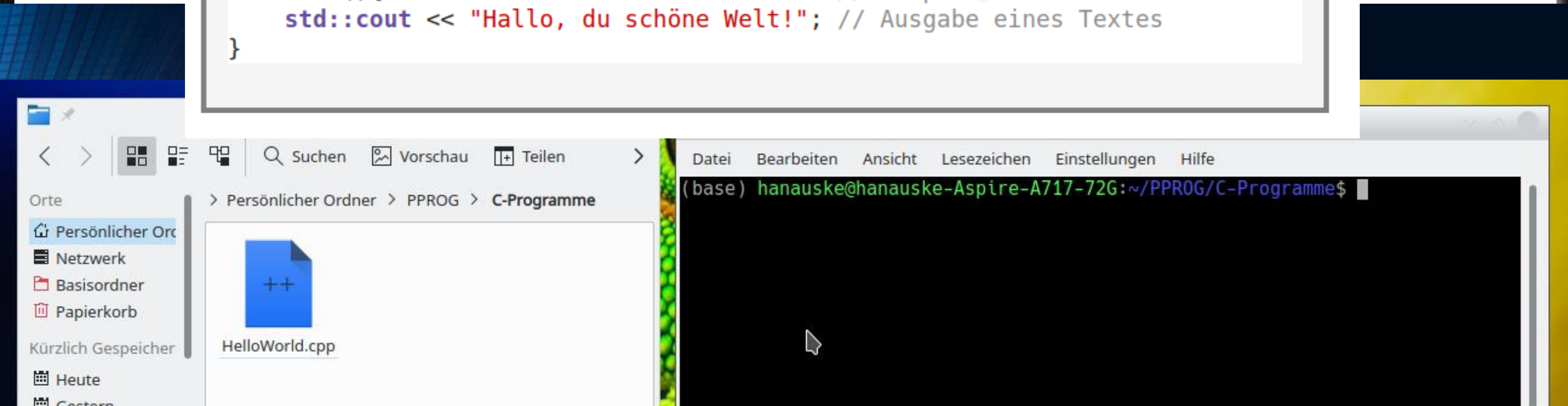
C++ ist eine kompilierte Programmiersprache, d.h. der vom Programmierer geschriebene *Quelltext* des Computerprogramms muss zunächst mittels eines *Compilers* in ein ausführbares Programm umgewandelt werden, damit es genutzt werden kann. Dieser Kompilierungsprozess und die Textausgabe des ausführbaren *Hello World Programms* im Linuxterminal wird im Folgenden vorgestellt.

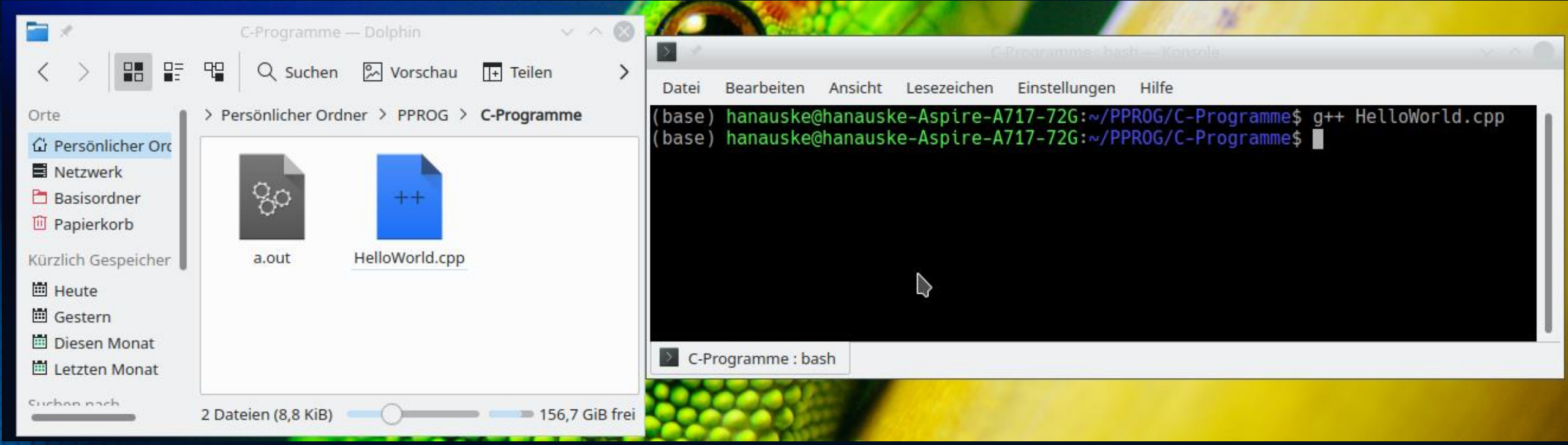
Der Quelltext eines C++ Programms kann in einem normalen Editor geschrieben, betrachtet und editiert werden (hier wurde z.B. der Editor Kate unter Linux verwendet). Bei umfangreicheren Quelltexten ist es jedoch vorteilhafter und komfortabler einen Editor mit einer Integrierten Entwicklungsumgebung (IDE) zu nutzen (z.B. Geany, Eclipse oder Visual Studio Code).
Der Quelltext des wohl einfachsten C++ Programms (Hello World Programm) ist in der folgenden Box dargestellt und Sie können sich das kurze Programm durch Klicken auf den Link in der oberen linken Seite der Box herunterladen:

HelloWorld.cpp

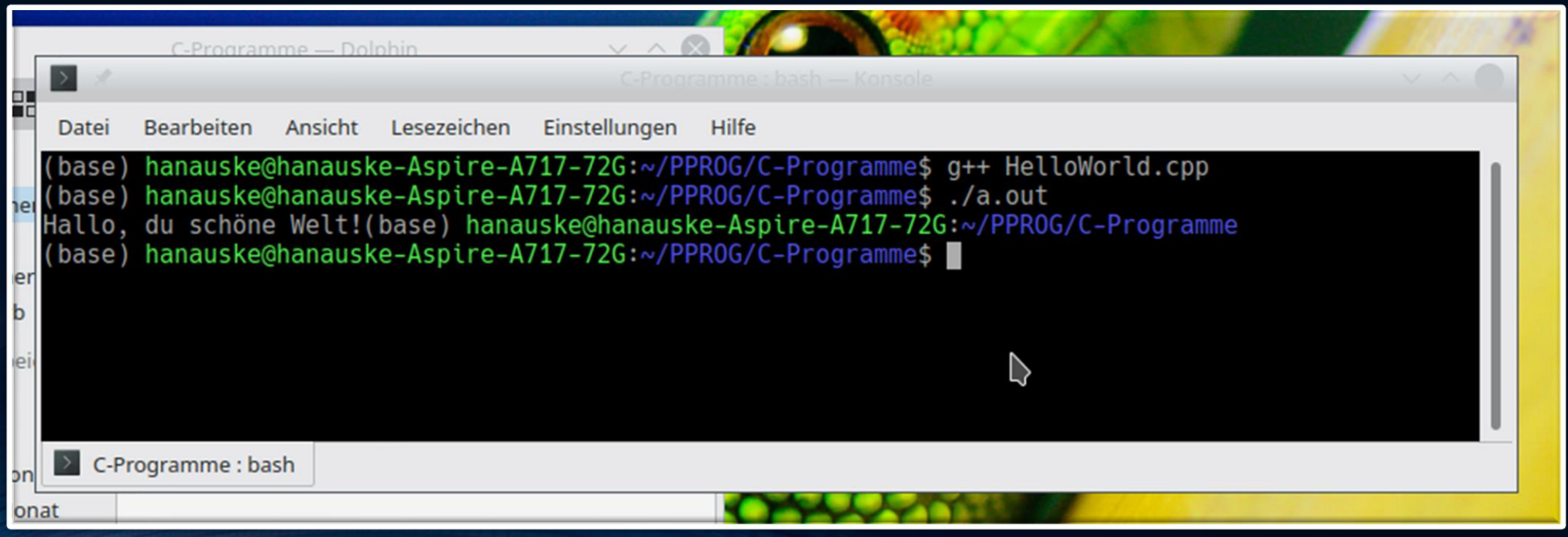
```cpp
#include <iostream>                          // Ein- und Ausgabebibliothek

int main(){                                  // Hauptfunktion
    std::cout << "Hallo, du schöne Welt!"; // Ausgabe eines Textes
}
```





g++ Helloworld.cpp



HelloWorld.cpp

```
#include <iostream>                          // Ein- und Ausgabebibliothek

int main(){                                  // Hauptfunktion
    std::cout << "Hallo, du schöne Welt!"; // Ausgabe eines Textes
}
```

Das Programm gibt den Text "Hallo, du schöne Welt!" im Linux-Terminal Fenster aus. Wir beschreiben zunächst die einzelnen Ausdrücke und Befehle des Quelltextes:

C++ Programm	Bedeutung
`#include <iostream>`	Damit das Programm die Ausgabe eines Textes produzieren kann, müssen zunächst die Deklarationen der standardmäßigen E/A-Implementierungen aus der C++ Standardbibliothek <iostream> eingebunden werden.
`// Ein- und Ausgabebibliothek`	Komentare, die eine einzelne Zeile im Code beschreiben werden mit // gekennzeichnet. Der Compiler ignoriert alles was nach diesen beiden Zeichen geschrieben ist.
`int main(){ ... }`	Jedes C++ Programm hat genau eine globale Hauptfunktion "main()" und das Programm startet indem diese Funktion aufgerufen wird. Die Bezeichnung int vor der Hauptfunktion gibt den Rückgabewert der Funktion an das System an. int bezeichnet hier einen Zahlenwert aus der Menge der natürlichen Zahlen $\mathbb{Z}$ an (näheres siehe Datentypen, Variablen und Arithmetik). Wenn die Hauptfunktion keinen Wert zurück gibt (wie in unserem Fall), bekommt das System einen Wert, der den erfolgreichen Abschluss anzeigt.
`std::cout << "Hallo, du schöne Welt!";`	In dieser Zeile des Programms wird der eigentliche Text "Hallo, du schöne Welt!" mittels des Befehls **std::cout** ausgegeben. Das Element **std::** gibt an, dass der Befehl der Standardausgabe **cout** im Namespace der Standardbibliothek zu finden ist. Der Operator << hat die Bedeutung "sende an" und schreibt sein zweites Argument auf sein erstes. Am Ende von jeder Befehlszeile eines C++ Programms muss ein Semikolon stehen.

Die Terminalausgabe

mittels „cout <<" oder „printf(...)"

HelloWorld_1.cpp

```cpp
#include <iostream>                          // Ein- und Ausgabebibliothek
using namespace std;                         // Benutze den Namensraum std

int main(){                                  // Hauptfunktion
    cout << "Hallo, du schöne Welt!" << endl; // Ausgabe eines Textes
}
```

```
C-Programme : bash — Konsole
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ g++ HelloWorld_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ./a.out
Hallo, du schöne Welt!
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$
```

HelloWorld_2.cpp

```cpp
#include <stdio.h>                           // Standard Input- und Output Bibliothek in C, z.B. printf(...)

int main(){                                  // Hauptfunktion
    printf("Hallo, du schöne Welt!\n");      // Ausgabe eines Textes mit printf(...)
}
```

Datentypen und Variablen

Unser Hauptanliegen in dieser Vorlesungsreihe besteht darin, die Eigenschaften von physikalischen Systemen und die Lösung von mathematischen Problemstellungen mittels des Computers zu simulieren. Hat man ein physikalisches Problem in einer oder mehreren mathematischen Gleichungen formuliert und quantifiziert, muss man diese Ausdrücke in eine dem Computer verständlichen Form bringen, z.B. die Gleichungen in einem C++ Programm implementieren. Die in den Gleichungen auftretenden Variablen beschreiben Eigenschaften des physikalischen Systems, die das Programm dann berechnen soll.
Nehmen wir zum Beispiel das mathematische Problem der Berechnung der Kreiszahl $\pi = 3.141592653589793...$ und betrachten die im Jahre 1682 von Gottfried Wilhelm Leibniz vorgestellte iterative Annäherung an diese irrationale Zahl in Form der Gleichung der Leibniz-Reihe:

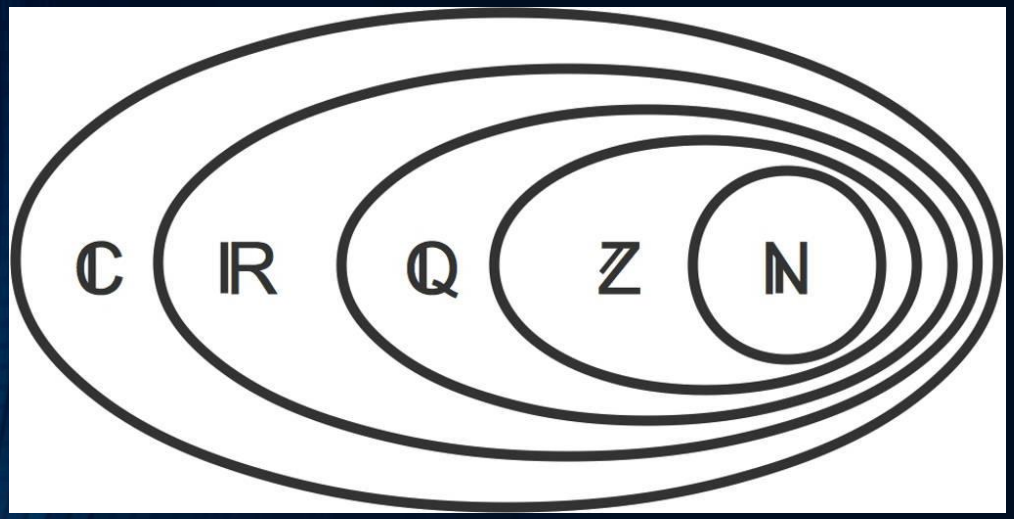
$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{3} - \frac{1}{5} + \ldots + \frac{(-1)^N}{2N+1}, \quad \text{mit:} \quad \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$

Möchte man diese Gleichung in einem C++ Programm implementieren, muss man das Ergebnis und die Summenformel, inklusive ihrer Variablen (k und N), im C++ Programm deklarieren. Ähnlich, wie bei einer präzisen mathematischen Definition, deklariert man die Variablen in einem C++ Programm auch anhand ihrer Zahlenmengen-Zugehörigkeit. Die Variablen k und N sind Element der natürlichen Zahlen ($k \in \mathbb{N}$ und $N \in \mathbb{N}$) und eine entsprechende Deklaration der Variable k in einem C++ Programm lautet z.B.:
Deklaration der Variable k: "int k ;"
Der Zusatz int am Anfang bedeutet, dass die Variable k aus der Zahlenmenge der ganzen Zahlen $\mathbb{Z}$ stammt und das Semikolon am Ende markiert einfach das Ende eines jeden C++ Ausdruckes. Genau genommen hätte man die Variable k wohl besser als unsigned int deklarieren sollen (siehe untere Tabelle der C++ Datentypen), da die Indexvariable k in der obigen Leibniz-Reihe nur positive ganze Zahlenwerte annehmen kann. Durch die Deklaration "int k ;" wird der Typ der Variable als ganzzahliger "Integer Wert" festgelegt und im Hauptspeicher (RAM) des Computers ein gewisser Speicherplatz für den Wert der Variable reserviert.

$$\sum_{k=0}^{N} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{3} - \frac{1}{5} + \ldots + \frac{(-1)^N}{2N+1}, \quad \text{mit:} \quad \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$

Der Zahlenraum des Computers

Aufbau der Zahlenmengen in der Mathematik



Ähnlich, wie bei einer präzisen mathematischen Definition, deklariert man die Variablen in einem C++ Programm auch anhand ihrer Zahlenmengen-Zugehörigkeit.

In gleicher Weise, wie es in der Mathematik die unterschiedlichen Zahlenmengen (z.B. $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{R}$, $\mathbb{C}$) gibt und man bei mathematischen Berechnungen den Wertebereich einer Variable oder Funktion vorher definieren muss, ist dies auch beim Programmieren in C++ nötig (in Python ist dies nicht nötig). In dieser Vorlesung werden die Begriffe *Deklaration einer Variable*, *Datentyp*, *Wert einer Variable* und die *Initialisierung einer Variable* diskutiert.

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{\pi}{4}$$

Die Variable k ist ein Element der natürlichen Zahlen ($k \in \mathbb{N}$) und eine entsprechende Deklaration der Variable k in einem C++ Programm lautet z.B.: "int k ;" Der Zusatz int am Anfang bedeutet, dass die Variable k aus der Zahlenmenge der ganzen Zahlen $\mathbb{Z}$ stammt und das Semikolon am Ende markiert einfach das Ende eines jeden C++ Ausdruckes. Genau genommen hätte man die Variable k wohl besser als unsigned int deklarieren sollen (siehe Tabelle der C++ Datentypen, auf der nächsten Folienseite), da die Indexvariable k in der nebenstehenden Leibniz-Reihe nur positive ganze Zahlenwerte annehmen kann.

Datentyp	Bedeutung	Speicherbedarf in Byte
bool	boolescher Wert (True oder False)	1
char	Zeichen, z.B. "a", "z" oder "9"	1
int	Ganzzahliger Wert $\in \mathbb{Z}$	4
short	Kurzer ganzzahliger Wert $\in \mathbb{Z}$	2
long	Langer ganzzahliger Wert $\in \mathbb{Z}$	8
unsigned int	Positiver ganzzahliger Wert $\in \mathbb{N}$	4
unsigned short	Positiver kurzer ganzzahliger Wert $\in \mathbb{N}$	2
unsigned long	Positiver langer ganzzahliger Wert $\in \mathbb{N}$	8
float	Reellwertiger Wert einfacher Genauigkeit $\in \mathbb{R}$ (6 Stellen)	4
double	Reellwertiger Wert doppelter Genauigkeit $\in \mathbb{R}$ (15 Stellen)	8
long double	Reellwertiger Wert noch höherer Genauigkeit $\in \mathbb{R}$ (19 Stellen)	16

Die Initialisierung einer Variable

Im Programm Datentypen_1.cpp hatten wir bereits die Initialisierung von zuvor deklarierten Variablen kennengelernt. Diese Initialisierung (Festlegung des nummerischen Wertes) erfolgte mittels des Gleichheitszeichens, z.B. "reelle_zahl = 2.4573;". Man hätte diese Initialisierung auch direkt bei der Deklaration der Variable durchführen können und man spricht dann von einer Variablen-Definition. Man kann die gleichzeitige Deklaration und Initialisierung einer Variable auf unterschiedliche Arten schreiben und die folgenden drei Varianten Definieren alle die ganze Zahl 4 als Integer Typ

```
int ganze_zahl = 4;      oder    int ganze_zahl (4);     oder    int ganze_zahl {4};
```

,wobei die letzte Form eine universelle Art der Initialisierung darstellt, die auch bei Initialisierungslisten verwendet wird. Initialisiert man gleich bei der Deklaration, ist es nicht mehr nötig den Typ explizit anzugeben und man benutzt dafür dann den Bezeichner **auto** welcher beim Initialisierungsprozess den Typ automatisch festlegt. Im folgenden Programm (Datentyp_auto.cpp) werden die unterschiedlichen Initialisierungsschreibweisen und der Bezeichner **auto** benutzt:

```cpp
#include <iostream>                         // Ein- und Ausgabebibliothek
using namespace std;                        // Benutze den Namensraum std

int main(){                                 // Hauptfunktion
    bool b;                                 // Deklaration der boolschen Variable 'b'
    char zeichen;                           // Deklaration der char Variable 'zeichen'
    int ganze_zahl;                         // Deklaration der ganzen Zahl Variable 'ganze_zahl'
    unsigned int natueliche_zahl;           // Deklaration der natürlichen Zahl Variable 'natueliche_zahl'
    short kurze_ganze_zahl;                 // Deklaration der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    long lange_ganze_zahl;                  // Deklaration der langen ganzen Zahl Variable 'lange_ganze_zahl'
    float reelle_zahl;                      // Deklaration der reellen Zahl Variable 'reelle_zahl'
    double reelle_zahl_doppeltgenau;        // Deklaration der reellen Zahl Variable 'reelle_zahl_doppeltgenau' mit doppelter Genauigkeit
    long double reelle_zahl_long;           // Deklaration der reellen Zahl Variable 'reelle_zahl_long' mit 19 Stellen

    b = "True";                             // Initialisierung (Festlegung des Wertes) der boolschen Variable 'b'
    zeichen = 'G';                          // Initialisierung der char Variable 'zeichen'
    ganze_zahl = -30256;                    // Initialisierung der ganzen Zahl Variable 'ganze_zahl'
    natueliche_zahl = 3278978;              // Initialisierung der natürlichen Zahl Variable 'natueliche_zahl'
    kurze_ganze_zahl = 245;                 // Initialisierung der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    lange_ganze_zahl = 327244582478947248;  // Initialisierung der langen ganzen Zahl Variable 'lange_ganze_zahl'
    reelle_zahl = 2.4573;                   // Initialisierung der reellen Zahl Variable 'reelle_zahl'
    reelle_zahl_doppeltgenau = 2453.4573545742;  // Initialisierung der reellen Zahl Variable 'reelle_zahl_doppeltgenau'
    reelle_zahl_long = 7653.3467587475842L;      // Initialisierung der reellen Zahl Variable 'reelle_zahl_long'

    cout << "Die Variable 'b' hat den Wert " << b;                    // Ausgabe eines Textes im Terminal
```

Datentypen_1.cpp

```cpp
#include <iostream>                                 // Ein- und Ausgabebibliothek
using namespace std;                                // Benutze den Namensraum std

int main(){                                         // Hauptfunktion
    bool b;                                         // Deklaration der boolschen Variable 'b'
    char zeichen;                                   // Deklaration der char Variable 'zeichen'
    int ganze_zahl;                                 // Deklaration der ganzen Zahl Variable 'ganze_zahl'
    unsigned int natueliche_zahl;                   // Deklaration der natürlichen Zahl Variable 'natueliche_zahl'
    short kurze_ganze_zahl;                         // Deklaration der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    long lange_ganze_zahl;                          // Deklaration der langen ganzen Zahl Variable 'lange_ganze_zahl'
    float reelle_zahl;                              // Deklaration der reellen Zahl Variable 'reelle_zahl'
    double reelle_zahl_doppeltgenau;                // Deklaration der reellen Zahl Variable 'reelle_zahl_doppeltgenau' mit doppelter Genauigkeit
    long double reelle_zahl_long;                   // Deklaration der reellen Zahl Variable 'reelle_zahl_long' mit 19 Stellen

    b = "True";                                     // Initialisierung (Festlegung des Wertes) der boolschen Variable 'b'
    zeichen = 'G';                                  // Initialisierung der char Variable 'zeichen'
    ganze_zahl = -30256;                            // Initialisierung der ganzen Zahl Variable 'ganze_zahl'
    natueliche_zahl = 3278978;                      // Initialisierung der natürlichen Zahl Variable 'natueliche_zahl'
    kurze_ganze_zahl = 245;                         // Initialisierung der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    lange_ganze_zahl = 327244582478947248;          // Initialisierung der langen ganzen Zahl Variable 'lange_ganze_zahl'
    reelle_zahl = 2.4573;                           // Initialisierung der reellen Zahl Variable 'reelle_zahl'
    reelle_zahl_doppeltgenau = 2453.4573545742;     // Initialisierung der reellen Zahl Variable 'reelle_zahl_doppeltgenau'
    reelle_zahl_long = 7653.3467587475842L;         // Initialisierung der reellen Zahl Variable 'reelle_zahl_long'

    cout << "Die Variable 'b' hat den Wert " << b;                  // Ausgabe eines Textes im Terminal
    cout << " und für sie wurde ein Speicherplatz von ";            // ...
    cout << sizeof(b) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'zeichen' hat den Wert " << zeichen;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(zeichen) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'ganze_zahl' hat den Wert " << ganze_zahl;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(ganze_zahl) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'natueliche_zahl' hat den Wert " << natueliche_zahl;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(natueliche_zahl) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'kurze_ganze_zahl' hat den Wert " << kurze_ganze_zahl;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(kurze_ganze_zahl) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'lange_ganze_zahl' hat den Wert " << lange_ganze_zahl;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(lange_ganze_zahl) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'reelle_zahl' hat den Wert " << reelle_zahl;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(reelle_zahl) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'reelle_zahl_doppeltgenau' hat den Wert " << reelle_zahl_doppeltgenau;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(reelle_zahl_doppeltgenau) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'reelle_zahl_long' hat den Wert " << reelle_zahl_long;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(reelle_zahl_long) << " Byte im Hauptspeicher reserviert." << endl;
}
```

Datentypen_1.cpp

```cpp
#include <iostream>                                 // Ein- und Ausgabebibliothek
using namespace std;                                // Benutze den Namensraum std

int main(){                                         // Hauptfunktion
    bool b;                                         // Deklaration der boolschen Variable 'b'
    char zeichen;                                   // Deklaration der char Variable 'zeichen'
    int ganze_zahl;                                 // Deklaration der ganzen Zahl Variable 'ganze_zahl'
    unsigned int natueliche_zahl;                   // Deklaration der natürlichen Zahl Variable 'natueliche_zahl'
    short kurze_ganze_zahl;                         // Deklaration der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    long lange_ganze_zahl;                          // Deklaration der langen ganzen Zahl Variable 'lange_ganze_zahl'
    float reelle_zahl;                              // Deklaration der reellen Zahl Variable 'reelle_zahl'
    double reelle_zahl_doppeltgenau;                // Deklaration der reellen Zahl Variable 'reelle_zahl_doppeltgenau' mit doppelter Genauigkeit
    long double reelle_zahl_long;                   // Deklaration der reellen Zahl Variable 'reelle_zahl_long' mit 19 Stellen

    b = "True";                                     // Initialisierung (Festlegung des Wertes) der boolschen Variable 'b'
    zeichen = 'G';                                  // Initialisierung der char Variable 'zeichen'
    ganze_zahl = -30256;                            // Initialisierung der ganzen Zahl Variable 'ganze_zahl'
    natueliche_zahl = 3278978;                      // Initialisierung der natürlichen Zahl Variable 'natueliche_zahl'
    kurze_ganze_zahl = 245;                         // Initialisierung der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    lange_ganze_zahl = 327244582478947248;          // Initialisierung der langen ganzen Zahl Variable 'lange_ganze_zahl'
    reelle_zahl = 2.4573;                           // Initialisierung der reellen Zahl Variable 'reelle_zahl'
    reelle_zahl_doppeltgenau = 2453.4573545742;     // Initialisierung der reellen Zahl Variable 'reelle_zahl_doppeltgenau'
    reelle_zahl_long = 7653.3467587475842L;         // Initialisierung der reellen Zahl Variable 'reelle_zahl_long'

    cout << "Die Variable 'b' hat den Wert " << b;                  // Ausgabe eines Textes im Terminal
    cout << " und für sie wurde ein Speicherplatz von ";            // ...
    cout << sizeof(b) << " Byte im Hauptspeicher reserviert." << endl;
    cout << "Die Variable 'zeichen' hat den Wert " << zeichen;
```

Es werden zunächst diverse Variablen unterschiedlicher Datentypen deklariert (bool b; , char zeichen; , ...) und danach wird ihnen ein ihnen spezifischer Wert zugewiesen - eine sogenannte *Initialisierung* der Variablen (b = "True"; , zeichen = 'G'; , ..., reelle_zahl_long = 7653.3467587475842L;), wobei das L am Ende des Initialisierungswertes der long double Variable "reelle_zahl_long" bedeutet, dass es sich um einen long double Initialisierungswert handelt. In den folgenden Zeilen des Quelltextes werden dann die Werte der einzelnen, nun definierten Variablen und ihr im Hauptspeicher reservierter Speicherplatz in Byte ausgegeben. Der Speicherplatz einer Variable wird hierbei mittels des Befehls **sizeof**(*Variablenname*) ermittelt.

Datentypen_1.cpp

```cpp
#include <iostream>                                 // Ein- und Ausgabebibliothek
using namespace std;                                // Benutze den Namensraum std

int main(){                                         // Hauptfunktion
    bool b;                                         // Deklaration der boolschen Variable 'b'
    char zeichen;                                   // Deklaration der char Variable 'zeichen'
    int ganze_zahl;                                 // Deklaration der ganzen Zahl Variable 'ganze_zahl'
    unsigned int natueliche_zahl;                   // Deklaration der natürlichen Zahl Variable 'natueliche_zahl'
    short kurze_ganze_zahl;                         // Deklaration der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    long lange_ganze_zahl;                          // Deklaration der langen ganzen Zahl Variable 'lange_ganze_zahl'
    float reelle_zahl;                              // Deklaration der reellen Zahl Variable 'reelle_zahl'
    double reelle_zahl_doppeltgenau;                // Deklaration der reellen Zahl Variable 'reelle_zahl_doppeltgenau' mit doppelter Genauigkeit
    long double reelle_zahl_long;                   // Deklaration der reellen Zahl Variable 'reelle_zahl_long' mit 19 Stellen

    b = "True";                                     // Initialisierung (Festlegung des Wertes) der boolschen Variable 'b'
    zeichen = 'G';                                  // Initialisierung der char Variable 'zeichen'
    ganze_zahl = -30256;                            // Initialisierung der ganzen Zahl Variable 'ganze_zahl'
    natueliche_zahl = 3278978;                      // Initialisierung der natürlichen Zahl Variable 'natueliche_zahl'
    kurze_ganze_zahl = 245;                         // Initialisierung der kurzen ganzen Zahl Variable 'kurze_ganze_zahl'
    lange_ganze_zahl = 327244582478947248;          // Initialisierung der langen ganzen Zahl Variable 'lange_ganze_zahl'
    reelle_zahl = 2.4573;                           // Initialisierung der reellen Zahl Variable 'reelle_zahl'
    reelle_zahl_doppeltgenau = 2453.4573545742;     // Initialisierung der reellen Zahl Variable 'reelle_zahl_doppeltgenau'
    reelle_zahl_long = 7653.3467587475842L;         // Initialisierung der reellen Zahl Variable 'reelle_zahl_long'

    cout << "Die Variable 'b' hat den Wert " << b;                  // Ausgabe eines Textes im Terminal
    cout << " und für sie wurde ein Speicherplatz von ";            // ...
    cout << sizeof(b) << " Byte im Hauptspeicher reserviert." << endl;
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ls
Datentypen_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ g++ Datentypen_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ls
a.out  Datentypen_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ./a.out
Die Variable 'b' hat den Wert 1 und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'zeichen' hat den Wert G und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'ganze_zahl' hat den Wert -30256 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'natueliche_zahl' hat den Wert 3278978 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'kurze_ganze_zahl' hat den Wert 245 und für sie wurde ein Speicherplatz von 2 Byte im Hauptspeicher reserviert.
Die Variable 'lange_ganze_zahl' hat den Wert 327244582478947248 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl' hat den Wert 2.4573 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_doppeltgenau' hat den Wert 2453.46 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_long' hat den Wert 7653.35 und für sie wurde ein Speicherplatz von 16 Byte im Hauptspeicher reserviert.
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ls
a.out  Datentypen_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ./a.out
Die Variable 'b' hat den Wert 1 und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'zeichen' hat den Wert G und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'ganze_zahl' hat den Wert -30256 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'natueliche_zahl' hat den Wert 3278978 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'kurze_ganze_zahl' hat den Wert 245 und für sie wurde ein Speicherplatz von 2 Byte im Hauptspeicher reserviert.
Die Variable 'lange_ganze_zahl' hat den Wert 327244582478947248 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl' hat den Wert 2.4573 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_doppeltgenau' hat den Wert 2453.46 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_long' hat den Wert 7653.35 und für sie wurde ein Speicherplatz von 16 Byte im Hauptspeicher reserviert.
```

Die Ausgabe nach dem Ausführen des Programms ist meist korrekt, jedoch erkennt man bei den letzten beiden Ausgabezeilen, dass die ausgegebenen Werte der Variablen "reelle_zahl_doppeltgenau" und "reelle_zahl_long" nicht den initialisierten Werten entsprechen. Dies hat jedoch lediglich mit den Eigenschaften des Ausgabebefehls "cout" zu tun, da standardmäßig nur eine begrenzte Anzahl von Stellen bei Werten von Variablen ausgegeben wird. Die Genauigkeit der cout-Ausgabe kann man mit dem Befehl "**std::cout**.precision(...);" verändern und im Programm Datentypen_1a.cpp wurde eine Zeile mit dem Befehl "**std::cout**.precision(40);" vor den cout-Befehlen dem Quelltext hinzugefügt und somit die Genauigkeit der cout-Ausgabe auf 40 festgelegt. Kompiliert man dieses Programm (beim Komilierungsprozess wird das alte ausführbare Programm "a.out" überschrieben, siehe unteres rechtes Bild)

```
reelle_zahl = 2.4573;
reelle_zahl_doppeltgenau = 2453.4573545742;
reelle_zahl_long = 7653.3467587475842L;
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ls
a.out  Datentypen_1a.cpp  Datentypen_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ g++ Datentypen_1a.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ls
a.out  Datentypen_1a.cpp  Datentypen_1.cpp
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ./a.out
Die Variable 'b' hat den Wert 1 und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'zeichen' hat den Wert G und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'ganze_zahl' hat den Wert -30256 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'natueliche_zahl' hat den Wert 3278978 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'kurze_ganze_zahl' hat den Wert 245 und für sie wurde ein Speicherplatz von 2 Byte im Hauptspeicher reserviert.
Die Variable 'lange_ganze_zahl' hat den Wert 327244582478947248 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl' hat den Wert 2.4572999477386474609375 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_doppeltgenau' hat den Wert 2453.4573545742000533209647983312606811152 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_long' hat den Wert 7653.346758747584200222036088234744966603 und für sie wurde ein Speicherplatz von 16 Byte im Hauptspeicher reserviert.
```

Der Zahlenraum $\mathbb{R}_c$ des Computers

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$ ./a.out
Die Variable 'b' hat den Wert 1 und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'zeichen' hat den Wert G und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'ganze_zahl' hat den Wert -30256 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'natueliche_zahl' hat den Wert 3278978 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'kurze_ganze_zahl' hat den Wert 245 und für sie wurde ein Speicherplatz von 2 Byte im Hauptspeicher reserviert.
Die Variable 'lange_ganze_zahl' hat den Wert 327244582478947248 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl' hat den Wert 2.4572999477386474609375 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_doppeltgenau' hat den Wert 2453.4573545742000533209647983312606811152 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'reelle_zahl_long' hat den Wert 7653.346758747584200222036088234744966603 und für sie wurde ein Speicherplatz von 16 Byte im Hauptspeicher reserviert.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/C-Programme$
```

Die ausgegebenen Werte der Variablen "reelle_zahl_doppeltgenau" und "reelle_zahl_long" stimmen nun gut mit den initialisierten Werte überein. Jedoch fällt auf, dass die ausgegebenen Werte nicht exakt mit den initialisierten übereinstimmen. Dies liegt daran, dass im Computer deklarierte Gleitkommazahlen einer Zahlenmenge entstammen (sagen wir $\mathbb{R}_c$), die nur eine echte Teilmenge der reellen Zahlen $\mathbb{R}$ ist $\mathbb{R}_c \subsetneq \mathbb{R}$. Die möglichen Gleitkommazahlen, die in einem C++ Programm definiert werden können, sind diskrete Werte mit einer begrenzten Anzahl von Nachkommastellen und die reellwertigen Zahlen, die sich zwischen zwei Maschinenzahlnachbarn befinden, können nicht exakt abgebildet werden. Der reellwertige Raum $\mathbb{R}$ der Mathematik kann somit mittels Computersimulationen nicht abgebildet werden und mathematische Begriffe wie z.B. ∞ und transzendente Zahlen wie z.B. die Kreiszahl π können mit dem Computer nicht exakt realisiert werden.

Datentypen_1.cpp

```cpp
#include <iostream>
using namespace std;

int main(){
    bool b;
    char zeichen;
    int ganze_zahl;
    unsigned int natueliche_zahl;
    short kurze_ganze_zahl;
    long lange_ganze_zahl;
    float reelle_zahl;
    double reelle_zahl_doppeltgenau;
    long double reelle_zahl_long;

    b = "True";
    zeichen = 'G';
    ganze_zahl = -30256;
    natueliche_zahl = 3278978;
    kurze_ganze_zahl = 245;
    lange_ganze_zahl = 327244582478947248;
    reelle_zahl = 2.4573;
    reelle_zahl_doppeltgenau = 2453.4573545742;
    reelle_zahl_long = 7653.34675874758L;

    cout << "Die Variable 'b' hat den Wert " << b;
    cout << " und für sie wurde ein Speicherplatz v
    cout << sizeof(b) << " Byte im Hauptspeicher re
```

Es können sogar nicht alle Brüche (Zahlen aus der Menge der rationalen Zahlen $\mathbb{Q} \subsetneq \mathbb{R}$) genau dargestellt werden und diese werden in einem C++ Programm nur mit einer gewissen Genauigkeit approximiert (z.B. 1/3=0.333...). Näheres finden Sie am Ende des noch folgenden Unterkapitels

Die Ein- und Ausgabe und im Unterkapitel Die Computerarithmetik und der Fehler in numerischen Berechnungen.

Initialisierung (Festlegung des numerischen Wertes) erfolgte mittels des Gleichheitszeichens, z.B. "reelle_zahl = 2.4575;". Man hätte diese Initialisierung auch direkt bei der Deklaration der Variable durchführen können und man spricht dann von einer Variablen-Definition. Man kann die gleichzeitige Deklaration und Initialisierung einer Variable auf unterschiedliche Arten schreiben und die folgenden drei Varianten Definieren alle die ganze Zahl 4 als Integer Typ

```
int ganze_zahl = 4;      oder    int ganze_zahl (4);      oder    int ganze_zahl {4};
```

,wobei die letzte Form eine universelle Art der Initialisierung darstellt, die auch bei Initialisierungslisten verwendet wird. Initialisiert man gleich bei der Deklaration, ist es nicht mehr nötig den Typ explizit anzugeben und man benutzt dafür dann den Bezeichner **auto** welcher beim Initialisierungsprozess den Typ automatisch festlegt. Im folgenden Programm (Datentyp_auto.cpp) werden die unterschiedlichen Initialisierungsschreibweisen und der Bezeichner **auto** benutzt:

Eine automatische Typfestlegung mittels **auto**

Datentyp_auto.cpp

```
#include <iostream>

using namespace std;                                    // Benutze den Namensraum std

int main(){                                             // Hauptfunktion
    int ganze_zahl_1 = 4;                               //Deklaration(Definition) einer ganzen Zahl und gleichzeitige Initialisierung mit dem Wert 4
    int ganze_zahl_2 (3);                               //Deklaration(Definition) einer zweiten ganzen Zahl und gleichzeitige Initialisierung mit dem
    int ganze_zahl_3 {5};                               //Deklaration(Definition) einer dritten ganzen Zahl und gleichzeitige Initialisierung mit dem

    auto zahl_1 (3);                                    // Definition einer Integer Zahl mittels auto
    auto zahl_2 (2.4568858679456457956);                // Definition einer langen Gleitkommazahl mittels auto
    auto zahl_3 (2.4568858679456457956L);               // Definition einer langen Gleitkommazahl mittels auto

    std::cout.precision(20);                                // Festlegung der Genauigkeit der cout-Ausgabe

    cout << "Die Variable 'ganze_zahl_1' hat den Wert " << ganze_zahl_1;          // Ausgabe eines Textes im Terminal
    cout << " und für sie wurde ein Speicherplatz von ";                         // ...
    cout << sizeof(ganze_zahl_1) << " Byte im Hauptspeicher reserviert." << endl;
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ g++ Datentyp_auto.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Die Variable 'ganze_zahl_1' hat den Wert 4 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher re
Die Variable 'ganze_zahl_2' hat den Wert 3 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher re
Die Variable 'ganze_zahl_1' hat den Wert 5 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher re
Die Variable 'zahl_1' hat den Wert 3 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'zahl_2' hat den Wert 2.4568858679456457 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
Die Variable 'zahl_3' hat den Wert 2.4568858679456457956 und für sie wurde ein Speicherplatz von 16 Byte im Hauptspeicher reserviert.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$
```

Man erkennt in der Terminalausgabe, dass mittels des Bezeichners **auto** die automatische Deklaration und Typfestlegung durch den Initialisierungsprozess bewerkstelligt wurde.

Konstante Datentypen

Es ist manchmal hilfreich, eine Variable im Programm zu definieren, die sich während der gesamten Berechnung nicht verändern soll. Eine solche konstante Variable, wie z.B. die Gravitationskonstante oder der Wert der Lichtgeschwindigkeit, wird im Programm mittels des Typzusatzes const gekennzeichnet. Dieser Zusatz kann vor jeden der Datentypen bei ihrer Definition geschrieben werden (z.B. const int N = 12;). Ein weiterer Ausdruck, der in ähnlicher Weise konstante Werte für den Compiler kennzeichnet, ist der Bezeichner **constexpr** und dieser wird ebenfalls als Zusatz vor den jeweiligen Typ bei der Deklaration geschrieben (z.B. constexpr double x = 1.4*N;). Der Wert der double-Variable x wird während des Kompilierungsprozesses zu dem konstanten Wert 1.4*N=16.8 initialisiert. An dieser Stelle wurde die Zahl 1.4 mit N multipliziert, wobei formal der arithmetische Operator der Multiplikation verwendet wurde. Auf die Arithmetik und Operatoren werden wir im nächsten Unterkapitel näher eingehen (siehe Arithmetik und Operatoren).

Arithmetik und Operatoren

Im vorigen Unterkapitel (Datentypen und Variablen) haben wir gelernt, wie man eine Variable nach ihrem Datentyp deklariert und mit einem nummerischen Wert initialisiert. Möchte man mit diesen Variablen Berechnungen durchführen, muss man zunächst geeignete Operatoren einführen, die die Variablen miteinander kombinieren (z.B. die Multiplikation zweier Variablen). Man unterscheidet hierbei die *arithmetischen Operatoren*, die *logischen Vergleichsoperatoren* und weitere *spezifische Operatoren*. Die folgende Tabelle listet diese Arten von Operatoren auf und verdeutlicht ihre Bedeutung mittels eines Beispiels.

Arithmetischen Operatoren

Operator	**Beispiel**
Addition	x + y
Vorzeichen +	+x
Subtraktion	x - y
Vorzeichen -	-x
Multiplikation	x * y
Division	x / y
Divisionsrest bei ganzen Zahlen	x % y

Logischen Vergleichsoperatoren

Operator	**Beispiel**
Gleich	x == y
Ungleich	x != y
Kleiner als	x < y
Größer als	x > y
Kleiner als oder gleich	x <= y
Größer als oder gleich	x >= y

Weitere spezifische Operatoren

Beispiel	**Bedeutung**
x += y	entspricht x = x+y
++x	entspricht x = x+1
x -= y	entspricht x = x-y
--x	entspricht x = x-1
x *= y	entspricht x = x*y
x /= y	entspricht x = x/y
x %= y	entspricht x = x%y

Die arithmetischen Operatoren sind weitgehend selbsterklärend, wobei der Divisionsrest bei ganzen Zahlen (x % y : x Modulo y) nur auf zwei ganze Zahlen angewendet werden darf. Die logischen Vergleichsoperatoren sind ebenfalls weitgehend selbsterklärend und diese werden in der nächsten Vorlesung, im Unterkapitel C++ Anweisungen: Auswahlanweisungen mit if und switch im Detail besprochen. Die in der dritten Tabelle aufgelisteten "Weiteren spezifischen Operatoren" sind allgemein nicht so geläufig, werden jedoch in der Programmierung häufig verwendet. Speziell der Inkrementierungsoperator ++x und Dekrementierungsoperator --x ist wichtig und diese werden in der nächsten Vorlesung, im Unterkapitel C++ Anweisungen: Die while- und for-Schleife häufig verwendet. In dem folgenden Quelltext des C++ Programms "Operatoren.cpp" wird die Verwendung einiger Operatoren (z.B. der Multiplikationsoperator "*", das unitäres Minus "-" (Minus Vorzeichen), der Inkrementierungsoperator "++x" oder der Divisionsrest (Modulo Operator %)) verwendet. Zusätzlich werden die, im vorigen Unterkapitel (Datentypen und Variablen) diskutierten Bezeichner **const**, **constexpr** und **auto** benutzt und auf die Verwendung von vordefinierten Funktionen der Standardbibliothek <cmath>

Arithmetik und Operatoren

Möchte man mit Datentypen und Variablen mathematische Berechnungen durchführen oder die Datenwerte miteinander vergleichen, ist es zunächst nötig eine Arithmetik (die zum Zählen oder Rechnen gehörige Kunst) zu definieren. Hierzu wurden in C++ geeignete Operatoren definiert, die dem Programmierer neben den arithmetischen Grundrechenarten (Multiplikation, Addition, ...) noch diverse weitere integrierte Operatoren zur Verfügung stellen. Man unterscheidet hierbei die *arithmetischen Operatoren*, die *logischen Vergleichsoperatoren* und weitere *spezifische Operatoren*.

Arithmetischen Operatoren

Operator	**Beispiel**
Addition	x + y
Vorzeichen +	+x
Subtraktion	x - y
Vorzeichen -	-x
Multiplikation	x * y
Division	x / y
Divisionsrest bei ganzen Zahlen	x % y

Logischen Vergleichsoperatoren

Operator	**Beispiel**
Gleich	x == y
Ungleich	x != y
Kleiner als	x < y
Größer als	x > y
Kleiner als oder gleich	x <= y
Größer als oder gleich	x >= y

Weitere spezifische Operatoren

Beispiel	**Bedeutung**
x += y	entspricht x = x+y
++x	entspricht x = x+1
x -= y	entspricht x = x-y
--x	entspricht x = x-1
x *= y	entspricht x = x*y
x /= y	entspricht x = x/y
x %= y	entspricht x = x%y

Die Standardbibliothek <cmath>

Zusätzlich sind einige wichtige mathematische Funktionen (Sinus, Cosinus, ..) in der Standardbibliothek <cmath> vordefiniert .

Operatoren.cpp

```cpp
#include <iostream>                                  // Ein- und Ausgabebibliothek
#include <cmath>                                     // Bibliothek für mathematisches (Sinus, e-Funktion, ...)

using namespace std;                                 // Benutze den Namensraum std

int main(){                                          // Hauptfunktion
    const int ganze_zahl_1 = 1;                      // Deklaration einer konstanten ganzen Zahl und Initialisierung mit 1
    const int ganze_zahl_2 = 3;                      // Deklaration einer konstanten zweiten ganzen Zahl und Initialisierung mit 3
    int i = 4;                                       // Deklaration einer integer Variable und Initialisierung mit 4
    double x, y, z;                                  // Deklaration dreier double Variablen

    constexpr auto zahl_1 = ganze_zahl_1 * ganze_zahl_2;                 // Deklaration eines konstanten Ausdruckes mittels auto
    constexpr auto zahl_2 = double(ganze_zahl_1) / double(ganze_zahl_2); // Deklaration eines konstanten Ausdruckes mittels auto

    ++i;                                             // Inkrementierung: i = i + 1

    x = 5.34 * (-zahl_1);                            // Multiplikationsoperator und unitäres Minus (Minus Vorzeichen)
    y = i * sin(x);                                  // Sinusfunktion von <cmath>
    z = 1.1 * pow(10,-ganze_zahl_2);                 // Potenzfunktion von <cmath> ( a^b := pow(a,b) )

    std::cout.precision(20);                         // Festlegung der Genauigkeit der cout-Ausgabe

    cout << "Die Variable 'zahl_1' hat den Wert " << zahl_1;          // Ausgabe eines Textes im Terminal
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(zahl_1) << " Byte im Hauptspeicher reserviert." << endl;

    cout << "Die Variable 'zahl_2' hat den Wert " << zahl_2;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(zahl_2) << " Byte im Hauptspeicher reserviert." << endl;

    cout << "i = " << i << endl;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;
    cout << "5 % 3 = " << i % ganze_zahl_2 << endl; // Divisionsrest (Modulo Operator)
}
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ g++ Operatoren.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Die Variable 'zahl_1' hat den Wert 3 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'zahl_2' hat den Wert 0.33333333333333331483 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
i = 5
x = -16.019999999999999574
y = 1.5349882533687984054
z = 0.0011000000000000000663
5 % 3 = 2
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$
```

Am Anfang der "main()"-Funktion werden zunächst zwei konstante ganzzahlige Variablen deklariert ("ganze_zahl_1" und "ganze_zahl_2"), die weiter unten bei der Definition der **constexpr** Ausdrücke der Variablen "zahl_1" und "zahl_2" verwendet werden. Durch den Bezeichner **auto** wird beim Initialisierungsprozess den Typ automatisch festlegt. Der durch den Multiplikationsoperator (zahl_1 = ganze_zahl_1 * ganze_zahl_2) erzeugte Wert ist ebenfalls wieder eine ganze Zahl, sodass der erzeugte Datentyp einer konstanten Integerzahl entspricht. Bei der Verwendung des Divisionsoperators muss man als Programmierer aufpassen, dass eine Typenumwandlung der Integerwerte stattfinden muss, da man sonst keine Gleitkommazahl, sondern eine Integerzahl beim Initialisierungsprozess erzeugen würde. Hier wurde die Typenumwandlung (von int nach double) wie folgt erzeugt "zahl_2 = double(ganze_zahl_1) / double(ganze_zahl_2)"; hierauf wird in einem späteren Unterkapitel genauer eingegangen. Der Inkrementierungsoperator wird auf die ganze Zahlenvariable "i" angewandt, die zuvor, bei ihrer Deklaration, auf den Wert 4 initialisiert wurde und somit nach der Inkrementierung den Wert 5 besitzt. Eine solche Inkrementierung hätte man z.B. mit der konstanten Variable "ganze_zahl_1" nicht machen können. Den deklarierten double-Variablen x, y und z werden mittels unterschiedlicher Operatoren verschiedene Werte zugewiesen. Am Ende des Programms werden einige Größen der definierten Variablen ausgegeben und auch der Modulo Operator "%" direkt bei der Ausgabe verwendet.

Operatoren.cpp

```cpp
#include <iostream>                                 // Ein- und Ausgabebibliothek
#include <cmath>                                    // Bibliothek für mathematisches (Sinus, e-Funktion, ...)

using namespace std;                                // Benutze den Namensraum std

int main(){                                         // Hauptfunktion
    const int ganze_zahl_1 = 1;                     // Deklaration einer konstanten ganzen Zahl und Initialisierung mit 1
    const int ganze_zahl_2 = 3;                     // Deklaration einer konstanten zweiten ganzen Zahl und Initialisierung mit 3
    int i = 4;                                      // Deklaration einer integer Variable und Initialisierung mit 4
    double x, y, z;                                 // Deklaration dreier double Variablen

    constexpr auto zahl_1 = ganze_zahl_1 * ganze_zahl_2;                 // Deklaration eines konstanten Ausdruckes mittels auto
    constexpr auto zahl_2 = double(ganze_zahl_1) / double(ganze_zahl_2); // Deklaration eines konstanten Ausdruckes mittels auto

    ++i;                                            // Inkrementierung: i = i + 1

    x = 5.34 * (-zahl_1);                           // Multiplikationsoperator und unitäres Minus (Minus Vorzeichen)
    y = i * sin(x);                                 // Sinusfunktion von <cmath>
    z = 1.1 * pow(10,-ganze_zahl_2);                // Potenzfunktion von <cmath> ( a^b := pow(a,b) )

    std::cout.precision(20);                        // Festlegung der Genauigkeit der cout-Ausgabe

    cout << "Die Variable 'zahl_1' hat den Wert " << zahl_1;          // Ausgabe eines Textes im Terminal
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(zahl_1) << " Byte im Hauptspeicher reserviert." << endl;

    cout << "Die Variable 'zahl_2' hat den Wert " << zahl_2;
    cout << " und für sie wurde ein Speicherplatz von ";
    cout << sizeof(zahl_2) << " Byte im Hauptspeicher reserviert." << endl;

    cout << "i = " << i << endl;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;
    cout << "5 % 3 = " << i % ganze_zahl_2 << endl; // Divisionsrest (Modulo Operator)
}
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ g++ Operatoren.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Die Variable 'zahl_1' hat den Wert 3 und für sie wurde ein Speicherplatz von 4 Byte im Hauptspeicher reserviert.
Die Variable 'zahl_2' hat den Wert 0.33333333333333331483 und für sie wurde ein Speicherplatz von 8 Byte im Hauptspeicher reserviert.
i = 5
x = -16.019999999999999574
y = 1.5349882533687984054
z = 0.0011000000000000000663
5 % 3 = 2
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$
```

Die Ein- und Ausgabe

In diesem Unterpunkt werden wir die Eingabe von Datenwerten durch den Benutzer eines ausführbaren C++ Programms vorstellen und die formatierte Ausgabe von Datenwerten im Terminalfenster näher betrachten. Die Programmiersprache C++ stellt mehrere integrierte Möglichkeiten der Benutzereingabe und Ausgabe bereit. Der zur cout-Ausgabe korrespondierende Eingabebefehl lautet "cin" und das folgende kleine Programm zeigt die Verwendung dieses Befehls:

EingabeInt.cpp

```cpp
#include <iostream>                                    // Ein- und Ausgabebibliothek
using namespace std;                                   // Benutze den Namensraum std

int main(){                                            // Hauptfunktion
    int zahl;                                          // Deklaration der Integer Variable 'zahl'
    cout << "Geben Sie bitte eine ganze Zahl ein: ";   // Ausgabe eines Textes
    cin  >> zahl;                                      // Einlesen der Zahl mittels der Tastatur
    cout << "Das doppelte Ihrer Zahl ist: " << 2*zahl << endl; // Ausgabe eines Textes
}
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Geben Sie bitte eine ganze Zahl ein: 4
Das doppelte Ihrer Zahl ist: 8
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Geben Sie bitte eine ganze Zahl ein: 4.9
Das doppelte Ihrer Zahl ist: 8
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Geben Sie bitte eine ganze Zahl ein: 5.0
Das doppelte Ihrer Zahl ist: 10
```

Ein- und Ausgabe mittels "cin" und "cout"

Bevor man eine Zahleneingabe durch den Benutzer machen kann, muss die zugehörige Variable, in welcher die eingegebene Zahl gespeichert wird, deklarieren. In dem nebenstehenden Programm wird deshalb zunächst eine Integer Variable "zahl" deklariert. Der Benutzer wird dann aufgefordert eine ganze Zahl einzugeben und mittels der Zeile "cin >> zahl;" wird der vom Benutzer eingegebene Zahlwert in der Variable "zahl" gespeichert. Falls der Benutzer hier eine 'falsche', dem Typ der Variable nicht entsprechende Eingabe macht (z.B. eine Gleitkommazahl eingibt), geschieht automatisch eine Typumwandlung (implizite Typkonvertierung). Am Ende wird dann das doppelte der eingegebenen Zahl im Terminal ausgegeben.
Die nebenstehende Abbildung verdeutlicht die Funktionsweise des Programms und zeigt eine dreimalige Benutzung des kompilierten ausführbaren Programms im Linux-Terminal. Beim ersten Ausführen des Programms gibt der Benutzer die ganze Zahl '4' ein und das doppelte dieser Zahl wird richtig ausgegeben. Beim zweiten Ausführen jedoch gibt der Benutzer die Gleitkommazahl '4.9' ein und Intern geschieht eine implizite Typkonvertierung, die effektiv zu einem Abrunden der eingegebenen Zahl führt, sodass die Ausgabe des doppelten Wertes der Zahl wieder '8' ist (näheres zur impliziten und expliziten Typkonvertierung wird später im Unterpunkt Typkonvertierung... besprochen).

In diesem Unterpunkt werden wir und die Eingabe von Zahlenwerten durch den Benutzer vorstellen und die formatierte Ausgabe von Datenwerten im Terminalfenster näher betrachten. Die Programmiersprache C++ stellt mehrere integrierte Möglichkeiten der Benutzereingabe und Ausgabe bereit und es werden im speziellen die Eingabe mittels "cin" und "scanf(...)" und die Ausgabe mittels "cout" und "printf(...)" betrachtet.

Eingabe mittels "cin" und Ausgabe mittels "cout"

Bevor man eine Zahleneingabe durch den Benutzer machen kann, muss die zugehörige Variable, in welcher die eingegebene Zahl gespeichert wird, deklariert werden.

In dem nebenstehenden Programm wird deshalb zunächst eine Integer Variable "zahl" deklariert. Der Benutzer wird dann aufgefordert eine ganze Zahl einzugeben und mittels der Zeile "cin >> zahl;" wird der vom Benutzer eingegebene Zahlwert in der Variable "zahl" gespeichert.

EingabeInt.cpp

```cpp
#include <iostream>                                          // Ein- und Ausgabebibliothek
using namespace std;                                         // Benutze den Namensraum std

int main(){                                                  // Hauptfunktion
    int zahl;                                                // Deklaration der Integer Variable 'zahl'
    cout << "Geben Sie bitte eine ganze Zahl ein: ";         // Ausgabe eines Textes
    cin  >> zahl;                                            // Einlesen der Zahl mittels der Tastatur
    cout << "Das doppelte Ihrer Zahl ist: " << 2*zahl << endl; // Ausgabe eines Textes
}
```

Falls der Benutzer hier eine 'falsche', dem Typ der Variable nicht entsprechende Eingabe macht (z.B. eine Gleitkommazahl eingibt), geschieht automatisch eine Typumwandlung (implizite Typkonvertierung). Am Ende wird dann das doppelte der eingegebenen Zahl im Terminal mittels „cout" ausgegeben.

Die nebenstehende Abbildung verdeutlicht die Funktionsweise des Programms und zeigt eine dreimalige Benutzung des kompilierten ausführbaren Programms im Linux-Terminal. Beim ersten Ausführen des Programms gibt der Benutzer die ganze Zahl '4' ein und das doppelte dieser Zahl wird richtig ausgegeben. Beim zweiten Ausführen jedoch gibt der Benutzer die Gleitkommazahl '4.9' ein und Intern geschieht eine implizite Typkonvertierung, die effektiv zu einem Abrunden der eingegebenen Zahl führt, sodass die Ausgabe des doppelten Wertes der Zahl wieder '8' ist.

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Geben Sie bitte eine ganze Zahl ein: 4
Das doppelte Ihrer Zahl ist: 8
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Geben Sie bitte eine ganze Zahl ein: 4.9
Das doppelte Ihrer Zahl ist: 8
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Geben Sie bitte eine ganze Zahl ein: 5.0
Das doppelte Ihrer Zahl ist: 10
```

Näheres zur impliziten und expliziten Typkonvertierung wird in einer noch folgenden Vorlesung besprochen.

Eingabe mittels „scanf(...)" und Ausgabe mittels „printf(...)"

Die Ein- und Ausgabebefehle "cin" und "cout", die wir im vorigen Programm benutzt hatten, können auch mittels der Ausgabefunktionen "scanf(...)" und "printf(...)" programmiert werden. Die in der C-Standardbibliothek definierte Ein- und Ausgabefunktionen "scanf(...)" und "printf(...)" werden auch oft in C++ Programmen verwendet und im weiteren Verlauf der Vorlesung werden wir die Terminalausgabe auch meist mittels des Befehls "printf(...)" machen.

Das auf der rechten Seite abgebildete Programm stellt die entsprechende scanf-printf-Version des vorigen Programms EingabeInt.cpp dar. Diesmal wird die vom Benutzer eingegebene Zahl mittels der Zeile "scanf("%i", &zahl);" eingelesen und in der Variable "zahl" gespeichert; oder genauer unter der Adresse der Variable "zahl" ("&zahl") abgelegt.

EingabeInt_1.cpp

```
#include <iostream>                                    // Ein- und Ausgabebibliothek

int main(){                                            // Hauptfunktion
    int zahl;                                          // Deklaration der Integer Variable 'zahl'
    printf("Geben Sie bitte eine ganze Zahl ein: ");   // Ausgabe eines Textes
    scanf("%i", &zahl);                                // Einlesen der Zahl mittels der Tastatur
    printf("Das doppelte Ihrer Zahl ist: %i \n",2*zahl);   // Ausgabe eines Textes
}
```

Das Symbol & welches vor der Variable steht, ist der sogenannte Adressenoperator (näheres siehe „Adressen, Zeiger und Referenzen"). Der erste Eintrag in der scanf-Funktion ("%i") spezifiziert das Format der zu erwartenden Eingabe und beginnt stets mit dem Prozentzeichen %, wobei %i eine Integerzahl spezifiziert. Die gleiche Vorgehensweise wird danach bei der printf-Funktion verwendet, wobei nun an der Stelle %i das Resultat von 2*zahl ausgegeben wird.

Die möglichen, unterschiedlichen Formatspezifikationen findet man in den Standardbibliotheken von scanf und printf (siehe auch wikipedia zu scanf und wikipedia zu printf).

Printf.cpp

```cpp
#include <iostream>                                  // Ein- und Ausgabebibliothek

int main(){                                          // Hauptfunktion
    bool b = "True";                                 // Definition der boolschen Variable 'b'
    char zeichen = 'G';                              // Definition der char Variable 'zeichen'
    int ganze_zahl = -30256;                         // Definition der int Variable 'ganze_zahl'
    float reelle_zahl = 2.4573;                      // Definition der float Variable 'reelle_zahl'
    double reelle_zahl_doppeltgenau = 2453.4573545742;  // Definition der double Variable 'reelle_zahl_doppeltgenau'
    long double reelle_zahl_long = 7653.3467587475842L; // Definition der long double Variable 'reelle_zahl_long'

    printf("Die Variable 'b' hat den Wert %i und für sie wurde ein Speicherplatz von %li Byte im Hauptspeicher reserviert. \n",b ,sizeof(b));
    printf("Die Variable 'zeichen' hat den Wert %c und für sie wurde ein Speicherplatz von %li Byte im Hauptspeicher reserviert. \n \n",zeichen ,sizeof(zeichen));

    printf("Die drei reellen Variablen Werte lauten: \n");
    printf("  reelle_zahl    reelle_zahl_doppeltgenau    reelle_zahl_long \n");
    printf("%13.4f %27.10f %19.13Lf \n",reelle_zahl, reelle_zahl_doppeltgenau, reelle_zahl_long);

    printf("\n... und in exponentieller Schreibweise: \n");
    printf("  reelle_zahl    reelle_zahl_doppeltgenau    reelle_zahl_long \n");
    printf("%13.4e %27.10e %19.13Le \n",reelle_zahl, reelle_zahl_doppeltgenau, reelle_zahl_long);
}
```

Es werden zunächst diverse Variablen unterschiedlicher Datentypen deklariert (bool b; , char zeichen; , ...) und ihnen gleichzeitig ein spezifischer Wert zugewiesen (Initialisierung). Der Wert der boolschen Variable 'b' wird als Integer Wert ausgegeben (%i), wobei durch die zuvor erfolgte "True"-Initialisierung diese Variable den Wert "1" haben wird. Es wird zusätzlich der reservierte Speicherplatz dieser Variable in Byte ausgegeben. Da der zurückgegebene Typ der durch "**sizeof**(b)" ermittelten Speicherplatzbelegung "long unsigned int" ist, muss man bei der Ausgabe den Formatbezeichner %li verwenden.

Danach wird der Wert der Variable 'zeichen' mittels %c ausgegeben und am Ende zwei Zeilenumbrüche verwendet (\n \n). Die drei reellwertigen Zahlenvariablen werden dann in einer speziellen Formatierung ausgegeben: printf("%13.4f %27.10of %19.13Lf \n",reelle_zahl, reelle_zahl_doppeltgenau, reelle_zahl_long);

So wird z.B. für die float-Variable "reelle_zahl" der Formatspezifizierer %13.4f verwendet. Das f am Ende des Ausdruckes kennzeichnet, dass es sich um eine Gleitkommazahl handelt, die auch in Komma-Schreibweise ausgegeben werden soll. Die davor geschriebene Zahl (13.4) bedeutet, dass 13 Leerzeichen für die Ausgabe reserviert und 4 Nachkommastellen ausgegeben werden sollen. Die Anzahl der reservierten Leerzeichen wurde so gewählt, dass der Wert der Zahl gerade rechtsbündig unter dem in der vorigen Zeile ausgegebenen Variablennamen stehen soll. In gleicher Weise wurden die zweite und dritte reelle Zahl ausgegeben, wobei hier eine unterschiedliche Anzahl der Nachkommastellen gewählt wurde und der Formatbezeichner bei der Variable "reelle_zahl_long" als eine 'lange' Gleitkommazahl Lf spezifiziert wurde, da ihr Typ long double ist. Die letzten drei printf(...) Anweisungen stellen ebenfalls die Werte der drei reellwertigen Variablen dar, dies jedoch in einer exponentiellen Darstellung (mittels e und Le).

Printf.cpp

```cpp
#include <iostream>

int main(){
    bool b = "True";
    char zeichen = 'G';
    int ganze_zahl = -30256;
    float reelle_zahl = 2.4573;
    double reelle_zahl_doppeltgenau = 2453.4573545742;
    long double reelle_zahl_long = 7653.3467587475842L;

    printf("Die Variable 'b' hat den Wert %i und für sie
    printf("Die Variable 'zeichen' hat den Wert %c und für sie wurde ein Speicherplatz von %li Byte im Hauptspeicher reserviert. \n \n",zeichen ,sizeof(zeichen));

    printf("Die drei reellen Variablen Werte lauten: \n");
    printf("  reelle_zahl    reelle_zahl_doppeltgenau    reelle_zahl_long \n");
    printf("%13.4f %27.10f %19.13Lf \n",reelle_zahl, reelle_zahl_doppeltgenau, reelle_zahl_long);

    printf("\n... und in exponentieller Schreibweise: \n");
    printf("  reelle_zahl    reelle_zahl_doppeltgenau    reelle_zahl_long \n");
    printf("%13.4e %27.10e %19.13Le \n",reelle_zahl, reelle_zahl_doppeltgenau, reelle_zahl_long);
}
```

Formatspezifizierer %13.4f

Das f am Ende des Ausdruckes kennzeichnet, dass es sich um eine Gleitkommazahl handelt, die auch in Komma-Schreibweise ausgegeben werden soll. Die davor geschriebene Zahl (13.4) bedeutet, dass 13 Leerzeichen für die Ausgabe reserviert und 4 Nachkommastellen ausgegeben werden sollen.

In diesem Programm werden einige Formatspezifikationen und Anwendungen der Ausgabefunktion "printf(...)" dargestellt:

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ g++ Printf.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Die Variable 'b' hat den Wert 1 und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.
Die Variable 'zeichen' hat den Wert G und für sie wurde ein Speicherplatz von 1 Byte im Hauptspeicher reserviert.

Die drei reellen Variablen Werte lauten:
  reelle_zahl    reelle_zahl_doppeltgenau    reelle_zahl_long
       2.4573              2453.4573545742  7653.3467587475842

... und in exponentieller Schreibweise:
  reelle_zahl    reelle_zahl_doppeltgenau    reelle_zahl_long
   2.4573e+00            2.4534573546e+03 7.6533467587476e+03
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$
```

Das Template <limits> der Standardbibliothek

Am Ende dieser Vorlesung möchte ich Ihnen noch eine hilfreiche Bibliothek vorstellen, die die numerischen Limitierungen bei numerischen Berechnungen in Form eines Templates bereitstellt (was ein Template eigentlich ist, werden wir erst später kennenlernen). Im Unterpunkt Datentypen und Variablen hatten wir gesehen, dass die Zahlenmenge $\mathbb{R}_C$ der mit einem C++ Programm abbildbaren Zahlen von ihrem grundsätzlichen Wesen her eine diskrete Abfolge von Zahlen und somit nicht-kontinuierlich ist. Die numerischen Grenzen, und somit der eigentliche Zahlenraum $\mathbb{R}_C$, hängt von dem gewählten Datentyp der definierten Variable ab. Die für die jeweiligen Typen definierten numerischen Grenzwerte sind in den Spezialisierungen des Templates "numeric_limits" enthalten, das sich in der Standardbibliothek <limits> befindet. Mittels "numeric_limits" lassen sich Fragen wie "Was ist die größte int-Zahl?" oder "Was ist die kleinste positive double-Zahl?" und vieles mehr beantworten (siehe Template "numeric_limits").

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ g++ NumLim.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Einige Grenzwerte des Datentyps int:
      Groesste Zahl        Kleinste Zahl
         2147483647          -2147483648

Einige Grenzwerte des Datentyps long:
           Groesste Zahl             Kleinste Zahl
     9223372036854775807      -9223372036854775808

Einige Grenzwerte des Datentyps double:
                Groesste Zahl                  Kleinste Zahl        Kleinste positive Zahl    Anzahl der Nachkommastellen            Maschinen epsilon
       1.797693134862316e+308        -1.797693134862316e+308        2.225073858507201e-308                             15         2.220446049250313e-16

Einige Grenzwerte des Datentyps long double:
                Groesste Zahl                  Kleinste Zahl        Kleinste positive Zahl    Anzahl der Nachkommastellen            Maschinen epsilon
   1.189731495357231765e+4932    -1.189731495357231765e+4932     3.362103143112093506e-4932                             18       1.084202172485504434e-19
```

NumLim.cpp

```cpp
// Numerische Grenzwerte des Templates "numeric_limits"
#include <iostream>     // Ein- und Ausgabebibliothek
#include <limits>       // Bibliothek mit std::numeric_limits
using namespace std;    // Benutze den Namensraum std

int main(){
    printf("Einige Grenzwerte des Datentyps int: \n");
    printf("%20s %20s \n","Groesste Zahl","Kleinste Zahl");
    printf("%20i %20i \n\n",numeric_limits<int>::max(),numeric_limits<int>::min());

    printf("Einige Grenzwerte des Datentyps long: \n");
    printf("%25s %25s \n","Groesste Zahl","Kleinste Zahl");
    printf("%25li %25li \n\n",numeric_limits<long>::max(),numeric_limits<long>::min());

    printf("Einige Grenzwerte des Datentyps double: \n");
    printf("%30s %30s %30s %30s %30s \n","Groesste Zahl","Kleinste Zahl","Kleinste positive Zahl","Anzahl der Nachkommastellen","Maschinen epsilon");
    printf("%30.15e %30.15e %30.15e ",numeric_limits<double>::max(),numeric_limits<double>::lowest(),numeric_limits<double>::min());
    printf("%30i %30.15e \n\n",numeric_limits<double>::digits10,numeric_limits<double>::epsilon());

    printf("Einige Grenzwerte des Datentyps long double: \n");
    printf("%30s %30s %30s %30s %30s \n","Groesste Zahl","Kleinste Zahl","Kleinste positive Zahl","Anzahl der Nachkommastellen","Maschinen epsilon");
    printf("%30.18Le %30.18Le %30.18Le ",numeric_limits<long double>::max(),numeric_limits<long double>::lowest(),numeric_limits<long double>::min());
    printf("%30i %30.18Le \n",numeric_limits<long double>::digits10,numeric_limits<long double>::epsilon());
}
```

Das C++ Programm stellt ein Anwendungsbeispiel des Templates <limits> dar und verwendet zusätzlich einen hilfreichen Trick der formatierten Ausgabe mittels der "printf(...)"-Funktion.

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ g++ NumLim.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/V1$ ./a.out
Einige Grenzwerte des Datentyps int:
       Groesste Zahl        Kleinste Zahl
          2147483647          -2147483648

Einige Grenzwerte des Datentyps long:
           Groesste Zahl              Kleinste Zahl
      9223372036854775807       -9223372036854775808

Einige Grenzwerte des Datentyps double:
                Groesste Zahl                 Kleinste Zahl        Kleinste positive Zahl    Anzahl der Nachkommastellen            Maschinen epsilon
       1.797693134862316e+308       -1.797693134862316e+308       2.225073858507201e-308                             15         2.220446049250313e-16

Einige Grenzwerte des Datentyps long double:
                Groesste Zahl                 Kleinste Zahl        Kleinste positive Zahl    Anzahl der Nachkommastellen            Maschinen epsilon
    1.189731495357231765e+4932    -1.189731495357231765e+4932    3.362103143112093506e-4932                             18      1.084202172485504434e-19
```

NumLim.cpp

```cpp
// Numerische Grenzwerte des Templates "numeric_limits"
#include <iostream>     // Ein- und Ausgabebibliothek
#include <limits>       // Bibliothek mit std::numeric_limits
using namespace std;    // Benutze den Namensraum std

int main(){
    printf("Einige Grenzwerte des Datentyps int: \n");
    printf("%20s %20s \n","Groesste Zahl","Kleinste Zahl");
    printf("%20i %20i \n\n",numeric_limits<int>::max(),numeric_limits<int>::min());

    printf("Einige Grenzwerte des Datentyps long: \n");
    printf("%25s %25s \n","Groesste Zahl","Kleinste Zahl");
    printf("%25li %25li \n\n",numeric_limits<long>::max(),numeric_limits<long>::min());

    printf("Einige Grenzwerte des Datentyps double: \n");
    printf("%30s %30s %30s %30s %30s \n","Groesste Zahl","Kleinste Zahl","Kleinste positive Zahl","Anzahl der Nachkommastellen","Maschinen epsilon");
    printf("%30.15e %30.15e %30.15e ",numeric_limits<double>::max(),numeric_limits<double>::lowest(),numeric_limits<double>::min());
    printf("%30i %30.15e \n\n",numeric_limits<double>::digits10,numeric_limits<double>::epsilon());

    printf("Einige Grenzwerte des Datentyps long double: \n");
    printf("%30s %30s %30s %30s %30s \n","Groesste Zahl","Kleinste Zahl","Kleinste positive Zahl","Anzahl der Nachkommastellen","Maschinen epsilon");
    printf("%30.18Le %30.18Le %30.18Le ",numeric_limits<long double>::max(),numeric_limits<long double>::lowest(),numeric_limits<long double>::min());
    printf("%30i %30.18Le \n",numeric_limits<long double>::digits10,numeric_limits<long double>::epsilon());
}
```

Das Maschinen-Epsilon

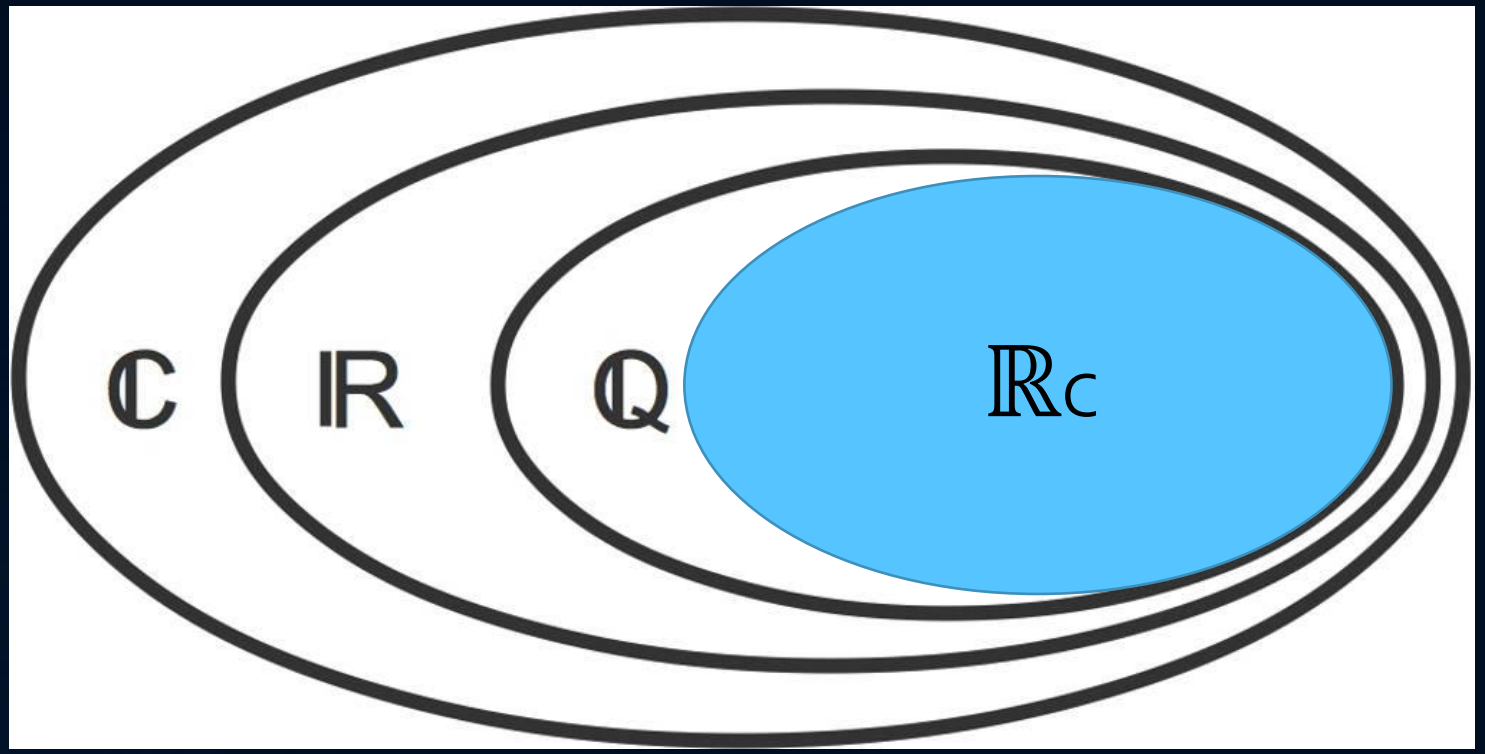
ϵ_M

Es werden einige Grenzwerte des Datentyps int, long, double und long double ausgegeben. Im Speziellen wird für die ganzzahligen Datentypen die größte und kleinste Zahl und für die reellwertigen Datentypen die größte und kleinste Zahl, die kleinste positive Zahl, die Anzahl der Nachkommastellen und das sogenannte "Maschinen-Epsilon" ausgegeben. Das Maschinen-Epsilon ϵ_M eines Gleitkommazahlen-Typs stellt die Differenz zwischen 1 und dem kleinsten darstellbaren Wert größer als 1 dar und somit ist der Wert $(1+\epsilon_M)$ die nächstliegende Zahl, die größer als 1 ist. Die Ausgabe mittels der "printf(...)"-Funktion benutzt hier einen hilfreichen Formatierungstrick um die dargestellten Zahlenwerte rechtsbündig unter ihre Beschreibung zu platzieren. Hierbei wird für die ausgegebene Zeile der Beschreibung die gleiche Anzahl von ausgegebenen Leerstellen reserviert, wie bei der Ausgabezeile der Datenwerte. Z.B. wird bei der Beschreibung der ganzzahligen Grenzwerte der Ausgabebefehl

"printf("%20s %20s \n","Groesste Zahl","Kleinste Zahl");"

benutzt. Mittels des Formatbezeichners %20s werden 20 Leerstellen reserviert und es wird der String der eigentlichen Bezeichnung separat nach dem Komma aufgelistet.

Der Zahlenraum $\mathbb{R}_C$ des Computers



Visualisierung der diskreten Zahlenmenge mithilfe dessen der Computer Berechnungen durchführt

In gleicher Weise, wie es in der Mathematik die unterschiedlichen Zahlenmengen (z.B. $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{R}$, $\mathbb{C}$) gibt und man bei mathematischen Berechnungen den Wertebereich einer Variable oder Funktion vorher definieren muss, ist dies auch beim Programmieren in C++ nötig.

Übungsblatt Nr. 2

Aufgabe 1 *(10 Punkte)*

Die Eulersche Zahl e kann mittels des folgenden Grenzwertes der Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := \left(1+\frac{1}{n}\right)^n$ definiert werden:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

Überprüfen Sie dies, soweit es die Computergenauigkeit erlaubt, mittels eines C++ Programms.
Deklarieren Sie dazu zunächst zwei reellwertige Variablen der Datentypen float und double, die die zwei approximierte Werte der Eulerschen Zahl e darstellen sollen. Zusätzlich deklarieren Sie eine Variable für die lange natürliche Zahl n, die Sie mit einem großen ganzzahligen Wert initialisieren (z.B. $n = 10000$). Initialisieren Sie danach die zwei reellwertigen Variablen mit dem Zahlenwert der Folge $\left(1+\frac{1}{n}\right)^n$. [Bemerkung: Hierzu müssen Sie zuvor die Standardbibliothek <cmath> eingebunden haben, in der die Funktion "pow(...)" definiert ist (näheres siehe Die Standardbibliothek <cmath>).]
Vergleichen Sie am Ende des Programms die berechneten Zahlenwerte mittels einer formatierten Ausgabe. Geben Sie zusätzlich den Fehler ($\mathcal{F} = e - e_{approx}$) mit 20 Nachkommastellen aus. [Bemerkung: Benutzen Sie hierbei für den wirklichen, exakten Wert von e, den in <cmath> definierten long double Zahlenwert "M_El" (M_El ist auf 18 Nachkommastellen genau).]
Vergrößern Sie nun den Wert von n und betrachten Sie die berechneten Fehler - was fällt Ihnen auf? Wie genau können Sie die Zahl e mittels der definierten Variablen bestimmen?

Aufgabe 2 *(5 Punkte)*

Berechnen Sie die folgenden Ausdrücke mittels eines C++ Programms und lassen Sie sich die berechneten Werte auf 10 und 20 Stellen genau ausgeben (benutzen Sie bei der Berechnung doppelte Maschinengenauigkeit).

$$\frac{64}{128}, \quad \frac{2\pi}{5}, \quad \cos\left(\frac{\pi}{2}\right), \quad e^{\left(5.84^{-12}\right)}, \quad \ln\left(e^{1.1}\right)$$

Die Eulersche Zahl e kann mittels des folgenden Grenzwertes der Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := \left(1+\frac{1}{n}\right)^n$ definiert werden:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

Überprüfen Sie dies, soweit es die Computergenauigkeit erlaubt, mittels eines C++ Programms.
Deklarieren Sie dazu zunächst zwei reellwertige Variablen der Datentypen float und double, die die zwei approximierte Werte der Eulerschen Zahl e darstellen sollen. Zusätzlich deklarieren Sie eine Variable für die lange natürliche Zahl n, die Sie mit einem großen ganzzahligen Wert initialisieren (z.B. $n = 10000$). Initialisieren Sie danach die zwei reellwertigen Variablen mit dem Zahlenwert der Folge $\left(1+\frac{1}{n}\right)^n$. [Bemerkung: Hierzu müssen Sie zuvor die Standardbibliothek <cmath> eingebunden haben, in der die Funktion "pow(...)" definiert ist (näheres siehe Die Standardbibliothek <cmath>).]
Vergleichen Sie am Ende des Programms die berechneten Zahlenwerte mittels einer formatierten Ausgabe. Geben Sie zusätzlich den Fehler ($\mathcal{F} = e - e_{approx}$) mit 20 Nachkommastellen aus. [Bemerkung: Benutzen Sie hierbei für den wirklichen, exakten Wert von e, den in <cmath> definierten long double Zahlenwert "M_El" (M_El ist auf 18 Nachkommastellen genau).]
Vergrößern Sie nun den Wert von n und betrachten Sie die berechneten Fehler - was fällt Ihnen auf? Wie genau können Sie die Zahl e mittels der definierten Variablen bestimmen?

Aufgabe 2 *(5 Punkte)*

Berechnen Sie die folgenden Ausdrücke mittels eines C++ Programms und lassen Sie sich die berechneten Werte auf 10 und 20 Stellen genau ausgeben (benutzen Sie bei der Berechnung doppelte Maschinengenauigkeit).

$$\frac{64}{128}, \quad \frac{2\pi}{5}, \quad \cos\left(\frac{\pi}{2}\right), \quad e^{\left(5.84^{-12}\right)}, \quad \ln\left(e^{1.1}\right)$$

A2_3F.cpp

```cpp
// Aufgabe 3 des Übungsblattes Nr.2
// Drei Fehler und viele unschöne Ausdrücke
#include <iostream>
#include <cmath>

int main(){
    int n;
    n = 0;
    double a = 0;
    a = pow(2,n);
    cout < a << endl;
    n = n + 1;
    a = pow(n,2);
    cout << a << endl;
    n = n + 1;
    a = pow(n,2);
    cout << a << endl;
    n = n + 1;
    a = pow(n,2);
    cout << a << endl;
}
```

Aufgabe 3 *(5 Punkte)*

Das neben stehende C++ Programm soll die Zahlenwerte der Folge $(a_n)_{n\in\mathbb{N}}$ mit $a_n := n^2$ für $n \in [0, 1, 2, 3]$ ausgeben. Leider hat der Programmierer drei Fehler eingebaut und zusätzlich besteht das Programm aus vielen, nicht optimal programmierten Ausdrücken. Welche der Zeilen ist falsch und muss korrigiert werden und welche Stellen sind unschön programmiert? Wie würde ihr entsprechendes Programm aussehen? [Bemerkung: Bitte verwenden Sie an dieser Stelle noch nicht die C++ Anweisungen der while- oder for-Schleifen; diese werden erst in der nächsten Vorlesung behandelt]

Die Musterlösung der Aufgaben des Übungsblatts Nr. 2 finden Sie unter dem folgenden Link:
Musterlösung Übungsblatt Nr. 2

Vorlesung 3

C++ Anweisungen: Die for-, while- und do-Schleifen

Möchte man als Programmierer ein Problem mittels eines C++ Programms lösen, so muss man dem Computer in Form von Anweisungen sagen, was er zu erledigen hat. In diesem Unterpunkt behandeln wir eine der wichtigsten Anweisungsarten, die sogenannten *Schleifenanweisungen*. Im Prinzip ist jede Programmzeile, die mit einem Semikolon endet, eine Anweisung an den Computer, jedoch stellen die *Schleifenanweisungen* eine besondere Art von iterativen Anweisungsprozessen dar, und sind ein oft verwendetes Hilfsmittel der prozeduralen Programmierung. Eine Schleifenanweisung kann als eine **for**-, **while**- oder **do**-Anweisung ausgedrückt werden (näheres siehe C++ Anweisungen: Die for-, while- und do-Schleifen).

Anwendungsbeispiel: Folgen und Reihen

Die im vorigen Unterpunkt besprochenen *Schleifenanweisungen* finden in vielen C++ Programmen ihre Anwendung. In diesem Unterpunkt wird ihre Anwendung im Bereich der mathematischen Folgen und Reihen diskutiert. Am Beispiel der konvergenten Folge der Eulersche Zahl e und der Leibniz-Reihe zur Berechnung der Kreiszahl π wird die Verwendung der while-Schleife vorgestellt. Das Umschreiben der Programme unter Verwendung einer for-Schleife ist Teil der Aufgabe 2 des (siehe Übungsblattes Nr. 3. Es wird unter anderem das Konvergenzverhalten der Folge für die ersten Folgenglieder untersucht und die von vom Programm ausgegebenen Werte visualisiert. Hierzu werden die ausgegebenen Daten in eine separate Datei umgeleitet und dann mittels Gnuplot bzw. Python-Matplotlib dargestellt (näheres siehe Anwendungsbeispiel: Folgen und Reihen).

Eine kleine Einführung in die Programmiersprache Python

Die Programmiersprache Python ist auch eine sehr gute Objekt-orientierte Programmiersprache und im Prinzip hätten wir die gesamte Vorlesung nur mittels Python gestalten können. Die in dieser Vorlesung behandelten Python-Skripte und Python Jupyter Notebooks werden jedoch lediglich zur Visualisierung von Daten und im Bereich der Illustration von mathematisch/physikalischen Gleichungen benutzt. Mittels des Python-Moduls "matplotlib" (siehe Matplotlib: Visualization with Python) können auf einem einfachen Weg Bilder and Animationen des zuvor mit C++ simulierten Systems erzeugt werden. Zusätzlich werden wir, die im nächsten Unterpunkt besprochene C++ Computerarithmetik mittels eines Python Jupyter Notebooks verdeutlichen und dabei die Verwendung von Listen, Arrays und for-Schleifen in der Programmiersprache Python kennenlernen (näheres siehe Eine kleine Einführung in die Programmiersprache Python).

Die Computerarithmetik und der Fehler in numerischen Berechnungen

In diesem Unterpunkt werden wir zwei Klassen von Fehlerquellen in numerischen Berechnungen besprechen, den sogenannten *Rundungsfehler*, der aufgrund der Computerarithmetik entsteht und der sogenannte *Approximierungsfehler* (*Abschneidefehler* bzw.'*Truncation error*'), der immer dann auftritt, wenn der Programmierer eine exakte mathematische Gleichung mittels approximativer Ausdrücke annähert (näheres siehe Die Computerarithmetik und der Fehler in numerischen Berechnungen).

Vorlesung 3

In dieser Vorlesung werden wir die wohl wichtigste Form von C++ Anweisungen, die sogenannten *Schleifenanweisungen*, kennenlernen. Die *Schleifenanweisungen* stellen einen iterativen Anweisungsprozess dar und können in Form von **for**-, **while**- oder **do**-Anweisung ausgedrückt werden. Möchte man z.B. die natürlichen Zahlen von Null bis 100 im Terminal ausgeben lassen, so kann man dies in einer einfachen Weise mittels einer Schleifenanweisung dem Computer sagen. Die Anwendung von Schleifenanweisung wird danach am Beispiel einer mathematischen Folge und Reihe diskutiert.
Die Visualisierung von Daten, die mittels C++ Programmen erstellt wurden, ist ein wichtiges Teilgebiet eines Programmierers im Bereich der Physik. In dieser Vorlesung werden diverse Python-Skripte und Python Jupyter Notebooks vorgestellt, die zur Visualisierung von Daten und im Bereich der Illustration von mathematisch/physikalischen Gleichungen benutzt werden. Mittels des Python-Moduls "matplotlib" (siehe Matplotlib: Visualization with Python) können auf einem einfachen Weg Bilder and Animationen des zuvor mit C++ simulierten Systems erzeugt werden.

Am Ende der Vorlesung kommen wir nochmals auf den, bereits im Unterkapitel Datentypen und Variablen angesprochenen Zahlenraum des Computers ($\mathbb{R}_C$) zurück und diskutieren die zwei wichtigsten Fehlerquellen bei nummerischen Berechnungen (*Rundungsfehler* und *Approximierungsfehler*). Der *Rundungsfehler* ist darin begründet, dass der Zahlenraum des Computers ($\mathbb{R}_C$), nur eine relativ kleine Teilmenge der reellen Zahlen benutzt und somit reellwertige Zahlen mit einer unendlichen Anzahl von Nachkommastellen nicht speichern kann. Diese Teilmenge $\mathbb{R}_C$ umfasst nur rationale Zahlen und speichert den gebrochenen Teil, der mit *Mantisse* bezeichnet wird, zusammen mit dem exponentiellen Teil, welchen man *Charakteristik* nennt als eine binäre Liste von Nullen und Einsen. Die zweite Klasse von nummerischen Fehlern, die *Approximierungsfehler* werden am Beispiel der approximativen Annäherung an die Kreiszahl π mittels der alternierenden Leibniz-Reihe besprochen. Bei dieser Art von Fehlern hat es der Programmierer selbst in der Hand, wie genau er in seinem Programm die Berechnung durchführen möchte (näheres siehe Die Computerarithmetik und der Fehler in numerischen Berechnungen).

Vorlesung 4

....

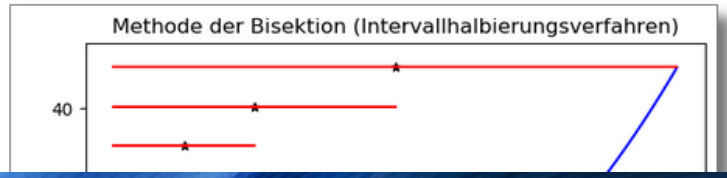
C++ Anweisungen: Auswahlanweisungen mit if und switch

Die Programmiersprache C++ bietet einen konventionellen und flexiblen Satz von Anweisungen. Im Prinzip ist jede Programmzeile, die mit einem Semikolon endet, eine Anweisung. In diesem Unterpunkt werden wir die sogenannten *Auswahlanweisungen* behandeln, wobei wir in der vorigen Vorlesung die *Schleifenanweisungen* vorstellt hatten (siehe C++ Anweisungen: Die for-, while- und do-Schleifen). *Auswahlanweisungen* werden auch als Verzweigung , Tests oder *bedingte Anweisungen* bezeichnet und sind immer dann anzuwenden, wenn das Programm bei einem gewissen Ereignis bzw. unter einer gewissen Bedingung etwas Bestimmtes tun soll. Es teilt somit das Programm in unterschiedliche Anweisungspfade auf. *Auswahlanweisungen* können in Form einer **if**-, (**if-else**)- oder **switch**-Anweisung ausgedrückt werden. C++ Anweisungen: Auswahlanweisungen mit if und switch

Funktionen in C++

Die Definition einer C++ Funktion ist im Grunde nichts Anderes, als eine Code-Block (*Anweisungsblock*) mit einem Funktionsnamen zu verbinden. C++ Funktionen werden außerhalb der main()-Hauptfunktion definiert und vereinfachen somit das Verständnis und die Lesbarkeit des Quelltextes. C++ Funktionen sind ein wichtiges Werkzeug, um den Quelltext eines Programms zu ordnen und wesentliche Algorithmen und zusammenhängende Anweisungsblöcke der main()-Hauptfunktion in einer zusammenhängenden Form auszulagern. Die Definition einer C++ Funktion besteht aus einer *Deklaration* und einem *Anweisungsblock* und sie ist der formalen Definition einer mathematischen Funktion nicht unähnlich: "Eine C++ Funktion ist eine Abbildung von dem Datenraum der *Argumentenliste* in den Datenraum des *Rückgabetyps*. Die dabei benutzte Abbildungsvorschrift findet sich in dem *Anweisungsblock* der Funktion. Der 'Rückgabe Typ' kann hierbei eine der schon besprochenen Datentypen (z.B. int oder doube) oder ein Daten-Array (siehe nächste Vorlesung) sein. Die *Argumentenliste* setzt sich aus einer Liste von Datentypen der formalen Argumente (Parameter) der Funktion zusammen, die jeweils mit einem Komma voneinander getrennt sind. In C++ hat das Wort Funktion eine allgemeinere Bedeutung als im Bereich der Mathematik und die in der Mathematik und Physik definierte Funktionen stellen eine echte Teilmenge der C++ Funktionen dar (näheres siehe Funktionen in C++).

Anwendungsbeispiel: Nullstellensuche einer Funktion



In diesem Unterpunkt möchten wir ein grundlegendes Problem der numerischen Mathematik behandeln, die Nullstellensuche einer Funktion. Wir betrachten im Speziellen die Funktion $f(x) = e^x - 10$ und wollen berechnen, bei welchem x-Wert die Funktion Null wird ($f(x) = 0$). Hat eine Funktion im Teilintervall $[a, b] \in \mathbb{R}$ eine

Vorlesung 4

....

Die weiteren Vorlesungen sind in Bearbeitung