



UNIVERSITÄT **BONN**

Algorithmen und Programmierung

Einleitung

Dr. Felix Jonathan Boes

boes@cs.uni-bonn.de

Institut für Informatik

Algorithmen und Programmierung | Universität Bonn | WS 22/23



Kurzes Q+A zu den Übungen

Einleitung

In den Vorlesungen zur imperativen Programmierung haben wir den folgneden Anspruch. Wir führen Sprachkonstrukte, Konzepte und Zusammenhänge ein wenn wir sie benötigen. Das gelingt sicherlich nicht immer perfekt.

Manchmal müssen wir Dinge verwenden die wir noch nicht verstehen und manchmal müssen wir Dinge einführen die wir nicht dringend brauchen aber die zum „allgemeinen guten Stil“ gehören.

Sie sollen hier erlernen **korrekten, gut verständlichen** Programmcode zu schreiben. Es reicht nicht **korrekten** aber schlecht verständlichen Programmcode zu schreiben.

Es ist für die Klausur und auch für Ihren weiteren Studienverlauf sehr wichtig, sowohl die Theorie als auch die Praxis zu vertiefen.

Einleitung

C++ Programme schreiben und ausführen

Erste Antworten auf Fragen:

Welche Werkzeuge verwendet man um ein C++-Programm zu schreiben?

Welche Werkzeuge verwendet man um ein C++-Programm „ans Laufen zu bringen“?



Wir wollen dieses einfache C++-Programm „ans Laufen“ bekommen.

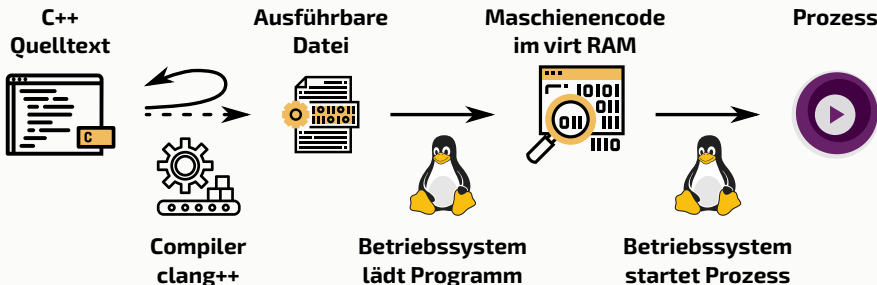
```
#include <iostream>

int main() {
    std::cout << "Hallo Welt!" << std::endl;
    return 0;
}
```

Was dazu nötig ist schauen wir uns jetzt an.

Vom Quellcode zum Prozess

Um ein C++-Quellcode auf unserem System „ans Laufen zu bringen“, durchlaufen wir folgende, typische Schritte.



Welches Betriebssystem soll verwendet werden?



Für die Vorlesung ist es unerheblich welches Betriebssystem Sie verwenden. Nutzen Sie das Betriebssystem mit dem Sie Ihre Aufgaben möglichst schnell und zuverlässig lösen können. In der Vorlesung sehen Sie oft Linux-Beispiele weil es am besten zu meinen typischen Aufgaben passt.

Wie bei Programmiersprachen gibt es nicht „das beste Betriebssystem“. Es gibt unter Umständen das „passendste Betriebssystem für eine vorgegebene Aufgabe“. Tauschen Sie sich gern über Ihre Anwendungsszenarien aus und teilen Sie Ihre Erfahrungen.

Die Konsole ist eine universelle Schnittstelle um mit einer Vielzahl von Programmen zu interagieren. Klare Vorteile sind:

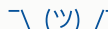
- Fast alle Konsolenprogramme sind koppelbar. Konsolenprogramme dienen als Bausteine für noch komplexere Konsolenprogramme.
- Vergleichsweise einfache Automatisierbarkeit von Aufgaben
- Zeitloses und plattformunabhängiges Look-And-Feel (keine Umgewöhnungen nötig)
- Verbrauchsarm



Um C++-Programme zu schreiben brauchen Sie einen **Texteditor** (am besten mit Syntaxhighlighting und Wortvervollständigung).

Um C++-Programme in ausführbare Programme umzuwandeln brauchen Sie einen **Compiler**. Das ist ein Computerprogramm das C++-Quellcode in ausführbare Programme übersetzt.

Die Kombination aus Texteditor, Compiler und weiteren (un)komfortablen Entwicklungswerkzeugen nennt man **Entwicklungsumgebung** (Integrated-Development-Environment IDE).

Es existieren sehr wenige gute Compiler, viele IDEs und sehr viele Texteditoren. Welche Kombination zu Ihrem Programmierprojekt passt müssen Sie selbst herausfinden.  Wir sprechen hier Empfehlungen für Anfänger:innen aus.



Als Compiler empfehlen wir eine aktuelle Version von **clang++**. Alternativen die wir in dieser Vorlesung nicht besprechen sind **g++** und **MSVC**.

Wie Sie **clang++** außerhalb von einer IDE installieren hängt von Ihrem Betriebssystem ab. Wie das im Detail funktioniert finden Sie bestimmt schnell selbst heraus (oder verwenden doch lieber eine IDE).



Als Texteditor empfehlen wir

- unter Windows **Notepad++**
- unter Linux **Kate**
- unter Mac **XCode**
- plattformübergreifend **atom**
- auf der Konsole **vim**

C++ Programme in Kate schreiben und mithilfe der Konsole kompilieren



Wie Sie ein C++-Programm in ein ausführbares Programm übersetzen (kompilieren) und ausführen zeigen wir hier exemplarisch.

Livedemo

C++ Programme in Kate schreiben und mithilfe der Konsole kompilieren



Um C++-Programme mit `clang++` in der C++-Version 17 zu kompilieren und auszuführen, geben Sie folgende Befehle ein.

```
clang++ -std=c++17 -o NAME_AUSGABE_DATEI NAME_EINGABE_DATEI.cpp  
./NAME_AUSGABE_DATEI
```

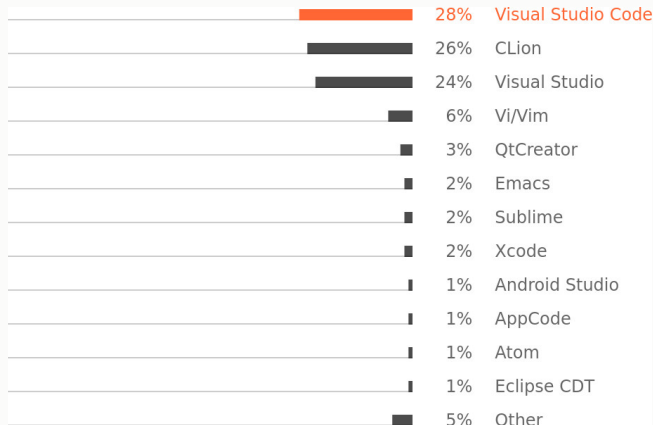


Wenn Sie lieber eine IDE verwenden wollen, dann empfehlen wir

- unter Windows **Visual Studio Code mit clang++** <https://code.visualstudio.com/docs/languages/cpp>
- unter Linux **Visual Studio Code** oder **QtCreator**
- unter Mac **XCode**



Die Verwendung von IDEs in 2021 finden Sie hier¹.



¹<https://www.jetbrains.com/idea/devecosystem-2021/cpp/>



Empfehlungen zum Tastaturlayout

Bis jetzt haben Sie Ihre Tastatur zum Schreiben von normalen Texten verwendet. Beim Programmieren brauchen Sie allerdings *unkonventionelle* Symbole wie zum Beispiel [] { } ^ ' / ... Diese sind auf ihrem üblichem Tastaturlayout evtl. umständlich einzutippen.

Wenn Sie regelmäßig viel Programmieren möchten und schon blind tippen können, empfehlen wir das Tastaturlayout „US international without dead keys“. Wenn Sie noch nicht blind tippen können empfehlen wir bei Ihrem Tastaturlayout zu bleiben.

Haben Sie Fragen?

Einleitung

Unser erstes C++ Programm

Ziel

In diesem Abschnitt schreiben wir unser erstes
C++-Programm

Auf dem Weg erlernen wir erste Sprachkonstrukte
von C++

Das Zielprogramm

Sie lernen jetzt einen Teil des Vokabulars kennen um folgendes Spiel in C++ zu implementieren. Die Benutzer:innen sollen das Geburtsjahr von Ada Lovelace raten.

INPUT: Geburtstagsjahr als ganze Zahl
OUTPUT: Rückmeldung ob geratenes Geburtsjahr korrekt/ zu klein / zu groß ist.
ALGORITHMUS (Geburtsjahr raten):

Speichere Ada Lovelace Geburtsjahr in der Ganzzahlvariable "Geburtsjahr"

Frage die Nutzer:innen nach dem Geburtsjahr

Speichere die Antwort in der Ganzzahlvariable "Rateversuch"

Wenn Rateversuch > Geburtsjahr dann:

Sag den Nutzer:innen "Ada ist älter"

Anderenfalls: Wenn Rateversuch < Geburtsjahr dann:

Sag den Nutzer:innen "Ada ist jünger"

Anderenfalls:

Sag den Nutzer:innen "Ja, genauso alt ist Ada"

ENDE VOM PROGRAMM

Die einzelnen Anweisungen von C++ nennen wir auch **Statements**. Wir betrachten zunächst eine Auswahl von

- **Declaration Statement** und
- **Expression Statements** und
- **Selection Statements** und
- **Iteration Statements.**

Declaration Statements

Declaration Statements sind Anweisungen die neue Variablen zur Verfügung stellen

- z.B. das Anlegen einer Ganzzahlvariable `int` Geburtsjahr;

Expression Statements sind Anweisungen die etwas Berechnen

- z.B. $42*3$;

oder Anweisungen die einer Variable einen Wert zuweisen:

- z.B. `Geburtsjahr = 1815`;

oder Anweisungen die eine *Funktion aufrufen*

- z.B. `drucke_den_text("Wie alt ist Ada?")`;

oder Anweisungen die etwas tun was wir erst später lernen.

Jedes Expression Statement wird in C++ mit dem Semikolon ; abgeschlossen.



INPUT: Geburtstagsjahr als ganze Zahl
OUTPUT: Rückmeldung ob geratenes Jahr korrekt / zu klein / zu groß
ALGORITHMUS (Geburtsjahr raten):

```
int Geburtsjahr = 1815;  
drucke_den_text("In welchem Jahr wurde Ada Lovelace geboren?");  
int Rateversuch = lies_eingabe();  
Wenn Rateversuch > Geburtsjahr dann:  
    drucke_den_text("Ada ist älter.");  
Anderenfalls: Wenn Rateversuch < Geburtsjahr dann:  
    drucke_den_text("Ada ist jünger");  
Anderenfalls:  
    drucke_den_text("Ja, genauso alt ist Ada");  
ENDE VOM PROGRAMM
```

Selection Statements

Selection Statements sind Anweisungen die Programmcode in Abhängigkeit von einer Bedingung ausführen. Das wichtigste Selection Statement ist das `if-else` Statement.

```
if (BEDINGUNG IST ERFÜLLT?) {  
    Statements des ersten Blocks  
} else {  
    Statements des zweiten Blocks  
}
```

Wenn die Bedingung erfüllt ist, werden alle Befehle in ersten `{}`-Block ausgeführt. Anderenfalls werden alle Befehle im zweiten `{}`-Block ausgeführt.



INPUT: Geburtstagsjahr als ganze Zahl
OUTPUT: Rückmeldung ob geratenes Jahr korrekt / zu klein / zu groß
ALGORITHMUS (Geburtsjahr raten):

```
int Geburtsjahr = 1815;
drucke_den_text("In welchem Jahr wurde Ada Lovelace geboren?");
int Rateversuch = lies_eingabe();
if (Rateversuch > Geburtsjahr) {
    drucke_den_text("Ada ist älter.");
} else {
    if (Rateversuch < Geburtsjahr) {
        drucke_den_text("Ada ist jünger");
    } else {
        drucke_den_text("Ja, genauso alt ist Ada");
    }
}
}
ENDE VOM PROGRAMM
```

Um C++ zu erklären, dass ein Text nur als Kommentar für die Programmierer:innen gemeint ist, nutzt man die folgenden zwei Konstrukte.

```
// Der Text bis zum Zeilenende ist ein Kommentar  
/* Der Text bis zum "Stern/" ist ein Kommentar.  
   Auch über mehrere Zeilen */
```

Kommentare werden nicht als Programmcode interpretiert.



```
// INPUT:  Geburtstagsjahr als ganze Zahl
// OUTPUT: Rückmeldung ob geratenes Jahr korrekt / zu klein / zu groß
// ALGORITHMUS (Geburtsjahr raten):

int Geburtsjahr = 1815;
drucke_den_text("In welchem Jahr wurde Ada Lovelace geboren?");
int Rateversuch = lies_eingabe();
if (Rateversuch > Geburtsjahr) {
    drucke_den_text("Ada ist älter.");
} else {
    if (Rateversuch < Geburtsjahr) {
        drucke_den_text("Ada ist jünger");
    } else {
        drucke_den_text("Ja, genauso alt ist Ada");
    }
}
// ENDE VOM PROGRAMM
```

Jedes Computerprogramm hat einen fest definierten **Startpunkt**. Ein C++-Programm beginnt seine Arbeit in der `main`-Funktion (nachdem die Ausführungsumgebung vollständig initialisiert wurde, das schauen wir uns in dieser Vorlesung aber nicht an).

Ist das Ende der `main`-Funktion erreicht, beendet der Prozess seine Arbeit (nachdem die Ausführungsumgebung vollständig deinitialisiert wurde, das schauen wir uns in dieser Vorlesung aber nicht an).

Es gibt drei typische Varianten der `main`-Funktion. Die Einfachste ist diese.

```
// Hier beginnt der Prozess die Arbeit
int main() {
    Statements ...
}
```



```
// INPUT:  Geburtstagsjahr als ganze Zahl  
// OUTPUT: Rückmeldung ob geratenes Jahr korrekt / zu klein / zu groß  
// ALGORITHMUS (Geburtsjahr raten):
```

```
int main() {  
    int Geburtsjahr = 1815;  
    drucke_den_text("In welchem Jahr wurde Ada Lovelace geboren?");  
    int Rateversuch = lies_eingabe();  
    if (Rateversuch > Geburtsjahr) {  
        drucke_den_text("Ada ist älter.");  
    } else {  
        if (Rateversuch < Geburtsjahr) {  
            drucke_den_text("Ada ist jünger");  
        } else {  
            drucke_den_text("Ja, genauso alt ist Ada");  
        }  
    }  
}
```




Wir wollen unser Zielprogramm wie folgt verändern. Wir wollen die Nutzer:innen so lange spielen lassen bis das Geburtsjahr erraten wurde. Hier reicht uns die **if-else**-Konstruktion nicht aus, weil wir nicht wissen wie wir **zurückspringen** können.

Iteration Statements

Iteration Statements werden genutzt um ein Stück Programmcode immer wieder auszuführen solange eine gegebene Bedingung erfüllt ist. Es gibt in C++ drei verschiedene Iteration Statements, wir lernen hier die **while**-Schleife kennen.

```
while (BEDINGUNG) {  
    Statements des Blocks  
}  
Statement nach der while-Schleife
```

Beim Ausführen der **while**-Schleife wird zuerst die Bedingung geprüft. Falls die Bedingung erfüllt ist werden die Statements des Blocks ausgeführt. Am Ende des Blocks wird an den Anfang der **while**-Schleife gesprungen und der Vorgang wird wiederholt.

Falls die Bedingung der **while**-Schleife bei der Überprüfung nicht erfüllt ist, wird das Statement nach der **while**-Schleife ausgeführt.



```
// INPUT:  Geburtstagsjahr als ganze Zahl
// OUTPUT: Rückmeldung ob geratenes Jahr korrekt / zu klein / zu groß
// ALGORITHMUS (Geburtsjahr raten):

int main() {
    int Geburtsjahr = 1815;
    int Rateversuch = 0;
    // Mit == prüft man Gleichheit. Mit != prüft man Ungleichheit.
    while (Rateversuch != Geburtsjahr) {
        drucke_den_text("In welchem Jahr wurde Ada Lovelace geboren?");
        Rateversuch = lies_eingabe();
        // ...
        // Rest wie oben
        // ...
    }
}
```



Um unser erstes Programm lauffähig zu machen fehlt noch ein wenig Magie. Wir verstehen später was die Magie im Detail macht.

```
// Wir beginnen mit Magie, reiner Magie.
#include <iostream>
#include <string>

void drucke_den_text(const std::string& text) {
    std::cout << text << std::endl;
}

int lies_eingabe() {
    std::string text;
    std::cin >> text;
    int zahl = 0;
    try { zahl = std::stoi(text); }
    catch (const std::invalid_argument& e) { zahl = -1000; }
    return zahl;
}

int main() { ...
```



Um unser erstes Programm lauffähig zu machen, fehlt noch ein wenig Magic. Wir verstehen

```
→ wie_alt_ist_ada_lovelace git:(main) x clang++ -std=c++17 ada.cpp -o ada
→ wie_alt_ist_ada_lovelace git:(main) x ./ada
In welchem Jahr wurde Ada Lovelace geboren?
1900
Ada ist älter.
In welchem Jahr wurde Ada Lovelace geboren?
1800
Ada ist jünger
In welchem Jahr wurde Ada Lovelace geboren?
1850
Ada ist älter.
In welchem Jahr wurde Ada Lovelace geboren?
1825
Ada ist älter.
In welchem Jahr wurde Ada Lovelace geboren?
1813
Ada ist jünger
In welchem Jahr wurde Ada Lovelace geboren?
1815
Ja, genauso alt ist Ada

// Wir b
#include
#include
void dru
std::c
}
int lies
std::s
std::c
int za
try {
catch
return
}

int main() { ...
```



Dinge die jetzt noch reine Magie sind

Zu diesem Zeitpunkt verstehen wir folgende Dinge noch nicht.

- Die Magie am Anfang des Programms
- Was Funktionen sind und wie sie genau genutzt werden
- Was das `int` am Anfang der `main` bedeutet



Sie haben jetzt bereits genug „C++-Vokabeln“ gelernt um fast jeden Algorithmus der Welt zu implementieren. Zumindest in der Theorie.

Aber Ihr Programm wird dann quasi unlesbar, unwartbar und ineffizient sein. Sie lernen im weiteren Verlauf der Vorlesung wie sie sich **verständlich ausdrücken**. Außerdem lernen Sie noch mehr über den Speicher. Genauer lernen Sie wie Daten im Speicher dargestellt und verarbeitet werden.

Haben Sie Fragen?

Zusammenfassung

C++-Programme sind eine Folge von Statements (und Dingen die wir noch lernen)

Sie haben erste Expression Statements, Selection Statements und Iteration Statements gelernt

Das Programm beginnt seine Arbeit in der main-Funktion

Kommentare erhöhen die Lesbarkeit des Programmcodes