Institut für Mathematik
Universität zu Köln

C++
Eine Einführung

Skript zur C++ Vorlesung für Flüchtlinge

Update vom 25.07.2017

Inhaltsverzeichnis

1 Zur Verwendung dieses Skripts

Dieses Skript ist das begleitendes Skript für die Vorlesung *C++ für Flüchtlinge*. Das Skript hat es zum Ziel, die Grundprinzipien der Programmierung mit C++ anschaulich und mit vielen Beispielen darzustellen und dient somit als Leitfaden für Anfänger in der Programmierung. **Es ist dabei weder eine umfassende Abhandlung der Programmiersprache C++, noch zeigt es die theoretischen Grundlagen in aller Breite auf.** An manchen Stellen wird im Skript etwas vorgegriffen, es ist also nicht als ein chronologischer Kurs, sondern eher als Nachschlagewerk zu sehen. Kursteilnehmer die sich tiefer mit den Themen und mit den theoretischen Konzepten der Vorlesung beschäftigen wollen wird empfohlen, sich zusätzlich zum Kurs an weiterführende Literatur zu wenden.

1.1 Empfohlene weiterführende Literatur

- **Breymann: Der C++ Programmierer**. Als Online-Ressource verfügbar im USB Köln Netz: http://www.hanser-elibrary.com/doi/book/10.3139/9783446449121
- **Stroustrup: Die C++ Programmiersprache**. Als Online-Ressource verfügbar im USB Köln Netz: http://www.hanser-elibrary.com/isbn/9783446439610

1.2 Format

An vielen Stellen im Skript sind C++ Code und Beispiele aufgeführt. Fertiger Code, welcher direkt ausführbar ist, ist dabei lila hinterlegt. An manchen Stellen ist auch das Ergebnis grün hinterlegt gezeigt, wie. z.B. in Beispiel 1.2.1. *Nur lila hinterlegter Code ist direkt ausführbar.* Text welcher hinter einem "//" oder zwischen "/*" und "*/" steht (und bei uns blau ist) ist ein Kommentar und wird nicht vom Programm gelesen. Er dient lediglich der Erklärung des Codes.

Beispiel 1.2.1. Hello World

```
1 #include <stdlib.h>
2 #include <iostream>
3
4 using namespace std;
5
6 int main(){ //Hier beginnt das Programm
7   cout << "Hello World!" << endl;
8   return 0; //Hier endet das Programm
9 }
```

```
Hello World!
```

An manchen Stellen erklären wir die Syntax von C++ Befehlen. Dies geschieht meist im Fließtext und ist daher nicht direkt in C++ ausführbar. In Abschnitt 2.4 definieren wir die if-Anweisung z.B. so:

```
if (<Bedingung>){
<Anweisung>
}
```

Alles was in den eckigen Klammern (<, >) steht ist dabei lediglich ein Bezeichner für die Art des Befehls und kein Programmcode.

Bei Fragen, Anregungen oder Anmerkungen zu Fehlern im Skript wenden Sie sich bitte an Laura Höller unter der E-Mail Adresse Laura-Hoeller@web.de.

2 Grundlagen

In diesem Kapitel werden die Grundlagen der C++ Programmierung besprochen. Die hier besprochenen Konzepte finden sich in ähnlicher Form in allen Programmiersprachen wieder.

2.1 Ein- und Ausgabe

Für alle Programme müssen wir mit dem Computer kommunizieren. Die Standard Ein- und Ausgabe funktioniert in C++ mit den folgenden beiden Befehlen:

- `cout`: Ist der Kommentar für die Ausgabe. Er wird gefolgt von <<.
- `cin`: Ist der Kommentar für die Eingabe. Er wird gefolgt von >>.

Die Verwendung von `cin/cout` wird im Folgenden anhand einiger Beispiele verdeutlicht.

2.1.1 Ausgabe mit cout

Wir beginnen mit der Standard-Ausgabe `cout`. Beispiel 2.1.1 zeigt den Code eines Programms, welches einfach den Text *"Hello World"* auf dem Bildschirm ausgibt.

Beispiel 2.1.1. Hello World 1

```
1 #include <stdlib.h> //Einbindern der Standard-Bibliothek
2 #include <iostream> //Einbindern der Standard Input- und Output
     Methoden
3
4 using namespace std; //Für die verwendung von cout
5
6 int main(){  //Hier beginnt die main-Funktion
7   cout << "Hello World!" << endl;  //Hier geben wir den Text aus
8   return 0; //Die Funktion gibt den Wert Null zurück
9 }
```

```
Hello World!
```

Ein paar Kommentare zu dein einzelnen Zeilen in Beispiel 2.1.1:

- Die ersten zwei Zeilen beginnen mit einem `"#"`. Dies sind Befehle um sogenannte **Header-Dateien** einzubinden. Z.B. funktioniert `cin` und `cout` nur wenn `#include <iostream>` am Anfang des Programms steht, wir also ein Programm welches `cin/cout` definiert **einbinden**. Wir kommen zur Einbindung von Header-Dateien in einem anderen Kapitel und wollen ab jetzt diese zwei Zeilen einfach hinnehmen.
- Zeile 4, `using namespace std`: Dies ist ein Befehl um im folgenden die Standard Ein-und Ausgabe mit `cin/cout` zu verwenden. Würden wir diese Zeile nicht schreiben so müssten wir anstatt `cin` `std::cin` schreiben. Den genauen Grund hierfür besprechen wir an einer anderen Stelle in diesem Skript.
- `int main(){...}`: Markiert den Beginn der Haupt-Funktion eines Programms und muss in jedem Programm **genau ein mal** vorkommen. Wichtig ist, dass der gesamte Programmcode innerhalb der geschweiften Klammern {...} steht. Das `int` vor `main()` besagt, dass `main` eine ganze Zahl zurückgeben muss. Wir verschieben die genaue Erklärung hierfür in den Abschnitt 5.5.
- Zeile 7: `cout` <<: Hier steht das eigentliche Programm. Wir geben den Text *"Hello World"* aus. Wichtig dabei ist:

  - `cout` wird immer von << gefolgt. Die "Pfeile" zeigen also immer in Richtung `cout`, also in Richtung Ausgabe.
  - Soll Text ausgegeben werden, so müssen diese zwischen Anführungszeichen `"..."` geschrieben werden.
  - Sollen verschiedene Objekte ausgegeben werden, so sind sie durch weitere << getrennt. Dies ist in Beispiel 2.1.2 veranschaulicht. Die Ausgabe sieht genau gleich aus wie in Beispiel 2.1.1.
  - Am Schluss einer Ausgabe steht meist `endl`, was für *end line* steht. Dadurch springt C++ bei der nächsten Ausgabe in die nächste Zeile. Dies ist in Beispiel 2.1.3 verdeutlicht.

- Am Ende von `main(){...}` steht `return 0`. Dies gibt als Ausgabe der `main`-Funktion die Zahl 0 zurück. Wir verschieben die Erklärung hiervon wieder in den Abschnitt 5.5.

- Wie wir im Beispiel sehen wird in C++ jeder Befehl von einem Semikolon ";" gefolgt. Zeilen die den Beginn einer Funktion (wie z.B. `int main()`) markieren, jedoch nicht!

Beispiel 2.1.2. Hello World 2

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7    cout << "Hello" << " World!" << endl; //Ausgabe durch zwei
       Textabschnitte
8    return 0;
9  }
```

```
Hello World!
```

Beispiel 2.1.3. Hello World 3

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7    cout << "Hello" << endl; //Getrennte Ausgabe der beiden Wörter
8    cout << " World!" << endl;
9    return 0;
10 }
```

```
Hello
World!
```

2.1.2 Eingabe mit cin

Wir wollen natürlich nicht nur Dinge ausgeben, sondern auch eingeben. Die Standard-Eingabe erfolgt in C++ über `cin`, welches immer von einem >> (also anders herum als bei `cout`) gefolgt wird. Beispiel 2.1.4 liest den Geburtstag vom Benutzer ein und gibt ihn anschließend aus. Wir greifen hierbei etwas voraus, und besprechen den **Datentyp** `int` in Abschnitt 2.3. Im Prinzip besagt der Befehl in Zeile 8 lediglich, dass wir drei Zahlen mit Namen "Tag", "Monat" und "Jahr" benötigen.

Beispiel 2.1.4. Geburtsdatum einlesen und ausgeben

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      int tag, monat, jahr; //Deklaration dreier Variablen für ganze
           Zahlen
9
10     cout << "Geben Sie bitte ihr Geburtsdatum ein:" << endl;
11     cout << "tag: ";
12     cin >> tag; //Hier wird der Tag eingegeben
13
14     cout << "monat: ";
15     cin >> monat; //Hier wird der Monat eingegeben
16
17     cout << "jahr: ";
18     cin >> jahr; //Hier wird das Jahr eingegeben
19
20     cout << "Sie haben am " << tag << "." << monat << "." << jahr <<
           " Geburtstag." << endl;
21
22     return 0;
23 }
```

Geben Sie bitte ihr Geburtsdatum ein:
tag: **10**
monat: **12**
jahr: **1985**
Sie haben am 10.12.1985 Geburtstag.

Im Beispiel 2.1.4 wird in Zeile 11 nach dem Tag gefragt und dieser wird in Zeile 12 eingelesen. Danach hat die Variable `tag` den Wert 10. Genauso wird für den Monat und das Jahr verfahren.
Ein weiteres Beispiel zur Ein-und Ausgabe ist in Beispiel 2.1.5 gezeigt. Hier werden wie oben Variablen eingelesen. In der Ausgabe wird dann die Summe und das Produkt der beiden Zahlen ausgegeben - C++ funktioniert also auch als ganz gewöhnlicher Taschenrechner.

Beispiel 2.1.5. Einfache Rechenoperationen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      int zahl1, zahl2;
9
10     cout << "Bitte geben Sie die beiden Zahlen ein: " << endl;
11     cin >> zahl1;
12     cin >> zahl2;
```

```
13
14      cout << "Die Summe der Zahlen ist " << zahl1 + zahl2 << endl;
15
16      cout << "Das Produkt der Zahlen ist " << zahl1*zahl2 << endl;
17
18      return 0;
19  }
```

2.2 Kommentare

Programmiercode kann manchmal sehr unübersichtlich werden. Vor allem wenn man will dass andere den Code verstehen ist es daher wichtig dass man Kommentare verwendet, da diese die Lesbarkeit deutlich erhöhen. Dabei gilt:

- Einzeilige Kommentare werden mit // eingeleitet.
- Mehrzeilige Kommentare beginnen mit /* und enden mit */

Das folgende Beispiel 2.2.1 zeigt wie wir Kommentare verwenden.

Beispiel 2.2.1. Arbeiten mit Kommentaren

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    //Dies ist ein Kommentar und wird nicht vom Compliler gelesen
9
10   cout << "Dies ist ein unkommentierter Text!" << endl;
11
12   /*
13   Alles was in diesen Zeilen steht wird nicht gezeigt
14   cout << "Dies ist ein auskommetierter Text" << endl;
15   */
16
17   return 0;
18 }
```

```
Dies ist ein unkommentierter Text!
```

2.3 Variablen und Datentypen

Eine Variable ist eine Stelle im Speicher des Computers, an welcher ein Wert eines bestimmten **Datentyps** gespeichert werden kann. Vor der Benutzung müssen Variablen immer mit Namen und Datentyp definiert werden. Die Syntax geht folgendermaßen:

```
<Datentyp> <Variablenname>;
```

Der Variablenname ist dabei ein beliebiger frei wählbarer Name. Wollen wir C++ z.B. unser Alter einlesen lassen, so müssen wir vorher Platz für die Zahl unseres Alters schaffen, d.h. eine **Variable deklarieren**. Dies wird am Besten mit einem Beispiel (Beispiel 2.3.1) erläutert.

Beispiel 2.3.1. Variablen 1: Einlesen von Zahlen in `int`

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int alter; //Hier deklarieren wir die Variable "alter" als integer
         , also als ganze Zahl
9
10   cout << "Bitte geben Sie Ihr Alter ein: " << endl;
11   cin >> alter; //Wir belegen die Variable "alter" mit einem
        eingegebenen Wert
12   cout << "Sie sind " << alter << " Jahre alt." << endl;
13   return 0;
14 }
```

```
Bitte geben Sie Ihr Alter ein:
32
Sie sind 32 Jahre alt.
```

In Beispiel 2.3.1 wird in Zeile 8 eine Variable mit dem Namen `alter` angelegt. Wir können über diesen Namen immer auf den Wert der Variablen zugreifen. Das `int` vor dem Variablen-Namen macht klar um welche Art von Variable es sich handelt (z.B. Text, ganze Zahl, Gleitkommazahl, Buchstabe etc.). `int` steht für **integer** (engl. ganze Zahl). Im Beispiel wird die Variable `alter` durch eine Benutzer-Eingabe belegt. `int` nennt man hierbei den **Datentyp** der Variablen `alter`.
Ein weiterer wichtiger Datentyp sind Buchstaben. Der entsprechende Datentyp heißt `char` (von engl. character). Wollen wir zum Beispiel zwei Buchstaben einlesen, so kann man das wie in Beispiel 2.3.2 tun.

Beispiel 2.3.2. Variablen 2: Einlesen von Buchstaben in `char`

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    char buchst1, buchst2;
9
10   cout << "Bitte geben Sie den ersten Buchstaben ein: " << endl;
11   cin >> buchst1;
12   cout << "Bitte geben Sie den zweiten Buchstaben ein: " << endl;
13   cin >> buchst2;
14   return 0;
15 }
```

Wir sehen im Beispiel 2.3.2 in Zeile 8 dass wir mehrere Variablen vom gleichen Datentyp auch in einer Zeile gemeinsam definieren können: `char buchst1, buchst2;`. Die beiden Namen sind dann durch ein Komma getrennt.
Ein weiterer Datentyp ist der `string`. Ihn benutzt man, wenn man mit einer Zeichenkette arbeiten will, zum Beispiel einem Wort oder einem Satz. Mit `string` lässt sich zum Beispiel leicht der Name und Vorname eingeben, siehe Beispiel 2.3.3.

Beispiel 2.3.3. Variablen 3: Einlesen von Worten in einen `string`

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    string nachname, vorname;
9
10   cout << "Bitte geben Sie Ihren Vornamen ein: " << endl;
11   cin >> vorname;
12   cout << "Bitte geben Sie Ihren Nachnamen ein: " << endl;
13   cin >> nachname;
14   cout << "Sie heissen " << vorname << " " << nachname << endl;
15   return 0;
16 }
```

```
Bitte geben Sie ihren Vornamen ein:
David
Bitte geben Sie ihren Nachnamen ein:
Wichmann
Sie heissen David Wichmann
```

Ein paar wichtige Punkte zu Variablennamen in C++:

- In C++ ist die Groß- und Kleinschreibung wichtig. Eine Variable mit dem Namen `alter` ist nicht das gleiche wie `Alter` oder `alTeR`.
- Ein Variablenname besteht aus einer Zeichenkette (d.h. Buchstaben, Ziffern, Unterstrich "_" etc.), darf aber kein Leerzeichen enthalten.
- Variablennamen dürfen nicht mit Ziffern beginnen (`int 2alter` ist also verboten!)
- Außerdem sind manche Namen wie z.B. `main` oder `int` logischerweise verboten.

Man beachte, dass die Eingabe eines `string` mit `cin` lediglich für zusammenhängende Zeichenketten gedacht ist und nicht für Sätze oder mehrere Worte. Will man mehrere Worte in einen einzelnen `string` einlesen, so erfolgt die Eingabe mit der Funktion `getline()`. Dies ist in Beispiel 2.3.4 gezeigt.

Beispiel 2.3.4. Variablen 4: Einlesen mehrerer Worte in einen `string`

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    string name;
9
10   cout << "Bitte geben Sie Ihren Namen ein: " << endl;
11   getline(cin, name);
12   cout << "Sie heissen " << name << endl;
13   return 0;
14 }
```

Bitte geben Sie ihren Namen ein:
David Wichmann
Sie heissen David Wichmann

Weitere wichtige Datentypen sind `float` und `double`. Dies sind Gleitkommazahlen (im Gegensatz zu `int`, was nur ganze Zahlen beinhaltet). Dabei kann `double` größere Zahlen speichern als `float`. Die Verwendung von `float` und `double` funktioniert genau gleich wie von `int`. Tabelle 1 zeigt die wichtigsten Datentypen und ihren Wertebereich.

Datentyp	**Erklärung**	**Wertebereich**
`int`	Ganze Zahlen	Systemabhängig. ± 32768 oder $\pm 2.147 \cdot 10^9$
`long`	Ganze Zahlen	Systemabhängig. Normalerweise $\pm 2.147 \cdot 10^9$
`float`	Gleitkommazahlen	$\pm 3.4 \cdot 10^{38}$
`double`	Gleitkommazahlen	$\pm 1.7 \cdot 10^{308}$
`char`	Zeichen und Buchstaben	Ein Zeichen oder ein Buchstabe
`string`	Zeichenketten	Worte oder ganze Sätze

Tabelle 1: Liste der wichtigsten Datentypen. Siehe auch z.B. unter diesem LINK (darauf klicken!).

Es ist wichtig, dass man den richtigen Datentyp deklariert, anderenfalls ist ein Programm meist fehlerhaft! Dies ist in Beispiel 2.3.5 gezeigt. Hier berechnen wir den Durchschnitt zweier ganzer Zahlen `int` - das Ergebnis ist falsch (abgerundet)! Für das richtige Ergebnis hätten wir Kommazahlen wie `float` oder `double` verwenden sollen. Man muss also immer vorher wissen mit welcher Art von Objekten man es zu tun hat!

Beispiel 2.3.5. Variablen 5: Ein Problem mit den Datentypen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int zahl1, zahl2;
9
10   cout << "Bitte geben Sie die Zahlen ein: " << endl;
11   cin >> zahl1;
12   cin >> zahl2;
13   cout << "Der Mittelwert der Zahlen ist: " << (zahl1 + zahl2)/2 <<
        endl;
14   return 0;
15 }
```

Bitte geben Sie die Zahlen ein:
5
8
Der Mittelwert der Zahlen ist 6

2.4 Die if-Anweisung und logische Operatoren

2.4.1 Die if-Anweisung

Wir wissen jetzt wie man Variablen anlegen und Werte der Variablen über die Benutzeroberfläche eingeben und ausgeben kann. Natürlich wollen wir mit den Variablen auch etwas machen. Ein sehr wichtiges Werkzeug für fast alle Programme ist die **if-Anweisung**. Mit ihr können wir testen, ob etwas richtig ist oder nicht. Die allgemeine Syntax lautet so:

```
if (<Bedingung>){
  <Anweisung>;
}
```

Wir können zum Beispiel zwei Zahlen einlesen wie im letzten Abschnitt und dann testen ob die erste Zahl kleiner oder größer ist als die zweite. Dies ist in Beispiel 2.4.1 veranschaulicht. Dies beinhaltet drei if-Abfragen, welche alle Möglichen Anordnungen der Variablen a und b testen.

Beispiel 2.4.1. If-Anweisung 1: Zahlen vergleichen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a, b;
9
10   cout << "Bitte geben Sie Zahl 1 ein: " << endl;
11   cin >> a;
12   cout << "Bitte geben Sie Zahl 2 ein: " << endl;
13   cin >> b;
14
15   if (a<b){ //Aufruf der if-Anweisung. Getestet wird ob a kleiner
        als b ist
16     cout << a << " ist kleiner als " << b << endl;
17   }
18
19   if (a>b){ //Jetzt wird getestet ob a größer b ist
20     cout << a << " ist groesser als " << b << endl;
21   }
22
23   if (a==b){ //Und zum Schluss testen wir noch ob beide Zahlen
        gleich groß sind
24     cout << "Die Zahlen sind gleich gross" << endl;
25   }
26
27   return 0;
28 }
```

```
Bitte geben Sie Zahl 1 ein:
16
Bitte geben Sie Zahl 2 ein:
15
16 ist groesser als 15
```

Der folgende Abschnitt erklärt welche Art von Bedingungen wir in der if-Anweisung verwenden können und wie man diese in C++ formuliert.

2.4.2 Vergleichsoperatoren und logische Operatoren

Die Bedingung in der if-Anweisung wird meist mit sogenannten **Vergleichsoperatoren** formuliert. Diese sind in Beispiel 2.4.1 das “<” (kleiner als) in Zeile 15, das “>“ (größer als) in Zeile 19 oder das ”==“. Es gibt folgende Vergleichsoperatoren (Tabelle 2):

C++ Befehl	Bedeutung
<, >	kleiner als, größer als
<=, >=	kleiner gleich, größer gleich
==	gleich
!=	ungleich

Tabelle 2: Liste der Vergleichsoperatoren.

Beachten Sie, dass der Vergleichsoperator ”==“, mit welchem wir testen ob zwei Objekte identisch sind, zwei Gleichheitszeichen hat! Wir wollen lediglich testen ob zwei Objekte gleich sind und keine Werte zuweisen (dies machen wir mit nur einem ”=“). Die Funktionsweise von ”==“ und ”!=“ ist in Beispiel 2.4.2 verdeutlicht.

Beispiel 2.4.2. If-Anweisung 2: Zahlen vergleichen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a, b;
9
10   cout << "Bitte geben Sie Zahl 1 ein: " << endl;
11   cin >> a;
12   cout << "Bitte geben Sie Zahl 2 ein: " << endl;
13   cin >> b;
14
15   if (a==b){
16     cout << a << " ist gleich " << b << endl;
17   }
18
19   if (a!=b){
20     cout << a << " ist ungleich " << b << endl;
21   }
22
23   return 0;
24 }
```

```
Bitte geben Sie Zahl 1 ein:
16
Bitte geben Sie Zahl 2 ein:
15
16 ist ungleich 15
```

Für if-Anweisungen sind zusätzlich zu den Vergleichsoperatoren sogenannte **logische Operatoren** wichtig. Tabelle 3 listet die logischen Operatoren auf. Beachten Sie dass der ODER Operator || kein exklusives Oder ist, also nicht mit "Entweder Oder" zu verwechseln ist.

C++ Befehl	Bedeutung
&&	Und
||	Oder
!	Nicht

Tabelle 3: Liste der Logischen Operatoren.

Die Bedeutung dieser Operatoren wird am besten mit eine paar Beispielen verdeutlicht. Beispiele 2.4.3 bis 2.4.5 lesen jeweils drei Zahlen ein:

- In Beispiel 2.4.3 erfolgt nur dann eine Ausgabe, wenn alle Zahlen gleich sind.
- In Beispiel 2.4.4 erfolgt nur dann eine Ausgabe, wenn mindestens eine der Zahlen ungleich Null ist.
- In Beispiel 2.4.5 erfolgt nur dann eine Ausgabe, wenn a gleich Null ist. Bei der Anweisung `if(a)` ist die Bedingung erfüllt, wenn a ungleich Null ist. `if(!a)` ist also genau dann erfüllt, wenn a gleich Null ist.

Beispiel 2.4.3. If-Anweisung 3: UND(&&)

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a, b, c;
9
10   cout << "Bitte geben Sie Zahl 1 ein: " << endl;
11   cin >> a;
12   cout << "Bitte geben Sie Zahl 2 ein: " << endl;
13   cin >> b;
14   cout << "Bitte geben Sie Zahl 3 ein: " << endl;
15   cin >> c;
16
17   if ((a==b) && (a==c)){ //Unsere Bedingung testet ob a=b UND a=c
         ist. Ist dies der Fall ist auch a=c.
18     cout << "Alle Zahlen sind gleich" << endl;
19   }
20
21   return 0;
22 }
```

Beispiel 2.4.4. If-Anweisung 4: ODER(||) und NICHT(!)

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
```

```
8    int a, b, c;
9
10   cout << "Bitte geben Sie Zahl 1 ein: " << endl;
11   cin >> a;
12   cout << "Bitte geben Sie Zahl 2 ein: " << endl;
13   cin >> b;
14   cout << "Bitte geben Sie Zahl 3 ein: " << endl;
15   cin >> c;
16
17   if ((a!=0) || (b!=0) || (c!=0)){ //So testen wir ob mindestens
        eine Zahl verschieden von Null ist
18     cout << "Mindestens eine der Zahlen ist ungleich Null." << endl;
19   }
20
21   return 0;
22 }
```

Beispiel 2.4.5. If-Anweisung 5: NICHT (!)

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a, b, c;
9
10   cout << "Bitte geben Sie Zahl 1 ein: " << endl;
11   cin >> a;
12   cout << "Bitte geben Sie Zahl 2 ein: " << endl;
13   cin >> b;
14   cout << "Bitte geben Sie Zahl 3 ein: " << endl;
15   cin >> c;
16
17   if (!a){ //Test ob a Null ist
18     cout << "Die erste Zahl ist Null." << endl;
19   }
20
21   return 0;
22 }
```

Das letzte Beispiel 2.4.5 erscheint etwas unnötig (man kann diese Bedingung leicht über `if(a==0)` testen). Für spätere Zwecke ist diese Art des Befehls jedoch durchaus nützlich.

2.4.3 if - else

Manchmal wollen wir nicht nur eine bestimmte Bedingung testen, sondern auch entscheiden was passiert wenn die Bedingung nicht eintrifft. Dies wird mit der **`if-else`-Anweisung** gemacht. Alles was in den geschweiften Klammern {} nach `else` steht, wird genau dann ausgeführt wenn die Bedingung in der vorherigen `if`-Anweisung nicht erfüllt ist:

```
if (<Bedingung>){
  <Anweisung 1>;
```

```
}
else{
  <Anweisung 2>;
}
```

Wir können damit zum Beispiel einfach testen ob zwei Zahlen gleich sind oder nicht (Beispiel 2.4.6):

Beispiel 2.4.6. If-Anweisung 6: `if` und `else`

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a, b;
9
10   cout << "Bitte geben Sie Zahl 1 ein: " << endl;
11   cin >> a;
12   cout << "Bitte geben Sie Zahl 2 ein: " << endl;
13   cin >> b;
14
15   if (a==b){ //Test ob a gleich b ist
16     cout << "Die eingegebenen Zahlen sind gleich." << endl;
17   }
18   else{ //Else wird ausgeführt wenn a ungleich b ist
19     cout << "Die eingegebenen Zahlen sind nicht gleich." << endl;
20   }
21
22   return 0;
23 }
```

```
Bitte geben Sie Zahl 1 ein:
16
Bitte geben Sie Zahl 2 ein:
15
Die eingegebenen Zahlen sind nicht gleich.
```

2.5 Einfache Rechenoperationen

Oft wollen wir mit C++ einfache Dinge ausrechnen. Wir haben schon ein paar Beispiele mit Rechen-Operatoren behandelt. In C++ verwenden wir einfach "*" als Multiplikation, "/" für die Division und "+/-" für die Addition und Subtraktion. Das folgende Beispiel 2.5.1 liest eine Variable ein und speichert das Quadrat der Zahl in einer anderen Variable.

Beispiel 2.5.1. Rechenoperatoren 1: Quadrat berechnen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
```

```
8    int a, b; //Deklaration beider Variablen
9
10   cout << "Bitte geben Sie eine Zahl ein: " << endl;
11   cin >> a;
12
13   b = a*a; //Hier belegen wir b mit dem Quadrat von a
14
15   cout << "Das Quadrat der Zahl ist: " << b << endl; //Ausgabe von b
16
17   return 0;
18 }
```

Bitte geben Sie eine Zahl ein:
9
Das Quadrat der Zahl ist: 81

2.5.1 Der Modulo-Operator

Eine wichtige Anwendung in der Programmierung ist der *Modulo-Operator*. Dies entspricht in der Mathematik dem Rest nach einer Division. Zum Beispiel ist 11 geteilt durch 4 gleich 2 Rest 3. Wir schreiben z.B.:

$$\begin{aligned} 11 \ \ mod \ \ 4 &= 3 \\ 20 \ \ mod \ \ 5 &= 0 \\ 15 \ \ mod \ \ 4 &= 3 \\ 34 \ \ mod \ \ 12 &= 10 \end{aligned}$$

Eine Zahl ist genau dann durch eine andere Zahl teilbar, wenn das Ergebnis der Modulo Operation Null ist (also kein Rest übrig bleibt). In C++ wird diese Operation mit dem Prozent-Zeichen `%` durchgeführt. Es gilt dann also z.B. $20\%3 = 2$, $50\%5 = 0$ etc. Das folgende Programm testet, ob eine eingegebene Zahl durch drei teilbar ist, und falls nicht gibt sie den Rest aus.

Beispiel 2.5.2. Rechenoperatoren 2: Teilbarkeit durch 3

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a;
9
10   cout << "Bitte geben Sie eine Zahl ein: " << endl;
11   cin >> a;
12
13   if (a%3 ==0){ //Test ob a modulo 3 = 0
14     cout << a << " ist durch 3 teilbar." << endl;
15   }
16   else{
17     cout << a << " ist nicht durch 3 teilbar. Der Rest der Division
           ist" << a%3 << endl;
18   }
```

```
19
20    return 0;
21  }
```

```
Bitte geben Sie eine Zahl ein:
20
20 ist nicht durch 3 teilbar. Der Rest der Division ist 2
```

Beachten Sie, dass der Modulo-Operator nur auf Zahlen vom Typ `int` anwendbar ist.

2.6 Aufgaben

2.6.1 Eingabe, Ausgabe, Variablen

Aufgabe 2.6.1. Finden Sie die Fehler in dem folgenden Programmcode:

```
1  #include <iostream>
2  using namespace std;
3
4  int Main(){
5          cout<<Hallo Welt!endl;
6          return 0;
7  }
```

Aufgabe 2.6.2. Der folgende Code beinhaltet sowohl Syntax-Fehler (d.h. Fehler welche den Code nicht ausführbar machen) als auch Semantik-Fehler (d.h. Logik Fehler, welche falsche Ausgaben produzieren). Das Programm soll den Namen und die Größe von zwei Personen einlesen und anschließend ausgeben, um wie viel größer Person 1 ist. Finden Sie die Fehler!

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  main(){
7
8    int groesse1, groesse2;
9    string name1, name2;
10
11   cout << "Bitte Namen von Person 1 eingeben: " << endl;
12   cin << groesse1;
13   cout << "Bitte Namen von Person 2 eingeben: " << endl;
14   cin << groesse2;
15
16   cout << "Bitte Größe (in m) von Person 1 eingeben: " << endl
17   cin >> name2;
18   cout << "Bitte Größe (in m) von Person 2 eingeben: " << endl
19   cin >> name2;
20
21   cout << name1 << " ist " << groesse1/groesse2 << " mal so groß wie
         " << name2 << endl;
22
23   return 0;
24 }
```

Aufgabe 2.6.3. Schreiben Sie ein Programm, welches Sie zunächst auffordert ihren Nachnamen, und dann Ihren Vornamen einzugeben.
Das Programm soll ihren Vor- und Nachnamen einlesen, und anschließend ausgeben.
Hinweis: Als Datentyp für ihren Namen verwenden Sie (Typ `string`).

Aufgabe 2.6.4. Erweitern Sie Ihr Programm aus Aufgabe 2.6.3. Nun soll das Programm auch Ihren Geburtsort und Ihr Geburtsdatum einlesen. Verwenden Sie für den Tag und das Geburtsjahr den Datentyp (Typ `int`), für den Monat den Datentyp (Typ `string`).

Die Ausgabe des Programms sollte wie folgt aussehen:
Vorname: ...
Name: ...
Geburtsort: ...
Geburtsdatum: tt. Monat jjjj

Aufgabe 2.6.5. Schreiben Sie ein Programm, welches Ihr Geburtsdatum, sowie das aktuelle Datum einliest und anschließend ihr aktuelles Alter (in Jahren) ausgibt.

Aufgabe 2.6.6. Schreiben Sie ein Programm, welches fünf Zahlen (Typ `int`) einliest und:

- Die Summe aller fünf Zahlen,
- das Produkt,
- die Zahlen in umgekehrter Reihenfolge wieder ausgibt.

Aufgabe 2.6.7. Definieren Sie drei ganze Zahlen x,y und z und weisen Sie den Variablen die Werte 15, 17 und 21 zu. Geben Sie dann den Mittelwert der drei Werte aus. Achten Sie auf die Wahl der Datentypen!

2.6.2 `if`-Anweisung

Aufgabe 2.6.8. Wiederholung if-Anweisung: Finde die Fehler
Der folgende Code beinhaltet sowohl Syntax-Fehler (d.h. Fehler welche den Code nicht ausführbar machen) als auch Semantik-Fehler (d.h. Logik Fehler, welche falsche Ausgaben produzieren). Das Programm soll drei Zahlen `a,b,c` einlesen und diejenigen Zahlen die größer als Null sind wieder ausgeben. Hier ist der Code:

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6    int a,b,c;
7
8  int main(){
9
10
11   cout << "Bitte drei Zahlen eingeben" << endl;
12   cin >> a, b, c;
13
14   cout << "Die Zahlen die größer als 0 sind: " << endl;
15   if a>0
16     cout << a << endl;
17
18    if (b>0){
19     cout << b << endl;
```

```
20
21        if (c>0) {
22          cout << c << endl;
23        }
24      }
25
26     return 0;
27  }
```

Aufgabe 2.6.9. Schreiben Sie ein Programm, welches 2 Zahlen einliest (Typ `int`) und die größte der beiden Zahlen wieder ausgibt. Verwenden Sie hierfür eine `if`-Anweisung.

Aufgabe 2.6.10. Schreiben Sie ein Programm, welches die Punktzahl einer Klausur einliest. Es waren insgesamt 60 Punkte zu erreichen.

- Falls die Punktzahl unter 30 Punkten liegt, ist der Teilnehmer durchgefallen.
- Falls die Punktzahl über 30 Punkten liegt oder genau 30 ist, hat der Teilnehmer bestanden.
- Falls die Punktzahl über 55 Punkten liegt, hat der Teilnehmer mit einer sehr guten Leistung bestanden.

Nachdem der Teilnehmer seine Punktzahl eingegeben hat, soll auf dem Bildschirm ausgegeben werden, mit welcher Leistung er bestanden hat oder nicht bestanden hat.
Beispiel:

Eingabe: **56**
Herzlichen Glückwunsch! Sie haben 56 Punkte erreicht. Sie haben die Klausur mit einer sehr guten Leistung bestanden!

Aufgabe 2.6.11. Schreiben Sie nun ein Programm welches 3 Zahlen einliest (Typ `int`) und die Zahlen der Reihenfolge nach aufsteigend sortiert wieder ausgibt. Verwenden Sie auch hierfür `if`-Anweisungen. Versuchen Sie, die Anzahl der if-Abfragen durch die Verwendung von Vergleichsoperatoren zu verringern.

Aufgabe 2.6.12. C++ als Taschenrechner
Schreiben Sie ein Programm, welches zwei Zahlen (Typ `double`) und anschließend eines der vier Zeichen +,-,*,/ (Typ `char`) einliest. Das Programm soll dann je nach Eingabe die Summe, Differenz, Produkt oder den Quotienten der beiden Zahlen ausgeben. Testen Sie außerdem bei der Bildung des Quotienten auf ungültige Eingaben (Teilen durch Null).

Aufgabe 2.6.13. Teilbarkeit mit der if-Anweisung
Schreiben Sie ein Programm, welches ermittelt ob eine eingegebene Zahl durch 7 teilbar ist. Verwenden Sie hierfür den in der Vorlesung besprochenen Modulo-Operator '%'.

Aufgabe 2.6.14. Schreiben Sie ein Programm, welches die Fakultät einer einzugebenden Zahl n ausgibt, wobei $n \leq 5$ gilt. Die Fakultät einer Zahl ist so definiert:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1$$

z.B. ist

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

Definieren Sie hierfür zu Beginn Ihres Programms eine Variable *Ergebnis*, welche Sie mit dem Wert 1 initialisieren. Verwenden Sie dann eine Reihe von 5 `if`-Anweisungen, welche hintereinander stehen und je nach Größe der Zahl das Ergebnis verändern.

Aufgabe 2.6.15. if-Anweisung I
Schreiben Sie ein Programm, welches den Absolutbetrag einer einzugebenden Zahl ausgibt. Verwenden Sie hierfür eine `if`-Anweisung.

Aufgabe 2.6.16. if-Anweisung II
Schreiben Sie ein Programm, welches den Namen und das Alter von 4 Personen einliest. Anschließend soll das Programm einen Namen einlesen und das entsprechende Alter auf dem Bildschirm ausgeben.

Aufgabe 2.6.17. Wiederholung if-Anweisung, Modulo-Operator
Schreiben Sie ein Programm, welches berechnet ob ein eingegebenes Jahr ein Schaltjahr ist (also 366 statt 365 Tage hat). Für die Festlegung eines Schaltjahrs gelten folgende Regeln:

- Ist die Jahreszahl durch 4 teilbar, so ist das Jahr ein Schaltjahr, jedoch mit der Ausnahme:
- Wenn die Jahreszahl durch 100 teilbar ist, so ist das Jahr doch kein Schaltjahr. Auch dies kommt mit einer Ausnahme:
- Ist die Jahreszahl durch 400 teilbar, so ist das Jahr doch ein Schaltjahr.

Aufgabe 2.6.18. if-Anweisung und sortieren
Lesen Sie drei Zahlen `a1,a2,a3` ein und sortieren Sie diese der Größe nach. Am Ende soll die Variable `a1` die kleinste Zahl, `a2` die mittlere Zahl und `a3` die größte Zahl beinhalten.

Aufgabe 2.6.19. Potenzen mit der if-Anweisung
Schreiben Sie ein Programm das zwei integer-Zahlen n, k einliest, wobei $k \leq 5$ gelten soll. Anschließend soll die Potenz n^k ausgegeben werden. Definieren Sie analog am Anfang eine Variable `Ergebnis`, welches Sie mit dem Wert n initialisieren. Schreiben Sie anschließend eine Reihe von fünf if-Anweisungen, welche je nach Eingabe das Ergebnis anpassen.

3 Schleifen

3.1 Die for-Schleife

Schleifen sind dafür da etwas sehr oft durchzuführen. Dies ist in fast allen Bereichen wichtig. Die Syntax der for-Schleife geht folgendermaßen:

```
for (<Anfangsbefehl>; <Fortsetzungsbedingung>; <Iteration>){
<Anweisung>
}
```

Will man zum Beispiel alle Zahlen von 1 bis 10 ausgeben, so schreibt man das mit folgendem Code:

Beispiel 3.1.1. For-Schleife 1

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    for (int i=1; i<=10; i++){ //Beginn der for-Schleife
9      cout << i << endl; //Anweisung innerhalb der for-Schleife
10   } //Ende der for-Schleife
11
12   return 0;
13 }
```

```
1
2
3
4
5
6
7
8
9
10
```

Zu Beginn der Schleife, in Zeile 8 in Beispiel 3.1.1 wird eine `int`-Variable `i` deklariert und mit dem Wert 1 belegt. Der Schleifen-Körper (die Anweisung) wird so lange ausgeführt, bis die Fortsetzungsbedigung (hier i<=10) nicht mehr erfüllt ist. Nach jedem Durchlauf der Schleife wird die Schleifenvariable i um 1 erhöht (Iteration), und *anschließend* die Abbruchbedingung getestet. Wir können die Iteration auf zwei völlig gleichwertige Arten schreiben: “i=i+1” oder “i++”. Letzteres ist lediglich eine Kurzschreibweise. Mit for-Schleifen, if-Anweisung und Rechenoperatoren lassen sich schon sehr viele Programme verwirklichen. Zum Beispiel können wir für eine beliebige Zahl testen, durch welche Zahlen sie teilbar ist. Dies in im folgenden Beispiel 3.1.2 verdeutlicht. Dort wird eine if-Anweisung innerhalb des Schleifenkörpers ausgeführt, also bei jedem Schleifendurchgang.

Beispiel 3.1.2. For-Schleife 2: Ermitteln aller Zahlenteiler

```
1  #include <stdlib.h>
2  #include <iostream>
```

```
3
4  using namespace std;
5
6  int main(){
7
8    int a;
9
10   cout << "Bitte eine Zahl eingeben: " << endl;
11   cin >> a;
12
13   for (int i=1; i<=a; i++){ //Beginn der Schleife. Die Schleife lä
        uft so lange bis i>a erreicht ist.
14     if (a%i==0){ //Test ob die Teilung Null ergibt
15       cout << a << " ist durch " << i << " teilbar." << endl; //
           Ausgabe wenn die Teilung null ergibt, also a durch i
           teilbar ist
16     }
17   }
18
19   return 0;
20 }
```

```
Bitte eine Zahl eingeben:
20
20 ist durch 1 teilbar
20 ist durch 2 teilbar
20 ist durch 4 teilbar
20 ist durch 5 teilbar
20 ist durch 10 teilbar
20 ist durch 20 teilbar
```

Oder wir können ganz schnell die Summe der Zahlen kleiner gleich einer beliebigen Zahl ausgeben (Beispiel 3.1.3):

Beispiel 3.1.3. For-Schleife 3: Summe Berechnen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a;
9    int summe=0; //Variable summe wird mit 0 initialisiert
10
11   cout << "Bitte eine Zahl eingeben: " << endl;
12   cin >> a;
13
14   for (int i=1; i<=a; i++){
15     summe = summe + i; //Bei jedem Schleifendurchgang addieren wir i
         zur Summe
16   }
17
```

```
18    cout << "Die Summe der Zahlen kleiner gleich" << a << " ist " <<
          summe << endl;
19
20    return 0;
21  }
```

Bitte eine Zahl eingeben:
10
Die Summe der Zahlen kleiner gleich 10 ist 55

3.1.1 `break` Anweisung

Manchmal wollen wir auch während der Ausführung einer Schleife diese anhalten können, z.B. wenn wir während des Durchlaufs feststellen dass wir unser Ziel bereits erreicht haben und daher nicht mehr weiter rechnen müssen. Dies wollen wir im folgenden verdeutlichen. Eine sehr interessante Anwendung all dieser Konzepte ist die Ermittlung von Primzahlen. Primzahlen sind Zahlen, welche nur durch 1 und sich selbst teilbar sind. Da es bis heute keine Möglichkeit gibt mit einer mathematischen Formel alle Primzahlen zu berechnen bietet sich hierfür die Verwendung eines Programms an. Beispiel 3.1.4 liest eine Zahl vom Benutzer ein und testet ob es sich um eine Primzahl handelt.

Beispiel 3.1.4. For-Schleife 4: Primzahltest

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a;
9    int prim=1; //Deklaration einer Variable prim. prim=1 wenn a eine
        Primzahl ist, und prim=0 wenn nicht. Wir gehen am Anfang davon
        aus dass es sich um eine Primzahl handelt.
10
11   cout << "Bitte eine Zahl eingeben: " << endl;
12   cin >> a;
13
14   for (int i=2; i<a; i++){ //for-Schleife läuft von 2 bis a-1
15     if (a%i==0){ //Wenn dies erfüllt ist für irgendein i, dann ist a
           keine Primzahl
16       prim=0; //Wir setzen dann prim=0 und
17       break; //beenden die Schleife
18       }
19   }
20
21   if (prim==1){
22     cout << a << " ist eine Primzahl!" << endl;
23     }
24   else{
25     cout << a << " ist keine Primzahl!" << endl;
26     }
27
28   return 0;
29 }
```

```
Bitte eine Zahl eingeben:
727
727 ist eine Primzahl!
```

Die Verwendung von `break` ist immer dann sinnvoll, wenn sonst unnötig viel Rechenzeit verschwendet würde, wie in unserem Primzahl-Beispiel. Falls mehrere Schleifen ineinander verschachtelt sind, so beendet `break` immer nur die innerste Schleife innerhalb welcher der `break`-Befehl steht.

3.2 Die while-Schleife

Eine alternative zur for-Schleife ist die while-Schleife. Die Syntax ist:

```
while (<Bedingung>){
<Anweisung>
}
```

Anders als bei der for-Schleife gibt es bei der while-Schleife lediglich eine Bedingung, welche vor jedem Schleifendurchlauf getestet wird. Ist diese Bedingung erfüllt so wird die Schleifenanweisung ausgeführt. Es gibt also keinen vordefinierten Platz für Iteration und Initialisierung einer Schleifenvariablen. Beispiel 3.2.1 zeigt den Code eines Programms, welche die Zahlen zwischen zwei Zahlen a und b ausgibt.

Beispiel 3.2.1. While-Schleife 1

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a, b;
9
10   cout << "Bitte Zahl 1 eingeben: " << endl;
11   cin >> a;
12   cout << "Bitte Zahl 2 eingeben: " << endl;
13   cin >> b;
14
15   while(a<b){ //Ausführung so lange bis a>=b
16     cout << a << endl;
17     a=a+1;
18   }
19
20   return 0;
21 }
```

```
Bitte Zahl 1 eingeben:
10
Bitte Zahl 2 eingeben:
16
10
11
12
13
14
15
```

Ein interessanteres Beispiel für die Verwendung einer while-Schleife ist in folgendem Beispiel 3.2.2 gezeigt. Eine Zahl wird hier so lange durch 2 geteilt, bis es nicht mehr möglich ist.

Beispiel 3.2.2. While-Schleife 2: Durch 2 teilen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a;
9
10   cout << "Bitte Zahl eingeben: " << endl;
11   cin >> a;
12
13   while(a %2 == 0){ //Die while Schleife wird so lange ausgeführt
        bis a nicht mehr durch 2 teilbar ist.
14     a = a /2; //Wir definieren den Wert von a um..
15     cout << a << endl;
16   }
17
18   return 0;
19 }
```

```
Bitte Zahl eingeben:
1024
512
256
128
64
32
16
8
4
2
1
```

Obwohl bei einem solchen Beispiel die Verwendung einer while-Schleife Vorteilhaft zu sein scheint, können wir das selbe Programm mit einer for-Schleife umsetzen. Dies ist in Beispiel 3.2.3 veranschaulicht. Hierbei lassen wir die Felder für Initialisierung und Inkrement einfach leer (die Semikolons sind aber dennoch wichtig!).

Beispiel 3.2.3. For vs. while

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a;
9
```

```
10    cout << "Bitte Zahl eingeben: " << endl;
11    cin >> a;
12
13    for(;a %2 == 0;){ //Eine for-Schleife dient als while-Schleife
14      a = a /2;
15      cout << a << endl;
16    }
17
18    return 0;
19  }
```

Generell lässt sich jede for-Schleife auch als while-Schleife schreiben und umgekehrt. Ob man for oder while benutzt ist also eher Geschmacksache. Eine interessante Anwendung der while-Schleife ist in Beispiel 3.2.4 gezeigt. Hier geben wir so lange eine Zahl ein, bis die eingegebene Zahl durch 10 teilbar ist.

Beispiel 3.2.4. For vs. while

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a=1;
9
10   while (a %10!=0){
11     cout << "Bitte Zahl eingeben: " << endl;
12     cin >> a;
13   }
14
15   cout << a << " ist durch 10 teilbar. Programm wird beendet." <<
         endl;
16
17   return 0;
18 }
```

```
Bitte Zahl eingeben:
17
Bitte Zahl eingeben:
81
Bitte Zahl eingeben:
100
100 ist durch 10 teilbar. Programm wird beendet.
```

Zum Schluss wollen wir anhand unseres Primzahltests nochmal demonstrieren, dass `while` und `for` alternativ immer verwendbar sind (siehe Beispiel 3.2.5).

Beispiel 3.2.5. Primzahltest mit `while`

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
```

```
5
6  int main(){
7
8    int a;
9    int prim=1;
10
11   cout << "Bitte Zahl eingeben" << endl;
12   cin >> a;
13
14   int i=2;
15   while ((prim==1)&&(i<a)){ //Der Test der Schleife ersetzt die
         Verwendung von break.
16     if (a%i==0){
17       prim=0;
18       }
19     i++;  //Unsere Iteration über i steht nun im Schleifen-Körper
20   }
21
22   if(prim==1){
23     cout << a << " ist eine Primzahl!" << endl;
24   }
25   else{
26     cout << a << " ist keine Primzahl!" << endl;
27   }
28
29   return 0;
30 }
```

```
Bitte Zahl eingeben:
727
727 ist eine Primzahl!
```

3.3 Aufgaben

Aufgabe 3.3.1. Wiederholung for-Schleife
Schauen Sie sich den folgenden Programmcode genau an. Wie sieht die Ausgabe aus?

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main (){
7      for (int i = 1; i<=5; i++){
8          for (int j=1; j<=i; j++){
9              cout<<"*";
10         }
11     cout<< endl;
12     }
13     return 0;
14 }
```

Aufgabe 3.3.2. Berechnen Sie mit Hilfe von zwei verschachtelten for-Schleifen die folgende Summe:

$$\sum_{i=1}^{5}\sum_{j=1}^{5}(i+j)$$

Aufgabe 3.3.3. Schreiben Sie ein Programm, welches

- alle geraden Zahlen von 1 bis 30 ausgibt
- alle geraden Zahlen von 1 bis n ausgibt, wobei die Zahl n vom Benutzer eingegeben werden soll
- alle durch 3 teilbaren Zahlen von 1 bis n ausgibt, wobei die Zahl n vom Benutzer eingegeben werden soll

Aufgabe 3.3.4. Erzeugen Sie mit Hilfe zwei verschachtelter for-Schleifen die folgende Ausgabe:
1
12
123
1234
12345

Aufgabe 3.3.5. Erzeugen Sie mit Hilfe zwei verschachtelter for-Schleifen die folgende Ausgabe:
1
22
333
4444
55555

Aufgabe 3.3.6. Primzahlen II
Schreiben Sie ein Programm, welches alle Primzahlen bis zu einer einzugebenden Zahl n ausgibt. Sie brauchen hierfür zwei verschachtelte for-Schleifen.

Aufgabe 3.3.7. While-Schleife
Das folgende Programm gibt die Fakultät einer einzugebenden Zahl n aus. Schreiben Sie das Programm so um, dass sie anstatt der `for`-Schleife, eine `while`-Schleife benutzen.

```
1  int main(){
2      int fakultaet=1;
3      cout<<"Bitte geben Sie eine Zahl ein :"<<endl;
4      cin>>n;
5      for (int i=1;i<=n;i++){
6          fakultaet=fakultaet*i;
7      }
8      cout<<fakultaet;
9
10     return 0;
11 }
```

Aufgabe 3.3.8. Kleines Einmaleins
Schreiben Sie ein Programm welches das kleine Einmaleins berechnet und ausgibt. Hierfür benötigen Sie zwei verschachtelte for-Schleifen.

Aufgabe 3.3.9. Potenzen mit der for-Schleife
Schreiben Sie ein Programm das zwei integer-Zahlen n, k einliest. Anschließend soll die Potenz n^k ausgegeben werden. Definieren Sie am Anfang eine Variable `Ergebnis`, welches Sie mit dem Wert 1 initialisieren. Schreiben Sie anschließend eine `for`-Schleife, welche dieses Ergebnis Schritt für Schritt anpasst.

Aufgabe 3.3.10. Summen mit der for-Schleife
Berechnen Sie die Summe aller natürlichen Zahlen von 1 bis zu einer einzugebenden Zahl n. Testen Sie explizit mit C++ die Relation:

$$1 + 2 + 3 + \cdots + (n-1) + n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Aufgabe 3.3.11. Schere, Stein Papier
Schreiben Sie ein Programm, welches den Computer gegen Sie "Schere, Stein, Papier" spielen lässt. Der Computer wählt dabei völlig zufällig eine der drei Möglichkeiten aus, der Benutzer kann seine Auswahl zu jeder Runde neu eingeben. Wer zuerst fünf mal gewinnt hat das Spiel gewonnen.
Tipp: Sie müssen dem Programm sagen wer wann gewinnt. Verwenden Sie verschachtelte `if`-Anweisungen um die einzelnen Ergebnisse auszuwerten.

Aufgabe 3.3.12. Fibonacci-Folge
Schreiben Sie ein Programm, welches die ersten 25 Folgenglieder der Fibonacci-Folge f_n berechnet. Jede Fibonacci-Zahl ist gleich der Summe der beiden vorhergehenden Fibonacci-Zahlen. Die beiden ersten Elemente sind $f_0 = 0, f_1 = 1$. Für alle weiteren Elemente gilt $f_n = f_{n-1} + f_{n-2}$.
Speichern Sie die Fibonacci-Zahlen in einem Array a. Initialisieren Sie dafür die ersten beiden Folgenglieder wie folgt: $a[0] = 0, a[1] = 1$.

Aufgabe 3.3.13. If-Anweisung, Schleifen und Zufallszahlen
Schreiben Sie ein Programm "Zahlenraten", bei welchem der Computer eine Zufallszahl zwischen 1 und 100 generiert und Sie durch eine Eingabe die Zahl erraten müssen. Nach jedem Versuch soll der Computer sagen ob die eigentliche Zahl größer oder kleiner als Ihre Vermutung war.
Mit C++ können Zufallszahlen folgendermaßen generiert werden:

- Binden Sie den Header time.h ein. (`#include <time.h>`)
- Eine Zufallszahl zwischen 1 und 100 wird über folgenden Befehl generiert: `int a = rand()%100+1`. Rufen Sie zuvor einmalig folgenden Befehl `srand(time(NULL))` auf, um die Zufallszahlen zu initialisieren (ohne diesen Befehl ergibt sich jedes mal dieselbe Zufallszahl).

Die Eingabe einer Zahl findet in einer Schleife statt, wobei die Abbruchbedingung von der Eingabe abhängt. Verwenden Sie die `break`-Anweisung zum Abbrechen der Schleife.

Aufgabe 3.3.14. Jede beliebige Zahl kann in Primfaktoren zerlegt werden (siehe Übung). Schreiben Sie ein Programm das die Primfaktorzerlegung einer beliebigen Zahl berechnet. Dafür brauchen wir eine for-Schleife. Allerdings müssen wir jetzt nicht nur auf Teilbarkeit testen, sondern die Division dann auch ausführen.
Tipp: Bei allen Schleifen können Variablen in der Schleifen-Initialisierung und der Schleifen-Bedingung während des Durchlaufens angepasst werden.

4 Arrays, Vektoren und externe Daten

4.1 Arrays

Ein Array ist eine Folge von Variablen gleichen Typs. Über einen Index kann man auf die Elemente eines Arrays zugreifen. Die Syntax ist:

- Deklaration:

```
<Datentyp> <Name>[<Anzahl Elemente>]
```

- Zugriff:

```
<Name>[<Index>]
```

Wir können uns einen Array wie eine Reihe von Boxen vorstellen, in welchen wir die Elemente des Arrays platzieren (siehe Abbildung 1). In Beispiel 4.1.1 legen wir einen Array vom Typ `string` an, welcher fünf Elemente enthält. Anschließend belegen wir den Array mit verschiedenen Namen und geben diese dann in einer for-Schleife wieder aus.

Beispiel 4.1.1. Arrays 1

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    string name[5]; //Deklaration des Arrays
9
10   name[0]="Alfred"; //Wir belegen die 5 Einträge 0,...,4 mit Werten
         (strings)
11   name[1]="Berta";
12   name[2]="Claus";
13   name[3]="Dorothea";
14   name[4]="Emil";
15
16   for (int i=0; i<5; i++){ //Ausgabe der Werte im Array
17     cout << "Nummer " << i << " ist: " << name[i] << endl;
18   }
19
20   return 0;
21 }
```

```
Nummer 0 ist: Alfred
Nummer 1 ist: Berta
Nummer 2 ist: Claus
Nummer 3 ist: Dorothea
Nummer 4 ist: Emil
```

Wichtig bei der Verwendung von Arrays sind folgende Punkte:

- Arrays erkennt man grundsätzlich an den eckigen Klammern [].
- Jeder Array beinhaltet nur Einträge des selben Datentyps. Dieser wird bei der Deklaration (hier `string`) festgelegt.

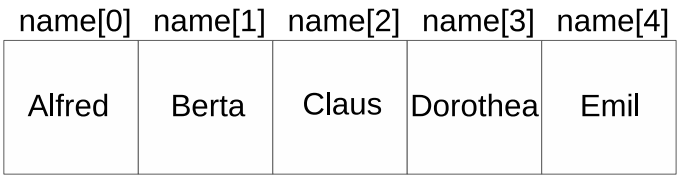

Abbildung 1: Veranschaulichung eines Arrays vom Typ string als Zusammengesetzte Boxen mit Inhalt.

- Die Anzahl der Elemente in der Deklaration ist eine positive ganzzahlige Konstante.
- Der Index, über welchen auf die Einträge im Array zugegriffen wird, beginnt bei Null! Das heißt dass die Deklaration "`int a[5]`" die Variablen a[0], a[1], a[2], a[3] und a[4] definiert. a[5] existiert nicht! Zugriffe auf Einträge außerhalb des Index-Bereichs sind verboten und führen meist zu Fehlern!

Arrays sind immer dann wichtig, wenn wir mit einer größeren Menge von Variablen *gleichen Typs* arbeiten. Zum Beispiel können wir mit einem Array und einer for-Schleife schnell 10 Zahlen einlesen und in umgekehrter Reihenfolge wieder ausgeben, wie Beispiel 4.1.2 zeigt.

Beispiel 4.1.2. Arrays 2: Ein- und Ausgabe

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int a[10];
9
10   //Einlesen der 10 Zahlen im Array
11   for (int i=0; i<10; i++){ //Einlesen der Array-Einträge
12     cout << "Eintrag " << i << " eingeben: ";
13     cin >> a[i];
14     cout << endl;
15   }
16
17   //Rueckgabe in umgekehrter Reihenfolge
18   for (int i=9; i>=0; i--){ //Ausgabe der Array-Einträge in
        umgekehrter Reihenfolge
19     cout << a[i] << endl;
20   }
21
22   return 0;
23 }
```

Für jeden Datentyp kann man Arrays definieren.
Insbesondere lassen sich Zeichenketten entweder als einzelner `string` oder als Array von `char` einlesen, wie Beispiel 4.1.3 zeigt. Beachten Sie dass ähnliches nicht mit anderen Arrays in dieser Form funktioniert.

Beispiel 4.1.3. Arrays 3: Ein- und Ausgabe von `char` Arrays

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
```

```
6  int main(){
7
8    char abk[3];
9
10   cout << "Bitte Abkürzung eingeben" << endl;
11   cin >> abk;
12
13   cout << abk;
14
15   return 0;
16 }
```

```
Bitte Abkürzung eingeben
USA
USA
```

4.1.1 Sortieren von Arrays

Arrays sind im Prinzip nichts anderes als Listen, deren Einträge Objekte des selben Datentyps sind. Eine wichtige Anwendung in Bezug auf Listen ist das **Sortieren** von Listen. Stellen Sie sich vor Sie haben eine Liste von 100 Zahlen und sollen diese der Reihenfolge aufsteigend Ordnen. Wie gehen Sie vor? In der Programmierung gibt es eine Reihe von Sortier-Algorithmen, und an dieser Stelle soll ein solcher Algorithmus besprochen werden: der sogenannte **Selectionsort**.
Wir stellen uns eine Liste mit n Zahlen vor. Beim Selectionsort wird zunächst, angefangen beim ersten Element des Arrays, die ganze Liste durchsucht und das kleinste Element ermittelt. Dieses wird dann mit dem ersten Element im Array vertauscht. Anschließend wird nun, beginnend beim zweiten Element des Arrays, das kleinste Element der restlichen $n-1$ Einträge ermittelt und mit dem zweiten Element im Array vertauscht. Man beginnt nun beim dritten Element und sucht nach dem kleinsten Element in den verbleibenden $n-2$ Zahlen. Dies macht man bis man am Ende angekommen ist. Beispiel 4.1.4 zeigt den Selectionsort am Beispiel einer Liste mit 10 `int` Einträgen.

Beispiel 4.1.4. Arrays 3: Ein- und Ausgabe von `char` Arrays

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7
8    int liste[10]; //Deklaration des Arrays mit 10 integern
9
10   liste[0] = 202; //Wir belegen die Array-Einträge mit Werten
11   liste[1] = 17;
12   liste[2] = 290;
13   liste[3] = 224;
14   liste[4] = 197;
15   liste[5] = 83;
16   liste[6] = 537;
17   liste[7] = 920;
18   liste[8] = 871;
19   liste[9] = 422;
20
21   int pos; //pos ist die Position des kleinsten Elements
```

```
22    int speicher; //Eine dummy variable um Werte zu tauschen
23
24    for (int i=0; i<10; i++){ //Schleife 1
25      pos=i; //Wir nehmen an dass das Element i das kleinste ist
26      for (int j=i; j<10; j++){ //Schleife 2
27
28        if(liste[j]<liste[pos]){ //Wenn das Element kleiner ist als
              das an der Position pos wird pos neu definiert
29          pos=j;
30        }
31      }
32      speicher = liste[i]; //Vertauschen der Einträge
33      liste[i]=liste[pos];
34      liste[pos]=speicher;
35    }
36
37    cout << "Die Sortierte Liste ist:" << endl;
38    for (int i=0; i<10; i++){ //Ausgabe des sortierten Arrays
39      cout << liste[i] << endl;
40    }
41
42    return 0;
43  }
```

```
Die Sortierte Liste ist:
17
83
197
202
224
290
422
537
871
920
```

4.1.2 Dynamische Speicherallokation

Bisher haben wir nur Arrays mit fester Größe verwendet. In Beispiel 4.1.4 z.B. mussten wir bei der Deklaration `int liste[10]` angeben dass der Array 10 Elemente enthält. Was wenn wir die Anzahl der Element nicht von vornherein festlegen, sondern z.B. vom Benutzer eingeben lassen wollen? In diesem Fall müssen wir die Größe des Arrays **dynamisch** festlegen, d.h. während das Programm ausgeführt wird. Dies ist in Beispiel 4.1.5 veranschaulicht. Die dabei verwendeten Pointer (alles was mit einem "*" im Code steht) werden in Abschnitt 6 genauer erklärt, das genaue Verständnis ist für unsere Zwecke an dieser Stelle aber nicht wichtig.

Beispiel 4.1.5. Array mit dynamischer Länge einlesen I

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
```

```
7
8       int n;
9
10      cout << "Bitte die Anzahl der Einträge eingeben: " << endl;
11      cin >> n;
12
13      int *liste; //Wir deklarieren einen Zeiger
14      liste = new int[n]; //Hier wird nun ein Array an der Speicher-
           Adresse erstellt, zu welcher liste zeigt
15
16      for (int i=0; i<n; i++){ //Eingabe der Liste
17         cin >> liste[i];
18        }
19
20      for (int i=0; i<n; i++){ //Ausgabe der Liste
21          cout << liste[i];
22      }
23
24      delete[] liste; //Wir löschen die Liste wenn wir sie nicht mehr
           brauchen
25      return 0;
26  }
```

Beispiel 4.1.5 fragt zu Beginn in Zeile 10 nach der Anzahl der Einträge und schafft dann einen Array mit dieser Anzahl von Einträgen in den Zeilen 13 und 14. Anschließend werden die Einträge des Array vom Benutzer eingegeben und wieder ausgegeben. Man bemerke, dass viele Compiler eine vereinfachte Version der dynamischen Arraylänge verstehen. Dies ist in Beispiel 4.1.6 veranschaulicht. Hierfür ist kein Zeiger notwendig.

Beispiel 4.1.6. Array mit dynamischer Länge einlesen II

```
1   #include <cstdlib>
2   #include <iostream>
3
4   using namespace std;
5
6   int main() {
7
8       int n;
9
10      cout << "Bitte die Anzahl der Einträge eingeben: " << endl;
11      cin >> n;
12
13      int liste[n]; //Wir deklarieren einen Array mit Länge n
14
15      for (int i=0; i<n; i++){ //Eingabe der Liste
16         cin >> liste[i];
17        }
18
19      for (int i=0; i<n; i++){ //Ausgabe der Liste
20          cout << liste[i];
21      }
22
23      delete[] liste; //Wir löschen die Liste wenn wir sie nicht mehr
           brauchen
```

```
24      return 0;
25  }
```

4.2 Vektoren

Ein Vektor (englisch und in C++ *vector*) ist ein Container welcher eine Liste von Daten speichern kann. Wie bei einem Array kann durch einen Index auf die verschiedenen Einträge zugegriffen werden. Anders als bei Arrays ist jedoch die Länge von Vektoren grundsätzlich dynamisch: Man muss sie nicht zu Beginn des Programms festlegen. Zudem haben Vektoren einige angenehme Eigenschaften - man kann z.B. neue Elemente ans Ende oder an eine bestimmte Position im Vektor einfügen. Die Syntax zur Deklaration ist:

```
vector <(Datentyp)> <Name>
```

Zusätzlich muss der Header `vector` eingebunden werden:

```
#include <vector>
```

Als Beispiel behandeln wir einen Vektor vom Typ `int`. Wie Arrays können Vektoren aber auch komplexere Objekte (z.B. `structs`) beinhalten.

Beispiel 4.2.1. Vektoren 1

```
1  #include <stdlib.h>
2  #include <iostream>
3  #include <vector>
4
5  using namespace std;
6
7  int main(){
8
9    vector <int> v;
10
11   v.push_back(2);
12   v.push_back(5);
13
14   for (int i=0; i<v.size(); i++){
15     cout << "Eintrag Nr. " << i << ": " << v[i] << endl;
16   }
17
18   cout << "*************" << endl;
19   v.push_back(6);
20
21   for (int i=0; i<v.size(); i++){
22     cout << "Eintrag Nr. " << i << ": " << v[i] << endl;
23   }
24
25   return 0;
26 }
```

```
2
5
*************
2
5
6
```

Wir sehen im Beispiel 4.2.1 zwei wichtige Befehle für Vektoren: `v.size()` gibt die Länge des Vektors zurück. Wie bei Arrays können wir über den Befehl `v[i]` auf die einzelnen Elemente zugreifen. Ist die Länge eines Vektors n, so läuft der Index von 0 bis $n-1$, wie bei Arrays. Außerdem sehen wir im Beispiel den Befehl `v.push_back(objekt)`. Die Zahl die in der Klammer steht wird an das Ende des Vektors angefügt. Die Länge wird dadurch um 1 erhöht. Man beachte dass das Objekt das hinzugefügt wird vom selben Datentyp wie der Vektor sein muss.
Die Übergabe eines Vektors an eine Funktion erfolgt am einfachsten per Referenz, wie in Beispiel 4.2.2 gezeigt ist.

Beispiel 4.2.2. Übergabe eines Vektors per Referenz

```
1  #include <stdlib.h>
2  #include <iostream>
3  #include <vector>
4
5  using namespace std;
6
7  void ausgabe(vector <int> &v){
8     for (int i=0; i<v.size(); i++){
9       cout << "Eintrag Nr. " << i << ": " << v[i] << endl;
10    }
11 }
12
13 int main(){
14
15   vector <int> v;
16
17   v.push_back(2);
18   v.push_back(5);
19   v.push_back(6);
20
21   ausgabe(v);
22
23   return 0;
24 }
```

```
2
5
6
```

4.3 Arbeiten mit externen Daten

In diesem Kapitel behandeln wir das Einlesen und die Ausgaben von externen Daten. Die Beispiele in diesem Kapitel verwenden meist das Konzept von structs, welche in Kapitel 7 eingeführt werden.

4.3.1 Daten Einlesen

Wenn wir Daten mit C++ analysieren wollen, so müssen wir diese in C++ lesen. Das Einlesen von Daten ist an sich nicht kompliziert, sondern beinhaltet lediglich ein paar neue Befehle. Im Prinzip lässt sich das Einlesen für unsere Zwecke in drei Schritte unterteilen:

1. Eine externe Datei wird als Objekt der Klasse `ifstream` geöffnet.
2. Es wird getestet ob die Datei erfolgreich geöffnet wurde. Dies geschieht mit dem Befehl `assert`. Dies muss nicht generell gemacht werden, ist aber hilfreich um mögliche Fehlerquallen ausfindig zu machen.

3. Die Daten werden eingelesen. Dies erfolgt ähnlich zum Einlesen mit `cin`, jedoch steht anstatt `cin` nun der Name des `ifstream`-Objekts am Anfang der Zeile.

Man beachte, dass zusätzliche `header-Dateien` eingebunden werden müssen, um mit `ifstream` zu arbeiten (wie im folgenden Beispiel verdeutlicht).
Im folgenden soll anhand eines Beispiels das Konzept des Daten Einlesens veranschaulicht werden. Wir haben einen Array von `structs` mit Namen `person`, welche als Komponenten einen Namen und ein Alter haben. Folgende Daten sollen in einen Array mit fünf Elementen eingelesen werden (Name und Alter sind jeweils nebeneinander). Die Daten liegen in der Datei `Beatles.txt` vor:

```
John 20
Paul 50
George 32
Ringo 38
```

Beispiel 4.3.1. Daten Einlesen mit ifstream

```
1  #include <stdlib.h>
2  #include <iostream>
3  #include <fstream> // Fuer das Einlesen der Dateien
4  #include <cassert> // Fuer den Test of die Datei erfolgreich geö
       ffnet wurde
5
6  using namespace std;
7
8  struct person{
9    string Name;
10   int Alter;
11 };
12
13 int main(){
14   person liste[4]; //Anlegen des Arrays von Personen
15
16   ifstream myfile("Beatles.txt"); // Hier öffnen wir die Datei als
         ifstream-Objekt und nennen es myfile
17
18   assert (myfile.is_open()); // Test ob das File geöffnet wurde.
         Wenn dies nicht der Fall ist erfolgt eine Fehlermeldung.
19
20   // Nun beginnen wir mit der Eingabe durch eine for-Schleife
21   for (int i=0; i<4; i++){
22     myfile >> liste[i].Name >> liste[i].Alter;
23   }
24
25   // Anschliessend geben wir die Daten durch eine for-Schleife
         wieder aus
26
27   for (int i=0; i<4; i++){
28       cout << liste[i].Name << " " << liste[i].Alter << endl;
29     }
30
31   return 0;
32 }
```

Wir sehen dass die Eingabe genauso wie bei `cin` durch ein “>>“ erfolgt und die einzelnen Worte oder Zahlen durch weitere “>>“ getrennt sind. Ein weiteres Beispiel mit einer struct namens “Buch” ist in Beispiel 4.3.2 verdeutlicht.

Beispiel 4.3.2. Struct Buch

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4
5  using namespace std;
6
7  struct buch //Definition der struct
8  {
9    string autor;
10   string titel;
11   int jahr;
12
13 };
14
15 void eingabe_aus_datei(buch *buchliste,int laenge_liste) //Funktion
       zum Daten Einlesen aus externer Datei "buecher.dat"
16 {
17
18     ifstream read_file("buecher.dat"); //Öffnen der externen Datei
19
20     for (int i=0;i<laenge_liste;i++)
21       {
22       getline(read_file, buchliste[i].autor); //Einlesen des Autors
23       getline(read_file, buchliste[i].titel); //Einlesen des Titels
             in der nächsten Zeile
24       read_file >> buchliste[i].jahr; //Einlesen des Jahres in der n
             ächsten Zeile
25       read_file.ignore();
26       }
27
28     read_file.close(); //Schließen der Datei
29
30 }
31
32 void ausgabe(buch a) //Definition der Ausgabe-Funktion
33 {
34   cout << a.autor << ", " << a.titel << ", " << a.jahr << endl;
35 }
36
37 int main()
38 {
39   int i;
40   int n;
41   cout << "Wie viele B   cher  m   chten Sie einlesen?" << endl;
42   cin >> n;
43   cin.ignore();
44   buch *buchliste; //Deklaration eines Arrays mit Länge n
45   buchliste=new buch[n];
46
47
```

```
48     eingabe_aus_datei(buchliste,n);
49
50      cout << "-----------------------------------------" << endl;
51      cout << "| Ihre Bücherliste enthält folgende " << n << " Bücher
           | " <<endl;
52      cout << "-----------------------------------------" << endl;
53
54
55      for (i=0;i<n;i++)
56        {
57          ausgabe(buchliste[i]);
58        }
59
60      cout << endl;
61      cout << endl;
62
63      delete[] buchliste; //Löschen des Arrays um Speicher wieder
           freizugeben
64      return 0;
65  }
```

4.3.2 Daten in Datei schreiben

Das Schreiben in eine externe Datei erfolgt recht ähnlich zum Lesen aus einer Datei. Beispiel 4.3.3 zeigt ein Programm, welches das kleine Einmaleins in eine Datei namens “Einmaleins.dat” schreibt. Wie bei `cin/cout` haben wir beim Schreiben von Daten den Operator “<<” anstatt “>>”.

Beispiel 4.3.3. Struct Buch

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4
5  using namespace std;
6
7  int main(){
8
9    ofstream myfile;
10
11   myfile.open ("Zahlen.dat");
12
13   for (int i=0; i<10;i++){
14
15     for (int j=0; j<10; j++={
16
17         myfile << i*j;
18       }
19    myfile << endl;
20
21     }
22   myfile.close();
23   return 0;
24  }
```

Der Inhalt der Datei “Zahlen.dat” hat nach der Ausführung des Programms folgende Form:
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100

4.4 Aufgaben

Aufgabe 4.4.1. Arrays I
Schreiben Sie ein Programm, welches 5 Zahlen einliest, in einem Array speichert und anschließend die Summe, den Durchschnitt und das Produkt berechnet. Lesen Sie dabei die Einträge des Arrays über eine for-Schleife ein und berechnen Sie die auszugebenden Werte ebenfalls in einer for-Schleife.

Aufgabe 4.4.2. Arrays II
Schreiben Sie ein Programm, welches das Quadrat aller Zahlen $n \leq 20$ in einem Array abspeichert. Verwenden Sie hierfür eine for-Schleife und initialisieren Sie die Array-Einträge durch die Anweisung $a[i] = i*i$. Geben Sie anschließend in einer weiteren for-Schleife die Differenzen zwischen den benachbarten Array-Elementen aus. Was fällt auf? Können Sie die Beobachtung begründen?

Aufgabe 4.4.3. Überall Würfel
Stellen Sie sich vor sie haben 10 Würfel und werfen alle gleichzeitig. Die Summer der Augen ergibt eine Zahl z mit $10 \leq z \leq 60$. Wir interessieren uns für die Wahrscheinlichkeit dass die Summe eine bestimmte Zahl ist. Diese Aufgabe ist nicht trivial und erfordert zur exakten Lösung einiges an Kombinatorik. Wir wollen deshalb C++ für uns würfeln lassen:

- Ziel ist es, mit den 10 würfeln 10000 mal zu würfeln um eine gute Statistik zu bekommen.
- Definieren Sie einen Array `int summe[61]`, in dem Sie die Anzahl der Würfe einer gewissen Zahl z speichern. Ist die Summe z.B. $z = 20$, so soll die Zahl in `summe[20]` um 1 erhöht werden. Der Array muss am Anfang mit Null initialisiert werden.
- Lassen Sie nun C++ 10000 mal für sich würfeln (for-Schleife) und updaten Sie den Array bei jeder Runde.
- Machen Sie eine Abfrage am Schluss, welche die Wahrscheinlichkeit eine bestimmte Zahl z zu würfeln ausgibt. Diese ist für 10000 mal Würfeln einfach definiert als:

$$p(z) = summe[z]/10000$$

.

Aufgabe 4.4.4. Funktion auswerten
Wir betrachten die Funktion $f(x) = 2x^2$. Schreiben Sie ein Programm, welches die Funktion $f(x)$ in den Punkten 0,1,2,3,...20 auswertet und in einem Array a abspeichert, so dass $a[i] = f(i)$.

Aufgabe 4.4.5. Variable Array-Größe
Schreiben Sie nun ein Programm, dass ein Array mit Integer-Einträgen erzeugt, indem von Nutzer folgende Informationen abgefragt werden:
1. Anzahl der einzugebenden Zahlen n
2. n mal den Wert beim Nutzer abfragen
Anschließend soll dieses Array in umgekehrter Reihenfolge ausgegeben werden und für jeden Wert soll die Position innerhalb des Arrays ausgegeben werden.
Die Ausgabe könnte z.B. wie folgt aussehen:

```
1 Wie viele Werte wollen Sie eingeben? 3
2 Geben Sie den 1. Wert ein: 12
3 Geben Sie den 2. Wert ein: 2
4 Geben Sie den 3. Wert ein: 3
5 Wert an Position 2 im angelegten Array: 3
6 Wert an Position 1 im angelegten Array: 2
7 Wert an Position 0 im angelegten Array: 12
```

Aufgabe 4.4.6. Bubble-Sort
Belegen Sie ein Array beliebiger Größe mit Zufallszahlen von 1 bis 9. Sortieren Sie das Array dann aufsteigend mit dem **Bubble-Sort-Verfahren:** Jeweils zwei benachbarte Einträge werden vertauscht, wenn sie in der falschen Reihenfolge stehen. Prüfen sie hierzu von hinten beginnend, für jeden Eintrag, ob der davorstehende Eintrag größer ist als der Eintrag selbst.

Bsp: Sortiere 1 7 5 3 2 :

1. Iteration: 1 7 5 2 3
2. Iteration: 1 7 2 5 3
3. Iteration: 1 2 7 5 3
4. Iteration: 1 2 7 3 5
5. Iteration: 1 2 3 7 5
6. Iteration: 1 2 3 5 7

Aufgabe 4.4.7. Sortieren von Arrays mit *Selectionsort*
Definieren Sie einen `int`-Array variabler Größe n mit Zufallszahlen zwischen 1 und 9. Wir wollen diesen Array nun mit dem sogenannten *selectionsort*-Algorithmus sortieren. Dieser funktioniert wir folgt:

- Beginnen Sie beim ersten Element und suchen Sie zunächst nach dem kleinsten Element im Array. Vertauschen Sie dieses anschließend mit dem ersten Element.
- Beginnend beim zweiten Element im Array, suchen Sie wieder nach dem kleinsten Element und vertauschen Sie es mit dem zweiten Element im Array.
- Verfahren Sie damit, bis Sie ans Ende des Array gelangen.

Geben Sie den Array vor und nach dem Sortieren aus, um zu sehen ob ihr Code richtig arbeitet.

Bsp: Sortiere 7 4 1 3 2 :

1. Iteration: 1 4 7 3 2
2. Iteration: 1 2 7 3 4
3. Iteration: 1 2 3 7 4
4. Iteration: 1 2 3 4 7

5 Funktionen

5.1 Einführung

Die Verwendung von Funktionen ist eines der Hauptwerkzeuge in der Programmierung. Funktionen sind dabei unabhängige Unterprogramme eines Programms, welche eine spezielle Aufgabe ausführt. Folgende Eigenschaften sind wichtig:

- Jedes Programm hat genau eine `main`-Funktion. Dies ist die Haupt-Funktion eines jeden Programms und diese wird beim Programmstart ausgeführt. Andere Funktionen werden nur ausgeführt, wenn sie in `main` aufgerufen werden.
- Beim Aufruf einer Funktion können Daten (Funktionsparameter) übergeben werden.
- Jede Funktion kann auch andere Funktionen aufrufen.

Dieser Abschnitt führt Schritt für Schritt in die Verwendung von Funktionen ein.

5.2 Funktionen ohne Rückgabewerte (`void`)

Die Verwendung von Funktionen hat viele Vorteile. Wollen wir zum Beispiel das Quadrat einer Zahl a berechnen, so können wir das mit folgendem Programm machen:

Beispiel 5.2.1. Quadrat berechnen ohne Funktion

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(){
7    int a;
8
9    cout << "Bitte Zahl eingeben: " << endl;
10   cin >> a;
11
12   cout << "Das Quadrat von " << a << " ist " << a*a << endl;
13   return 0;
14 }
```

Wir können das gleiche Programm mit Hilfe einer Funktion `square` schreiben. Bevor wir das tun wollen wir kurz Funktionen in der Mathematik besprechen, da das Konzept im Prinzip dasselbe ist.
Die Idee der Funktion kommt, wie so vieles in der Programmierung, aus der Mathematik. Eine Funktion ist dort, grob formuliert, eine Zuordnung von einem Objekt zu einem anderen Objekt, meist von einer Zahl zu einer anderen Zahl. Die Mathematik ist voll von Funktionen. Berühmte Funktionen sind zum Beispiel:

- Die Fakultät $n! = n \cdot (n-1) \cdot ... \cdot 3 \cdot 2 \cdot 1$.
- Die Wurzelfunktion $f(x) = \sqrt{x}$.
- Die Exponentialfunktion $f(x) = e^x$.
- Das arithmetische Mittel zweier Zahlen $f(x, y) = (x + y)/2$.

All diese Funktionen lassen sich mit entsprechenden Algorithmen sehr einfach in C++ implementieren. Wir wollen die Quadratfunktion $f(x) = x^2$ etwas genauer besprechen. Die Funktion bekommt als **Argument** eine Zahl, in diesem Fall x. Diese wird von der Funktion aufgenommen und eine andere Zahl, der Funktionswert, wird zurückgegeben. So ist z.B. $f(2) = 4$. Hierbei ist 2 das Argument und 4 der Funktionswert.
Ganz Ähnlich funktioniert das in C++ - siehe Beispiel 5.2.2.

Beispiel 5.2.2. Quadrat berechnen mit einer Funktion

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  void square(int a){ //Funktionsdefinition
7    cout << "Das Quadrat von " << a << " ist " << a*a; //Funktionskö
       rper
8  }
9
10 int main(){
11   int a;
12
13   cout << "Bitte Zahl eingeben: " << endl;
14   cin >> a;
15
16   square(a);
17   return 0;
18 }
```

In Beispiel 5.2.2 definieren wir die Funktion `square` bereits vor `main` und in `main` wird die Funktion aufgerufen um das Quadrat einer Zahl auszugeben. Die Funktion `square` hat genauso wie die mathematische Quadratfunkion ein Argument, welches nach der Funktionsdefinition in der Klammer steht, im Beispiel `int a`, d.h. die Funktion bekommt eine ganze Zahl gespeichert in der Variablen a.
Ein paar Kommentare zu dein einzelnen Zeilen in Beispiel 5.2.2:

- In Zeile 6 beginnt die Definition der Funktion. Generell werden Funktionen mit folgender Syntax definiert:

  ```
  <Typ><Funktionsname>(<Typ Var1><Name Var1>,...){
    <Programmcode>
  }
  ```

  In unserem Beispiel ist die Funktion vom Typ `void`. Dies bedeutet, dass die Funktion keinen Wert zurück gibt, sondern lediglich einen Befehl ausführt. In unserem Beispiel gibt die Funktion das Quadrat aus, gibt den Wert an sich jedoch nicht wieder an `main` zurück.

- Die Variablen welche der Funktion übergeben werden sollen (im Beispiel `int a`) stehen in einer Klammer hinter dem Funktionsnamen und müssen stets mit dem entsprechenden Datentyp angegeben werden.

- Danach folgt in geschweiften Klammern der beim Aufruf der Funktion auszuführende Programmcode.

Wo ist der Vorteil der Verwendung von Funktionen? Die Programme in den beiden Beispielen 5.2.1 und 5.2.2 machen effektiv das gleiche, dennoch hat die Implementierung mit Funktion ein paar Vorteile:

- Das Programm `main` wird logisch von der Berechnung getrennt.

- Der Programmcode wird verständlicher und übersichtlicher.

- Das Testen des Programms wird erleichtert.

Außerdem haben Funktionen den entscheidenden Vorteil, dass wir sie im Code beliebig oft aufrufen können. Zum Beispiel:

Beispiel 5.2.3. Quadrat berechnen mit Funktion 2

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  void square(int a){
7    cout << "Das Quadrat von " << a << " ist " << a*a << endl;
8  }
9
10 int main(){
11   int a, b;
12
13   cout << "Bitte Zahl 1 eingeben: " << endl;
14   cin >> a;
15   cout << "Bitte Zahl 2 eingeben: " << endl;
16   cin >> b;
17
18   square(a);
19   square(b);
20
21   return 0;
22 }
```

```
Bitte Zahl 1 eingeben:
8
Bitte Zahl 2 eingeben:
5
Das Quadrat von 8 ist 64
Das Quadrat von 5 ist 25
```

Das Programm in Beispiel 5.2.3 gibt das Quadrat von zwei Zahlen a und b aus, obwohl wir nur einmal definiert haben wie man ein Quadrat berechnet! Umso komplexere Rechnungen man macht, umso besser ist es Funktionen zu verwenden um den Code übersichtlich zu halten.

5.3 Funktionen mit Rückgabewert

Wie besprochen ist eine Funktion in der Mathematik eine Zuordnung von Zahlen zueinander. Ist zum Beispiel die Funktion $f(x) = x^2$ gegeben, so ist $f(2) = 4$, $f(3) = 9$, usw. Wir geben der Funktion $f(x)$ die Werte 2,3, ... und erhalten Rückgabewerte, 4,9,... Das lässt sich so auch in C++ implementieren. Hierfür ersetzen wir das `void` durch den Datentyp, welchen die Funktion zurückgeben soll.

Beispiel 5.3.1. Quadrat berechnen mit Funktion mit Rückgabewert

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int square(int a){
7    return a*a;
8  }
9
10 int main(){
```

```
11    int a, b;
12
13    cout << "Bitte Zahl 1 eingeben: " << endl;
14    cin >> a;
15    cout << "Bitte Zahl 2 eingeben: " << endl;
16    cin >> b;
17
18    cout << "Das Quadrat von " << a << " ist " << square(a) << endl;
19    cout << "Das Quadrat von " << b << " ist " << square(b) << endl;
20
21    return 0;
22  }
```

```
Bitte Zahl 1 eingeben:
8
Bitte Zahl 2 eingeben:
5
Das Quadrat von 8 ist 64
Das Quadrat von 5 ist 25
```

In Zeile 6 in Beispiel 5.3.1 ist nun ein `int` anstatt ein `void` vor der Funktion `square` zu sehen. Dies bedeutet einfach, dass die Funktion jetzt einen Wert vom Typ `int` zurückgibt. Dieser steht immer hinter dem Befehl `return`. Ist der `return` Befehl erreicht, so wird die Funktion beendet. Der Vorteil hiervon ist, dass `square(a)` jetzt einfach wie eine Zahl behandelt werden kann (wie $f(x)$ in unserem Beispiel), wie das folgende Beispiel zeigt:

Beispiel 5.3.2. Quadrat berechnen mit Funktion mit Rückgabewert 2

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int square(int a){
7    return a*a;
8  }
9
10 int main(){
11
12   int a, b;
13
14   cout << "Bitte Zahl 1 eingeben: " << endl;
15   cin >> a;
16   cout << "Bitte Zahl 2 eingeben: " << endl;
17   cin >> b;
18
19   cout << "Das Quadrat von " << a << " ist " << square(a) << endl;
20   cout << "Das Quadrat von " << b << " ist " << square(b) << endl;
21
22   cout << "Die Summe der Qudrate ist: " << square(a)+square(b) <<
        endl;
23
24   return 0;
25 }
```

Mit Funktionen mit Rückgabewert kann man aber nicht nur Rechenoperationen, sondern auch Ein- und Ausgabe Befehle ausführen, wie folgendes Beispiel zeigt

Beispiel 5.3.3. Ein- und Ausgabe mit Funktionen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  string eingabe(){
7    string name;
8    cout << "Bitte geben Sie Ihren Namen ein" << endl;
9    getline(cin, name);
10   return name;
11 }
12
13 void ausgabe(string name){
14   cout << "Ihr Name ist: " << name << endl;
15 }
16
17 int main(){
18   string name;
19
20   name = eingabe();
21   ausgabe(name);
22
23   return 0;
24 }
```

```
Bitte geben Sie ihren Namen ein
David Wichmann
Ihr Name ist: David Wichmann
```

In Beispiel 5.3.3 sehen wir, dass der Code in der `main`-Funktion deutlich lesbarer ist als wenn man alles in `main` direkt reinschreibt. Man sieht auf einen Blick was in `main` gemacht wird!
Als letztes Beispiel wollen wir noch eine Funktion schreiben, welche testet ob eine Zahl eine Primzahl ist[1].

Beispiel 5.3.4. Primzahltest mit Funktion

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int prim(int a){ //Definition der Primzahlfunktion
7
8    int primzahl=1;
9
10   for (int i=2; i<a; i++){
11      if (a%i==0){
12         primzahl=0;
```

[1] Wie bereits erklärt sind Primzahlen Zahlen, welche nur durch 1 und sich selbst teilbar sind

```
13        }
14    }
15    return primzahl; //Wir geben eine 1 zurück wenn es sich um eine
          Primzahl handelt. Andernfalls eine 0.
16  }
17
18
19  int main(){
20
21    int zahl;
22
23    cout << "Bitte geben Sie eine Zahl ein" << endl;
24    cin >> zahl;
25
26    if (prim(zahl)){ //Wenn prim(zahl) nicht Null ist dann ist zahl
          eine Primzahl
27        cout << zahl<< " ist eine Primzahl" << endl;
28    }
29    else{
30        cout << zahl << " ist keine Primzahl" << endl;
31    }
32
33    return 0;
34  }
```

```
Bitte geben Sie eine Zahl ein
18757
18757 ist eine Primzahl
```

Das folgende Beispiel greift etwas voraus in Abschnitt 7.1. Hier wird ein struct vom Typ Buch erstellt, welches die Komponenten *Titel*, *Autor* und *Jahr* beinhaltet. Über Funktionen werden für ein Buch die jeweiligen Komponenten eingelesen und wieder ausgegeben.

Beispiel 5.3.5. Structs mit Funktionen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  struct buch{ //Definition der struct
7    string autor;
8    string titel;
9    int jahr;
10 };
11
12 buch eingabe(){
13
14   buch buch1;
15
16   cout << "Eingabe eines neuen Buchs: " << endl;
17   cout << "Autor: " << endl;
18   getline(cin, buch1.autor);
19   cout << "Titel: " << endl;
```

```
20    getline(cin, buch1.titel);
21    cout << "Jahr: " << endl;
22    cin >> buch1.jahr;
23
24    return buch1;
25  }
26
27  void ausgabe (buch buch1)
28  {
29     cout << "Folgende Daten wurden eingegeben:" << endl;
30     cout << buch1.autor << endl;
31     cout << buch1.titel << endl;
32     cout << buch1.jahr << endl;
33  }
34
35  int main(){
36
37    buch buch1 = eingabe();
38
39    ausgabe(buch1);
40
41    return 0;
42  }
```

Die Lesbarkeit des Codes in Beispiel 5.3.5 ist sehr gut. Wir haben effektiv die gesamte Berechnung in externe Funktionen ausgelagert, sodass `main` diese lediglich noch aufrufen muss.

5.4 Rekursive Funktionen

Funktionen haben eine sehr interessante Eigenschaft: sie können sich selbst aufrufen. Wenn eine Funktion sich selbst aufruft, so geschieht dies in *einer anderen Instanz*, d.h. wie als ob es dieselbe Funktion nochmal gäbe. Dies ist besonders interessant wenn man rekursive Definitionen von Funktionen hat. Was heißt das? Dies ist am Besten an einem berühmten Beispiel veranschaulicht - der Fakultät. Diese ist folgendermaßen definiert:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

Oder in **rekursiver** Form:

$$n! = n \cdot (n-1)!$$

Wissen wir also die Fakultät von $(n-1)$, so bekommen wir die Fakultät von n einfach indem wir mit n multiplizieren. Dies kann nun in C++ folgendermaßen implementiert werden:

Beispiel 5.4.1. Fakultät berechnen mit rekursiver Funktion

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int fakultaet(int a){
7    if(a==1)
8      return 1;
9    else
10     return a*fakultaet(a-1);
```

```
11  }
12
13
14  int main(){
15
16   int zahl;
17
18   cout << "Bitte Zahl eingeben:" << endl;
19   cin >> zahl;
20
21   cout << "Die Fakultät von " << zahl << " ist: " << fakultaet(zahl)
        << endl;
22
23    return 0;
24  }
```

```
Bitte Zahl eingeben:
6
Die Fakultät von 6 ist 720
```

Man bemerke dass wenn der Befehl `return` erreicht ist, eine Funktion beendet wird.
Eine weitere interessante Anwendung rekursiver Funktionen ist die *Fibonacci Folge*. Sie ist die Folge der Zahlen $1, 1, 2, 3, 5, 8, 13, 21, \ldots$. Ein Element ergibt sich also immer als Summ der letzten beiden Elemente. Die ersten beiden Element sind per Definition 1. Die folgende Funktion berechnet das n-te Fibonacci Glied.

Beispiel 5.4.2. Fibonacci Zahlen mit rekursiver Funktion

```
1   #include <stdlib.h>
2   #include <iostream>
3
4   using namespace std;
5
6   int fibonacci(int n){
7     if ((n==1)||(n==2))
8       return 1;
9     else
10      return fibonacci(n-1)+fibonacci(n-2);
11  }
12
13
14  int main(){
15
16   int zahl;
17
18   cout << "Bitte Zahl eingeben:" << endl;
19   cin >> zahl;
20
21   cout << "Das " << zahl << "-te Fibonacci Glied ist: " << fibonacci(
        zahl) << endl;
22
23    return 0;
24  }
```

```
Bitte Zahl eingeben:
8
Das 8-te Fibonacci Glied ist 21
```

5.5 Aufgaben

Aufgabe 5.5.1. Unsere erste Funktion: Quadrat berechnen
Tippen Sie folgenden Code in Netbeans und führen Sie es aus. Was macht ihr Programm?

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  void square(double a){
7    cout << "Das Quadrat von " << a << " ist " << a*a << endl;
8  }
9
10 int main(){
11   double a;
12   cout << "Bitte Zahl eingeben: " << endl;
13   cin >> a;
14
15   square(a);
16
17   return 0;
18 }
```

Aufgabe 5.5.2. Funktionen ohne Rückgabewert (`void`)

- Schreiben Sie eine Funktion `void maximum(double a, double b)`, welche das Maximum zweier Zahlen ausgibt (Sie benötigen hierfür eine `if`-Anweisung).
- Schreiben Sie eine Funktion `void summe(int a, int b)`, welche die Summe der beiden Zahlen a und b ausgibt.
- Zusatzaufgabe: Schreiben Sie dann eine Funktion `void sum(int a, int b)`, welche die Summe aller ganzen Zahlen i mit $a \leq i \leq b$ ausgibt (`for`-Schleife!).

Aufgabe 5.5.3. Quadrat berechnen mit Rückgabewert
Tippen Sie folgenden Code in Netbeans und führen Sie es aus. Was macht ihr Programm? Wo liegt der Unterschied zu Aufgabe 5.5.1?

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  double square(double a){
7    return a*a;
8  }
```

```
 9
10  int main(){
11    double a;
12
13    cout << "Bitte Zahl eingeben: " << endl;
14    cin >> a;
15
16    cout << "Das Quadrat von " << a << " ist " << square(a) << endl;
17
18    return 0;
19  }
```

Aufgabe 5.5.4. Funktionen mit Rückgabewert
Schreiben Sie die Funktion `maximum(double a, double b)` aus Aufgabe 5.5.2 nun als Funktion mit Rückgabewert. Verwenden Sie nun diese Funktion und die Funktion `square(double a)` aus der letzten Aufgabe, um das Maximum des Quadrats von zwei Zahlen a und b zu berechnen (also $\max(a^2, b^2)$). Erläutern Sie mit eigenen Worten wo der Vorteil einer Funktion mit Rückgabewert liegt.

Aufgabe 5.5.5. Funktion mit Rückgabewert

- Schreiben Sie eine Funktion `prod` mit Rückgabewert, welche das Produkt von 2 Variablen x und y zurückgibt.
- Schreiben Sie eine zweite Funktion `zufall` mit Rückgabewert und Eingabewerten a und b, welche eine Zufallszahl z, mit $a \leq z \leq b$ generiert.
  Tipp: Eine Zufallszahl in einem Intervall [`untereGrenze, obereGrenze`] wird wie folgt generiert:
  `int a = rand()%(obereGrenze-untereGrenze+1)+untereGrenze`
- Benutzen Sie die beiden Funktionen `zufall` und `prod`, um das Produkt von 5 Zufallszahlen zwischen 1 und 10 zu berechnen. Geben Sie zuerst die 5 Zufallszahlen aus, um zu sehen, ob Ihr Programm funktioniert
- Verändern Sie nun Ihr Programm so, dass die Anzahl der Zufallszahlen vom Benutzer eingegeben werden kann

Aufgabe 5.5.6. In dem folgenden Programmmcode sollen zwei Zahlen a und b eingelesen und das Produkt aus deren Quadraten wiedergegeben werden (also $a^2 \cdot b^2$). Es sind einige Syntax- sowie Logik Fehler eingebaut. Finden Sie die Fehler.

```
 1  #include <cstdlib>
 2  #include<iostream>
 3  #include<time.h>
 4  using namespace std;
 5
 6  void square(z){
 7      return z*z
 8  }
 9  int quotient(float a,float b){
10      return  a/b;
11  }
12
13  int main() {
14
```

```
15      cout<<"Geben Sie zwei Zahlen ein: "<<endl;
16      cin<<zahl1>>zahl2;
17
18      cout<<quotient(zahl1,zahl2);
19      return 0;
20  }
```

Aufgabe 5.5.7. Funktionen
In dieser Aufgabe wollen wir uns mit einfachen Funktionen beschäftigen.

- Schreiben Sie eine Funktion `max(int a, int b)` welche das Maximum zweier Zahlen zurück gibt.
- Schreiben Sie dann eine Funktion `sum(int a, int b)`, welche die Summe aller Zahlen i mit $a \leq i \leq b$ ausgibt.

Schreiben Sie beide Funktionen als Funktionen mit und ohne Rückgabewert. Worin besteht der Unterschied?

Aufgabe 5.5.8. Fakultäts-Funktion
Schreiben Sie eine Funktion (mit oder ohne Rückgabewert) `int fakultaet(int n)`, welche die Fakultät einer beliebigen Zahl `n` ausgibt. Die Fakultät einer Zahl ist wie folgt definiert:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1.$$

Zum Beispiel ist:

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24.$$

Sie benötigen zur Berechnung eine `for`-Schleife in der Funktion.

Aufgabe 5.5.9. Potenzfunktion

- Schreiben Sie eine Funktion `pow` mit Eingabewerten x und n, welche die n-te Potenz der Zahl x berechnet, also x^n. Benutzen Sie hierfür eine `for-Schleife`.
- Benutzen Sie Ihre Funktion `pow`, um die Reihe

  $$\sum_{k=0}^{n} x^k = 1 + x + x^2 + x^3 + \dots$$

  zu berechnen. Prüfen Sie mit Hilfe Ihres Programms die folgende Aussage für ein $x \neq 1$:

  $$\sum_{k=0}^{n} x^k = \frac{1 - x^{n+1}}{1 - x}$$

Aufgabe 5.5.10. Fibonacci-Folge
Schreiben Sie eine Funktion mit Rückgabewert, welche das n-te Fibonacci-Element zurückgibt, wobei `n` das Argument der Funktion sein soll.
Jede Fibonacci Zahl ist gleich der Summe der beiden vorhergehenden Fibonacci-Zahlen. Die beiden ersten Elemente sind $f_0 = 1$ und $f_1 = 1$. Für alle weiteren Elemente gilt $f_n = f_{n-1} + f_{n-2}$. Die ersten Folgenglieder sind also:

$$\begin{aligned} f_0 &= 1 \\ f_1 &= 1 \\ f_2 &= f_0 + f_1 = 1 + 1 = 2 \\ f_3 &= f_1 + f_2 = 1 + 2 = 3 \\ f_4 &= f_2 + f_3 = 2 + 3 = 5 \end{aligned}$$

usw.
Eine besondere Eigenschaft der Fibonacci-Folge ist, dass der Quotient von zwei aufeinander folgenden Folgengliedern (f_n/f_{n-1}) gegen den sogenannten goldenen Schnitt $\phi = 1.6180339887$ strebt.
Prüfen Sie diese Aussage an Ihrer Funktion.

Aufgabe 5.5.11. Rekursive Funktionen
Schreiben Sie Ihr Programm aus Aufgabe 5.5.10) zur Berechnung des n-ten Fibonacci Elements in eine **rekursive** Funktion um.
Eine rekursive Funktion ist eine Funktion, die sich selber wieder aufruft. Damit dies nicht zu einer Endlos-Schleife führt, brauchen Sie eine zusätzliche Abbruchbedingung.
Tipp: Für die Anfangselemente $f_0 = 1$ und $f_1 = 1$ kennen Sie die Fibonacci-Zahlen bereits. Nutzen Sie diese als Abbruchbedingung.

Aufgabe 5.5.12. *Für Alle*
Schreiben Sie vier Funktionen (ohne Rückgabewert, also *void*), welchen zwei Zahlen übergeben werden und

1. Den Mittelwert,
2. das Produkt,
3. das Minimum,
4. das Maximum

der beiden Zahlen berechnen und ausgeben. Greifen Sie dabei auf das Beispiel 5.2.2 zurück. Zwei Zahlen werden wie folgt übergeben:

```
void Mittelwert(int a, int b){
  ...
}
```

Aufgabe 5.5.13. *Für Fortgeschrittene*
Schreiben Sie eine rekursive Definition der Fibonacci-Reihe. Ziel ist es, das $n-te$ Element der Fibonacci-Reihe zu ermitteln. Es soll also eine Zahl n eingelesen, und dann das Element f_n ausgegeben werden. Zum Beispiel wäre $f_2 = 1$ und $f_6 = 8$.

6 Zeiger

6.1 Einführung

Jeder Speicherplatz eines Computers hat eine eindeutige Adresse, mit welcher auf den Inhalt von gespeicherten Objekte (z.B. `int`-Zahlen) zugegriffen werden kann. In fast jedem Programm haben wir bisher Variablen definiert, z.b. durch den Befehl `int zahl`. Später konnten wir dann direkt über den Namen der Variablen, nämlich `zahl` auf den Inhalt dieser Variablen zugreifen, z.B. durch den Befehl `zahl = 27`. In C++ kann man, anstatt direkt mit dem Namen einer Variablen zu arbeiten, auch die Speicheradresse selbst verwenden und über verschiedenen Operationen auf deren Inhalt zugreifen. Variablen welche die Adresse eines Speicherplatzes beinhalten nennt man **Zeiger** (oder englisch **pointer**.
Wichtig sind die folgenden drei Operationen für Zeiger:

- **Deklaration:** Die Deklaration eines Zeigers Namens p, welcher auf eine `int`-Zahl zeigt erfolgt durch den Befehl `int *p`. Die Variable `p` beinhaltet dann *die Adresse* einer anderen Variablen.
- **Dereferenzierung:** Wollen wir auf den Wert zugreifen, auf welchen der Zeiger `p` zeigt, so erfolgt dies über den *-Operator, also über `*p`. Dies sieht zwar gleich aus wie bei der Deklaration, hat mit dieser aber nichts zu tun!
- **Referenzierung**: Umgekehrt können wir für eine beliebige Variable deren Speicheradresse über den `&`-Operator erhalten: `&a`.

Bei der Deklaration eines Zeigers (und auch einer Variablen) wird so viel Speicher zugewiesen, wie der jeweilige Datentyp erfordert. Beispiel 6.1.1 zeigt die Verwendung von Zeigern anhand eines einfachen Beispiels.

Beispiel 6.1.1. Zeiger

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      int a =5;
9      int b=10;
10     int *z1; //Deklaration des Zeigers
11     int *z2;
12
13     z1 = &a; //z beinhaltet nun die Adresse von a
14     *z1 = b; //Der Wert am Speicherplatz auf den z zeigt (also a)
           wird gleich b gesetzt.
15
16     cout << "a: "<< a << endl;
17
18     z2=z1; //Jetzt zeigen beide Zeiger auf den gleichen
           Speicherplatz
19
20     cout << "z2:" << *z2 << endl;
21
22     *z2 = *z2 + b;
23
24     cout << "a:" << a << endl;
25
26     return 0;
27 }
```

```
a: 10
z2: 10
a: 20
```

Man sieht an Beispiel 6.1.1 auch gut dass die Verwendung von Zeigern gefährlich sein kann - der Wert der Variablen a wurde durch den Befehl `*z1 = b` überschrieben.
Zeiger können auch als Argumente an Funktionen übergeben werden. Beispiel 6.1.2 zeigt wie wir z.B. zwei Zahlen vertauschen können.

Beispiel 6.1.2. Zeiger und Funktionen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  void vertausche(int *p1, int *p2){ //Übergabe der Zeiger: Die *'s
       sagen lediglich dass Zeiger übergeben werden und haben nichts mit
        Dereferenzierung zu tun
7      int x;
8      x = *p1;
9      *p1 = *p2;
10     *p2 = x;
11 }
12
13 int main()
14 {
15     int a =5;
16     int b =10;
17
18     cout << "a: "<< a << "; b: " << b << endl;
19     vertausche(&a, &b); //Die Adressen der Variablen werden an die
          Funktion übergeben
20     cout << "a: "<< a << "; b: " << b << endl;
21
22     return 0;
23 }
```

```
a: 5; b: 10
a: 10; b: 5
```

6.2 Zeiger und Arrays

Wie wir in Kapitel 4.1 gesehen haben, können wir für eine Sammlung von Objekten gleichen Typs einen Array verwenden. Die einzelnen Einträge des Arrays sind dabei im Speicher "Nebeneinander" angeordnet, d.h. wenn wir die Adresse des ersten Elements sowie die Länge des Arrays kennen, so können wir auf alle Einträge im Array zugreifen. Dies ist in Beispiel 6.1.2 veranschaulicht.

Beispiel 6.2.1. Zeiger und Arrays

```
1  #include <stdlib.h>
2  #include <iostream>
```

```
3
4  using namespace std;
5
6  int main()
7  {
8      int *p;
9      int a[5];
10     a[0]=2;
11     a[1]=4;
12     a[2]=6;
13     a[3]=8;
14     a[4]=10;
15
16     cout << "Davor: " << endl;
17     for (int i=0; i<5; i++){
18         cout << "a(" << i << ")=" << a[i] << endl;
19
20     }
21
22     p = a; //Zuweisung der Adresse des ersten Array-Elements a[0] in
           die Zeiger-Variable
23     *p=0; //Der Wert von a[0] ist nun 0
24
25     p[2]=4; //Der Wert von a[2] ist nun 4
26
27     p++; //Wir springen einen Platz vor - auf a[1]
28
29     *p=10; //a[1] hat nun den Wert 10
30
31     cout << "Danach:" << endl;
32     for (int i=0; i<5; i++){
33         cout << "a(" << i << ")=" << a[i] << endl;
34
35     }
36
37     return 0;
38 }
```

```
Davor:
a(0)=2
a(1)=4
a(2)=6
a(3)=8
a(4)=10
Danach:
a(0)=0
a(1)=10
a(2)=4
a(3)=8
a(4)=10
```

Insbesondere beachte man, dass die Zuweisung einer Array-Adresse an einen Zeiger nicht wie bei einer Variable über den Referenzierungs-Operator & erfolgt, sondern direkt wie in Beispiel 6.2.1 zu sehen ist

über den Befehl `p=a`! Arrays und Zeiger sind also so ziemlich das gleiche!
Besonders wichtig sind Zeiger, wenn ganze Arrays an Funktionen übergeben werden sollen. Beispiel 6.2.2 zeigt, wie wir z.B. einen Array an eine Funktion übergeben, welche alle Elemente Null setzt und den Array ausgibt. Wichtig ist, dass auch immer die Länge des Arrays übergeben werden muss.

Beispiel 6.2.2. Zeiger, Arrays und Funktionen

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  void initialisieren(int *a, int laenge){
7    int *p = a;
8
9    for (int i=0; i<laenge; i++){
10   *p=0;
11   p++;
12  }
13 }
14
15 void ausgabe(int *a, int laenge){
16     for (int i=0; i<laenge; i++){
17     cout << a[i] << endl;
18     }
19 }
20
21 int main()
22 {
23     int a[5];
24
25     initialisieren(a,5);
26     ausgabe(a,5);
27
28     return 0;
29 }
```

Das Zusammenspiel von Arrays und Zeigern wird auch in der dynamischen Speicherallokation (siehe Kapitel **??**) deutlich. Zuletzt sei gesagt, dass Zeiger auch auf komplexere Objekte, wie z.B. Structs oder Klassen zeigen können!

7 Objektorientiertes Programmieren

7.1 Structs

Wir haben in den letzten Abschnitten einige Objekte, zum Beispiel Variablen, oder Sammlung von Objekten gleichen Typs (Arrays) kennen gelernt. In C++ ist es möglich, zusätzliche, benutzerdefinierte Objekte zu Verwenden. Zum Beispiel können wir uns ein neues Objekt namens “Person” vorstellen, welches die Eigenschaften Alter und Größe besitzt. In diesem Fall ist das neue Objekt lediglich eine Sammlung von Variablen. Um dies in C++ umzusetzen, verwenden wir eine `struct`.
Eine *struct* (von engl. Structure) ist nichts anderes als eine Zusammenfassung von Variablen. Die allgemeine Syntax geht so:

```
struct <Name>
{
<Komponenten>;
};
```

Haben wir z.B. den struct “person“ mit den Komponenten “Name“ und “Alter“, so schreibt man:

```
struct person
{
int Alter;
string Name;
};
```

Wir können dann auf die Komponenten eines structs über den “.“-Operator zugreifen:

```
int main()
{
person Student;
Student.Alter = 22;
Student.Name = "Martin";
}
```

Als Beispiel erstellen wir eine struct mit Namen “Person“, mit den Komponenten Name und Alter. Anschließend lesen wir für zwei Personen die Daten ein und geben sie über eine Funktion wieder aus. Die Übergabe von structs an Funktionen erfolgt analog zur Übergabe von üblichen Variablen.

Beispiel 7.1.1. Structs 1

```
1  #include <stdlib.h>
2  #include <iostream>
3
4  using namespace std;
5
6  struct person{
7   string name;
8   int alter;
9  };
10
11 person eingabe(){
12   person p; //Wir definieren ein neues Objekt "Person" mit Namen p.
13
14   cout << "Bitte geben Sie den Namen ein " << endl;
15   cin >> p.name;
16   cout << "Bitte geben Sie das Alter ein " << endl;
17   cin >> p.alter;
18
19   return p;
```

```
20  }
21
22  void ausgabe(person p){ //Funktion zur Ausgabe der Komponenten
23    cout << "Name: " << p.name << endl;
24    cout << "Alter: " << p.alter << endl;
25  }
26
27  int main(){
28
29    person p1, p2;
30
31    p1 = eingabe();
32    p2 = eingabe();
33
34    ausgabe(p1);
35    ausgabe(p2);
36
37    return 0;
38  }
```

7.2 Klassen

Kommt bei Bedarf...

7.3 Aufgaben

Aufgabe 7.3.1. Erstellen Sie einen struct “Land“, welcher die Komponenten ”Name“ und “Einwohnerzahl“. Erstellen Sie drei Länder, z.B. “Land1, Land2, Land3“, und lesen Sie über *cin/cout* für alle drei Länder den Namen und die Einwohnerzahl ein. Ermitteln Sie nun über *if-Abfragen* welches Land die höchste Einwohnerzahl hat, und geben Sie den Namen dieses Lands aus. Sie können dabei vereinfacht annehmen, dass zwei Länder immer verschiedene Einwohnerzahlen haben.
Bonus: Berechnen Sie die durchschnittliche Einwohnerzahl der drei Länder und geben Sie diese aus.

Aufgabe 7.3.2. Ziel ist es, ein Programm zu schreiben welches 10 Länder nach deren Einwohnerzahl ordnet. Erstellen Sie zuerst einen struct “Land“ mit den Komponenten “Name“ und “Einwohnerzahl“. Erstellen Sie einen Array von 10 Ländern und belegen Sie deren Komponenten mit Werten direkt im Code (ohne cin/cout). Wir wollen nun diesen Array so umordnen, dass der Array die Länder nach aufsteigender Einwohnerzahl beinhaltet. Gehen Sie dabei in folgenden Schritten vor:

1. Legen Sie einen Land-Array mit 10 Einträgen an, z.B. *Land Liste[10].*
2. Initialisieren Sie die Komponenten der 10 Länder, z.B. *Liste[0].Name = “USA“; Liste[0].Einwohnerzahl = 318900000*
3. Jetzt beginnt das Sortieren: Suchen Sie für $0 \leq i < 10$ nach Eintrag im Array, welcher die kleinste Einwohnerzahl hat und vertauschen Sie den Eintrag dort mit dem ersten ($i = 0$) Eintrag (Tipp: Zum Vertauschen brauchen Sie eine zusätzliche Variable).
4. Suchen Sie jetzt für $1 \leq i < 10$ nach dem Minimum der Einwohnerzahl und vertauschen Sie den Eintrag des Arrays mit dem zweiten ($i = 1$) Eintrag im Array.
5. Suchen Sie jetzt für $2 \leq i < 10$ nach dem Minimum der Einwohnerzahl und vertauschen Sie den Eintrag des Arrays mit dem zweiten ($i = 2$) Eintrag im Array.
6. Machen Sie so weiter bis Sie $i = 9$ erreicht haben.

Dieser Sortier-Algorithmus wird auch als **Selectionsort** bezeichnet.

Aufgabe 7.3.3. Schreiben Sie eine struct namens “Stadt“ mit folgenden Komponenten:

- Name
- Einwohner
- Mittlere Jahrestemperatur (C)
- Gesamtfläche (in km^2).

Schreiben Sie drei Funktionen welche zwei Städte einlesen und:

1. Die mittlere Einwohnerzahl, mittlere Temperatur und mittlere Gesamtfläche ermitteln und ausgeben. Sie können hierbei die Berechnung des Mittelwertes in eine weitere Funktion auslagern.
2. Die Stadt mit der größten Gesamtfläche ermittelt und ausgibt.
3. Die Temperatur in C einer Stadt in Fahrenheit (F) berechnet und ausgibt. Mit folgender Formel kann man die Temperaturen umrechnen:

$$T[^\circ F] = T[^\circ C] \times 1.8 + 32$$

Sie können dabei die zwei Städte im Code mit Werten belegen (also ohne cin/cout).

Tipp: Zwei Structs vom Typ “Stadt“ werden wie folgt übergeben:

```
void Fahrenheit(Stadt Stadt1, Stadt Stadt2){
  ...
}
```

Aufgabe 7.3.4. *Finde die Fehler!*
Das folgende Programm beinhaltet eine Funktion und ein struct "Land". Die struct soll als Komponenten den Namen und die Einwohnerzahl beinhalten. Die Funktion soll nun zwei Länder einlesen und das Land mit der höchsten Einwohnerzahl ausgeben. Der Code unten beinhaltet sowohl Syntax als auch Semantik-Fehler (also logische Fehler). Wie immer gibt's für alle gefundenen Fehler einen kleinen Preis.

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6   //Definition des struct
7  struct Land{
8    string name;
9    int einwohner;
10 }
11
12  //Definition der Funktion
13 void Max(){
14   if (land1.einwohner>land2.einwohner)
15     cout << land1.name << " hat mehr Einwohner als " << land2.name
          << endl;
16
17   if (land2.einwohner>land1.einwohner)
18       cout << land2.name << " hat mehr Einwohner als " << land1.name
             << endl;
19
20   if (land1.einwohner==land2.einwohner)
21       cout << land1.name << " hat gleich viele Einwohner wie " <<
            land2.name << endl;
22 };
23
24 int main() {
25
26   Land land1, land2;
27
28   land1.Name = "USA";
29   land2.Name = "China";
30   land1.einwohner = "319000000";
31   land2.einwohner = 1360000000;
32
33   Max(land1, land2);
34
35   return 0;
36 }
```

Aufgabe 7.3.5. Schreiben Sie analog zur Vorlesung eine **struct** vom Namen "Land" mit den folgenden Komponenten:

- Name (**string**)
- Einwohner (**int**)

Erstellen Sie nun einen Array vom Typ "Land" mit 4 Ländern und lesen Sie die Komponenten der einzelnen Länder über eine for-Schleife (for-Schleife in *main*) ein. Schreiben Sie für die Eingabe ebenfalls eine Funktion "Eingabe", welche über **cin/cout** die Komponenten eines Landes einliest und das Land

zurückgibt.
Schreiben Sie anschließend eine Funktion zur Ausgabe des Namens eines Landes und geben Sie die Länder im Array über eine for-Schleife (for-Schleife ist in *main*) aus. Im Gegensatz zur Vorlesung soll die Ausgabe aber nur dann erfolgen, wenn die Einwohnerzahl größer als ein bestimmter Wert (z.B. 20 000 000) ist. Andernfalls soll keine Ausgabe erfolgen. Dieser Grenzwert soll ebenfalls als Parameter an die Ausgabe-Funktion übergeben werden und über **cin/cout** eingegeben werden. Verwenden Sie für die bedingte Ausgabe eine if-Anweisung in der Ausgabe-Funktion.

Aufgabe 7.3.6. Schreiben Sie eine **struct** vom Namen "Land" mit den folgenden Komponenten:

- Name (**string**)
- Einwohner (**int**)

Erstellen Sie nun einen Array vom Typ "Land" mit 10 Ländern und lesen sie die beigefügt Datei "Countries.txt" analog zum Beispiel 4.3.1 oben ein. Geben Sie die Länder anschließend durch eine for-Schleife wieder aus. Es ist keine Funktion zur Eingabe oder Ausgabe notwendig, sondern schreiben Sie gleich wie im Beispiel die Schleife zur Eingabe direkt in die *main*-Funktion.
Wichtig: Speichern Sie die Datei *Countries.txt* im selben Ordner wie ihr C++ Projekt ab.

Aufgabe 7.3.7. Einlesen der Daten und Datenanalyse

1. Modifizieren Sie die **struct** *Land*, sodass sie folgende vier Komponenten enthält:
   - Code (**char-Array[3]**)
   - Einwohner2000 (**float**)
   - Einwohner2015 (**float**)
   - Wachstum (**float**)
2. Erstellen Sie nun einen Array vom Typ "Land" mit 217 Ländern und lesen sie den Ländercode und die Einwohner-Daten in der Datei ein (die Komponente "Wachstum" bleibt an dieser Stelle noch leer). Überlegen Sie sich, wo Sie im Code Änderungen vornehmen müssen.
3. Geben Sie zunächst zum Test die eingelesenen Daten über eine Ausgabe-Funktion aus.
4. Die Variable "Wachstum" soll das prozentuale Bevölkerungswachstum zwischen den Jahren 2000 und 2015 beinhalten. Überlegen Sie sich wie Sie dies berechnen, und belegen Sie die Variable für jeden Eintrag des Arrays durch eine for-Schleife.
5. Finden Sie durch eine for-Schleife das Land mit dem höchsten Bevölkerungswachstum.
6. Ermitteln Sie durch eine for-Schleife die Gesamtbevölkerung aller Länder im Jahr 2015.

Aufgabe 7.3.8. Beschreibung des Datensatzes
Die beigefügte Datei "BIP.txt" beinhaltet den Länder-Namen in einer Zeile, gefolgt von drei Spalten in der nächsten Zeile. Dies sind die BIP-Werte der jeweiligen Länder in Mrd. USD (US Dollar). Machen Sie sich mit dem Datensatz vertraut. Bearbeiten Sie folgende Aufgaben:

1. Modifizieren Sie die **struct** *Land*, sodass sie folgende vier Komponenten enthält:
   - Name (**string**)
   - BIP2000 (**float**)
   - BIP2005 (**float**)
   - BIP2010 (**float**)
2. Erstellen Sie nun einen Array vom Typ "Land" mit dynamischer Länge (genau wie im obigen Beispiel!). Lesen Sie über **cin** die Anzahl der Länder ein. Das beigefügte txt-File beinhaltet 193 Länder.

3. Schreiben Sie eine Funktion “Daten_Einlesen“ welche als Argument den Pointer auf den Länder-Array sowie dessen Länge erhält (so wie im obigen Beispiel). Beachten Sie, dass der Name der Länder und die BIP-Daten in getrennten Zeilen stehen. Lesen Sie den Namen über getline() ein (so wie im Buch-Beispiel in der Vorlesung).

4. Schreiben Sie eine Ausgabe-Funktion, welche ebenfalls den Pointer auf den Array sowie dessen Länge als Argumente bekommt und die eingelesenen Daten ausgibt.

5. Legen Sie einen neuen Array an welcher die Länder beinhaltet, welche ein BIP von mehr als 50 Mrd. USD haben. Schreiben Sie hierfür eine Funktion welche als Argumente zwei Pointer bekommt, einen auf die normale Liste und einen für den neuen Array. Bestimmen Sie zunächst die Anzahl der Länder mit BIP >50 Mrd. USD und erstellen Sie den Array dynamisch in der Funktion. Geben Sie den Array anschließend mit der Ausgabe-Funktion aus.

6. Bestimmen Sie das gesamte BIP aller Länder im Jahr 2010