Don't Panic!

Inhaltsverzeichnis

Kapitel 1

Einführung

> *" In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move. "*
>
> *– Douglas Adams*

Dieses Dokument ist als Einführung in die algorithmische Programmierung konzipiert. Es vermittelt die nötigen Werkzeuge um Programmieraufgaben im Stile der SOI, IOI oder verwandten Wettbewerben in Angriff zu nehmen. Die Erklärungen sind bewusst kurz gehalten und sollen dich zum eigenständigen Testen und Ausprobieren anregen. Unklarheiten besprichst du am besten mit anderen Mitprogrammierer/innen, deinem Coach oder dem freundlichen Computerfreak aus deinem Bekanntenkreis.

1.1 Programmierumgebung

Um selber Programme zu erstellen müssen einige Werkzeuge auf dem Computer installiert sein. In ihrer Gesamtheit nennt man diese die "Programmierumgebung". Sie besteht üblicherweise aus einem Editor, mit dem der Programmcode ("Quelltext") bearbeitet wird, und dem sogenannten "Compiler", der den Quelltext in Computersprache übersetzt. Die folgenden Abschnitte erläutern in groben Zügen die Installation von NetBeans und der GNU-Compiler Collection:

Windows

Die Installation auf einem Windows Rechner benötigt (sofern nicht vorhanden):

- Java Runtime Environment (JRE, z.B. Offline Version)
  `http://www.java.com/de/download/manual.jsp`
- Die NetBeans Programmierumgebung (C++ Version, Aktuell 6.9.1)
  `http://www.netbeans.com/downloads/index.html`
- Cygwin Umgebung mit dem Gnu Compiler
  `http://cygwin.com` $\Longrightarrow$ "Install or update now"

Lade und installiere die Java Umgebung "JRE" und das NetBeans Paket für C++ von den angegebenen Webseiten und benutze die vorgegebenen Standardeinstellungen beim Installieren. Von der Cygwin Webseite lädst du den Installer

`setup.exe` herunter und führst ihn aus. Klicke zunächst fünf mal `[Next]` (Installationspfad `C:\cygwin\bin`) und wähle anschliessend einen lokalen Server, z.b. `switch.ch`. Aus der folgenden Paketliste brauchst du vier Pakete:

```
Devel>
    gcc-core
    gcc-g++
    gdb
    make
```

Führe die Installation anschliessend aus und beende den Installer.

Wichtig: Damit der Cygwin Compiler von der Programmierumgebung gefunden wird, musst du dem System mitteilen, es solle in `C:\cygwin\bin` danach suchen. Dazu musst du die Systemvariable "`PATH`" verändern:

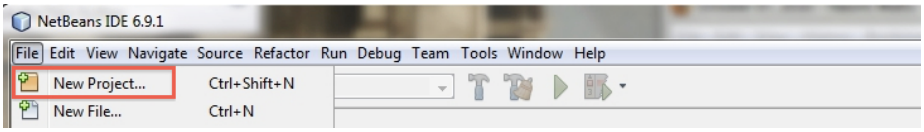
Öffne den Windows Explorer
→ Rechts-Klick auf "Computer"
→ "Eigenschaften"
→ "Erweiterte Eigenschaften"
→ "Umgebungsvariablen"
→ "Systemvariablen"

Verändere nun die Systemvariable "`Path`" oder "`PATH`", indem du den Eintrag für Cygwin zusätzlich anhängst (den alten Wert *nicht* ersetzen!):

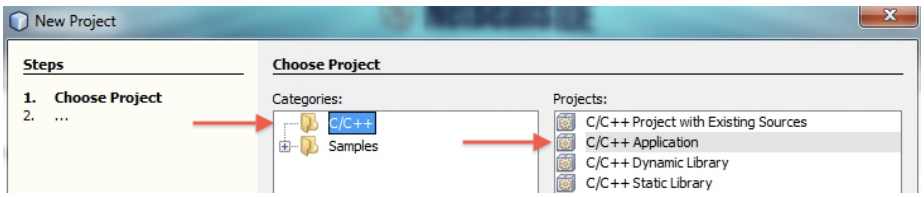
`{...Bisheriger Inhalt von Path...};C:\cygwin\bin\;`

1.2 NetBeans IDE

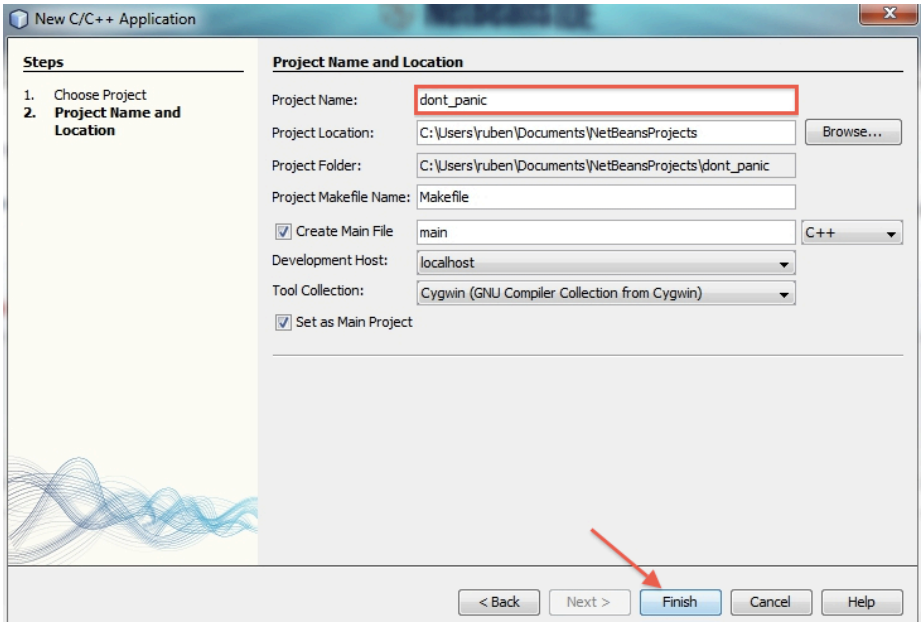
Die NetBeans Entwicklungsumgebung verwaltet Programme in Projekten. In jedem Projekt gibt es Quelltext Dateien (`.cpp`), die in ausführbare Dateien übersetzt werden, sowie gegebenenfalls weitere Dateien die zu diesem Programm gehören. Für ein neues Projekt müssen wir zuerst ein neues Projekt erstellen. Die Option dafür ist im "File"-Menü zu finden: "New Project...":



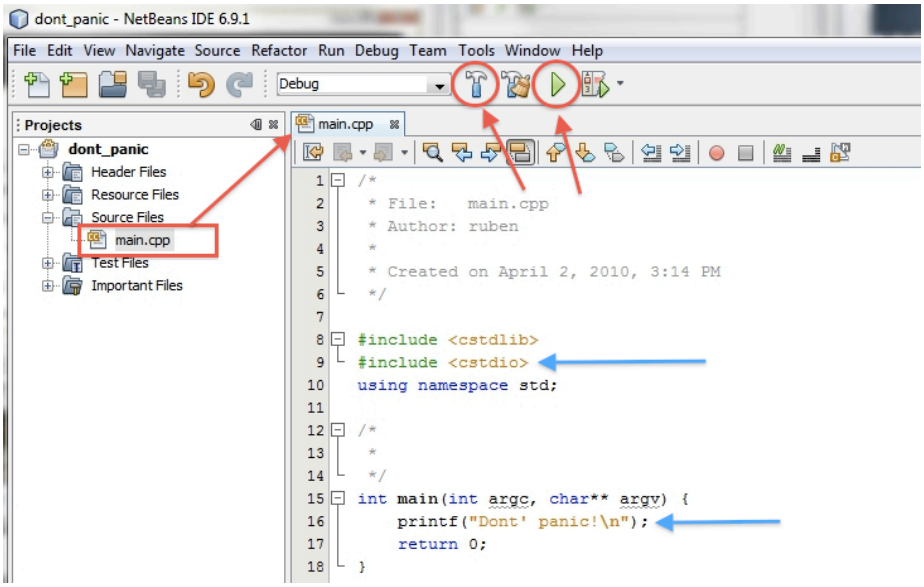
Dadurch wird ein Fenster geöffnet, indem wir die Art des neuen Projektes bestimmen können. Für den Rahmen dieser Einführung befassen wir uns mit der Wahl "C/C++ Application". Dabei handelt es sich um alleinstehende, ausführbare Programme:



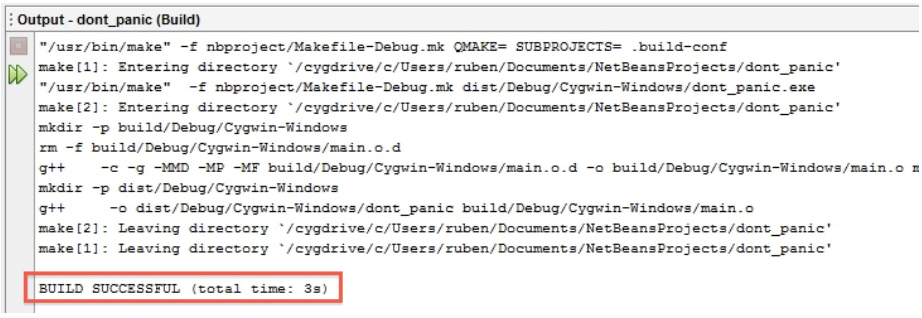
Im darauf folgenden Fenster mit Einstellungen wählen wir einen Projektnamen, die übrigen Einstellungen sollten bereits automatisch richtig gewählt sein:



Nun erscheint unser Projektordner in der linken Spalte. Im Unterordner "Source Files" finden wir eine Vorlage `main.cpp` für unser Programm. Diese können wir durch Doppelklicken im Arbeitsfenster öffnen:



Hier wurden bereits einige Sachen hinzugefügt, damit das Programm auch etwas macht (siehe Abschnitt "Dein erstes Programm"). Sobald der Code soweit fertig ist, muss dieser zuerst kompiliert werden mit dem "Build" Knopf (Hammer-Symbol). Dabei wird der Code in Maschinensprache übersetzt. Das klappt aber nur falls der Code auch Fehlerfrei ist – die entsprechende Meldung erscheint in der Konsole unter dem Arbeitsfenster:



Beachte: Es wird immer das aktuelle "Main Project" kompiliert (egal welche Datei offen ist!): Wähle das "Main Project" durch Rechtsklicken auf den Projektordner → "Set as Main Project". Falls erfolgreich kompiliert wurde, können wir das Programm mit dem "Run" Knopf (grünes Dreieck) testen. Es erscheint eine Konsole auf dem Bildschirm mit der Textausgabe des Programms:

Falls du das Programm nicht modifiziert hast, fehlt in diesem Fenster die Linie `Don't panic`. – im folgenden Abschnitt siehst du, wie man diese hinzufügt.

1.3 Dein erstes Programm

Traditionellerweise soll das erste Programm eine Nachricht in die Welt hinaus schicken. Es ist dies meist ein wenig einfallsreiches "Hallo Welt". Für unsere Zwecke wollen wir eine aufmunternde Botschaft wählen:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Don't␣Panic!" << endl;
7     return 0;
8 }
```

Die obenstehenden Zeilen haben diese Wirkung:

1-2 Teilt dem Compiler mit, dass wir die Bibliothek `iostream` und den Namensraum `std` verwenden wollen.

4 Bezeichnet den Anfang der `main`-Funktion (Hauptteil des Programms).

5-8 Alle Anweisungen der `main`-Funktion werden zwischen zwei geschwungenen Klammern zusammengefasst.

6 Mit `cout` schreiben wir die Textnachricht auf den Bildschirm (gefolgt von einem Zeilenumschlag, `endl`).

7 Der Rückgabewert Null ist das Signal für "kein Fehler".

Der obenstehende Quelltext kann in zwei Komponenten geteilt werden:

- Der Kopfteil gibt dem Compiler die nötige Information über die Bibliotheken die wir verwenden wollen (Bibliotheken enthalten verschiedene vordefinierte Funktionen die verwendet werden können. Die wichtigsten Bibliotheken werden direkt mit dem Compiler mitgeliefert, weitere können manuell installiert werden).
- In der `main()`-Funktion (die immer als erste durchlaufen wird) stehen die Anweisungen für den Rechner. Beachte, dass jede Anweisung mit einem Strichpunkt abgeschlossen '`;`' werden muss. Das Programm endet, wenn das Ende der `main()`-Funktion erreicht wird (oder ein `return` in der `main()`-Funktion.

Programm testen

Um das obenstehende Programm zu testen, kopiere einfach den Quelltext in ein neues Projekt, in die Programm-Datei mit der Endung "`.cpp`". Wähle anschliessend aus dem Menü "Compile and Run" – Übersetzen und Ausführen.

1.4 Übungen

1. **Grosse Variation für Tastatur und Bildschirm in drei Sätzen**

   Erweitere das obenstehende Programm so, dass drei Sätze (deiner Wahl) auf dem Bildschirm erscheinen – auf drei nacheinanderfolgenden Zeilen. Dazu kannst du entweder mehrere `cout`-Anweisungen einfügen oder den Text in der bestehenden Ausgabe erweitern.

2. **BUILD FAILED (exit value 2, total time: 42ms)**

   In C++ muss jede Instruktion mit einem Strichpunkt beendet werden. Wird dies nicht gemacht, so betrachtet der Kompiler die zwei nicht getrennten Instruktionen als eine und wird sich (meistens) daran verschlucken. Entferne aus deinem Programm einen Strichpunkt und versuche das Programm dann zu kompilieren. *Präge dir den Fehler gut ein, du wirst ihn in Zukunft sehr häufig sehen.*

3. **Debugger: A Programmer's Best Friend**

   Nicht alle Fehler fallen bereits beim Kompilieren auf. Falls dein Programm problemlos kompiliert werden kann, aber leider das falsche Resultat produziert, musst du auf Fehlersuche gehen. Diese Tätigkeit nennt sich neudeutsch "Debuggen", also das loswerden von Fehlern, "Bugs". Dazu kann ein Programm im "Debug"-Modus gestartet werden, und der Programmdurchlauf an bestimmten Stellen Unterbrochen werden.

   Benutze das Menü "Debug" um an einer bestimmten Stelle in deinem Programm einen "Breakpoint" einzusetzen. Starte anschliessend das Programm zum debuggen: "Debug Main Project". Das Programm läuft jetzt vorerst nur bis zum Breakpoint. Mit den Tabs "Variables", "Call Stack" und "Output" kannst du dir nun den aktuellen Zustand des Programms anschauen (um Fehler zu finden), bevor du es weiterlaufen lässt.

4. **Dein erstes Projekt**

   Nun sollst du ein zweites Projekt in der NetBeans IDE erstellen (mit einem anderen Namen). Dieses taucht als separater Ordner in der Spalte links auf und hat die gleiche Struktur wie das bisherige.

   Beachte: In NetBeans ist immer ein Projekt das aktive "Main Project". Ein klick auf "Build" oder "Run" bezieht sich immer auf das "Main Project", egal welche Datei gerade geöffnet ist im Arbeitsfenster. Falls also dein Programm nicht das macht, was es soll, und zudem verdächtig aussieht wie ein früheres Projekt, hast du evtl. vergessen das neue "Main Project" festzulegen.

   Nachdem du das neue Projekt erstellt hast, lege es als neues "Main Project" fest, füge eine `cout`-Ausgabe hinzu und Teste dein Werk.

Kapitel 2

Grundkenntnisse

> *“ The knack of flying is learning how to throw yourself at the ground and miss. ”*
>
> *– Douglas Adams*

Dieses Kapitel dient als Einführung in die Grundelemente von C++.

2.1 Variablen

Variablen sind Speicherorte für Daten mit denen das Programm arbeiten kann. In C++ muss jede Variable definiert werden, bevor man sie verwenden kann. Dazu spezifiziert man Typ und Name der gewünschten Variable:

```
1 int x; // definiere eine neue Variable ‘x’
2 x=42;  // weise der Variabel einen Wert zu
```

Definition einer Variable

Der Typ einer Variable (hier `int`) legt fest, welche Form von Information darin gespeichert werden kann. In dieser Ganzzahl-Variable (Integer) können nur ganze Zahlen gespeichert werden – Kommastellen werden bei der Zuweisung einfach ignoriert (es wird also immer abgerundet):

```
1 int x; // Die Variable ’x’ vom Typ ’int’ wird definiert
2 x=5/3; // In der Variable wird der Wert 5/3 gespeichert
3 cout << x << endl; //Der gespeicherte Wert wird ausgegeben
```

1 In der ersten Zeile wird eine neue Variable erstellt: Es wird soviel Speicherplatz reserviert, wie eine Variable vom Typ `int` braucht, und diesem Speicherort wird nun der Name `x` gegeben.

2 Wir versuchen 5/3 in `x` zu speichern, die 2/3 werden abgerundet.

3 Wir schreiben den in `x` gespeicherten Wert auf den Bildschirm. Die Ausgabe ist `1`, weil die Zuweisung in Zeile 2 abgerundet wurde.

Typ einer Variable

Wenn man Nachkommastellen speichern will, muss man einen anderen Variabeltyp wählen, z.B. `double` (Gleitkomma Zahlen mit doppelter Präzision). Für Buchstaben und andere Symbole gibt es den speziellen Typ `char` und für Zeichenfolgen den Datentyp `string`.

Ganzzahlen: `int`

Diese Variablen können eine Zahl aus dem Bereich $[-2'147'483'648, 2'147'483'647]$ darstellen – das ist gerade die Menge der Zahlen die man mit vier Byte (32 Bits) darstellen kann: $[-2^{31}, 2^{31}-1]$. Diese Beschränkung kann zu Problemen führen, wenn man mit sehr grossen Zahlen arbeitet.

Gleitkommazahlen: `double`

Wir haben das Problem des Rundens beim dividieren von ganzen Zahlen bereits kennengelernt. Mehr Präzision bietet die sogenannte Gleitkomma-Darstellung. Dabei wird eine Zahl zusammengesetzt aus einem Faktor zwischen Null und Eins (Mantisse m) und einer Zehnerpotenz: $m \cdot 10^e$. Nachkommastellen werden in m gespeichert, während der Exponent e die Grössenordnung der Zahl bestimmt. (Aber: Auch dieser Datentyp kann nicht beliebig genaue Zahlen speichern, denn die Genauigkeit von m und der Bereich von e sind beschränkt. Für die meisten Berechnungen sind `double`s präzise genug.)

Wichtig: Gleitkommazahlen sollten immer *mit Dezimalpunkt* angegeben werden, ansonsten werden sie vom Programm als ganze Zahlen betrachtet und ensprechend gehandhabt:

```
1 double x = 5/3;
2 double y = 5.0/3.0;
3 cout << "Der␣Wert␣von␣x␣ist␣" << x << endl;
4 cout << "und␣y␣hat␣den␣Wert␣" << y << endl;
```

```
Der Wert von x ist 1.000000
und y hat den Wert 1.666667
```

1 In der ersten Zeile wird zuerst die rechte Seite ausgerechnet. Dort stehen nur ganze Zahlen, also wird die Ganzzahldivision benutzt und das Resultat wird abgerundet. Erst dann wird es in der Variable `x` gespeichert (obwohl da auch Kommastellen Platz hätten).

2 Wenn wir hingegen einen Dezimalpunkt angeben, ist dem Programm klar, dass es nicht auf ganze Zahlen runden muss.

Bool'sche Werte: `bool`

Bool'sche Werte sind Wahrheitswerte – sie sind entweder wahr (`true`) oder falsch (`false`). Alle Zahlen werden als `true` interpretiert, wenn sie ungleich Null sind, als `false`, wenn sie gleich Null sind.

Einzelne Buchstaben: `char`

Buchstaben die in einer Variable vom Typ `char` gespeichert werden sind eigentlich Zahlen im Bereich $[-128, 127]$. Sie werden vom Computer gemäss der Ascii Zeichentabelle[1] als Buchstaben interpretiert. Man kann das Symbol in seinen Zahlwert umwandeln, indem man den Wert in einer Variable vom Typ `int` speichert:

```
1 char symbol='*';
2 int zahlwert=symbol;
3 cout << "Der␣Zahlwert␣von␣" << symbol;
4 cout << "␣ist␣" << zahlwert << endl;
```

1 Wir erstellen eine Variable vom Typ `char` und speichern darin das Stern-Symbol: '*'. *Ein einzelnes Symbol oder ein Buchstabe wird immer mit einfachen Apostroph angegeben.*

2 Wir erstellen eine Ganzzahl-Variable und geben ihr den gleichen Wert. Aus der Sicht des Computers enthalten nun beide Variablen die gleiche Zahl – nämlich 42. Beim ausgeben wird das eine aber als Zahl auf dem Bildschirm erscheinen, während das andere in ein Symbol übersetzt wird.

3-4 Wir geben das Symbol und seinen Zahlwert mit `cout` aus.

Zeichenfolge: `string`

Um den Datentyp `string` für das Abspeichern einer Zeichenfolge (i.e. Wort, Satz, Text) zu verwenden, brauchen wir die gleichnamige Bibliothek (und den Namensraum `std`):

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main(){
6     string msg="Don't␣panic!";
7     cout << msg << endl;
8 }
```

1-3 Zu verwendende Bibliotheken und Namensräume.

6 Wir erstellen eine Variable vom Typ `string` (Zeichenfolge) und speichern darin eine Nachricht. *Für Zeichenfolgen verwenden wir immer Anführungszeichen (Gänsefüsschen).*

7 Die gespeicherte Nachricht geben wir nun auf den Bildschirm aus.

[1] http://www.google.ch/search?q=ascii+table

Initialisieren

In C++ wird eine Variable beim definieren nicht automatisch auf einen sinnvollen Anfangswert gesetzt. Nach Möglichkeit sollte man sie mit einem guten Anfangswert "initialisieren", um nicht fälschlicherweise den unbekannten und somit zufälligen Wert in eine wichtige Berechnung einfliessen zu lassen. *Beispiel*: Für die Initialisierung kann man der Variable direkt in der Definitions-Anweisung einen Wert zuweisen:

```
1 double pi=3.14159265; // pi definieren und initialisieren
```

2.2 Ein-/Ausgabe

Wir haben bereits gesehen, wie man mit `cout` Ausgaben auf den Bildschirm bringen kann. Mit dem Shift-Operator '`<<`' können wir sowohl Text als auch Variablen an `cout` übergeben:

```
1 int zahl=42;
2 char buchstabe='a';
3 cout << "Zahl=" << zahl << endl;
4 cout << "Buchstabe:␣" << buchstabe << endl;
```

`cout` setzt anschliessend die verschiedenen Stücke zusammen und leitet sie an die Bildschirmausgabe weiter. Mit Steuerelementen wie `endl` können Zeilenumbrüche in die Ausgabe eingebaut werden.

Formatierung

Damit die Ausgabe besser aussieht können wir Steuerelemente aus der Bibliothek `iomanip` verwenden:

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main(){
6      double pi=3.14159265;
7      cout << setw(10) << 42 << endl;
8      cout << setw(10) << setprecision(3) << pi << endl;
9      cout << setw(10) << scientific << 0.000001 << endl;
10 }
```

Eingabe in einer Variable speichern

Das pendant zu `cout <<` für die Eingabe lautet `cin >>`.

```
1 cout << "Gib␣eine␣Zahl␣ein␣:";
2 int x;
3 cin >> x;
4 cout << "Du␣hast␣" << x << "␣eingegeben." << endl;
```

Mit `cin` lesen wir einen Wert von der Eingabe und speichern diesen in der Variable `x`. Wenn man das Programm laufen lässt, wartet der Computer an dieser Stelle um dem Benutzer die Möglichkeit zu geben, etwas einzutippen. Erst wenn `[Enter]` gedrückt wird, läuft das Programm weiter.

`cin` liest immer Zeichen ein, solange bis ein Leerschlag (oder Zeilenumbruch) kommt. Anschliessend versucht es die gelesenen Zeichen sinnvoll in den Datentyp der Variable umzuwandeln und speichert den ermittelten Wert anschliessend dort. Sollte dabei etwas schief gehen, wird dies in `cin.fail()` vermerkt und muss zuerst mit `cin.clear()` behoben werden, bevor man weiter einlesen kann.

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

int main(){
    int x;
    cout << "Bitte␣eine␣Zahl␣eingeben:␣";
    cin >> x;
    if (cin.fail()){
        cin.clear();
        string keineZahl;
        cin >> keineZahl;
        cout << keineZahl << "ist␣doch␣keine␣Zahl!" << endl;
    }else{
        cout << "Sie␣haben␣" << x << "␣eingegeben." << endl;
    }
    return 0;
}
```

Ganze Zeile einlesen

Beim Arbeiten mit `string`'s möchte man Oft eine ganze Zeile einlesen, und nicht nur eizelne Wörter. Dazu kann man die Funktion `getline()` verwenden:

```
string vollerName;
cout << "Geben␣Sie␣Vor-␣und␣Nachnamen␣ein:␣";
getline(cin,vollerName);
cout << "Ihr␣voller␣Name␣ist␣" << vollerName << endl;
```

2.3 Funktionen

Computer sind in erster Linie gut im Umgang mit Zahlen – alles weitere muss ihnen mit grosser Sorgfalt beigebracht werden. Mit den eben eingeführten Variablen kann man einfache Berechnungen direkt ausführen. Auch mathematische Funktionen können in C++ einfach abgebildet werden. Nehmen wir zum Beispiel die Quadratfunktion:

$$f(x) = x^2 \,.$$

Mathematisch ordnet diese Funktion $f()$ jeder Zahl x einen bestimmten anderen Wert x^2 zu, nämlich ihr Quadrat. Aus programmiertechnischer Sicht nimmt

die Funktion `f()` den Übergabewert `x`, berechnet dessen Quadrat[2] und gibt das Resultat als Rückgabewert zurück. Eine eigene Funktion kann neben der bestehenden `main()`-Funktion erstellt werden:

```
1  int f(int x){
2      return x*x;
3  }
4  int main(){
5      int a;
6      cout << "Gib␣eine␣Zahl␣ein:␣";
7      cin >> a;
8      int a2 = f(a);
9      cout << "Das␣Quadrat␣von␣" << x << "␣ist␣" << a2 << endl;
10     return 0;
11 }
```

```
Gib eine Zahl ein: 3
Das Quadrat von 3 ist 9
```

1 Wir definieren unsere eigene Funktion `int f(int x)`. Aus dieser ersten Zeile (Signatur der Funktion) ist abzulesen, dass die Funktion `f()` heisst, dass sie einen ganzzahligen Parameter `int x` hat und dass sie ein ganzzahliges `int`-Resultat zurück gibt.

2-3 Im Funktionskörper wird das Resultat ausgerechnet und mit dem Befehl `return` zurückgegeben. Die Klammer beendet unsere Funktion.

4 Nun folgt wie gewohnt die `main()`-Funktion. Der Signatur sieht man an, dass sie keine Parameter hat und einen ganzzahligen Wert als Resultat zurückgibt (ein Fehlercode, 0 heisst kein Fehler).

5-7 Wir erstellen eine Variable `a` und forderen der Benutzer zur Eingabe auf. Die eingegebene Zahl wird in `a` gespeichert.

8 Wir berechnen den Funktionswert und speichern das Resultat in der neu erstellten Variable `a2`.

9-10 Am Schluss geben wir die Zahl und ihr Quadrat aus und signalisieren mit dem Rückgabewert 0 “kein Fehler”.

Funktionen in Programmen können aber nicht nur dazu verwendet werden einen Zahlenwert auszurechnen, sondern um beliebige Instruktionen auszuführen. Wir können damit also auch eine Funktion schreiben, die Textausgabe auf den Bildschirm bringt oder die Daten im Speicher modifiziert.

2.3.1 Rückgabewert

Als Resultat kann eine Funktion immer nur einen Wert zurückgeben. Falls eine Funktion nur eine Instruktion ausführt und explizit *keinen* Rückgabewert gegeben werden soll, kann man diesen als `void` (leer) bezeichnen und die Funktion mit “`return;`” verlassen:

[2] Da potenzieren nicht zu den Basisberechnungen zählt, berechnet man das Quadrat am einfachsten mit `y=x*x`. Für höhrere Potenzen bietet die `cmath`-Bibliothek die Funktion `pow(,)`.

```
1 void sag_hallo(){
2     cout << "Hallo!" << endl;
3     return;
4 }
```

Falls die Funktion hingegen mehrere Resultate produziert, muss man entweder einen kombinierten Datentypen zurückgeben, oder mit veränderbaren Parametern arbeiten (siehe nächster Abschnitt).

2.3.2 Parameterliste

Eine Funktion kann Null, einen oder mehrere Übergabewerte, sogennante Parameter, akzeptieren. Dabei handelt es sich im Normalfall um die zu verarbeitenden Daten und/oder Instruktionen wie diese verarbeitet werden sollen. Dabei muss sowohl der Typ jedes einzelnen Parameters, als auch deren Reihenfolge genau eingehalten werden:

```
1 void zahl_ausgeben(double zahl, int prec)
2     cout << setprecision(prec) << zahl;
3     return;
4 }
```

Diese Funktion kann benutzt werden als `zahl_ausgeben(3.14159,2)`, aber nicht als `zahl_ausgeben(2,3.14159)`. Bei der zweiten Version interpretiert das Programm `3.14159` als ganze Zahl `3` (abgerundet) und gibt anschliessned die Zahl 2 mit Präzision 3 aus.

Für die letzten Parameter einer Funktion besteht die Möglichkeit, Standardwerte zu definieren, für den Fall dass keine Wert angegeben wird. Wenn ein Zahl im Normalfall mit Kommastellen geschrieben werden soll, können wir die Signatur oben wie folgt abändern:

```
1 void zahl_ausgeben(double zahl, int prec=6)
```

Dann sind die Aufrufe `zahl_ausgeben(3.14159,6)` und `zahl_ausgaben(3.14159)` identisch – in der zweiten Version wird für den Parameter `prec` der Standardwert `prec=6` gewählt.

2.3.3 Gültigkeitsbereich

Jede Variabel hat ihren Gültigkeitsbereich dort, wo sie definiert wurde. Auf Variablen aus `main()` kann man in einer Funktion nicht zugreifen und alle Parameter und eigene Variablen der Funktion existieren nach dem Beenden der Funktion nicht mehr. Nur durch die übergebenen Parameter und den Rückgabewert wird Information ausgetauscht. *Beachte*: Parameter werden standardmässig als Kopie übergeben - Veränderungen werden also in `main` nicht Sichtbar. Es gibt zwei Möglichkeiten dies zu umgehen:

- Variablen als Referenz übergeben (durch Angabe von &)

```
1 void vertausche(int &x, int &y){
2     int tmp=x; x=y; y=tmp;
3 }
```

Wären die Parameter `x` und `y` hier als Kopie übergeben worden, würde man die Vertauschung in der Funktion nicht feststellen (nur die Kopien würden vertauscht).

- Mit Speicheradressen (Pointern) arbeiten. Dazu werden wir in einem späteren Kapitel mehr erfahren. Vorerst nur die Warnung, dass alle Arrays auch zu dieser Klasse gehören – sie werden nicht kopiert beim Funktionsaufruf.

2.4 Entscheidungen

Damit das Programm Entscheidungen fällen kann, kann man gewisse Instruktionen nur dann ausführen wenn eine bestimmte Bedingung erfüllt ist. Dazu schliesst man sie in die `if(...)`-Klausel ein und spezifiziert in den Runden Klammern die notwenige Bedingung:

```
1  int main(){
2      int x=0;
3      scanf("␣%d",&x);
4      if(x==42){
5          cout << x << "␣ist␣die␣Antwort." << endl;
6      }else{
7          cout << x << "␣ist␣nicht␣die␣Antwort." << endl;
8      }
9      return 0;
10 }
```

2-3 Wir bitten den Benutzer um die Eingabe einer Zahl

4-8 Die `if`-Klausel führt den ersten Block aus (Zeile 5), falls `x` genau 42 ist, andernfalls wird der zweite Zahlenblock (Zeile 7) ausgeführt.

Das Resultat der Bedingung muss ein sogenannter "Boolscher Wert" sein, eine Aussage die entweder wahr (`true`) oder falsch (`false`) ist. Zusammen mit den Verfügbaren Vergleichsoperatoren und logischen Verknüpfungen können so beliebig komplizierte Entscheidungen getroffen werden.

Vergleichsoperatoren

- `(x==y)` ist wahr, falls x und y den gleich numerischen Wert haben.
- `(x<=y)` is true if x is smaller or equal to y.
- `(x<y)` is true if x is smaller than y.
- `(x!=y)` is true if the values of x and y are different.

Beachte: Der Vergleichsoperator "`==`" wird mit zwei Gleichheitszeichen geschrieben und muss vom Zuweisungsoperator "`=`" unterschieden werden. Ein häufiger Fehler besteht darin, nur ein Gleichheitszeichen in einem Vergleich zu verwenden (`x=y`), wodurch zuerst `x` der Wert von `y` zugewiesen wird und anschliessend dieser Wert als Wahrheitswert interpretiert wird. Nicht nur das Resultat ist also unter Umständen falsch, wir haben auch den Wert von `x` überschrieben.

Logische Verknüpfungen

Wir können auch mehrere solche Vergleiche miteinander Verknüpfen, mit sogenannten logischen Operatoren. In C++ werden diese wie folgt notiert: Logisches UND (`&&`), Logisches ODER (`||`)[3] und Logisches NICHT (`!`):

- `(x<3 || 9<x)` ist `true`, falls x kleiner ist als 3 oder grösser als 9 (also ausserhalb des Bereiches $[3, 9]$ liegt).
- `(3<x && x<9)` ist `true`, falls x grösser ist als 3 und gleichzeitig auch kleiner als 9 (also im Bereich $(3, 9)$ liegt).
- `( !(x<y) )` ist `true`, falls die innere Bedingung `false` ist, also falls $x < y$ nicht zutrifft (äquivalent zu $x \geq y$).

```
1  int main(){
2      cout << "Zahl␣zwischen␣1-10␣eingeben:␣";
3      int x=0;
4      cin >> x;
5      if (1<x && x<10){
6          cout << "Bravo,␣" << x << "␣liegt␣zwischen␣1-10";
7      }else if(x==1 || x==10){
8          cout << "Knapp,␣" << x << "␣liegt␣auf␣der␣Grenze";
9      }else{
10         cout << x << "␣liegt␣nicht␣in␣Bereich␣1...10";
11     }
12     cout << endl;
13     return 0;
14 }
```

2.5 Programmschleifen

Wir haben den Rechner bereits dazu gebracht, unsere Instruktionen auszuführen, aber wir mussten bislang jede Einzelne Aufgabe ausschreiben. Nun lernen wir, wie der Computer sich wiederholende Aufgaben mit immer dem gleichen Programmteil abarbeiten kann. Dazu setzt man “Schleifen” ein:

2.5.1 `while`-Schleife

In einer `while`-Schleife wird ein Codeblock solange ausgeführt, wie eine bestimmte Bedingung erfüllt ist. Damit können wir zum Beispiel den Benutzer solange um die eine Eingabe bitten, wie die Eingabe eine bestimmte Bedingung nicht erfüllt:

```
1  int main(){
2      int x=0;
3      while( x<1 || x>10 ){
4          cout << "Zahl␣eingeben␣zwischen␣1-10:␣";
5          cin >> x;
6          if (cin.fail()){
```

[3] Die Striche müssen *durchgehend* sein, nicht unterbrochen in der Mitte!

```
7               cin.clear(); string ignore; cin >> ignore;
8               cout << "Fehler␣beim␣Eingeben" << end;
9               continue;
10          }
11          if( x<1 ) cout << "Zahl␣ist␣zu␣klein." << endl;
12          else if( x>10 ) cout << "Zahl␣ist␣zu␣gross." << endl;
13      }
14      cout << "Na␣endlich␣" << x << "␣liegt␣im␣Bereich␣(1,10)!";
15  }
```

Das obenstehende Programm wird solange um die Eingabe einer Zahl bitten, bis diese die gewünschten Kriterien erfüllt:

3-11 Die `while`-Schleife wird durch die Schleifenbedingung initiiert und durch die zwei geschwungenen Klammern begrenzt.

5 Wir prüfen den Rückgabewert von `scanf()` ist dieser Null (also `false`), so wird der optionale Teil (5-8) ausgeführt. Dieser teilt den Fehler beim Einlesen mit und startet mit `continue;` die `while`-Schleife neu.

9-10 Falls die Zahl ausserhalb liegt, wird eine entsprechende Meldung angezeigt und die `while`-Schleife startet neu.

Endlosschleifen

Beim Benutzen von Schleifen ist Vorsicht geboten – es kann leicht passieren, dass eine Schleife nicht mehr verlassen werden kann, also endlos läuft. Das Programm muss beendet werden (mit `Ctrl-C` oder dem Task-Manager).

Kontrollwörter in der Schleife

Grundsätzlich wird die Schleife immer nach einem ganzen Durchlauf wiederholt, sofern die Schleifenbedingung noch erfüllt ist – andernfalls verlassen. Ein neuer Schleifendurchlauf kann an einer beliebigen Stelle mit `continue;` gestartet werden. Ausserdem kann mit `break;` die Schleife manuell verlassen werden.

`do-while`-Schleife

Eine Variante der `while` Schleife garantiert das einmalige Ausführen des Codeblockes in der Schleife. Dazu wird vor der Schleife nur die Anweisung "`do`" geschrieben, das `while` samt Bedingung folgt am Ende der Schleife.

```
1  int x=42;
2  do{
3      cout << "Wie␣lautet␣die␣Antwort?"
4      cin >> x;
5  }while( x!=42 );
```

Die Bedingung wird auch erst am Schluss der Schleife, also nach dem ersten Durchlauf geprüft (eine `while`-Schleife, deren Bedingung anfänglich nicht erfüllt ist, wird gar nicht ausgeführt).

2.5.2 `for`-Schleife

Mit `while`-Schleifen lässt sich auch Zählen:

```
1 int i=1;
2 while(i<10){
3     cout << i << endl;
4     i++;
5 }
```

Zählschleifen werden sehr oft verwendet, deswegen gibt es dafür eine Kurzform:

```
1 for(int i=0; i<10; i++){
2     cout << i;
3 }
```

In den runden Klammern der `for`-Schleife stehen drei Instruktionen:

`int i=0` Wird nur einmal ausgeführt, bevor die Schleife gestartet wird. Der Gültigkeitsbereich der Variable liegt nur *innerhalb* der Schleife.

`i<10` Die zweite Instruktion ist die Schleifenbedingung. Sie wird, wie bei `while`-Schleifen, auch vor dem ersten Durchlauf geprüft.

`i++` Die dritte Instruktion wird am Ende jedes Schleifendurchlaufs ausgeführt, noch bevor die Bedingung für den nächsten Durchlauf geprüft wird. (In diesem Fall wird der Wert von `i` um eins erhöht.)

2.6 Übungen

1. **Alles nur Zahlen**

   Auch Buchstaben werden im Computer nur als Zahlen dargestellt, lediglich für die Ausgabe auf den Bildschirm werden sie, mittels Zeichentabelle, in lesbare Symbole umgewandelt.

   Schreibe ein Programm, das zuerst eine ganze Zahl einliest, den Wert dann aber einer Variable vom Typ `char` zuweist, und diesen auf den Bildschirm ausgibt.

2. **Taschenrechner**

   Schreibe ein Programm, das zwei Zahlen einliest und ihre Summe, Differenz, Produkt und Quotient berechnet und ausgibt.

   ```
   Gib zwei Zahlen ein: 4 2
   Summe: 6
   Differenz: 2
   Produkt: 8
   Quotient: 2
   ```

3. **Weniger ist manchmal mehr**

   Schreibe eine Funktion `zahl_einlesen()` die das Einlesen einer Ganzzahl übernimmt und direkt deren Wert zurückgibt. Falls keine Zahl eingeben wurde (z.B. Text) soll sie eine entsprechende Fehlermeldung ausgeben und 0 retournieren.

Damit musst du nicht mehr jedes mal den ganzen Code schreiben, um eine Zahl einzulesen, sondern kannst einfach deine eigene Funktion aufrufen.

4. **Mit Buchstaben rechnen**

   Die Buchstaben des Alphabetes liegen praktischerweise nebeneinander in der ASCII Tabelle. Benutze eine for Schleife um alle Buchstaben des Alphabetes nacheinander auszugeben (jeweils Gross- und Kleinbuchstabe nebeneinander).

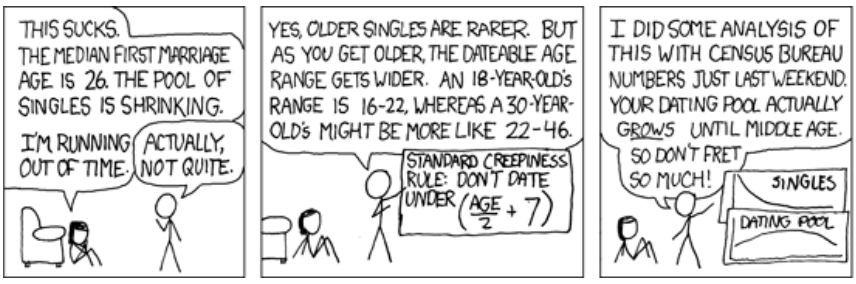
```
Aa
Bb
Cc
Dd
...
```

5. **Teilbar oder nicht?**

   Ganzzahlen werden immer abgerundet (auch die Zwischenresultate von Ganzzahlen!). Nutze dies um zu testen ob zwei Zahlen ohne Rest teilbar sind. (*Ja das geht später auch einfacher, mit dem Modulooperator*)

```
Gib zwei Zahlen ein: 7 4
7 ist nicht ohne Rest durch 4 teilbar.
```

6. **Dating Pool**[4]



   Angenommen das Mindestalter für potentielle Dates sei $x = (Alter/2+7)$, schreibe ein Programm, welches für ein gegebenes Alter die Unter- und Obergrenze des Alters des Partners ausrechnet und auf eine Kommastelle gerundet angibt:

```
Wie alt bist du? 28
Dein/e Partner/in sollte zwischen 21.0 und 42.0 sein.
```

7. **Raten auf Raten**

   Schreibe ein Programm, das eine ganze Zahl aus dem Bereich $[0 - 100]$ erraten kann. Dazu soll das Programm immer zwei Zahlen ausgeben und als Antwort die Zahl bekommen, die näher an der gesuchten Zahl ist – solange bis es die Zahl erraten kann. (Wenn beide gleich nah sind kann eine beliebige als Antwort gegeben werden.)

[4] `http://xkcd.com/314/`, nicht der ganze Comic wegen Platzmangel, Vollversion online.

```
Merke dir eine Zahl. Bereit? Los:
Ist 30 oder 50 naeher? 50
Ist 40 oder 50 naeher? 40
Ist 42 oder 43 naeher? 42
Ist 41 oder 42 naeher? 42
Ich glaub du hast dir 42 gemerkt!
```

Das Programm hatte viel Glück und hat nur 4 Fragen gebraucht. Schreibe dein Programm so, dass es die Antwort sicher findet – aber so wenig Fragen wie möglich stellt.

Kapitel 3

String

> *“ It can hardly be a coincidence that no language on earth has ever produced the expression “As pretty as an airport.” ”*
>
> *– Douglas Adams*

In diesem Kapitel befassen wir uns mit dem Einlesen, Bearbeiten und Ausgeben von Zeichenfolgen (Wörter, Sätze, Abschnitte).

3.1 string

Im vorherigen Kapitel haben wir kurz die `string`-Klasse aus der gleichnamigen Bibliothek kennengelernt. Sie erlaubt das einfache Handhaben von Zeichenfolgen in C++. Ausserdem bringt die Bibliothek auch einige Werkzeuge mit, mit denen wir die Zeichenfolgen untersuchen und bearbeiten können.

Eigenschaften eines Strings

Eine Variable vom Typ `string` kann nicht nur gelesen oder gesetzt werden, sie erlaubt auch das Abfragen relevanter Eigenschaften des momentan gespeicherten Wertes - z.B. die Länge der enthaltenen Zeichenfolge.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main(){
6      cout << "Hallo, wie heisst du?" << endl;
7      string name;
8      cin >> name;
9      cout << "Name hat " << name.length() << " Buchstaben";
10     cout << ", der erste ist " << name[0] << " an." << endl;
11     cout << "Ersten drei Buchstaben:" << name.substr(0,3);
12     cout << ", letzte drei " << name.substr(name.length()-3);
13 }
```

9 Die Länge einer Zeichenfolge können wir mit `length()` berechnen.

10 Auf einzelne Buchstaben kann mit eckigen Klammern zugegriffen werden.

11-12 `substr(x,y)` Gibt einen Teil der Zeichenfolge zurück, startend von x, mit y Zeichen Länge (oder von x bis zum Schluss, falls nur eine Zahl gegeben wird).

Einlesen

`string`'s können entweder mit `cin` oder `getline` eingelesen werden. Im ersten Fall wird jeweils nur ein Einzelnes Wort von der Eingabe gelesen, im zweiten Fall die ganze Zeile (mit Leerschlägen):

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main(){
6      cout << "Hallo, wie heisst du?" << endl;
7      string vorname, nachname;
8      cin >> vorname >> nachname;
9      cout << "Hallo Frau/Herr " << nachname <<". ";
10     cout << "Oder darf ich " << vorname << "sagen?" << end;
11
12     cout << "Wie lautet dein ganzer Name?" << endl;
13     string ganzername;
14     getline(cin,ganzername);
15     cout << "Der ganze Name lautet " << ganzername << endl;
16
17     if ( vorname+" "+nachname != ganzername ){
18         cout<<"Wieso hat sich dein Name geaendert?" << endl;
19     }
20 }
```

8 Mit `cin` werden einzelne Wörter eingelesen, Vorname und Nachname werden also separat eingelesen, auch wenn sie zusammen auf einer Zeile eingetippt werden.

14 Mit `getline` wird die ganze Zeile eingelesen und same Leerzeichen in einer Variable gespeichert.

Suchen

Es gibt auch einige Methoden um einzelne Buchstaben oder Zeichenfolgen in einem string zu suchen:

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

int main(){
    string s = "21␣ist␣nur␣die␣halbe␣Wahrheit";
    int start = s.find_first_of("0123456789");
    cout << "Ich␣habe␣eine␣Zahl␣gefunden!"
         << "␣Sie␣startet␣bei␣" << start << endl;
    cout << "Das␣Wort␣'nur'␣steht␣nur␣an␣"
         << "Position␣" << s.find("nur") << endl;
    return 0;
}
```

3.2 Übungen

1. **Selbstgespräche**

   Schreibe ein Programm, das eine kurze Diskussion führen kann. Es soll den Benutzer mit Namen begrüssen können und eine Meinung zu einem beliebigen Gesprächsthema abgeben können.

   Erweitert: Stelle ein, dass der Programmiere persönlich begrüsst wird.

2. **Palindrom**

   Schreibe ein Programm das ein Wort einliest und daraus ein Palindrom macht durch anhängen von Zeichen. Ein Palindrom ist ein Wort, das vorwärts und Rückwärts gleich aussieht. Wie lautet dein Name als Palindrom?

   Fortgeschrittene: Schreibe ein Programm, das in einer Zeichenfolge das längste Palindrom findet.

3. **Starwars Name**

   Schreibe ein Programm, das Vor- und Nachname, sowie den Mädchennamen der Mutter und den Geburtsort einliest und daraus den "Starwars"-Namen generiert. Das geht wie folgt:

   - Der "Starwars"-Vorname bildet sich aus den ersten drei Buchstaben des Nachnamens und den ersten zwei des Vornamens.
   - Der "Starwars"-Nachname bildet sich aus den ersten zwei Buchstaben des Mädchennamens der Mutter und den ersten drei Buchstaben des Geburtsortes.

Kapitel 4

Arithmetik

> *“ A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools. ”*
>
> *– Douglas Adams*

Für den folgenden Abschnitt brauchen wir sogenannte Arrays. Dabei handelt es sich nicht um einzelne Variablen, sondern einen ganzen Block davon. Mit einem zusätzlichen Index wählen wir, welche der Variablen aus dem Block wir benutzen möchten:

```
1  int main(){
2      //Ein Array von zehn Variablen
3      int a[10];
4
5      //Start der Fibonacci Folge
6      a[0]=1; a[1]=1;
7
8      //Zahlen generieren
9      for(int i=2; i<10; i++){
10         a[i]=a[i-1]+a[i-2];
11         cout<<a[i]<<endl;
12     }
13     return 0;
14 }
```

3 Wir definieren ein Array von 10 Variablen vom Typ Int.

6 Wir können die Variablen mit den Indizes $[0, \ldots, 9]$ ansprechen (Indizes starten in C++ immer bei Null, nicht Eins!). Wir setzten den Wert an position 0 und 1 auf 1.

9-11 Mit einer for-Schleife generieren wir die ersten Zehn Zahlen der Fibonacci-Folge.

10 Wie hier zu sehen ist, können wir als Index auch den Wert einer Variable (hier die Zählvariable i) verwenden.

Achtung: C++ überprüft *nicht*, dass der Wert des Index im richtigen Bereich liegt (hier $0-9$). Ist der Wert ausserhalb, ist der Wert beliebig und das Programm stürzt wahrscheinlich ab (mit einem Segfault oder Bus-Error).

4.1 Teilbarkeit ganzer Zahlen

Der Modulo Operator `%` erlaubt das Prüfen der Teilbarkeit einer Zahl durch eine andere: Er berechnet den Rest der Ganzzahldivision der zwei Zahlen. Verschwindet der Rest, so ist `a` durch `b` teilbar:

```
int main(){
    int a,b;
    printf("Bitte zwei Zahlen eingeben: ");
    scanf(" %d %d", &a, &b);
    if( a%b==0 ) printf("%d ist durch %d teilbar\n",a,b);
    else printf("%d durch %d gibt %d Rest %d\n",a,b,a/b,a%b);
}
```

Eine Zahl ist prim, falls sie nur durch sich und 1 ohne Rest Teilbar ist. Mithilfe einer `for`-Schleife können wir also bereit einen Primzahlentest konstruieren:

```
int main(){
    int a;
    printf("Bitte eine Zahl eingeben: ");
    scanf(" %d", &a);
    bool istPrim=true;
    for(int i=2; i<a; i++)
        if( a%i==0 ) istPrim=false;
    if( istPrim ) printf("%d ist prim\n",a);
    else printf("%d ist nicht prim\n",a);
}
```

Dieser Test, so wie er oben aufgeschrieben ist, macht genau $(N-2)$ Schritte um zu bestimmen ob N prim ist – hat also eine asymptotische Laufzeit von $O(N)$[1]

Um alle Primzahlen im Bereich $[0\ldots100]$ zu finden, könnten wir einfach den Test aus dem obenstehenden Abschnitt in eine Testfunktion packen und diese mit allen Zahlen aus dem Bereich aufrufen. Allerdings kannten bereits die Griechen eine bessere Methode: das *Sieb des Eratosthenes*.

```
int main(){
    bool isPrime[101];
    for(int i=0; i<=100; i++) isPrime[i]=true;
    isPrime[0]=isPrime[1]=false;
    for(int i=2; i<=100; i++){
        if(isPrime[i]){
            printf("%d\n",i);
            for(int j=i+i; j<=100; j+=i)
                isPrime[j]=false;
        }
    }
    return 0;
}
```

[1] Das ist nicht optimal, eigentlich müssten wir nur von $2\ldots\sqrt{N}$ testen. Wieso?

2 Wir brauchen dazu ein Array von bool'schen Variablen, nennen wir es `isPrime[]`, wobei der Wahrheitswert an der Position `isPrime[i]` anzeigen soll, ob `i` eine Primzahl ist.

3 Wir gehen Anfangs davon aus, das alle Zahlen ¿=2 prim sind

4 Null und Eins sind per Definition keine Primzahlen

5-11 Nun gehen wir, ausgehend vom Index `i=2` durch das Array und

6-7 geben alle Primzahlen aus, die wir antreffen.

8-9 Für jede gefundene Primzahl streichen wir sofort alle Vielfachen aus der Liste, damit wir sie später nicht antreffen.

Grösster gemeinsamer Teiler

Der grösste gemeinsame Teiler zweier Ganzzahlen ist die grösste ganze Zahl, durch die beide ohne Rest teilbar sind. Ein Programm, welches einfach alle Zahlen durchprobiert können wir sehr einfach schreiben, aber es gibt eine elegantere Lösung mit *Rekursion*. (Eine rekursive Funktion ist eine Funktion die sich selber aufruft.)

Falls x der grösste gemeinsame Teiler von a und b ist, muss die Differenz der beiden Zahlen auch gerade ein Vielfaches von x sein. Also haben a und $(b-a)$ den gleichen ggT wie a und b. Daraus ergibt sich die rekursive Funktion:

```
1 int ggT(int a, int b){
2     if(a==b) return a;
3     if(a<b) return ggT(a,b-a);
4     else return ggT(a-b,b);
5 }
```

Falls die beiden Zahlen nicht gleich sind, ziehen wir die grössere von der kleineren ab und rufen `ggT()` mit dieser Differenz und einer der beiden Zahlen auf. Dadurch wird der ggT *nicht* verändert aber die Zahlen werden immer kleiner – solange bis sie gleich sind und den Wert des ggT erreicht haben.

Kleinstes gemeinsames Vielfaches

Das kleinste gemeinsame Vielfache können wir einfach mithilfe der ggT berechnen. Wenn wir die zwei Zahlen multiplizieren ist gerade der ggT zweimal in das Produkt multipliziert worden. Wenn wir diesen entfernen, ist die Zahl immer noch ein Vielfaches beider Zahlen.

```
1 int kgV(int a, int b){
2     return a*b/ggT(a,b);
3 }
```

4.2 Übungen

1. **The Usual Suspects**

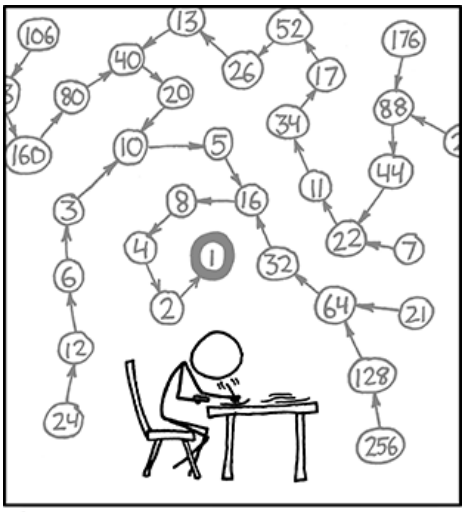
   Schreibe ein Programm das mit dem Sieb des Eratosthenes alle Primzahlen im Bereich $[a, b]$ ausgibt.

2. **Primezahlzerlegung**

   Schreibe ein Programm das eine Zahl in ihre Primfaktoren zerlegen kann. Das Programm soll alle Primfaktoren als Liste ausgeben:

   ```
   Geben sie ein Zahl ein: 420
   Primfaktoren: 2 2 3 5 7
   ```

3. **Collatz Conjecture**[2]**:** $3N + 1$

   THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

   Ausgehend von einer beliebigen Zahl gehen wir wie folgt vor:

   - Falls die Zahl gerade ist, dividiere sie durch zwei.
   - Falls die Zahl ungerade ist, multipliziere mit drei und addiere eins.

   Die Collatz-Vermutung sagt aus, dass mit diesem Vorgehen durch wiederholtes Anwenden immer die Zahl Eins erreicht wird.

   Schreibe ein Programm, welches ausgehend von einer Zahl die Zahlenfolge ausgibt bis die Zahl eins erreicht wird.

   Fortgeschrittene: Schreibe ein Programm, welches für einen gegebenen Bereich $[a, b]$ die Zahl mit der längsten Sequenz findet.

[2] Comic: `http://xkcd.com/710/`, Referenz: `http://de.wikipedia.org/wiki/Collatz-Problem`