

# C++ für Anfänger (BG 11)

## will not be continued

[http: worgtsone.scienceontheweb.net/worgtsone](http://worgtsone.scienceontheweb.net/worgtsone) - [mailto: worgtsone @ hush.com](mailto:worgtsone@hush.com)

13. Oktober 2011

## Inhaltsverzeichnis

<b>I</b>	<b>1. Quartal</b>	<b>2</b>
<b>1</b>	<b>Allgemeines</b>	<b>3</b>
1.1	Vom Problem zum Programm . . . . .	3
1.2	Hallo Welt . . . . .	3
1.3	Programmskelett . . . . .	3
<b>2</b>	<b>Speichern</b>	<b>3</b>
<b>3</b>	<b>Beurteilung von Leistungen</b>	<b>4</b>
<b>4</b>	<b>Variablen : Schnell und schmutzig</b>	<b>5</b>
4.1	Deklariere . . . . .	5
4.2	Namen . . . . .	5
4.3	Eingeben von Variablen . . . . .	5
4.4	Zuweisen . . . . .	6
4.4.1	Syntax von Zuweisungen - ++*() . . . . .	6
4.5	Ausgeben von Variablen . . . . .	6
4.6	Übung . . . . .	7
<b>5</b>	<b>Variablen : Lang und ausführlich</b>	<b>8</b>
5.1	Deklariere . . . . .	8
5.1.1	Datentypen . . . . .	8
5.2	Eingeben . . . . .	8
5.3	Zuweisen . . . . .	8
5.3.1	mit + - * ( ) . . . . .	8
5.3.2	Übung: Zuweisen im Struktogramm . . . . .	9
5.3.3	Division von Floats . . . . .	9
5.3.4	Division von Integers . . . . .	9

5.3.5	Übung Modulo % . . . . .	9
5.3.6	Der Große Bruchstrich . . . . .	10
5.4	Formatierung der Ausgabe mit <code>iomanip</code> . . . . .	10
5.5	Übung . . . . .	11
5.5.1	Wild . . . . .	11
5.5.2	Quadrat . . . . .	11
5.5.3	Kreis . . . . .	11
5.5.4	Quadratur . . . . .	12
5.5.5	Ausgabe . . . . .	12
5.5.6	Rechenaufgaben . . . . .	12
6	<code>iomanip</code> . . . . .	13
7	<b>Weitere Variablentypen</b> . . . . .	<b>14</b>
7.1	Integer-Typen . . . . .	14
7.1.1	Int-Variablen im Speicher . . . . .	14
7.1.2	<code>long int</code> . . . . .	14
7.1.3	<code>unsigned int</code> . . . . .	14
7.1.4	<code>unsigned long int</code> . . . . .	14
7.2	Float-Typen . . . . .	14
7.2.1	<code>double</code> . . . . .	14
7.3	Char-Typen . . . . .	15
7.3.1	<code>char</code> . . . . .	15
7.4	Boolean-Typen . . . . .	15
7.4.1	<code>bool</code> . . . . .	15
8	<b>Bedingte Verzweigung - unverknüpft</b> . . . . .	<b>17</b>
8.1	Einseitig, Beispiel: Untersuchen einer Zahl . . . . .	17
8.1.1	Schreibweise der Bedingung . . . . .	17
8.2	Übung: Notenauswertung . . . . .	18
8.3	Zweiseitige Verzweigung . . . . .	18
8.4	Übung: Einseitige Verzweigung . . . . .	19
8.4.1	Zahlenterror . . . . .	19
8.4.2	Gehalt . . . . .	19
8.5	Übung: Zweiseitige Verzweigung . . . . .	19
8.5.1	Reihenfolge . . . . .	19
8.6	Übung: Verknüpfte Bedingungen . . . . .	19
8.6.1	Bauspar . . . . .	19
8.6.2	Bank . . . . .	19
8.6.3	QuadrGl2 . . . . .	19
8.6.4	DreiName . . . . .	19
8.6.5	Geradengleichung . . . . .	20

<b>9 Verknüpfungs-Operatoren AND, OR, NOT</b>	<b>21</b>
9.1 Übung: Tabelle zum Ausfüllen . . . . .	21
9.2 Wiederholung: Wasserscheuer Lehrer ohne Schirm . . . . .	21
9.3 Beispiel: Wasserscheuer Lehrer . . . . .	22
9.4 Syntax . . . . .	22
9.4.1 Einseitig . . . . .	22
9.4.2 Zweiseitig . . . . .	23
9.4.3 Verschachtelt . . . . .	23
9.5 Übung: Schirm o.k.? . . . . .	23
<b>II 2. Quartal</b>	<b>24</b>
<b>10 cmath</b>	<b>25</b>
10.1 cmath . . . . .	25
10.2 Bogenmaß [optional] . . . . .	25
<b>11 Zählschleife (FOR)</b>	<b>26</b>
11.1 Summenbildung . . . . .	26
11.2 Übung . . . . .	28
11.2.1 FORDO01 . . . . .	28
11.2.2 FORDO02 . . . . .	28
11.2.3 Fakultät . . . . .	28
11.2.4 Großes Einmaleins . . . . .	28
11.2.5 ASCII-Tabelle . . . . .	28
11.2.6 SINUS-Verlauf . . . . .	28
<b>12 Modularisierung (VOID)</b>	<b>29</b>
<b>13 Kopf- und Endegesteuerte Schleife (WHILE / DO...WHILE)</b>	<b>30</b>
13.1 Menüprogramm . . . . .	30
13.2 Schleife mit Startbedingung . . . . .	30
13.3 Schleife mit Weitermach-Bedingung . . . . .	31
13.4 Übungen . . . . .	32
13.4.1 Summe der Potenzen von 1/2 . . . . .	32
13.4.2 Ken's benutzerfreundlicher Quader . . . . .	32
13.4.3 Summe(Kehrwerte(Quadrate)) . . . . .	32
13.4.4 Iteratives Wurzelziehen nach Lisa . . . . .	32
13.4.5 Diophant . . . . .	32
<b>14 Globale und Lokale Variablen - Arbeitsblatt</b>	<b>33</b>
14.1 Was der Compiler sah . . . . .	33
14.2 Was der Rechner ausspuckte . . . . .	33
14.3 Ankreuzen . . . . .	34

<i>INHALTSVERZEICHNIS</i>	4
<b>15 Bedingte mehrfache Verzweigung [optional]</b>	<b>35</b>
<b>16 Zufallszahlen erzeugen</b>	<b>36</b>
16.1 Übung . . . . .	36
16.1.1 Lotto . . . . .	36
16.1.2 Verteilung . . . . .	36
<b>17 Felder (arrays)</b>	<b>37</b>
<b>18 Sortieren</b>	<b>39</b>
18.1 Bubblesort . . . . .	39
18.1.1 Optimieren . . . . .	39
18.2 Minsort . . . . .	39
18.3 Quicksort [optional] . . . . .	40
18.3.1 Suchen des Medians [optional] . . . . .	40
<b>19 Zeitmessung im Millisekunden-Bereich [optional]</b>	<b>41</b>
19.1 Mit <code>mymillitime</code> - 8 Wochen alt und schon historisch . . . . .	41
19.2 Mit <code>time.h</code> . . . . .	41
19.3 Übung . . . . .	42
19.3.1 Lotto2 . . . . .	42
19.3.2 Verteilung2 . . . . .	42
19.3.3 Bubblesort . . . . .	42
19.3.4 Minsort . . . . .	42
<b>20 Funktionen mit Rückgabewert - call by value</b>	<b>43</b>
20.1 <code>int summe</code> . . . . .	43
20.2 Mechanismus [optional] . . . . .	43
20.3 Übung . . . . .	44
20.3.1 <code>produkt()</code> etc. . . . .	44
20.3.2 Schreiben Sie <code>lisaSqrt(float)</code> . . . . .	44
20.3.3 <code>readln</code> . . . . .	44
<b>III 3. Quartal</b>	<b>45</b>
<b>21 Multi-Referat Overkill</b>	<b>46</b>
21.1 Ablauf . . . . .	49
21.2 Fragen . . . . .	49
21.3 Hinweis für die Neugierigen . . . . .	50
<b>22 Webserver ärgern [optional]</b>	<b>51</b>
22.1 <code>txtenc</code> und <code>txtdec</code> . . . . .	51
22.2 Verbesserungsmöglichkeiten . . . . .	52
22.3 <code>txtenc.cpp</code> . . . . .	52

22.4	txtdec.cpp . . . . .	53
<b>23</b>	<b>Referat Staubtrocken</b>	<b>54</b>
23.1	Zuweisungsoperatoren . . . . .	54
23.2	Prä- und Post-In- und -dekrement . . . . .	54
23.2.1	Musterlösung zu Prä-Post-Zuweisung-Übung . . . . .	54
23.3	Vergleichsoperatoren . . . . .	54
23.4	Auswertung Boolescher Ausdrücke . . . . .	55
23.4.1	Musterlösung Auswertung . . . . .	55
23.5	Ternärer Operator ?: . . . . .	55
23.6	Übungen . . . . .	56
23.6.1	Zuweisung1 . . . . .	56
23.6.2	Prä Post Zuweisung . . . . .	56
23.6.3	Auswertung . . . . .	56
<b>24</b>	<b>Besenkammer</b>	<b>57</b>
24.1	Definition . . . . .	57
24.2	struct – Datensatz . . . . .	57
24.2.1	Übung Struct . . . . .	58
24.3	enum – Aufzählung [optional] . . . . .	60
24.4	typedef – Typdefinition [optional] . . . . .	60
24.5	Übung struct : Rechnungsverwaltung . . . . .	61
24.6	Übung enum : Attributverwaltung . . . . .	61
<b>25</b>	<b>Casts - Datentyp-Wechsel</b>	<b>62</b>
25.1	Übung : ASCII-Tabelle . . . . .	62
<b>26</b>	<b>iomani2 : dec oct hex right left showbase noshowbase (to be written)</b>	<b>63</b>
<b>27</b>	<b>strings (to be written)</b>	<b>63</b>
<b>28</b>	<b>Pointer (Zeiger)</b>	<b>64</b>
28.1	Wozu ist das gut? . . . . .	64
28.2	Was ist das? . . . . .	64
28.3	Operatoren . . . . .	64
28.4	Folie - Übungsblatt . . . . .	66
<b>29</b>	<b>Funktionen mit Adressübergabe (Call by reference)</b>	<b>67</b>
29.1	Beispiel: Eine Variable um 1 erhöhen . . . . .	67
29.2	Vorteile . . . . .	67
29.3	Nachteile . . . . .	67
29.4	Übung . . . . .	67
29.5	Übergeben von Feldern by reference [optional] . . . . .	68
29.5.1	Vorteile . . . . .	68

29.5.2 Nachteile . . . . . 68

**IV 4. Quartal 69**

**30 Objekte etc. 70**

30.1 Das Weltbild der objektorientierten Programmierung (OOP) . . . . . 70  
 30.2 Allgemeines Vorgehen . . . . . 70  
     30.2.1 Beispiel: Schüler . . . . . 70  
 30.3 Schreibweise . . . . . 70  
     30.3.1 Beispiel: Fenster . . . . . 71  
 30.4 Objekte . . . . . 71  
 30.5 Arbeitsblatt Klassen . . . . . 72

**31 Beispiel: Rationale Zahlen (Brüche) 73**

31.1 Rational.hpp . . . . . 74  
 31.2 Rational.hpp . . . . . 74  
 31.3 main.cpp . . . . . 74  
 31.4 Fragen . . . . . 75  
 31.5 ... und die Antworten (partly to be written) . . . . . 76

**32 worgtsone’s Note 77**

**33 Wir legen uns ein Lexikon an 77**

**34 datenkapselung: private und public (to be written) 77**

**35 konstruktor, destruktork (to be written) 77**

**36 vererbung (to be written) 77**

**37 verwendung (to be written) 77**

**38 Wir suchen nützliche Klassen und benutzen sie 77**

**39 Dynamisches Allozieren – (NEW und DELETE) 78**

39.1 new . . . . . 78  
 39.2 delete . . . . . 79  
 39.3 Liste mit dynamischem Speicherplatz . . . . . 79  
 39.4 Übung: Schülerverwaltung . . . . . 79  
 39.5 Übung: Indizieren . . . . . 79  
 39.6 Übung: Optimieren . . . . . 79

**40 Dynamisches Allozieren – (NEW und DELETE) 80**

40.1 new . . . . . 80  
 40.2 delete . . . . . 81

40.3	Liste mit dynamischem Speicherplatz . . . . .	81
40.4	Übung: Schülerverwaltung . . . . .	81
40.5	Übung: Indizieren . . . . .	81
40.6	Übung: Optimieren . . . . .	81
40.7	Arbeitsblatt new delete . . . . .	82
<b>41</b>	<b>Projektarbeit</b>	<b>84</b>
41.1	class Screen . . . . .	84
41.2	Lissajous . . . . .	84
41.3	Starfield . . . . .	84
41.4	Snake . . . . .	84

## Disclaimer

Wissen ist zum Teilen da. Ich teile mein Wissen mit Ihnen, lieber Kollege.

Ich bin aber nicht perfekt. Unter [worgtsone@hush.com](mailto:worgtsone@hush.com)  
nehme ich dankbar Ihre Verbesserungsvorschläge entgegen.

\*

**Legal Blurb:** Alle Informationen in diesem Dokument sind falsch, unvollständig, irreführend,  
irrelevant und / oder funktionieren einfach nicht.

Wenn Sie es trotzdem benutzen, und es geht dabei etwas kaputt, ist das Ihr Problem, nicht meins.

\*

**Bitte teilen Sie meine Web-Adresse nicht Ihren Schülern mit.**

Teil I

# 1. Quartal

# 1 Allgemeines

## 1.1 Vom Problem zum Programm

Rechner können uns bei der Arbeit helfen. Sie können umfangreiche Daten in kurzer Zeit verarbeiten und lesbar präsentieren.

Man muß ihnen dabei genau sagen, was sie machen sollen. Wir werden nach folgendem Schema vorgehen:

1. **Problem** lesen und verstehen.
2. **Analyse:** Welche Daten gehen rein, welche sollen herauskommen?  
In welche Teilprobleme kann man das Problem aufspalten?
3. **Algorithmus (Lösungsweg)** skizzieren als Struktogramm.
4. **Programm** schreiben, testen, benutzen.

## 1.2 Hallo Welt

Problem: Rechner soll Hallo Welt! ausgeben.

Analyse: keine Eingaben, keine Teilprobleme.

**Ausgabe ("Hallo Welt!")**

Algorithmus:

Programm schreiben: mit irgendeinem Texteditor, zB notepad. Anschließend umbenennen von \*.txt nach \*.cpp.

Programm compilieren und ausführen: mit Bloodshed Dev-C++.

## 1.3 Programmskelett

```
#include <iostream>           // Bibliothek für Ein- und Ausgabe
using namespace std;         // Pflicht

int main () {                // geschweifte Klammern fassen Blöcke
    // zusammen
    cout << "HaloWelt, worgtsone, 01.08.2005" << endl;
    // sagt, was das Programm macht

    system.pause();          // Pflicht
    return (0);              // Pflicht
}                             // Ende des Blockes main()
```

# 2 Speichern

Speichern Sie Ihre Programme unter H:/sprachen/cpp.

### 3 Beurteilung von Leistungen

Schülerleistungen sind

- mündliche Leistungen (Anwesenheit, Aufmerksamkeit, weiterführende Fragen und Ideen, Störgrad);
- Mappenführung;
- Projekte und Referate;
- Klausuren.
  1. Handschriftlich auf Papier. Nicht am Rechner.
  2. Hilfsmittel: Stift, Papier, Taschenrechner, Lineal. Basta.
  3. Übliche Aufgaben umfassen:
    - Vom Problem zum Struktogramm, und rückwärts.
    - Vom Struktogramm zum Programm, und rückwärts.
    - Beides, und rückwärts.

## 4 Variablen : Schnell und schmutzig

In C++-Programmen dürfen wir kleine Schmierzettel benutzen, um Gedanken festzuhalten. Die Schmierzettel heißen Variablen und haben einen Namen, einen Datentyp und ggf. einen Inhalt.

Mit Variablen kann man folgende Sachen machen:

- deklarieren;
- eingeben;
- zuweisen;
- ausgeben.

### 4.1 Deklarieren

Jede Variable, die man benutzen will, muß man zunächst deklarieren.

Variablen-Deklarationen schreibt man vor die erste `main()`-Funktion oder an deren Anfang.

Beispiel:

```
int i;           // int : Zahlen ohne Nachkommastellen
int hans=5;     // ein integer wird deklariert und initialisiert
float f;        // float : Zahlen mit Nachkommastellen
float u=9.4;    // C++ verwendet DezimalPUNKT.
```

### 4.2 Namen

Als Variablenamen sind nur amerikanische Kleinbuchstaben (ohne ä, ö, ü, ß) und Ziffern zulässig. Ziffern dürfen nicht am Anfang stehen.

Ich verlange sprechende Namen. Beispiele:

Eine Variable, in der eine Summe gebildet wird, heißt `summe`.

Eine Variable, die zählt, heißt `zaehler`.

### 4.3 Eingeben von Variablen



**Eingabe (radius)**

Im Struktogramm:

Im Programm:

```
cout << "Bitte geben Sie einen Wert für den Radius ein: ";
cin >> radius;           // wird von stdin in die variable ge"pumpt"
```

## 4.4 Zuweisen

<code>pi &lt;--- 3,1415926</code>
<code>umfang &lt;--- 2 pi r</code>
<code>flaeche &lt;--- pi r<sup>2</sup></code>

Im Struktogramm:

Im C++-Programm:

```
pi = 3.1415926;
umfang = 2 * pi * radius; // c++ rechnet den rechten Teil aus
flaeche = pi * radius * radius; // und weist ihn anschließend zu.
```

**Beim Zuweisen und Eingeben wird die Variable überschrieben.  
Der vorherige Wert ist für immer weg, der neue steckt drin.  
Man kann das so oft machen, wie man will.  
Die Variable geht davon nicht kaputt.**

### 4.4.1 Syntax von Zuweisungen - +=\*()

Wir verwenden den Bruchstrich (/) zunächst nicht, weil er Fallen eingebaut hat.

**Regel 1:** C++ verlangt zwischen zwei Operatoren STETS ein Verknüpfungszeichen. Ausnahme: Minuszeichen.

Aus  $3a^2 + 2b^3 - 4c$  wird `3*a*a + 2*b*b*b - 4*c`. Aus `-4` wird `-4`.

**Regel 2:** C++ kennt "Punktrechnung kommt vor Strichrechnung."

`1+2*3-4` ergibt richtigerweise 3.

**Regel 3:** C++ akzeptiert Klammern und rechnet die Terme von innen nach außen aus.

Aus `(1 + 2) * (3 - 2)` wird intern `3 * 1` und somit 3.

## 4.5 Ausgeben von Variablen

<code>Ausgabe (i)</code>
<code>Ausgabe ("Die Fläche eines Kreises mit Radius ", r, " = ", flaeche);</code>

Im Struktogramm:

**Alles, was man durch << nach cout pumpt, wird ausgegeben.**

Im Programm:

```
cout << flaeche; // schlecht
cout << "fläche zu " << r << " = " << flaeche << endl; // viel besser
```

## 4.6 Übung

Unsere ersten Programme folgen alle dem Schema **Eingabe - Verarbeitung - Ausgabe** oder **EVA**. Schreiben Sie auf Papier das Struktogramm und das Programm, das folgendes Problem löst:

Für ein Rechteck sind bei gegebener Länge und Breite die Fläche, der Umfang und die Diagonale gesucht.

## 5 Variablen : Lang und ausführlich

### 5.1 Deklarieren

Variablen-Deklarationen schreibt man vor die erste `main()`-Funktion oder an deren Anfang.

#### 5.1.1 Datentypen

Wir lernen heute die Datentypen `int` und `float`:

`int` enthält Zahlen ohne NKS und hat einen Wertebereich von  $-2^{31}..0..2^{31} - 1 = -2147483648..0..2147483647$ .  
`float` enthält Zahlen mit NKS und hat einen Wertebereich von  $-2^{128}..0..2^{128} = -3.403 \times 10^{38}..0..3.403 \times 10^{38}$ .

Beispiel:

```
int i, j, k;           // drei integer-Variablen in einem Rutsch
int integer1;        // noch eine
int hans=5;          // eine integer-variable wird definiert und gleich
                      // gesetzt
float r, s, t;        // drei float-variablen in einem Rutsch
float u=9.4;          // floats können Nachkommastellen. C++ verwendet
                      // DezimalPUNKT.
```

**Warnung:** Wenn die Variablen überlaufen, ist das Verhalten undefiniert.

Viele Compiler denken:  $2147483647 + 1 = -2147483648$ , d.h. sie laufen im negativen Zahlenbereich weiter.

### 5.2 Eingeben

Nichts Neues. Es bleibt dabei:

Wer vom User eine Eingabe will, muß sie im Programm deutlich anfordern.

### 5.3 Zuweisen

#### 5.3.1 mit + - \* ()

Links von den folgenden Anweisungen kommt eine Schreibtischtest-Tabelle hin:

```
i=6;           // lies: "i wird 6 zugewiesen."
j=k;           // k war undefiniert => j ist jetzt auch undefiniert.
j = i + i * i;
integer1 = 0;
j= (i + i) * i;
i = i + 1;
hans = hans * hans;
hans = hans - hans;
```

### 5.3.2 Übung: Zuweisen im Struktogramm

Die erste Anweisung im obigen Programmfragment sieht im Struktogramm so aus:



Lies: "i wird zugewiesen 6."

Zeichnen Sie Struktogramme für die anderen Anweisungen.

### 5.3.3 Division von Floats

Kein Problem: `float f = 1.0 / 9.0;` funktioniert wie erwartet.

### 5.3.4 Division von Integers

C++ kennt 2 Divisionszeichen für Integer: `/` und `%` (modulo).

Division wie in der Dritten Klasse:  $17 : 3 = 5$  Rest 2.

Die 5 erhalten wir in C++ mit `17 / 3`.

Die 2 erhalten wir in C++ mit `17 % 3`.

**Bei der Division zweier Integer ist das Ergebnis ohne Nachkommastellen.**

#### Lösung:

```
(int zaehler) / (int nenner)
```

#### stets ersetzen durch

```
1.0 * (int zaehler) / (int nenner)
```

Beispiel:

```
float a;
a = 4/5;           // funktioniert nicht
a = 1.0 * 4 / 5;  // funktioniert, weil von links nach rechts
a = 4 / 5 * 1.0;  // funktioniert nicht, weil von links nach rechts
a = 4.0/5;        // funktioniert auch
```

### 5.3.5 Übung Modulo %

Wir werden den Modulo-Operator bei der Primzahlenuntersuchung und beim Erzeugen von Zufallszahlen verwenden.

Übung: Berechnen Sie  $(10,11,12,13,15,23,60) \% (1,2,3,4,5,6)$ .



## 5.5 Übung

Hinweise:

`#include <cmath>` bringt dem Compiler u.a. bei, wie man Wurzeln zieht.

`sqrt (ZAHL)` gibt die Wurzel von ZAHL zurück.

### 5.5.1 Wild

Schreiben Sie zum folgenden Algorithmus Struktogramm, Schreibtischtest und Programm:  
Ich nehme 3 Int-Variablen  $i$ ,  $j$ ,  $k$  und weise allen 0 zu. Ich vergrößere  $i$  um 2,  $j$  um -3 und  $k$  um den Faktor 13. Ich stecke den Inhalt von  $j$  nach  $i$  und anschließend den Inhalt von  $i$  nach  $j$ . Ich quadriere  $k$ .

Vor jeder Zuweisung sage ich, was ich mache. Nach jeder Zuweisung gebe ich alles aus.  
Beispiel:

```
...
i = 5, j = 9, k = 13.
Quadriere k.
i = 5, j = 9, k = 169.
...
```

### 5.5.2 Quadrat

Schreiben Sie ein Struktogramm und ein Programm, das folgendes Problem löst:

Gegeben ist ein Quadrat mit Seitenlänge  $a$ .

Gesucht sind Fläche, Umfang und Durchmesser.

### 5.5.3 Kreis

Gegeben ist ein Struktogramm:

<code>p &lt;--- 3.1415926</code>
<code>a &lt;--- 10;</code>
<code>r &lt;--- wurzel (a / p)</code>
<code>u &lt;--- 2*p*r</code>
<code>Ausgabe "XXXX", a</code>
<code>Ausgabe "XXXX", r</code>
<code>Ausgabe "XXXX", u</code>

a) Welches Problem löst es? Wodurch müssen die XXXX ersetzt werden?

b) Schreiben Sie das passende Programm dazu.

### 5.5.4 Quadratur

Es ist nicht möglich, mittels Zirkel und Lineal zu einem gegebenen Kreis ein flächengleiches Quadrat zu zeichnen.

Schreiben Sie ein Struktogramm und ein C++-Programm, das zu einem Radius die passende Kantenlänge finden kann.

### 5.5.5 Ausgabe

Schreiben Sie ein Struktogramm und ein Programm, die die folgende Ausgabe erzeugen:

```

           alle spaltenbreiten           =           15
           sic      transit      gloria      mundi.
vierwortige      saetze      sind      langweilig.
           das      alphabet:      abcdefghijklmnopqrstuvwxyz
           2+2      =           4.000000000000000
           128*128      =           16383.000000000000000

```

### 5.5.6 Rechenaufgaben

Schreiben Sie die folgenden Rechenaufgaben in C++ **auf Papier**. Treffen Sie vorher Annahmen über den Datentyp:

$$\text{a) } \frac{4}{3} + 1,5 \quad \text{b) } 4,2 a^2 - \frac{5}{2} b \quad \text{c) } \frac{2a(5 + 3b)}{3c + d}$$

$$\text{d) } \sqrt{7} * \frac{a^2}{4x} : 0,2 b^2$$

## 6 iomanip

**Wer `iomanip` will, muß `iomanip` inkludieren.**

`iomanip` definiert Befehlswoorte, die den Ausgabestrom manipulieren.

<b>Befehl</b>	<b>Wirkung</b>
<code>setw(width)</code>	macht das folgende Element rechtsbündig, <code>width</code> Spalten breit.
<code>setprecision(digits)</code>	schaltet um auf <code>digits</code> angezeigte Ziffern.
<code>dez hex oct</code>	schaltet um auf Dezimal-, Hexadezimal-, Oktalsystem.
<code>right left internal</code>	schreibt Element nach rechts — nach links — Vorzeichen links, Rest rechts.
<code>showpos noshowpos</code>	zeigt positives Vorzeichen.
<code>showbase noshowbase</code>	zeigt Basis des verwendeten Zahlensystems. 0x für hex, 0 für oktal.
<code>scientific</code>	zeigt <code>float</code> in Exponential-Schreibweise an.
<code>showpoint noshowpoint</code>	zeigt / verbirgt Dezimalpunkt eines <code>float</code> .

## 7 Weitere Variablentypen

### 7.1 Integer-Typen

#### 7.1.1 Int-Variablen im Speicher

Der Compiler reserviert für eine `int`-Variable 4 Byte = 32 Bit im Speicher, wobei das linke Bit (B31) für das Vorzeichen verwendet wird.

(Das rechte Bit heißt Bit0. Informatiker fangen bei 0 an zu zählen.)

Die größte `int`-Zahl ist daher  $01111111111111111111111111111111_B$ , das ist  $2^{31} - 1$ .

Die kleinste `int`-Zahl ist  $11111111111111111111111111111111_B$ , das ist  $-2^{31}$ .

#### 7.1.2 `long int`

`long int` hat auch 4 Byte = 32 Bit und damit denselben Wertebereich.

#### 7.1.3 `unsigned int`

`unsigned int` hat auch 4 Byte = 32 Bit. Das linke Bit wird aber nicht für das Vorzeichen verwendet und steht daher als Binärstelle zur Verfügung.  
Wertebereich:  $0..2^{32} - 1 = 0 .. 4294967295$ .

#### 7.1.4 `unsigned long int`

`long int` hat 4 Byte und kein Vorzeichen (wie `unsigned int`).  
Wertebereich:  $0..2^{32} - 1 = 0 .. 4294967295$ .

### 7.2 Float-Typen

#### 7.2.1 `double`

`double` verbraucht 8 Bytes Speicherplatz und kann daher Zahlen von  $-10^{38}$  bis  $+10^{38}$  anzeigen.

`float` kann auf meinem System (Redhat 6.2) ca. 8 gültige Stellen, `double` kann 15.

## 7.3 Char-Typen

### 7.3.1 char

**char ist ein beliebiges Computerzeichen.  
Es wird intern als Zahl von 0 bis 255 gespeichert.**

- Computerzeichen werden hintereinander eingegeben und durch ein <Return> abgeschlossen.
- Ein Leerzeichen zählt nicht als Computerzeichen, es wird ausgeblendet.
- Wenn man zu wenig Zeichen eingibt, springt der Cursor in die nächste Zeile und wartet.
- Wenn man zu viele Zeichen eingibt, nimmt C++ nur das oder die ersten.

Das macht chars sehr unhandlich bei Strings (d.h. Zeichenketten, zB "Hans Müller", "Aka-  
zienstr. 7", "05432-234567"). Dieses Problem besprechen wir später.

## 7.4 Boolean-Typen

### 7.4.1 bool

**bool ist entweder wahr (1) oder falsch (0).  
Bei Zuweisung einer Zahl ungleich 0 an eine bool-Variable wird sie  
1.**

Zu besseren Verwirrung nehmen wir ein Listing:

```

#include <iostream>
int main () {
    int i;
    float f;
    bool b;
    char c;

    i = -1;
    cout << "i=" << i;
    if (i) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    i = 0;
    cout << "i=" << i;
    if (i) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    i = 1;
    cout << "i=" << i;
    if (i) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    f = -0.5;
    cout << "f=" << f;
    if (f) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    f = -0.0;
    cout << "f=" << f;
    if (f) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    f = +1000.0 / 11;
    cout << "f=" << f;
    if (f) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    b = (5 == 4);
    cout << "(5 == 4) = " << b << endl;

    b = (5 != 4);
    cout << "(5 != 4) = " << b << endl;

    c = '@';
    cout << "c = " << c;
    if (c) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    c = '0';
    cout << "c = " << c;
    if (c) cout << " TRUE"; else cout << " FALSE";
    cout << endl;

    c = 0;
    cout << "c = " << c;
    if (c) cout << " TRUE"; else cout << " FALSE";
    cout << endl;
}

***** Sun Sep 11 10:34:48 EDT 2005 ***** c++ start *****
i=-1 TRUE
i=0 FALSE
i=1 TRUE
f=-0.5 TRUE
f=-0 FALSE
f=90.9091 TRUE
(5 == 4) = 0
(5 != 4) = 1
c = @ TRUE
c = 0 TRUE
c = FALSE
***** Sun Sep 11 10:34:49 EDT 2005 ***** c++ end *****

```

## 8 Bedingte Verzweigung - unverknüpft

Rechner wären langweilig, wenn sie nicht Fallunterscheidungen treffen könnten, wie zB

"Sie haben eine Kantenlänge  $< 0$  eingegeben. Fehler!"

"Ich darf keine Wurzel aus einer negativen Zahl ziehen."

"Ich darf nicht durch 0 teilen. *Niemals!*"

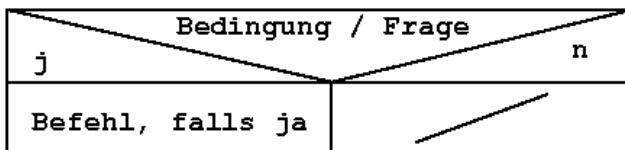
### 8.1 Einseitig, Beispiel: Untersuchen einer Zahl

Gegeben sei ein Schüler, der einen Jahresdurchschnitt von 2 im Zeugnis haben will. Natürlich schreibt er sich ein C++-Programm, in das er seine Zeugnisnoten eingibt und das ihm dann den Durchschnitt nennt.

Statt "Jahresdurchschnitt = 2.763" wünscht er sich aber etwas Motivierenderes, zB

- Dein Jahresdurchschnitt ist besser als 2! Geh sofort Fußball spielen!
- Dein Jahresdurchschnitt ist exakt 2! Weiter so!!!
- Dein Jahresdurchschnitt ist schlechter als 2. Besondere Anstrengungen sind in Mathe, Physik und Geschichte nötig.

Und das geht. Struktogramm und Programm:



```
if (bedingung) {
    // befehl, falls ja
}
```

**Falls die Bedingung erfüllt ist, geht das Programm durch den linken Kasten, aber NICHT durch den rechten.**

**Falls die Bedingung NICHT erfüllt ist, geht das Programm durch den rechten Kasten, aber NICHT durch den linken.**

**In jedem Fall geht es unter dem Kasten weiter.**

**Die Bedingte Verzweigung ist selbst ein Kasten und darf überall dort stehen, wo auch ein Kasten stehen darf.**

#### 8.1.1 Schreibweise der Bedingung

```
i < 0           // untersucht, ob i kleiner 0
```

```
j > 13         // untersucht, ob j größer 13
```

```
k == 256       // untersucht, ob k den Wert 256 hat (schlechte Schreibung)
```

```
256 == k       // untersucht, ob k den Wert 256 hat (gute Schreibung)
```

```
l != 12        // untersucht, ob l ungleich 12 ist
```

```
//=====
```

```
k = 256      // weist k 256 zu und ist immer WAHR
```

```
256 = k     // gibt einen Compilerfehler
```

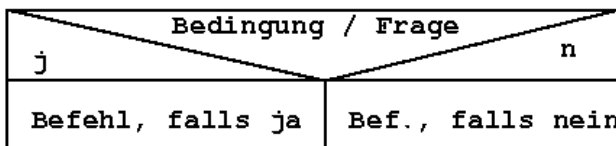
## 8.2 Übung: Notenauswertung

Der oben genannte Schüler habe sein Struktogramm bis zum Kasten "durchschn <- summe / anzahl" fertig.

Schreiben Sie Struktogramm und Programm fertig.

**An dieser Stelle die Übungen zur einseitigen Verzweigung.**

## 8.3 Zweiseitige Verzweigung



```
if (bedingung) {
    // befehl, falls ja
} else {
    // befehl, falls nein
}
```

Übung: Ein Programm bekommt `zaehler` und `nenner` eingegeben. Wenn das Ergebnis definiert ist, wird `quotient` ausgegeben, sonst eine Fehlermeldung. Struktogramm reicht.

**An dieser Stelle die Übung zur zweiseitigen Verzweigung.**

## 8.4 Übung: Einseitige Verzweigung

Zeichnen Sie stets zunächst ein Struktogramm. Übersetzen Sie anschließend das Struktogramm nach C++.

### 8.4.1 Zahlenterror

Untersuchen Sie, ob eine eingegebene float-Zahl  $<0$ ,  $=0$  oder  $>0$  ist.

### 8.4.2 Gehalt

Die Mitarbeiter der BIC KG bekommen eine Gehaltserhöhung: +3,5%, aber mindestens +50 EUR. Das Programm soll nach Eingabe des Gehalts die Gehaltserhöhung und das neue Gehalt ausgeben.

## 8.5 Übung: Zweiseitige Verzweigung

### 8.5.1 Reihenfolge

Schreiben Sie ein Programm, das zu 2 eingegeben Zahlen die richtige Ausgabe "zahl1 kommt vor zahl2." erzeugt.

## 8.6 Übung: Verknüpfte Bedingungen

### 8.6.1 Bauspar

Beim Bausparen erhalten Verheiratete für Sparleistungen bis zu 1600 EUR eine Prämie, Alleinstehende nur bis zur Hälfte dieses Betrages. Die Prämie beträgt 14% der Sparleistung. Für jedes Kind werden 2% zusätzlich gewährt. Es gibt halbe Kinder, aber keine negativen Kinder.

Schreiben Sie ein Programm, das nach Eingabe von Sparleistung, Familienstand und Kinderanzahl die Prämie ermittelt.

### 8.6.2 Bank

Eine Bank führt bei jedem Girokonto die ersten 10 Buchungen kostenlos aus. Für die nächsten 10 berechnet sie 30 Cent pro Buchung, für jede weitere 20 Cent.

Schreiben Sie ein Programm, das zur Anzahl Buchungen die Gebühr berechnet.

### 8.6.3 QuadrGl2

Es werde die allgemeine quadratische Gleichung  $x^2 + px + q = 0$  gelöst.

Berücksichtigen Sie die Sonderfälle  $p=0$  und/oder  $q=0$ .

### 8.6.4 DreiName

Drei eingegebene Zahlen sollen in aufsteigender Reihenfolge ausgegeben werden.

**8.6.5 Geradengleichung**

fragt nach  $m$  und  $b$  und gibt die Nullstelle aus und ob der Graph steigend oder fallend ist.

## 9 Verknüpfungs-Operatoren AND, OR, NOT

Wir verwenden die `bool`-Variablen `b1 b2 b3 b4`.

**AND:** (`b1 && b2`) ist WAHR, wenn `b1` und `b2` wahr sind.

**OR:** (`b1 || b2`) ist WAHR, wenn `b1` oder `b2` wahr ist.

**Klammern:** Ausdrücke mit Klammern werden von innen nach außen ausgerechnet.

**NOT:** `!` (Ausrufezeichen) invertiert den folgenden `bool`-Ausdruck.

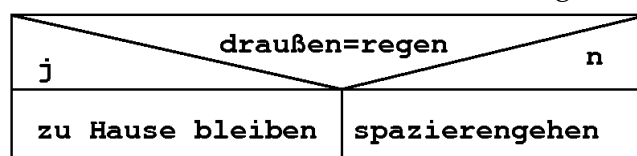
**Reihenfolge:** Logische Ausdrücke werden von links nach rechts ausgewertet.

### 9.1 Übung: Tabelle zum Ausfüllen

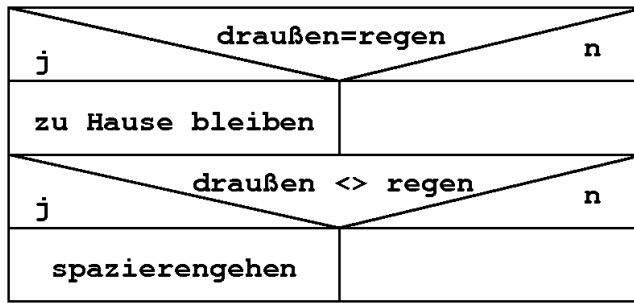
		b1=0,	b1=1,	b1=0,	b1=1,
		b2=0,	b2=0,	b2=1,	b2=1,
		b3=0,	b3=1,	b3=0,	b3=1,
		b4=1	b4=0	b4=0	b4=1
b1 && b2					
b3 && b4					
(b1 && b2) && (b3 && b4)					
b1 && (b2 && b3) && b4					
b1    b2					
b3    b4					
(b1    b2)    (b3    b4)					
b1    (b2    b3)    b4					
!b1 && b2    b3					
(!b1 && b2)    b3					
!b1 && (b2    b3)					
(!b1 && (b2)    b3)					
!(!b1 && !b2)					
!(!b3 && !b4)					
!(!b1    !b2)					
!(!b3    !b4)					
b1 && b2    b3 && b4					
(b1 && b2)    (b3 && b4)					
b1 && (b2    b3) && b4					
b1 && (b2    (b3 && b4))					

### 9.2 Wiederholung: Wasserscheuer Lehrer ohne Schirm

WENN es regnet, DANN bleibe ich zu Hause, SONST gehe ich spazieren.



Struktogramm:

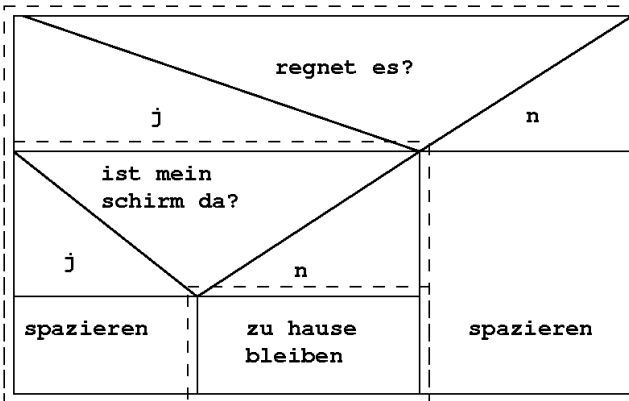


Struktogramm ohne SONST:

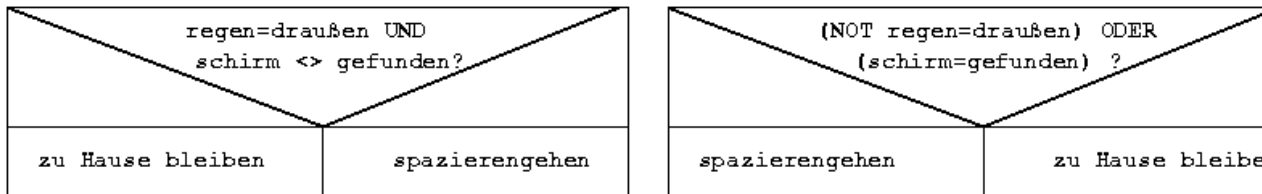
### 9.3 Beispiel: Wasserscheuer Lehrer

Wenn es NICHT regnet ODER ich endlich meinen Schirm finde, gehe ich spazieren, sonst bleibe ich zu Hause.

Wenn es regnet UND mein Schirm ist NICHT da, bleibe ich zu Hause, IN JEDEM ANDEREN FALL gehe ich spazieren.



Verschachteln:



Verknüpfen:

Übung: Malen Sie die Kästen bunt an.

**Abhängig von der Bedingung geht das Programm entweder den linken oder den rechten Weg. Niemals beide. Niemals keinen.**

Übung: Denken Sie sich in verschiedenen Farben Startbedingungen aus, und malen Sie die entsprechenden Wege bunt an.

### 9.4 Syntax

#### 9.4.1 Einseitig

```
if (bedingung) {           // die ( ) müssen sein
    then-block
}
```

### 9.4.2 Zweiseitig

```
if (bedingung) {  
    then-block  
} else {  
    else-block  
}
```

ODER

```
if (bedingung) {  
    then-block  
}  
if (! bedingung) {  
    else-block  
}
```

### 9.4.3 Verschachtelt

```
if (bedingung1) {  
    if (bedingung2) {  
        then-block  
    }  
}
```

ODER

```
if (bedingung1 && bedingung2) {  
    then-block  
}
```

## 9.5 Übung: Schirm o.k.?

Definieren Sie die drei Variablen *regen*, *sgefunden*, *sdefekt*.

Formulieren Sie die Regel, nach der der Lehrer spazierengeht / die Regel, nach der der Lehrer zu Hause bleibt / in Prosa, Struktogramm und C++.

Legen Sie eine Tabelle an mit den Spalten *regen*, *sgefunden*, *sdefekt*, *spazieren*, *zuHause*. Füllen Sie die ersten drei Spalten mit allen möglichen Kombinationen. Prüfen Sie Ihre Regeln an den beiden letzten Spalten.

Teil II

## 2. Quartal

## 10 cmath

### 10.1 cmath

```
#include <cmath>
```

gibt die folgenden mathematischen Funktionen:

sin, cos, tan, sqrt, log, log10, ceil, floor, exp, pow.

sin(float)	gibt den Sinus einer float-zahl	sin(0) = 0 sin(1) = 0.84147
cos(float)	gibt den Cosinus einer float-zahl	cos(0) = 1 cos(1) = 0.54030
tan(float)	gibt den Tangens einer float-zahl	tan(0) = 0 tan(1) = 1.5574
sqrt(float)	gibt die Wurzel einer float-zahl	sqrt(65) = 8.06226
log(float)	gibt den Natürlichen Logarithmus (zur Basis e, e=2.7182818) einer float-zahl	log(1) = 0 log(10000) = 9.21034
log10(float)	gibt den Zehner-Logarithmus (zur Basis 10) einer float-zahl	log10(1) = 0 log10(10000) = 4 log10(200) = 2.30103
ceil(float)	(ceiling, engl.: Himmel) gibt die nächstgrößere ganze Zahl	ceil(4.5) = 5 ceil(-4.5) = -4
floor(float)	(floor, engl.: Fußboden) gibt die nächstkleinere ganze Zahl	ceil(4.5) = 4 ceil(-4.5) = -5
exp(float x)	gibt $e^x$	$e^1 = 2.7182818$ $e^0 = 1$

**Übung:** Schreiben Sie ein Programm, daß die Beispiele bestätigt.

### 10.2 Bogenmaß [optional]

Es hat die Mathematiker schon lange geärgert, daß die herkömmlichen Winkelbezeichnungen eine Einheit (grad) mit sich herumschleppen.

Deshalb haben sie eine neue Winkelbezeichnung **ohne Einheit** erfunden: das Bogenmaß.

**Definition: Bogenmaß gibt den Winkel an durch den Quotienten aus Bogenlänge und Radius. Es hat keine Einheit. Zur besseren Lesbarkeit wird häufig rad (wie Radianten) dahintergesetzt.**

0 grad = 0 rad. 90 grad =  $\pi/2$  rad. 180 grad =  $\pi$  rad. 360 grad =  $2\pi$  rad.

$\pi$  ist ungefähr 3.1415926. Damit ist ein Vollkreis nicht mehr 360 grad, sondern 6.28 rad.

## 11 Zählschleife (FOR)

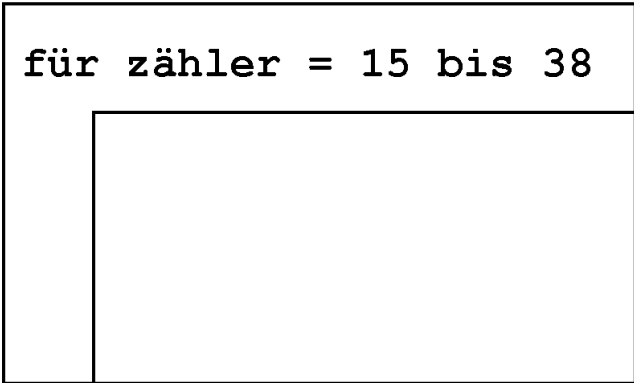
**Beispiel:** Wir wollen die Quadratzahlen von 1 bis 5 ausgeben. Das geht u.a. wie folgt:

```
cout << 1 << " ist quadriert: " << 1*1 << endl;
cout << 2 << " ist quadriert: " << 2*2 << endl;
cout << 3 << " ist quadriert: " << 3*3 << endl;
cout << 4 << " ist quadriert: " << 4*4 << endl;
cout << 5 << " ist quadriert: " << 5*5 << endl;
```

Für die Quadratzahlen von 1 bis 100 oder gar 1000 ist diese Methode jedoch inakzeptabel.

Wie nett wäre es, wenn man dem Rechner sagen könnte:

"He, Rechner! i sei zunächst mal 1. Führe nun die folgenden Befehle aus: ... Und jetzt erhöhst du i um 1 und machst das ganze nochmal, bis i>9."



**für zähler = 15 bis 38**

C++ ist nett. Struktogramm:

Im C++-Programm sieht das etwas anders aus. Wir haben eine Initialisierung, eine Weitermachbedingung und einen Weiterzähler, alles zusammengefaßt in einer Zeile:

```
for (j=9 ; j<=29 ; j=j+1) {
    ...
}
```

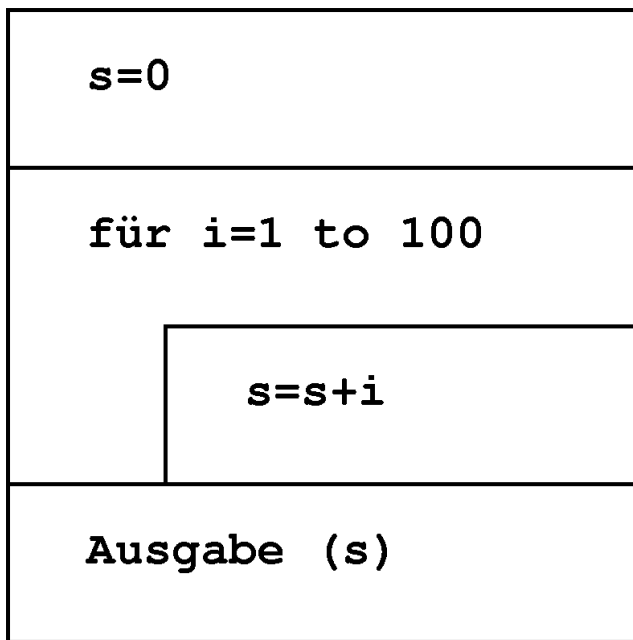
**Erläuterung:** j wird zunächst auf 9 gesetzt - anschließend wird dieser Befehl nicht mehr angeschaut.

\*\*\* Falls der zweite Term erfüllt ist, wird der Anweisungsteil (...) ausgeführt. Dann wird weitergezählt. Weiter bei \*\*\*.

Daraus folgt übrigens, daß die Variable j mit dem Wert 30 aus der obigen Zählschleife herauskommt: sie wurde hochgezählt und erfüllte anschließend die Bedingung nicht.

### 11.1 Summenbildung

Als nächstes wollen wir die Zahlen von 1 bis 100 zusammenzählen. Wir gehen vor wie folgt:



Machen Sie jetzt einen Schreibtischttest für die Zahlen von 1 bis 10.

## 11.2 Übung

### 11.2.1 FORDO01

Das folgende Programm soll 20 Zeilen ausgeben:

in der ersten Spalte die Zahlen 1..20, in der zweiten  $1^2$  bis  $20^2$ ,

in der dritten  $1^3$  bis  $20^3$ , in der vierten  $1^4$  bis  $20^4$ .

Es sind integer zu verwenden. Spaltenbreite = 10.

Falls Sie negative Werte erhalten, kommentieren Sie das Ergebnis schriftlich, und geben Sie den Kommentar in einer weiteren Programmzeile aus.

### 11.2.2 FORDO02

Schreiben Sie ein Struktogramm und anschließend ein Programm, das 21 Zeilen ausgibt:

in der ersten Spalte die Zahlen -5 bis 5 in Schrittweite 0,5,

in der zweiten Spalte deren Quadrat,

in der dritten Spalte den ganzzahligen Anteil des Wertes der 1. Spalte,

in der 4. Spalte den Kehrwert der ersten Spalte (sofern möglich).

Benutzen Sie Spaltenbreite 8.

### 11.2.3 Fakultät

Schreiben Sie ein Struktogramm und anschließend ein Programm, das  $n!$  (lies: enn Fakultät) berechnet. Das Programm soll  $10!$  fehlerfrei berechnen.  $n$  wird eingegeben.

Hinweis:  $0! = 1$ .

$$n! = 1 * 2 * 3 * .. * (n-1) * n \text{ für } n > 0.$$

### 11.2.4 Großes Einmaleins

Schreiben Sie ein Struktogramm und anschließend ein Programm, das das große Einmaleins (von  $1*1$  bis  $20*20$ ) in Tabellenform auf den Bildschirm schreibt.

### 11.2.5 ASCII-Tabelle

Schreiben Sie ein Programm mit der Integer-Variable `lauf` und der Char-Variable `c`.

`lauf` soll nacheinander die Zahlen von 32 bis 255 durchlaufen. Außerdem wird `c` `lauf` zugewiesen. Dann wird `lauf`, gefolgt von einem Leerzeichen, `c` und noch drei Leerzeichen in übersichtlichen Spalten auf den Bildschirm geschrieben.

### 11.2.6 SINUS-Verlauf

Schreiben Sie ein Programm, das einen um  $90^\circ$  gedrehten Sinus-Verlauf von 0 bis  $2*PI$  auf den Bildschirm malt. Benutzen Sie dazu die Formatierung innerhalb einer `writeln`-Anweisung.

**Hinweis:** `sin()` steckt im Paket `cmath` und funktioniert im Bogenmaß (rad): 0 grad = 0 rad. 180 grad = pi rad. 360 grad = 2pi rad = 1 Vollkreis.

## 12 Modularisierung (VOID)

Manchmal möchte man Codeteile auslagern. Das ist besonders sinnvoll, wenn man Programmteile mit völlig verschiedenen Aufgaben voneinander trennen will.

Beispiel:

```
int i=9;

void sagTschuess () {                               // Modul sagTschuess
    cout << "Tschüs." << endl;
}

void sagHallo () {                                  // Modul sagHallo
    cout << "Hallo." << endl;
}

int main () {                                       // main()
    sagHallo();
    sagTschuess();
}
```

Diese Programmbröckchen heißen Funktionen.

Unsere speziellen Funktionen hier geben keinen Wert zurück (`void`) und übernehmen auch keinen (leere Klammern). Die Klammern sind trotzdem Pflicht, sonst wäre es nur ein Variablenname.

Es ist sinnvoll, Funktionsnamen aus einem Verb und einem Hauptwort zusammensetzen.

## 13 Kopf- und Endegesteuerte Schleife (WHILE / DO...WHILE)

### 13.1 Menüprogramm

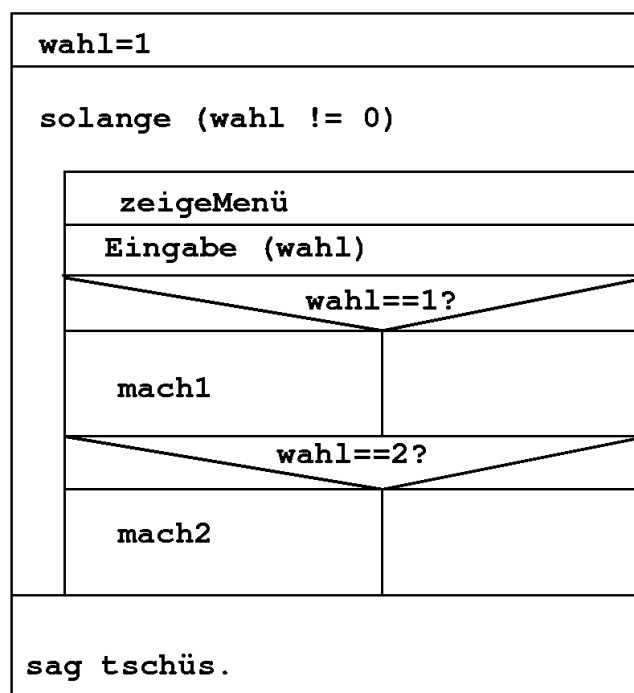
Manchmal möchte man ein Programm mehrmals durchlaufen lassen, zB wenn es ein interaktives Programm ist.

Beispiel:

```
Hallo.
1  Quadratzahlen von 0 bis 10 anzeigen
2  Kubikzahlen von 0 bis 10 anzeigen
0  Exit
Wähle:
```

### 13.2 Schleife mit Startbedingung

**Die Schleife mit Startbedingung wird wieder und wieder ausgeführt, solange die Startbedingung erfüllt ist.**



Struktogramm:

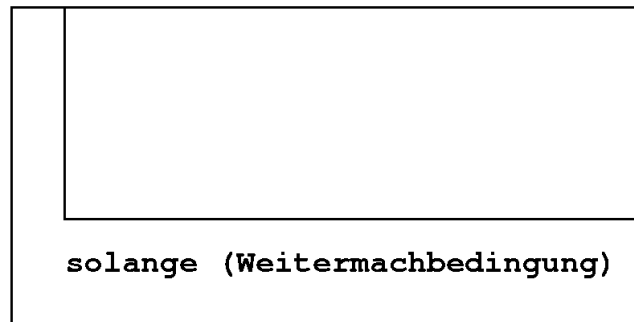
Programm (teilweise):

```
...
while (wahl != 0) {
    ...
}
```

Aufgabe: Simulieren Sie eine Zählschleife, in der *i* von 0 bis 5 läuft, mit einer while-Schleife.

### 13.3 Schleife mit Weitermach-Bedingung

**Die Schleife mit Weitermach-Bedingung wird zunächst ausgeführt. Solange die Weitermach-Bedingung erfüllt ist, wird sie erneut ausgeführt. Wenn man die Weitermach-Bedingung verneint, erhält man die Abbruchbedingung.**



Struktogramm:

Programm (teilweise):

```
...
do {
    // block
} while (Weitermach-Bedingung) ;           // Semikolon ist wichtig hier
...
```

Aufgabe: Simulieren Sie eine Zählschleife, in der  $i$  von 0 bis 5 läuft, mit einer while-Schleife.

Aufgabe: Schreiben Sie das Struktogramm für das obige Menü-Programm mit einer do-while-Schleife.

## 13.4 Übungen

Zu jeder Aufgabe gehört zuerst ein Struktogramm und dann erst ein C++-Programm.

### 13.4.1 Summe der Potenzen von 1/2

Berechnen Sie die Summe von  $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots$ , bis der Summand kleiner als 0.00001 ist.

### 13.4.2 Ken's benutzerfreundlicher Quader

Lassen Sie zuerst die Quaderlängen eingeben.

Ken's benutzerfreundliche Quaderberechnung

- 1 Oberfläche
- 2 Volumen
- 3 Kantenlänge
- 4 Raumdiagonale
- 5 neue Längen a,b,c eingeben
- 0 Exit

### 13.4.3 Summe(Kehrwerte(Quadrate))

Schreiben Sie ein Programm, das die Summe aller  $\frac{1}{x^2}$  für x von 1 bis ??? berechnet und diese Summe anzeigt, wenn  $\frac{1}{x^2}$  kleiner als 0.00001 ist.

Benutzen Sie im Programm die folgende Zeile: `gonzo = 1/i/i;`

### 13.4.4 Iteratives Wurzelziehen nach Lisa

Lisa sagt: Um die Wurzel w aus der zahl z zu ziehen, fange ich an mit a=1 und b=z. Dann berechne ich ein besseres a aus (a+b)/2 und ein passendes b aus z/a. Bis a nahe genug an b ist.

### 13.4.5 Diophant

Es sollen 12345 Personen befördert werden. Es stehen Busse mit 31, 41 und 43 Sitzen zur Verfügung.

Bestellen Sie möglichst wenig Busse so, daß alle Sitzplätze besetzt sind.

## 14 Globale und Lokale Variablen - Arbeitsblatt

### 14.1 Was der Compiler sah

```
// erstmal inkludieren
#include <iostream>

// programmstart
int i = 2; // globales i

void
mod2 ()
{
    int i = 3; // noch ein i, lokal in mod2()
    cout << "mod2: lokales i = " << i << endl;
}

int
main ()
{
    mod2 ();
    cout << "globales i = " << i << endl;
    int i = 5; // noch ein i, lokal in main()
    { // the following block is called raptor
        int i = 9;
        cout << "raptor: i = " << i << endl;
        mod2 ();
    } // end of raptor
    cout << "main: lokales i = " << i << endl;
    mod2 ();
}
```

### 14.2 Was der Rechner ausspuckte

```
***** Tue Dec 27 13:41:24 EST 2005 ***** c++ start *****
mod2: lokales i = 3
globales i = 2
raptor: i = 9
mod2: lokales i = 3
main: lokales i = 5
mod2: lokales i = 3
***** Tue Dec 27 13:41:24 EST 2005 ***** c++ end *****
```

### 14.3 Ankreuzen

Schreiben Sie R für Richtig oder F für Falsch daneben:

- Es gibt nur eine Variable `i`.
- Es gibt im Modul `mod2()` eine Variable `i`, die deklariert und auf 3 gesetzt wird.
- Jedesmal, wenn in `mod2()` das `i` ausgegeben wird, kommt dasselbe heraus.
- `mod2()` interessiert sich für das `i`, das in `main()` definiert wird.
- `mod2()` interessiert sich für das `i`, das am Programmmanfang definiert wird.
- `main()` kann das `i`, das am Programmmanfang definiert wird, sehen und ausgeben.
- `main()` kann sich selber ein `i` definieren.
- `main()` kann anschließend das `i`, das am Programmmanfang definiert wird, sehen und ausgeben.
- `mod2()` interessiert sich für die anderen `i`.
- Innerhalb eines Moduls kann man lokale Variablen definieren.
- "Lokalere" Variablen decken "globalere" zu.
- `raptor` ist ein Block, weil das so im Kommentar steht.
- `raptor` ist ein Block, weil er in Klammern steht.
- `raptor` ist ein Block, weil er eingerückt ist.
- Von `raptor` aus kann ich solange das "nächst-globalere" `i` sehen, wie ich kein `raptor`-eigenes definiert habe.
- Von `raptor` aus kann ich nur das globale `i` sehen, wenn ich ein `raptor`-eigenes definiert habe.
- Modul `mod2()` interessiert sich dafür, ob in seinem aufrufenden Kontext schonmal ein `i` definiert wurde.
- In allen meinen Modulen darf ich `i` als Schleifenzähler einsetzen, ohne mit dem Hauptprogramm durcheinanderzukommen.
- In jeder Schleife darf ich `i` neu definieren, wenn ich sie in einen Block setze.

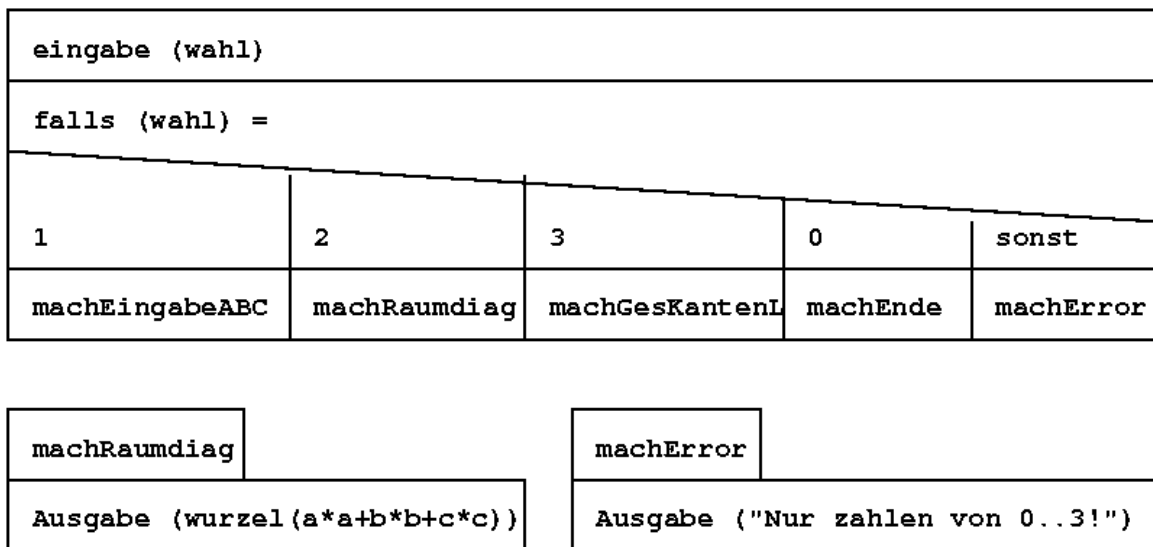
## 15 Bedingte mehrfache Verzweigung [optional]

Gegeben sei ein Menü-Programm, das folgenden Menübildschirm hat:

Ken's benutzerfreundliche Quaderberechnung

- 1 neue a,b,c, eingeben
- 2 Raumdiagonale ausgeben
- 3 Oberfläche ausgeben
- 0 Exit

Struktogramm für Bedingte Mehrseitige Verzweigung:



Und der C++-Code dazu sieht aus wie folgt:

```

switch (wahl) {
    case 1 : machEingabe();           // schalte weiche abhängig von wahl
           break;                    // bei 1 mache Eingabe,
    case 2 : machRaumdiag();         // dann verlasse dieses switch
           break;                    // bei 2 mache Raumdiag,
    case 3 : machGesKantL();        // dann verlasse dieses switch
           break;                    // bei 3 mache Eingabe,
    case 0 : break;                 // dann verlasse dieses switch
           // bei 0 mach nix, die while-Schleife
           // wird das Programm beenden
    default: machError();
}                                     // end of switch

```

**switch vergleicht die Variable nur mit Konstanten.  
Wenn man das break wegläßt, werden auch die folgenden Anweisungen ausgeführt.**

**Beispiel:** Wenn man im obigen Beispiel das `break` bei `case 0` wegläßt, wird das Programm kurz vor dem Beenden eine häßliche Fehlermeldung ausspucken.

## 16 Zufallszahlen erzeugen

1. Zunächst Zufallszahlengenerator neu setzen mit `srand(time(NULL))` ;
2. Zufallszahl von 0 bis n-1 abrufen mit `rand() % n`

### 16.1 Übung

#### 16.1.1 Lotto

Erzeugen Sie 6 Zufallszahlen und eine Zusatzzahl, alle zwischen 1 und 49.

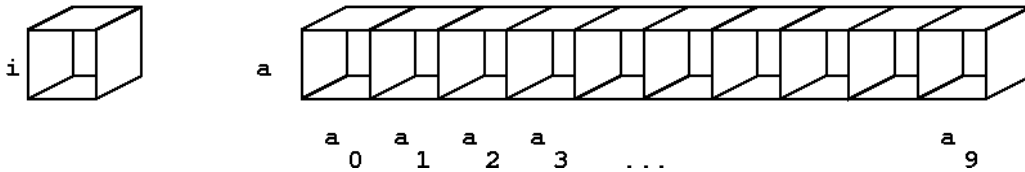
#### 16.1.2 Verteilung

Zufallszahlen nützen nur etwas, wenn sie gleichverteilt sind.

Schreiben Sie ein Programm, das eine Million Zufallszahlen zwischen 0 und 4 erzeugt, jede zählt, und am Ende die Anzahlen jeder Zahl auswirft.

## 17 Felder (arrays)

Wie nett wäre es, wenn man in C++ Vektoren definieren könnte. Das sind Speicher für Zahlen, die alle denselben Namen haben und sich nur im Index unterscheiden.

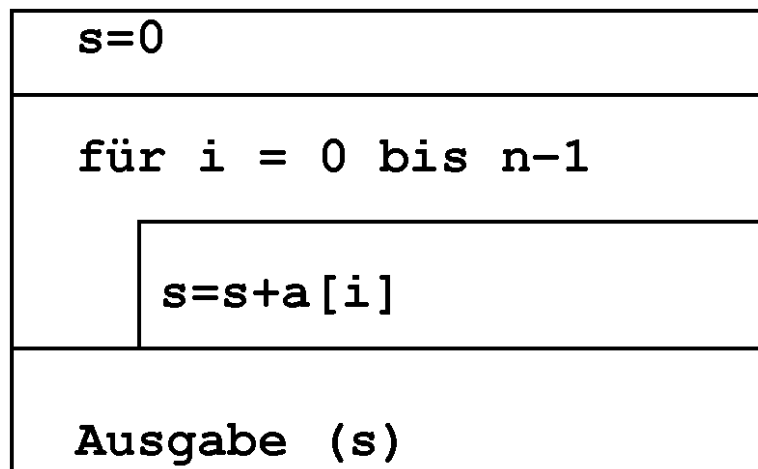


C++ ist nett.

Wir werden Felder definieren, initialisieren, zuweisen, ein bißchen damit rechnen und ausgeben.

**Felder von  $n$  Elementen haben die Indices 0 bis  $n-1$ .  
 Einzelne Feldelemente werden genauso angesprochen (eingegeben, zugewiesen, ausgegeben, als Schleifenzähler benutzt) wie "normale" Variablen auch.**

Und hier ist noch, wie man alle Elemente eines Feldes zusammenzählt:



```
// alles mit feldern. zieh eine folie, oder mach fotokopien
#include <iostream>

int n = 9;           // ich nehme gerne n für die anzahl der elemente
int a[9];           // jetzt gibt es a[0] bis a[8]
                    // andere namen (b, meinFeld, array, ...) sind auch erlaubt
int i;             // andere namen (j,k,l,ii,... ) sind auch erlaubt

int main ()
{

// initialisierung
  for (i = 0; i <= n - 1; i++) // falls ich die anzahl der elemente
                              // erhöhen will, muß ich bloß
                              // 2 zeilen ändern
  {                            // i++ ist dasselbe wie i=i+1
    a[i] = 0;                  // jetzt sind alle Elemente 0
  }

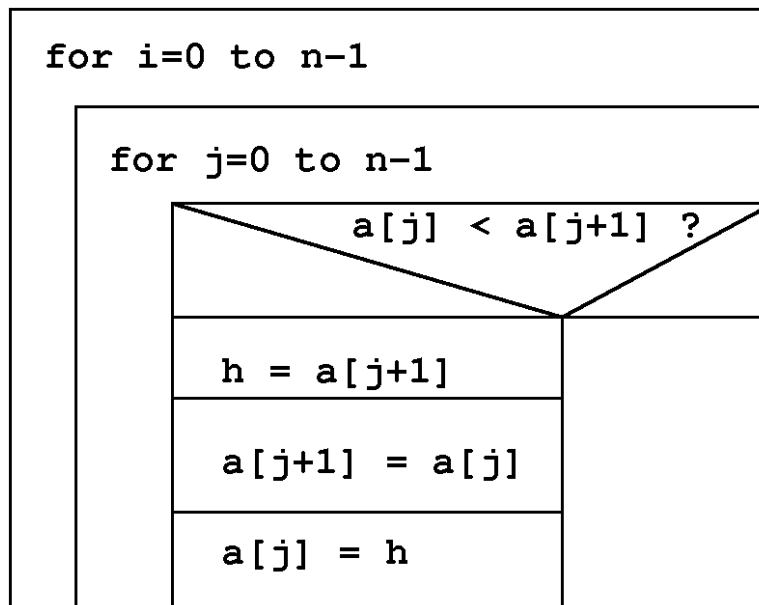
// zuweisen
  for (i = 0; i <= n - 1; i++) // i++ ist dasselbe wie i=i+1
  {
    a[i] = i;                  // in a[i] steht jetzt i
  }

// nochmal zuweisen
  for (i = 0; i <= n - 1; i++)
  {
    a[i] = a[i] * a[i];        // in a[i] steht jetzt i^2
  }

// ausgeben, rückwärts
  for (i = n - 1; i >= 0; i--) // i-- ist dasselbe wie i=i-1;
  {
    if (a[i] < 10)
      cout << " ";
    cout << a[i] << " " << i << endl;
  }
}
```

## 18 Sortieren

### 18.1 Bubblesort



**Übung:** Schreibtischtest für n=8.

**Das größte Element blubbert bereits im ersten Durchgang nach oben, das zweitgrößte im zweiten, usw. Die Idee ist, daß die großen (schweren) die kleinen (leichten) verdrängen. Und das funktioniert ja auch.**

#### 18.1.1 Optimieren

- Im 2. Durchlauf von i braucht man das j nicht mehr bis n-2 laufen zu lassen, denn das größte Element ist ja schon dort.
- man braucht für i maximal n-2 Durchläufe. Man braucht vielleicht weniger. Das Feld ist sortiert, wenn es bei einem kompletten j-Durchlauf nichts mehr zu tauschen gab. Das kann man mit einer `while`-Schleife realisieren.

## 18.2 Minsort

Bubblesort führt sehr viele Vergleiche und Zuweisungen durch. Minsort ist ein schnelleres Verfahren.

Minsort durchsucht das ganze Feld nach dem Minimum und tauscht anschließend das Minimum mit dem ersten Element.

Dann verfährt es analog mit dem Rest des Feldes.

## 18.3 Quicksort [optional]

Quicksort ist ein elegantes, schnelles, rekursives Sortierverfahren.

Es bekommt einen Ausschnitt des Feldes übergeben, von  $l$  bis  $r$ . Hier sucht es sich einen Median oder Schwellenwert.

Dann schiebt es innerhalb des Ausschnitts alle Werte, die kleiner sind als der Median, nach links, und alle anderen (also alle, die größer ODER gleich sind als der Median) nach rechts. Dabei merkt es sich, wo der Übergang  $ü$  ist.

Anschließend sortiert es die beiden Teilfelder, sowohl von  $l$  bis  $ü$  als auch von  $ü+1$  bis  $r$ .

### 18.3.1 Suchen des Medians [optional]

Das Finden eines effektiven Medians ist eine Wissenschaft für sich. Ich verwende das linke Element.

Dies erzeugt ein hochinteressantes Problem, falls das linke Element das Minimum des Teilfeldes ist. Dann gibt es nämlich kein neues linkes Teilfeld, und das neue rechte Teilfeld ist das original übergebene Teilfeld. `sort()` würde sich dann unendlich oft mit demselben Teilfeld aufrufen...

Daher prüfe ich, ob es ein linkes Teilfeld gibt. Wenn ja, verfare ich wie oben. Wenn nein, rufe ich nur `sort(l+1, r)`, denn das linke Element sitzt ja schon an der richtigen Stelle.

## 19 Zeitmessung im Millisekunden-Bereich [optional]

### 19.1 Mit `mymillitime` - 8 Wochen alt und schon historisch

```
// this is mymillitime.cpp
#include <iostream>
#include <cstdlib>
#include <sys/timeb.h>
#include <iomanip>
#include <ctime>

/** mymillitime returns the number of MilliSeconds
 * since the start of the epoch (01.01.1970, 00:00:00 hours).
 */
long long mymillitime ()
{
    timeb tb;          // this is a struct, defined in timeb.h
    ftime (&tb);      // functions called on structs must give the struct
                      // address as parameter, then the struct is changed.
    long long mytb = tb.time;
    mytb = mytb * 1000 + tb.millitm; // because mytb*1000 is real big.
    return (mytb);
}
```

`mymillitime()` kann inkludiert und zur Zeitmessung im Millisekundenbereich herangezogen werden:

```
#include "mymillitime.cpp"
...
long long startzeit = mymillitime();
... // hier der Vorgang der gemessen werden soll
long long endzeit = mymillitime();
int millisecs = endzeit - startzeit;
...
```

### 19.2 Mit `time.h`

```
#include <time.h>
...
    clock_t start = clock();
// do something
    clock_t ende = clock();
    float zeit = (float)(ende-start) / CLOCKS_PER_SEC;
/* * 1000 wenn man millisekunden mag */
...
```

## 19.3 Übung

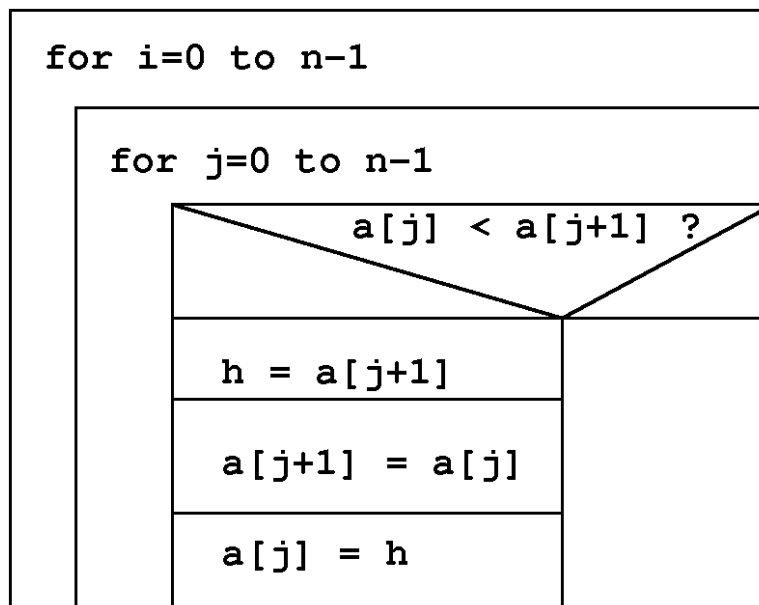
### 19.3.1 Lotto2

Wie Lotto. Bloß daß keine Zahl doppelt auftreten darf.

### 19.3.2 Verteilung2

Wie Verteilung. Nur mit 20 verschiedenen Zahlen.

### 19.3.3 Bubblesort



Sortieren Sie ein Feld aus 10000 Integern mit Bubblesort.

### 19.3.4 Minsort

Sortieren Sie ein Feld aus 10000 Integern mit Minsort, und messen Sie die Zeit. Wie?

1. Inkludieren Sie `ctime`.
2. Setzen Sie vor dem Sortieren `starttime = time(NULL);`.
3. Setzen Sie nach dem Sortieren `endtime = time(NULL);`.
4. Geben Sie `endtime-starttime` aus.

## 20 Funktionen mit Rückgabewert - call by value

Wir benutzen bereits Funktionen, zB die Funktion `sqrt()`.

Sie bekommt einen Wert übergeben und gibt einen anderen Wert zurück.

Der übergebene Wert verschwindet auf Nimmerwiedersehen. Der zurückgegebene Wert ist das Resultat des Funktionsaufrufes.

Das können wir auch.

### 20.1 `int summe`

```
int summe (int a, int b) {  
    // hallo c++! ich definiere eine Funktion namens summe. sie gibt ein  
    // integer zurück und bekommt zwei integer a und b übergeben.  
    // diese integer sind vom Rest des programms nicht zu sehen, da sie  
    // in dieser funktion deklariert werden. du kannst im aufrufenden  
    // programm so viele a und b verwenden wie du willst.  
  
    return (a + b);  
    // du sollst a und b zusammenzählen und das ergebnis zurückgeben.  
}  
// fertig.
```

**Übung:** Ein Schüler ist die Funktion `summe` und wird vom Lehrer aufgerufen. Er soll ihm die beiden Integer abringen (zB 5 und 6) und ihm das Ergebnis zurückgeben (zB 11).

### 20.2 Mechanismus [optional]

Das Hauptprogramm wirft die beiden Integer und seine Rücksprungadresse auf den Stapel. Dann springt es an den Anfang der Funktion `summe()`.

Die Funktion `summe()` rettet den Rücksprungzeiger vom Stapel, holt sich die beiden Integer vom Stapel, zählt sie zusammen, wirft das Ergebnis auf den Stapel und springt die Rücksprungadresse an.

Das Hauptprogramm findet das Ergebnis auf dem Stapel und freut sich.

Intelligente Programmierer nutzen schlampig programmierte String-Bibliotheken oder TCP-IP-Stacks, um Programmcode an bestimmten Stellen im Speicher zu überschreiben, sodann dorthinzuspringen und ihren böartigen Code mit interessanten Benutzer-Rechten auszuführen.

## 20.3 Übung

### 20.3.1 produkt() etc.

Schreiben Sie die Funktionen `differenz()`, `produkt()` und `quotient()` für je zwei übergebene Integer. Wählen Sie passende Datentypen.

### 20.3.2 Schreiben Sie `lisaSqrt(float)`

Lisa sucht die Wurzel  $w$  aus der Zahl  $z$  wie folgt:

Sie setzt  $a=z$  und  $b = 1$ .

\*

Sie setzt  $a=(a+b)/2$  und  $b=z/a$  (so daß stets  $a*b=z$ ).

Wenn  $a$  und  $b$  noch nicht ähnlich genug sind, macht sie bei \* weiter.

Finden Sie `lisaSqrt(5.0)` **auf Papier**, bevor Sie mit Coden anfangen.

### 20.3.3 readln

C++ liest strings nur bis zum ersten Leerzeichen ein. Programmieren Sie

```
string readln()
```

Es liest Tastatureingaben in die Variable `char c` mittels `cin.get()`, bis der User Enter drückt, d.h. (`c == 10`).

```
// ... string readln() { ... } // selber herausfinden ...

int main () {
    cout << "readln() by worgtsone, 25.02.2006" << endl;
    cout << "Sag mir deinen Namen mit Leerzeichen: ";
    string mystring = readln ();
    // Hoppla, schlampig programmiert, das chr(10) hängt auch noch an mystring
    cout << "Hallo " << mystring << endl;
}

***** Sun Feb 26 12:13:51 EST 2006 ***** c++ start *****
readln() by grm, 25.02.2006
Sag mir deinen Namen mit Leerzeichen: john doe user
Hallo john doe user

***** Sun Feb 26 12:13:55 EST 2006 ***** c++ end *****
```

Teil III

## 3. Quartal

# 21 Multi-Referat Overkill

7of9 in normal, verpixelt, 5 Graustufen, mehr Kontrast.



```

File: 7of9.raw.pgm          Col 0          307239 bytes          0%
P5
# Created by IrfanView

640 480
255
.....C.....iBEJE623)/11/)&(*#".....
%(+,., (" ...$23$. # &66+-%",7:<=JKOS[ `aaXddZ\lqkbejptvyx}.....
.....
.....

```

```

File: 7of9.raw.pgm          Offset 0x00000000  307239 bytes          0%
00000000 50 35 0A 23 x 20 43 72 65 x 61 74 65 64 x 20 62 79 20          P5.# Created by
00000010 49 72 66 61 x 6E 56 69 65 x 77 0A 0A 36 x 34 30 20 34          IrfanView..640 4
00000020 38 30 0A 32 x 35 35 0A 14 x 14 14 14 x 14 14 14 16          80.255.....
00000030 16 15 14 14 x 13 12 12 12 x 12 12 12 12 x 12 12 12 12          .....

```

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

/**
dieses programm saugt eine datei namens infile.pgm ins feld a[[]],
manipuliert die pixel und schreibt sie ins feld outfile.pgm.

int main () {
    getInfile();
    pixelize();
    writeOufile();
}
*/

const int width = 640;
const int height = 480;
int a[width][height];
int i, j, k;
string s;
char c;

void getInfile () {
    ifstream infile ("7of9.raw.pgm");

    // save the first 5 zeils in string s
    s = "";
    j = 0;
    while (j < 5) {
        k = infile.get ();
        s += k;
        if (k == 10)
            j++;
    }

    for (int zeil = 0; zeil < height; zeil++)
        for (int spal = 0; spal < width; spal++) {
            k = infile.get ();
            a[spal][zeil] = k;
        }
    infile.close ();
}

void pixelize () {
    for (int zeil = 0; zeil < height; zeil++)
        for (int spal = 0; spal < width; spal++)
            a[spal][zeil] = (a[spal][zeil] * 2) % 256;
}

void graustuf (int gs) {
    gs = 255 / gs;
    for (int zeil = 0; zeil < height; zeil++)
        for (int spal = 0; spal < width; spal++)
            a[spal][zeil] = (a[spal][zeil] / gs) * gs + gs / 2;
}

void morecontr () {
    for (int zeil = 0; zeil < height; zeil++)
```

```
    for (int spal = 0; spal < width; spal++) {
        a[spal][zeil] = (a[spal][zeil] - 127) * 1.5 + 127;
        if (a[spal][zeil] < 0)
a[spal][zeil] = 0;
        if (a[spal][zeil] > 255)
a[spal][zeil] = 255;
    }}

void invert () {
    for (int zeil = 0; zeil < height; zeil++)
        for (int spal = 0; spal < width; spal++)
            a[spal][zeil] = 255 - a[spal][zeil];
}

void writeOufile () {
    ofstream outfile ("7of9.raw.pixed.pgm");
    // write the first 4 zeils
    outfile << s;

    for (int zeil = 0; zeil < height; zeil++)
        for (int spal = 0; spal < width; spal++) {
            c = a[spal][zeil];
            outfile << c;
        }
    outfile.close ();
}

void whatsup () {
    for (int zeil = 0; zeil < height; zeil++) {
        for (int spal = 0; spal < width; spal++)
            cout << hex << a[spal][zeil];
        cout << endl;
    }
    cout << endl;
}

int main () {
    getInfile ();
    // whatsup ();
    // pixelize ();
    // graustuf (32);
    morecontr ();
    // invert ();
    // whatsup ();
    writeOufile ();
}
```

## 21.1 Ablauf

Am Mo sucht sich jeder Schüler ein paar Fragen sowie 2 Vertreter aus.

Am Do kann man Rückfragen an den Lehrer richten.

Am Mo werden die Fragen der Reihe nach beantwortet. Falls ein Schüler krank ist, übernimmt ein Vertreter. Falls der Vertreter nichts zustande bringt, leiden beider Noten entsetzlich.

## 21.2 Fragen

1. Was ist ein 256-Graustufen-Bild?
2. Welche Formate sind üblich?
3. Wie ist ein raw-Graustufen-pgm aufgebaut?
4. Was ist ein hexeditor?
5. Was ist ein hexdump?
6. Was ist ein fstream? Wie wird er oben benutzt?
7. Was ist ein String? Wie wird string s oben benutzt?
8. Was ist das für ein Feld a[][]? Wozu wird es oben benutzt?
9. Wie wird die infile eingelesen?
10. Wie wird die outfile geschrieben?
11. Was bedeutet `variablenname.close()`?
12. Wie funktioniert `pixelize()`?
13. Wie funktioniert `graustuf(int)`?
14. Wie funktioniert `morecontr()`?
15. Wie funktioniert `invert()`?
16. Was ist das für eine merkwürdige Fehlermeldung im Modul `morecontr()`? Wie kann man sie vermeiden?
17. Was ist ein `DosNavigator`? Was kann er?
18. Was ist `IrfanView`? Was kann er?
19. Was ist `Gimp`? Was kann er?
20. Was ist Komprimierung?

### 21.3 Hinweis für die Neugierigen

Die Funktion `graustuf()` funktioniert nicht für sehr helle Werte.

Beispiel: Für 5 Graustufen gibt sie für 254 eine 288 (korrekt), für 255 jedoch eine 280 (zu viel).

Verbesserungsvorschläge per Email sind willkommen. Scheint mit der unsauberen Berechnung der Intervallbreite zu tun zu haben.

## 22 Webserver ärgern [optional]

Die Windos-Gemeinde von 2006 ist paranoid geworden und erlaubt via Proxy den Download von binären Dateien nicht mehr, um sich nicht noch mehr Viren, Würmer, Trojaner, Mal- und Spyware einzufangen.

Das ist furchtbar lästig, wenn man binäre Dateien herunterladen will, zB den Lieblings-Editor, -Ripper, -Recoder, -Compiler oder was auch immer.

Gottlob dürfen noch zwei Dateitypen in jeden Browser: \*.html und \*.txt.

Es ist auch nicht weiter schwer, binäre Dateien nach text umzusetzen. ZB: Man nehme das erste Byte, kodiere es nach hex (aus 0 wird 00, aus 127 wird 7F, usw.) und schreibe diese hex-Zahlen in eine neue Datei. Bis Dateiende.

Fehlt nur noch das Programm, mit dem man das erledigt. Der Download von \*.exe fällt wohl aus, also downloaden wir lieber den Quelltext und übersetzen ihn mittels des installierten C++-Compilers.

Der komplette Proxy-Betrug sieht also aus wie folgt:

- User erzeugt Binärdatei, zB MyFavoriteCompiler.zip.
- User benennt sie um nach dummy.bin.
- txtenc.
- User benennt um: dummy.txt nach MyFavoriteCompiler.txt.
- User lädt hoch auf öffentliche Site.
- User geht ans andere Ende der Welt und benutzt Windos-Client, der angeblich hinter Proxy, Firewall, etc. pp. "sicher" ist.<sup>1</sup>
- User lädt runter MyFavoriteCompiler.txt, winzip70.txt, txtdec.txt.
- User erzeugt ausführbares txtdec, zB mit `cpp -o txtdec txtdec.cpp`
- User benennt um... txtdec... done.

### 22.1 txtenc und txtdec

worgtsone ist etwas radikaler (oh Wunder) und benutzt statt 0..9 und a..f lieber a..p, das ist leichter zu programmieren.

worgtsone kämpft noch mit den Dateinamen<sup>2</sup>, daher gibt es 2 Programme:

- txtenc encodiert die binäre Datei dummy.bin nach dummy.txt;
- txtdec decodiert die Text-Datei dummy.txt nach dummy.bon.

<sup>1</sup>Ich verwende die Worte 'Windos' und 'sicher' nie in einem Satz.

<sup>2</sup>und sehnt sich manchmal nach PASCAL zurück...

## 22.2 Verbesserungsmöglichkeiten

Encodieren:

- Man sollte den Namen der Datei eintippen können.
- Oder: Es encodiert automatisch alle Dateien im aktuellen Verzeichnis (außer sich selbst).
- Der gegebene Algorithmus benutzt nur 16 Zeichen (plus ein paar Zeilenende-Zeichen), die encodierte Datei ist also doppelt so groß.

Wenn man 64 Zeichen (also 6 Bit) verwendet, ist die encodierte Datei nur um ein Drittel größer.

- In der ersten Zeile steht hinter // der Name der Original-Datei, in klar oder (bei neugierigen Proxies, die Textdateien scannen) verschlüsselt.
- Nur ein Programm namens `txtcodec`, das mittels Schalter `-d` oder `/d` in decode-Modus versetzt wird.

### 22.3 txtenc.cpp

```
// txtenc.cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>

unsigned short b;
int i = 0;                               // 68 Spalten breit

int
main ()
{
    cout << "txtenc by grm, 25.01.2006" << endl;
    cout << "Encodes dummy.bin to dummy.txt" << endl;
    ifstream inf ("dummy.bin");
    ofstream ouf ("dummy.txt");
    while (!inf.eof ())
    {
        b = inf.get ();
        if (i++ > 34)
        {
            ouf << endl;
            i = 0;
        }
        if (!inf.eof ())
        {
            ouf << (char) (b / 16 + 'a') << (char) (b % 16 + 'a');
        }
    }
}
```



## 23 Referat Staubtrocken

### 23.1 Zuweisungsoperatoren

Wir kennen das Zuweisungszeichen =.

Wir kennen auch die Operatoren + - \* / %

Wir müssen manchmal schreiben  $a = a + b$ .

Wir möchten das a aus Faulheit nicht zweimal hinschreiben ( $a = + b$ , aber so wird uneindeutig bis falsch).

Also schreiben wir  $a += b$ . Das ist in C und C++ okay.

Übung Zuweisung1

### 23.2 Prä- und Post-In- und -dekrement

++ erhöht um 1, -- erniedrigt um 1.

Wenns davor steht, wird erst erhöht/erniedrigt, und dann die Anweisung ausgeführt.

Wenns danach steht, wird erst die Anweisung ausgeführt, und dann erhöht/erniedrigt.

**Beispiel:**

```
...
    int a, b;
    a = 1; b = a++ * 2; cout << a << " " << b << endl;
    a = 1; b = ++a * 2; cout << a << " " << b << endl;
...
***** Sun Feb 26 13:40:00 EST 2006 ***** c++ start *****
2 2
2 4
***** Sun Feb 26 13:40:00 EST 2006 ***** c++ end *****
```

#### 23.2.1 Musterlösung zu Prä-Post-Zuweisung-Übung

2-1 - 3-4 - 4-7 - 4-196 - 201-197 - 200-397 - 5-398 - 13-398

### 23.3 Vergleichsoperatoren

Jeder Befehl in C++ hat einen Rückgabewert. Beispiel:

```
#include <iostream>
int main () {
    int i = 2;
    cout << 2 << (i = 2) << (i = i = 2) << (i == 2) << (i + 2) << endl;
}
***** Sun Feb 26 13:50:30 EST 2006 ***** c++ start *****
22214
***** Sun Feb 26 13:50:30 EST 2006 ***** c++ end *****
```

Interessant: Der Vergleich gibt 1 oder 0 zurück.

## 23.4 Auswertung Boolescher Ausdrücke

Boolesche Ausdrücke werden von links nach rechts ausgewertet.

Die Auswertung wird abgebrochen, sobald das Ergebnis klar ist.

**Regel: Benutze keine ++, -- oder Zuweisungen in Bedingungen!!!**

```
...
  i = 1; if (i++ || i--) cout << i << endl;
...
***** Sun Feb 26 13:55:59 EST 2006 ***** c++ start *****
2
***** Sun Feb 26 13:55:59 EST 2006 ***** c++ end *****
```

### 23.4.1 Musterlösung Auswertung

1 2

0 0

2 1

2 -1

0 -1

3 -1

## 23.5 Ternärer Operator ?:

Hallo C++! Wenn Bedingung erfüllt ist, dann weise wert\_true zu, sonst wert\_falsch.

```
i = (bedingung) ? wert_true : wert_falsch ;
```

zum Beispiel

```
i = (j==4) ? 4*i+3 : sqrt(k-17) ;
```

## 23.6 Übungen

### 23.6.1 Zuweisung1

a um 1 erhöhen

a um 5 erhöhen

a mit 17 multiplizieren

a durch 19 teilen

rest von a / b herausfinden.

### 23.6.2 Prä Post Zuweisung

```
a=2; b=1;
b=a+++a;
b+=a++;
b*=a*b;
a=++a+b++;
b=--a+b;
a+=++a-b++;
a=(a++)+(++a);
```

a

b

### 23.6.3 Auswertung

```
a=0; b=0; if (a++||++b)b++;
a=-1; b=-1; if (a++||++b)b++;
a=1; b=0; if (a++||++b)b++;
a=0; b=0; if (++a&&--b)a++;
a=-1; b=-1; if (++a&&--b)a++;
a=1; b=0; if (++a&&--b)a++;
```

a

b

## 24 Besenkammer

### 24.1 Definition

In einer Besenkammer kann man hochinteressante Sachen finden, die sich hervorragend für spezialisierte Routineaufgaben eignen, wie zB Aufwischen, Zusammenkehren, Staubsaugen.

In diesem Kapitel finden wir Struct, Enum und Typedef.

### 24.2 struct – Datensatz

Rechner können prima große Mengen gleichartiger Daten verarbeiten.

Leider liegen die meisten Daten in etwas komplexerer Form vor, z.B. die Zeile aus einem Meßprotokoll aus dem Motoren-Versuchslabor:

```
timestamp           Luftt.  Sensor1 Sensor2 Sensor3 ...  
01.01.2003,10:00:00  20.00   73.49   54.87   93.78   ...
```

Die Zeile ist aber recht aufgeräumt, die anderen Zeilen sehen praktisch genauso aus. Will sagen: Die Werte sind verschieden, aber **die Datentypen sind gleich**.

### 24.2.1 Übung Struct

Ordnen Sie die numerierten Kommentare den passenden Zeilen zu, indem Sie die Kommentarnummer zu der Zeile schreiben.

1. Hallo C++, ich möchte ein Struct.
2. Jetzt bin ich fertig mit Struct definieren.
3. Hier gebe ich die Temperatur von Sensor3 in der 2. Zeile aus. Die Variable \_\_\_\_ hat dabei den Wert \_\_\_\_\_.
4. Hier verschönere ich die auszugebende Tabelle.
5. Hier fälsche ich die Lufttemperatur.
6. Hier setze ich die Lufttemperatur in der 4. Zeile. Die Variable \_\_\_\_ hat dabei den Wert \_\_\_\_\_.
7. Manche Werte waren nicht wie spezifiziert, deshalb habe ich sie hier (c) geändert, und hier (d) sieht man die Änderung.
8. Das erste Feld ist ein `int` namens `secs`, das zweite ist ein `float` namens `lufttemp`, und dann noch 5 floats namens `sensor[i]`.
9. Ich möchte eine Tabelle mit 9 Zeilen anlegen, die Zeilen heißen `meineZeile[0]` bis `meineZeile[8]`.
10. Mein Struct heißt Datenzeile, weil .....
11. Hier stehen gleiche Datentypen untereinander, deshalb kann ich ein Feld nehmen. (Einkreisen)
12. Hier stehen gleiche Datentypen nebeneinander, deshalb kann ich ein Feld nehmen. (Einkreisen)
13. Ich muß erst den Spaltenaufbau definieren, bevor ich vieleviele Zeilen dieses Aufbaues bestellen kann. Das mache ich von hier (a) bis hier (b).
14. Wenn ich nur eine Zeile hätte habe wollen, hätte ich hier kein Feld definiert.

```
#include <iostream>

struct datenzeile
{
    int secs;
    float lufttemp;
    float temp[5];
};

datenzeile meineZeile[9];
int i, j;

int
main ()
```

```

{
  srand (time (NULL));
  int secs = 1;
  for (i = 0; i < 9; i++) {
    meineZeile[i].secs = secs++;
    meineZeile[i].lufttemp = 20 + (rand () % 10) / 10.0 + 0.03;
    for (j = 0; j < 5; j++) {
      meineZeile[i].temp[j] = 60 + (rand () % 200) / 10.0 + 0.03;
    }
  }
}

meineZeile[8] = meineZeile[5];

cout << "secs  luftt  sens01 sens02 sens03 sens04 sens05" << endl;
for (i = 0; i < 9; i++) {
  cout << meineZeile[i].secs << ".000";
  cout << "  " << meineZeile[i].lufttemp;
  for (j = 0; j < 5; j++) {
    cout << "  " << meineZeile[i].temp[j];
  }
  cout << endl;
}
}

```

```

***** Sun Mar 26 22:11:08 EST 2006 ***** c++ start *****
secs  luftt  sens01 sens02 sens03 sens04 sens05
1.000  20.23  76.03  78.93  68.63  62.43  66.93
2.000  20.63  71.73  66.93  75.23  65.23  73.13
3.000  20.23  60.03  67.43  75.33  69.33  61.53
4.000  20.33  69.13  65.83  64.93  75.43  78.33
5.000  20.13  69.03  78.93  75.63  66.13  67.33
6.000  20.43  74.33  63.33  75.53  78.23  60.93
7.000  20.43  60.83  72.63  69.43  76.03  73.03
8.000  20.73  76.43  73.13  60.33  67.03  77.63
6.000  20.43  74.33  63.33  75.53  78.23  60.93
***** Sun Mar 26 22:11:08 EST 2006 ***** c++  end  *****

```

### 24.3 enum – Aufzählung [optional]

Mit Enum kann man Integer-Konstanten Pseudo-Namen zuweisen, zB Wochentagen oder Monaten.

```
#include <iostream>

enum Wochentag { Mo, Di, Mi, Do, Fr, Sa, So }; // erlaubt von 0 bis 6
enum Monat
{ Jan = 1, Feb, Mrz, Apr, Mai, Jun, Jul = 10, Aug, Sep, Okt, Nov, Dez };

int main () {
    Wochentag wochentag = Mo;
    cout << wochentag << endl;
    wochentag = (Wochentag) (Di + Do + Do + Di);
    cout << wochentag << endl;
    cout << Jan << Feb << Mrz << Apr << Mai << Jun << Jul << Aug << Sep << Okt
        << Nov << Dez << endl;
}
***** Wed Apr 26 11:49:01 EDT 2006 ***** c++ start *****
0
8
123456101112131415
***** Wed Apr 26 11:49:01 EDT 2006 ***** c++ end *****
```

### 24.4 typedef – Typdefinition [optional]

Sie dürfen Datentypen umbenennen wie folgt: typedef alterNameUndGGFDefinition NeuerName

```
struct Adresse {
    string strasseNr;
    string plzOrt;
};
typedef unsigned long long int Tmyui; // ein T davor ist guter Brauch
typedef struct Adresse TmyAdresse;
typedef struct Adresse2 {
    string strasse;
    string nr;
    Tmyui plz;
    string ort;
} TmyAdresse2;
```

## 24.5 Übung struct : Rechnungsverwaltung

Erzeugen Sie 1000 Artikel per Zufallswert wie folgt:

struct Artikel : Nummer Bezeichnung Beschreibung Einzelpreis

Erzeugen Sie anschließend auf dem Bildschirm 10 Rechnungen wie diese:

```

===== Rechnung 00004 =====
                        Rodgau, 02.04.2006

Pos      Stck  Beschr.                EP      Preis
-----
 01      20    hdfvvhgkjlkfajd kjdqpjdv dvcv nq      123,00  2460,00
 02      13    jjöök jhpoj poh poijhphjhp          2,00    26,00
 ...
 10       9    lfifbvjbfbv jsdh fbjkdfjkbösdfölsdfböl 9000,00 81000,00
-----
Gesamt, Netto                                99865,14
MWSt (16%)                                  1598,45

Gesamt, Brutto                                101463,59

===== Rechnung 00005 =====

```

## 24.6 Übung enum : Attributverwaltung

Sie wollen in einer völlig aus der Mode gekommenen Entwicklungsumgebung einen bunten DOS-Bildschirm bedienen, der Zeichen heller, blinkend und in einer von 8 Vordergrundfarben und 8 Hintergrundfarben darstellen kann.

Dazu müssen Sie je ein Zeichen mit `cout` ausgeben und dann das passende Attribut mit `aout` ausgeben.

Das Attribut hat 8 Bit, durchnumeriert von Bit7 bis Bit0.

Wenn Bit7 gesetzt ist, blinkt das Zeichen.

Wenn Bit6 gesetzt ist, ist es heller.

Bit5-Bit3 enthalten den Bitcode für die Vordergrundfarbe, Bit2-Bit0 den für die Hintergrundfarbe.

000 Schwarz – 001 Blau – 010 Grün – 011 Cyan – 100 Rot – 101 Magenta – 110 Orange – 111 Hellgrau

Schreiben Sie Enums, so daß die folgende Programmzeilen funktionieren wie gedacht:

```

cout << "a";
aout << normal + VSchwarz + HOrange;
cout << "b";
aout << blink + heller + VOrange + HGrün;

```

## 25 Casts - Datentyp-Wechsel

Casts (Datentypwechsel) entstehen beim Zuweisen oder beim Ausgeben. Sie können zu sehr interessanten Effekten führen.

```
#include <iostream>
int main () {
    int i = 33 - 256;
    cout << i;
    cout << (char) i;
    cout << (float) i;
    cout << (unsigned int) i;
    cout << (unsigned short) (i);
    cout << (float) (int) (char) (unsigned short) i;
    cout << (char) (float) (int) (unsigned short) i;
    float f = (char) i;
    cout << f;
    i = f;
    cout << i << endl;
}
casts.cpp: In function int main():
casts.cpp:16: warning: assignment to int from float
***** Wed Mar  8 21:09:06 EST 2006 ***** c++ start *****
-223 ! -223 4294967073 65313 33 ! 33 33
***** Wed Mar  8 21:09:06 EST 2006 ***** c++ end *****
```

**Ein cast ist ein Datentyp, der in Klammern vor einem Wert steht.  
Eine Compilerwarnung gibt es nur bei Zuweisung von int an float.**

### 25.1 Übung : ASCII-Tabelle

Geben Sie die Zahlen von 33 bis 255 aus - zuerst als Zahl, dann als char.

32	33 !	34 "	35 #	36 \$	37 %	38 &	39 '
40 (	41 )	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [	92 \	93 ]	94 ^	95 _
96	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124	125 }	126 ~	127

**26 iomanip2 : dec oct hex right left showbase noshowbase  
(to be written)**

**27 strings (to be written)**

## 28 Pointer (Zeiger)

### 28.1 Wozu ist das gut?

Zeiger werden bei Feldern, Objekten, Dateilesen und -schreiben und bei Funktionen verwendet.

Sie machen Programme schneller und sparen Speicher.

### 28.2 Was ist das?

Ein Zeiger ist wie eine normale Variable,

d.h. er sitzt an einer bestimmten Stelle im RAM und hat einen bestimmten Wert.

**aber er zeigt immer auf die Adresse einer anderen Variable.**

Wenn er letzteres nicht tut, zB weil er nicht initialisiert wurde, zeigt er wild in den Speicher und führt höchstwahrscheinlich zum Programmabsturz.

**Beispiel:** Ein Zeiger auf die Variable x.

```
int *px;
```

x ist eine Integer-Variable. Zeiger erhalten ein \* (weil sie Zeiger sind) und ein p (für Pointer).

### 28.3 Operatoren

**& liefert die Adresse.**

**\* liefert den Wert, der in einer Adresse liegt.**

Eselsbrücke:

**Ampersand liefert die Addresse.**

**Stern liefert den Wert, der in einer Adresse liegt.**

Noch'n Beispiel

```
...
int a = 99;           // 1
int *pa1;           // 2
pa1 = &a;           // 3
int *pa2 = &a;      // 4
*pa2 = 50;          // 5
cout << "&a = " << &a << endl;
cout << "&*pa1 = " << &*pa1 << endl;
cout << "&*pa2 = " << &*pa2 << endl;
cout << "&pa1 = " << &pa1 << endl;
cout << "&pa2 = " << &pa2 << endl;
cout << "a = " << a << endl;
```

```

...
***** Sun Mar 12 22:34:14 EST 2006 ***** c++ start *****
&a = 0xbffffae4
&*pa1 = 0x8049824
&*pa2 = 0xbffffae4
&pa1 = 0xbffffae0
&pa2 = 0xbffffadc
a = 50
***** Sun Mar 12 22:34:14 EST 2006 ***** c++ end *****

```

Zeile 1: Variable a wird angelegt und mit 99 belegt. Sie sitzt an einer unbekanntem Adresse im Speicher.

Zeile 2: Ein Integer-Zeiger wird angelegt, aber nicht initialisiert. Das ist böse.

Zeile 3: pa1 wird eine Adresse zugewiesen. Nun ist er gut.

Zeile 4: Noch ein Zeiger auf a, im selben Moment deklariert und initialisiert.

Zeile 5: Lies: "Der Inhalt von zeiger pa2 wird auf 90 gesetzt."

**Übung** Legen Sie eine Tabelle an mit den Spaltenköpfen Zeile, Deklaration, Initialisierung, Var.Name, Wert, Adresse, Zuweisung.

Füllen Sie sie für Zeile 1 bis 5 aus.

Legen Sie noch eine Tabelle an, die anzeigt, welche Variable welchen Wert hat und wo sie im Speicher sitzt.

**Noch ne Übung** Kommentieren Sie jede Zeile, und geben Sie die Ausgabe an.

```

#include <iostream>

int main () {
    int x, *z;
    x = 10;
    cout << "x = " << x << endl;
    z = &x;
    cout << " z = " << z << endl;
    *z = 20;
    cout << "x = " << x << endl;
    cout << "z = " << z << endl;
    x = 30;
    cout << "x = " << x << endl;
    cout << "&x = " << &x << endl;
    cout << "z = " << z << endl;
    cout << "&z = " << &z << endl;
    cout << "*z = " << *z << endl;
}

```

## 28.4 Folie - Übungsblatt

### Noch'n Beispiel

```

...
int a = 99;           // 1
int *pa1;            // 2
pa1 = &a;            // 3
int *pa2 = &a;       // 4
*pa2 = 50;           // 5
cout << "&a = " << &a << endl;
cout << "&*pa1 = " << &*pa1 << endl;
cout << "&*pa2 = " << &*pa2 << endl;
cout << "&pa1 = " << &pa1 << endl;
cout << "&pa2 = " << &pa2 << endl;
cout << "a = " << a << endl;

```

```

...
***** Sun Mar 12 22:34:14 EST 2006 ***** c++ start *****
&a = 0xbffffae4
&*pa1 = 0x8049824
&*pa2 = 0xbffffae4
&pa1 = 0xbffffae0
&pa2 = 0xbffffadc
a = 50
***** Sun Mar 12 22:34:14 EST 2006 ***** c++ end *****

```

**Noch ne Übung** Kommentieren Sie jede Zeile, und geben Sie die Ausgabe an.

```

#include <iostream>

int main () {
    int x, *z;
    x = 10;
    cout << "x = " << x << endl;
    z = &x;
    cout << " z = " << z << endl;
    *z = 20;
    cout << "x = " << x << endl;
    cout << "z = " << z << endl;
    x = 30;
    cout << "x = " << x << endl;
    cout << "&x = " << &x << endl;
    cout << "z = " << z << endl;
    cout << "&z = " << &z << endl;
    cout << "*z = " << *z << endl;
}

```

## 29 Funktionen mit Adressübergabe (Call by reference)

Funktionen sind definitionsgemäß so gebaut, daß sie **einen** Wert zurückgeben.

Manchmal möchte man aber an mehreren Variablen gleichzeitig gebastelt haben, zB beim Vertauschen zweier Variablen.

C++ bietet eine Lösung: Man kann einer Funktion die **Adressen** von Variablen übergeben (statt der Werte). Man kann das alles sogar wild in der Parameterliste mischen.

### 29.1 Beispiel: Eine Variable um 1 erhöhen

```
#include <iostream>

void inc (int &i) {
    i++;
}

int main () {
    int i = 9;
    inc (i);
    cout << "i = " << i << endl;
}
```

### 29.2 Vorteile

Die Funktion weiß, daß sie eine Adresse bekommt<sup>3</sup>, und kann mit der Variable arbeiten wie mit jeder anderen auch.

Für eine Funktion `vertausche(int, int)` ist das wie gemacht.

Zudem muß der Wert nicht großartig über den Stack übergeben und in eine anzulegende Variablen kopiert werden. Falls oftmals und/oder zahlreiche Variablen übergeben werden, spart das Zeit.

### 29.3 Nachteile

Das aufrufende Programm weiß leider nicht, daß es seine Variablen zum Bearbeiten freigibt: man sieht es im Funktionsaufruf einfach nicht.

Spaßvögeln ist Tür und Tor geöffnet:

```
void inc (int &i) {
    if (rand()==136482736) i=0;
    else i++;
}
```

### 29.4 Übung

Schreiben Sie eine Funktion `vertausche`.

<sup>3</sup>naja – eigentlich weiß es der Compiler, wenn er das `&` sieht, und sorgt dafür.

## 29.5 Übergeben von Feldern by reference [optional]

```
...
int a[9]; cout << "a = " << a << endl;
...
a = 0xbffffacc
...
```

Überraschung: in a steckt die Basisadresse des Feldes.

Damit können wir komplette Felder zum Sortieren übergeben. Wir sollten aber den niedrigsten und den höchsten Index dazulegen.

```
...
void bubble(int a[], int l, int r) { // Übergib
    ... // 1) eine Adresse, die als Feld-Basisadresse interpretiert
        // werden soll
} // 2) und 3) linke und rechte Grenze

int main () {
    int f[87436];
    ...
    bubble(f, 0, 87435); // rufe auf mit Basisadresse-Pointer und 2 Werten
}
```

### 29.5.1 Vorteile

Man kann auf diese Weise wunderbar und äußerst effektiv das schnellste Sortierverfahren (QuickSort) rekursiv schreiben.

### 29.5.2 Nachteile

Man kann in Speicherbereichen, die gar keine richtigen Daten enthalten, herumschreiben, und wird – je nachdem – durch einen CoreDump oder ein BSOD aufgehalten.

Ein Pointer auf ein Feld ist nur eine Möglichkeit, Variablen als Adresse zu übergeben – man übergibt auch gerne C-Strings oder Videostreams. Ein Fehler in einer der System-Bibliotheken ermöglicht in diesem Fall häufig das Ausführen von Code unter dem Benutzernamen des Bibliotheks-Besitzers, zB *root*, *system*, *bin*, *disk* oder *Administrator*.

Teil IV

## 4. Quartal

## 30 Objekte etc.

### 30.1 Das Weltbild der objektorientierten Programmierung (OOP)

Moderne Programme bilden Objekte der realen Welt (Zahlen, Schüler, Filme, ...) ab. Sie lassen sich besser verwalten und einfacher handhaben.

Unsere bisherigen Datenstrukturen schaffen das nur eingeschränkt und treten sich mit ihren Funktionen und Zeigern bisweilen gegenseitig auf die Zehen.

OOP lenkt den Fokus auf die modellierten Objekte.

#### Vorteile der OOP:

- **Alles, was ein Objekt hat und kann, wandert in seine Klasse.**
- **Die Klasse bietet eine einheitliche Schnittstelle nach außen.**
- **Falls Änderungen erforderlich werden, werden sie in der Klasse gemacht, und fertig.**
- **Gut geschriebene und dokumentierte Klassen können beliebig oft wiederverwendet werden.**

### 30.2 Allgemeines Vorgehen

- Herausfinden, welche nützlichen Klassen wo auf dem Rechner vorhanden sind.
- Herausfinden, wie man sie verwendet, und ggf. ihre Eigenschaften erweitert.
- Ins Hauptprogramm einbinden, instanzieren, benutzen. Dabei **nicht** darüber nachdenken, wie sie intern funktionieren.

#### 30.2.1 Beispiel: Schüler

Schreiben Sie einige Sachen auf, die ein Schüler über sich weiß, und einige Sachen, die er kann.

Lösung: Vorname, Nachname, Geburtsdatum, Handynr, Wohnort. kichern(), schwaetzen(), mitPapierRascheln(), zuSpaetKommen(Minute m), unterrichtFolgen().

### 30.3 Schreibweise

- **Schreiben Sie Klassennamen groß.**
- **Schreiben Sie Objekte, Daten und Methoden klein.**
- **Schreiben Sie bei einem neuen Wort einen Großbuchstaben.**

### 30.3.1 Beispiel: Fenster

Schreiben Sie Sachen und Datentypen auf, die jedes Fenster über sich weiß. Schreiben Sie auf, was es kann, und die Parameter, die es dazu braucht.

Hinweis: Eine dimension ist ein Paar aus zwei Integer-Zahlen.

Lösung:

- string Titel,
- dimension Position,
- dimension Größe,
- bool resizable;
- void setTitle(string neuerTitel),
- string getTitle(),
- dimension getPosition(),
- void setPosition(dimension d),
- get / set Groesse(),
- int getFlaecheInPixel(),
- dimension getRechteUntereEcke(),
- neuesFenster(dimension position, dimension groesse, bool resizable).

## 30.4 Objekte

Von jeder Klasse (Schablone, Bauplan) kann ich beliebig viele Objekte (Schüler, Fenster, Häuser) instanzieren.

Normalerweise nimmt man die höchstentwickelten, schlauesten Klassen. Diese haben bereits viel geerbt und sind durch eigene Methoden (Know-How) noch schlauer geworden.

Sie stehen normalerweise ganz unten in der Objekthierarchie. Die Objekthierarchie (Vererbungsbeziehungen) werden normalerweise als ein nach unten wachsender Baum dargestellt, so wie ein Verzeichnisbaum auf dem Rechner.

### 30.5 Arbeitsblatt Klassen

Notieren Sie, welcher Satzanfang zu welchem Satzende gehört.

1. Eine Klasse ...
2. Ein Objekt ...
3. Eine Objekthierarchie ...
4. Vererbung ...
5. Spezialisierung ...
6. Generalisierung ...
7. Abstraktion ...

- 
1. ist eine Instanz einer Klasse.
  2. ist z.B., wenn ich die Klassen `Knopf` und `CheckBox` von der Klasse `ClickableObject` ableite, denn beide müssen Mausklicks empfangen und verarbeiten.
  3. ist eine Sammlung von Daten und Methoden.
  4. ist, wenn Klassen von schlaun Klassen erben und durch zusätzliche Mitglieder noch schlauer werden.
  5. gibt eine verpflichtende Sammlung zu belegender Datenfelder.
  6. erfordert das Aufsetzen einiger neuer `get-` und `set-`Methoden.
  7. ist, wenn neue, sehr schlaue Klassen nicht mehr zu allgemeineren Aufgaben eingesetzt werden können.
  8. ist, wenn man sich Richtung Objekt-Hierarchie-Wurzel bewegt.
  9. wirft die Frage auf: Was weiß die Klasse über sich selbst?
  10. ist das Entwerfen neuer Klassen.
  11. ist z.B., wenn ich nach einem Bauplan 5 gleiche Wohnhäuser in der Seestr. 13-17 baue.
  12. ist eine Art Schaltplan und zeigt, wie ein Ding aufgebaut ist.
  13. ist z.B., wenn ich nach einem Bauplan 6 Doppelhaushälften baue: eine mit Eckturm, eine mit Flachdach, vier mit Doppelgarage, zwei mit Tiefgarage.
  14. ist, wenn man sich von der Objekt-Hierarchie-Wurzel wegbewegt.
  15. ist das schwierige Herausfinden der Member einer neu zu erstellenden Klasse.
  16. umfaßt die Frage: Was kann die Klasse?
  17. ist eine Art Verhaltensmuster und zeigt, was ein Ding kann.
  18. definiert sich durch ihre Mitglieder (values und functions).
  19. ist z.B., wenn ich die Klassen `Affe`, `Mensch` und `Wal` von der Klasse `Saeugetier` ableite, denn alle Säugetiere können atmen, essen und sich Geschlechtlich Fortpflanzen.

## 31 Beispiel: Rationale Zahlen (Brüche)

Wir wollen eine Klasse für Brüche schreiben und sie anschließend auch benutzen.

**Übung:** Welche Fragen stellen sich?

**Lösung:**

- Was weiß eine Rationale Zahl über sich selbst? — Zähler, Nenner, istDefiniert.
- Was kann eine Rationale Zahl? — Entstehen, kürzen, eineRationaleZahlDazuaddieren, -Abziehen, -Multiplizieren, -Teilen.
- Was soll ich nur erben? — Gar nichts.
- Wie schreibt man das in C++?
- Wie benutzt man das?
- Welchen Zugriff erlaube ich von außen? — getZaehler, getNenner, getDezimal.

**31.1** Rational.hpp

```

class Rational {
private :
    int zaehler;
    int nenner;
    int ggt(int, int);
    void kuerzen();

public :
    Rational (int z, int n);
    Rational();
    void addiereRational(Rational r);
    void subtrahiereRational(Rational r);
    void multipliziereRational(Rational r);
    void dividiereDurchRational(Rational r);
    float getDezimal();
};

#include "Rational.cpp"

```

**31.2** Rational.hpp

```

Rational::Rational(int z=0, int n=0) {
    zaehler=z; nenner=n; kuerzen();
}

Rational::Rational() {
    Rational(1,0);
}

int Rational::ggt(int i, int j) {
    if (j>i) return ggt(j,i);
    if (i % j == 0) return j;
    return (ggt (i-j, j));
}

void Rational::kuerzen() {
    int h=ggt(zaehler, nenner);
    zaehler /= h; nenner /= h;
}

float Rational::getDezimal() { return (1.0); }

```

**31.3** main.cpp

```

#include "Rational.hpp"
#include <iostream>

```

```
int main () {
    Rational r1 (4, 5);
    cout << r1.getDezimal () << endl;
}
***** Sun Apr 16 11:57:58 EDT 2006 ***** c++ start *****
1
***** Sun Apr 16 11:57:58 EDT 2006 ***** c++ end *****
```

### 31.4 Fragen

1. Wie definiert man in C++ eine Klasse?
2. Was bedeutet `private:` ?
3. Was bedeutet `public:` ?
4. Wie werden die Dateien aufgeteilt?
5. Wie bindet man Dateien ein, die im gleichen Verzeichnis liegen?
6. Wie instanziiert man eine Klasse?
7. Wie instanziiert man ein Objekt?
8. Wie greift man auf ein Objekt zu?
9. Welche Methoden im obigen Beispiel sind fertig, welche nicht? Bitte begründen.
10. Welche Änderungen sind noch nötig, damit die Klasse `Rational` funktioniert wie gewünscht? Nennen und schreiben Sie sie.
11. Schreiben Sie eine Methode `talkAboutYourself()`.
12. Schreiben Sie ein Demonstrationsprogramm, das die Klasse `Rational` vorführt.
13. Der Chef wünscht, daß die Klasse geschwätzig (G) wird, zB ansagt, wenn sie durch etwas kürzt, wenn sie einen undefinierten Bruch hat oder sonst etwas Schreckliches passiert.  
Schreiben Sie die Klasse `GRational`.

### 31.5 ... und die Antworten (partly to be written)

1

2 Darf man von außen nicht drauf zugreifen.

3 Darf man von außen drauf zugreifen. DAS Mittel zur Datenkapselung.

4 eine \*.hpp, die Member (Daten und Methoden) deklariert, und eine \*.cpp, die die Methoden definiert.

5 mit Anführungszeichen.

6 gar net.

7 naja — bis jetzt durch `Klassenname myKlasse(...);` .8 Über seine `public` Methoden. Syntax: `myKlasse.myMethod(...);`.

9 unfertig:

alle: sollten meckern wenn etwas Schreckliches passiert — s. Frage 13.

Rational() - sollte Fehler ausgeben

Rational(int) - sollte Rational(int, 1) aufrufen

getDezimal - sollte zaehler/nenner zurückgeben

10

zu schreiben: `addi`, `subtrahi`, `multi`, `divi`, `getZaehler`, `getNenner`.So. Und was kann ein Bruch, der einen anderen Bruch erhält und dazu den Befehl: `addiere?`- Er kann ihn **zu sich selber und niemand sonst** addieren.

11

```
void Rational::talkAboutYourself() { cout <<
    "Ich bin ein Bruch! Ich kann aus ein oder zwei Integer entstehen, \n
    andere Brüche mit mir + - * / und mich dezimal ausgeben. \n
    Außerdem meckere ich über cerr, wenn ich nicht definiert bin. \n"; }
```

12

```
Rational zufallsBruch {
    srand(...);
    return Rational (rand(...), rand(...) + 1);    // :>)
}

int main () {
    .....
}
```

## 32 worgtsone's Note

Uff. Somit können wir Klassen anlegen und benutzen. Darauf kann man alles andere aufbauen, denn ich halte diese Basis für tragfähig.

Comments welcome.

Ansonsten weiß ich, daß 90 Seiten zuviel für 2 Halbjahre sind — aber wo soll ich nur anfangen zu streichen???

## 33 Wir legen uns ein Lexikon an

OOP verwendet viele merkwürdige Wörter. Wir müssen ein Extra-Blatt Papier hernehmen und darauf Sachen festhalten wie

**Vererbung** : ist wenn eine Klasse von einer Oberklasse erbt. Die neue Klasse ist spezialisierter als die Oberklasse.

**Abstraktion** : ist, wenn man ein komplexes System modellieren will und sich überlegt, welche Klassen man dazu braucht und was die wissen und können müssen.

...

## 34 datenkapselung: private und public (to be written)

## 35 konstruktor, destruktorkon (to be written)

## 36 vererbung (to be written)

## 37 verwendung (to be written)

## 38 Wir suchen nützliche Klassen und benutzen sie

Schaffen wir zeitlich zwar nicht, wäre aber wünschenswert.

Dev-C++ bringt eine SimpleWindow.cpp mit, deren Main in C++ geschrieben ist. Ausgedruckt hab ich sie schon, aber bis zum Verstehen ist es offensichtlich noch weit.....

## 39 Dynamisches Allokieren – (NEW und DELETE)

Wie wir im Kapitel Pointer gesehen haben, kann man in C++ Felder sehr bequem (wenn auch fehlerträchtig) durch Zeiger verwalten.

Solange alle Code-Teile, d.h. `main()` und die aufgerufenen Funktionen, wissen, was hinter dem Pointer steckt, läuft das prima. Man kann auf diese Weise hochkomplexe **Objekte** verwalten<sup>4</sup>. Beispiele:

```
Schüler: String Nachname, String Vorname, int Nummer, String Handy,
         Abschluss AngestrebterAbschluss, ...
```

```
Fenster: Point linkeObereEcke, Point rechteUntereEcke, Style style,
         Color titelbalkenFarbe, ...
```

Bei diesen Beispielen wird klar:

**Ich weiß nicht im voraus, wieviele Objekte (Schüler, Fenster, ...) ich werde verwalten müssen.**

Wir möchten also einen Befehl für "Ich brauch noch einen Schüler." oder "Ich möchte dieses Fenster jetzt schließen und vergessen." Oder, allgemein: "Hallo C++, ich möchte einen **neuen** Zeiger auf ein (Datentyp)."

### 39.1 new

```
int *p_int = new int(8);
struct schueler {
    int nummer; string nachname; string vorname; string handy;
};

schueler *p_hans = new schueler;
(*p_hans).nummer = 71453; // gibt eine total unverständliche
                          // Fehlermeldung, wenn man die Klammern wegläßt
p_hans->nachname = "müller"; // legale, elegante Schreibweise
p_hans->vorname = "hans"; // hmmm... redundant
p_hans->handy = "0177-18367452876345";
cout << p_hans->vorname << " " << p_hans->nachname << endl;

int SIZE = 5;
int i;
int *pi = new int[SIZE]; // Array mit 5 int anlegen
for (i = 0; i < SIZE; i = i + 1) // füllen mit 0,2,4,6,8
    pi[i] = 2 * i;
for (i = 0; i < SIZE; i = i + 1) // Ausgabe von 0,2,4,6,8
    cout << pi[i] << endl;
```

<sup>4</sup>Das war schon in C möglich, aber man konnte sich dabei leicht in den Fuß schießen, zB einen Schüler als Fenster interpretieren lassen...

## 39.2 delete

```
...
delete p_int;           // ohne *
delete p_hans;
delete[] pi;           // delete [] für arrays!
```

## 39.3 Liste mit dynamischem Speicherplatz

Mit diesen Mitteln können wir unsere Objekte schön verwalten.

Schön bedeutet: so, daß nicht mehr Speicherplatz verwendet wird, als wir wirklich brauchen.

1. Zunächst brauchen wir einen Zähler, der die Anzahl der Objekte aufbewahrt. Er sollte mit 0 initialisiert werden.
2. Sodann ein Feld vom Datentyp (welcher auch immer), das unsere Objekte aufnimmt.
3. Falls ein Objekt dazukommt, erzeugen wir ein neues Feld, das um 1 größer ist als das bisherige, und kopieren unsere Objekte, inklusive dem neuen, dort hinein.
4. Alternativ speichern wir sie in einem vorübergehenden Feld, löschen das alte, eröffnen ein genügend großes unter dem alten Namen, und speichern unsere Objekte darin.
5. Das Herauswerfen von Objekten geht analog. Das Ändern geht sowieso.

Das Ganze wird interessanter, wenn wir die Objekte **einsortieren** wollen, damit sie schneller gefunden werden können.

## 39.4 Übung: Schülerverwaltung

Schreiben Sie ein Programm zur Schülerverwaltung mit den Menüpunkten (1) Schüler nach Nummer sortiert ausgeben, (2) Schüler nach Nachname sortiert ausgeben, (3) neuer Schüler, (4) Schüler löschen, (5) Härtetest (10000mal eine zufällige Aktion 1 bis 4) mit Zufallsdaten), (0) Exit.

## 39.5 Übung: Indizieren

Legen Sie zusätzlich einen Index an: in der ersten Spalte der Nachname (sortiert), in der zweiten Spalte die Schülernummer.

## 39.6 Übung: Optimieren

Bei **jedem** Anlegen oder Löschen zwei komplette Felder zu kopieren ist zeitaufwendig.

Schreiben Sie das Programm so um, daß im Bedarfsfall 1% der bisherigen Einträge dazualloziert bzw. gelöscht wird. Sie benötigen hierbei wahrscheinlich zwei Zähler: einen für die allozierten Objekte, und einen für die wirklichen Schüler.

## 40 Dynamisches Allokieren – (NEW und DELETE)

Wie wir im Kapitel Pointer gesehen haben, kann man in C++ Felder sehr bequem (wenn auch fehlerträchtig) durch Zeiger verwalten.

Solange alle Code-Teile, d.h. `main()` und die aufgerufenen Funktionen, wissen, was hinter dem Pointer steckt, läuft das prima. Man kann auf diese Weise hochkomplexe **Objekte** verwalten<sup>5</sup>. Beispiele:

```
Schüler: String Nachname, String Vorname, int Nummer, String Handy,
         Abschluss AngestrebterAbschluss, ...
```

```
Fenster: Point linkeObereEcke, Point rechteUntereEcke, Style style,
         Color titelbalkenFarbe, ...
```

Bei diesen Beispielen wird klar:

**Ich weiß nicht im voraus, wieviele Objekte (Schüler, Fenster, ...) ich werde verwalten müssen.**

Wir möchten also einen Befehl für "Ich brauch noch einen Schüler." oder "Ich möchte dieses Fenster jetzt schließen und vergessen." Oder, allgemein: "Hallo C++, ich möchte einen **neuen** Zeiger auf ein (Datentyp)."

### 40.1 new

```
int *p_int = new int(8);
struct schueler {
    int nummer; string nachname; string vorname; string handy;
};

schueler *p_hans = new schueler;
(*p_hans).nummer = 71453; // gibt eine total unverständliche
                          // Fehlermeldung, wenn man die Klammern wegläßt
p_hans->nachname = "müller"; // legale, elegante Schreibweise
p_hans->vorname = "hans"; // hmhhh... redundant
p_hans->handy = "0177-18367452876345";
cout << p_hans->vorname << " " << p_hans->nachname << endl;

int SIZE = 5;
int i;
int *pi = new int[SIZE]; // Array mit 5 int anlegen
for (i = 0; i < SIZE; i = i + 1) // füllen mit 0,2,4,6,8
    pi[i] = 2 * i;
for (i = 0; i < SIZE; i = i + 1) // Ausgabe von 0,2,4,6,8
    cout << pi[i] << endl;
```

<sup>5</sup>Das war schon in C möglich, aber man konnte sich dabei leicht in den Fuß schießen, zB einen Schüler als Fenster interpretieren lassen...

## 40.2 delete

```
...
delete p_int;           // ohne *
delete p_hans;
delete[] pi;           // delete [] für arrays!
```

## 40.3 Liste mit dynamischem Speicherplatz

Mit diesen Mitteln können wir unsere Objekte schön verwalten.

Schön bedeutet: so, daß nicht mehr Speicherplatz verwendet wird, als wir wirklich brauchen.

1. Zunächst brauchen wir einen Zähler, der die Anzahl der Objekte aufbewahrt. Er sollte mit 0 initialisiert werden.
2. Sodann ein Feld vom Datentyp (welcher auch immer), das unsere Objekte aufnimmt.
3. Falls ein Objekt dazukommt, erzeugen wir ein neues Feld, das um 1 größer ist als das bisherige, und kopieren unsere Objekte, inklusive dem neuen, dort hinein.
4. Alternativ speichern wir sie in einem vorübergehenden Feld, löschen das alte, eröffnen ein genügend großes unter dem alten Namen, und speichern unsere Objekte darin.
5. Das Herauswerfen von Objekten geht analog. Das Ändern geht sowieso.

Das Ganze wird interessanter, wenn wir die Objekte **einsortieren** wollen, damit sie schneller gefunden werden können.

## 40.4 Übung: Schülerverwaltung

Schreiben Sie ein Programm zur Schülerverwaltung mit den Menüpunkten (1) Schüler nach Nummer sortiert ausgeben, (2) Schüler nach Nachname sortiert ausgeben, (3) neuer Schüler, (4) Schüler löschen, (5) Härte-test (10000mal eine zufällige Aktion 1 bis 4) mit Zufallsdaten), (0) Exit.

## 40.5 Übung: Indizieren

Legen Sie zusätzlich einen Index an: in der ersten Spalte der Nachname (sortiert), in der zweiten Spalte die Schülernummer.

## 40.6 Übung: Optimieren

Bei **jedem** Anlegen oder Löschen zwei komplette Felder zu kopieren ist zeitaufwendig.

Schreiben Sie das Programm so um, daß im Bedarfsfall 1% der bisherigen Einträge dazualloziert bzw. gelöscht wird. Sie benötigen hierbei wahrscheinlich zwei Zähler: einen für die allozierten Objekte, und einen für die wirklichen Schüler.

**40.7 Arbeitsblatt new delete**

```

#include <iostream>
#include <string>
class Schueler
{
public:
    string name;
    int papier;
    Schueler (string n);
    Schueler (string n, int pap);
    ~Schueler ()
    {
        sprich ();
        cout << " und wurde gerade destrukturiert!" << endl;
    }
    void raschel ();
    void raschel (int laut);
    void sprich ();
};

Schueler::Schueler (string n){    name = n;    papier = 4; }

Schueler::Schueler (string n, int pap) {    name = n;    papier = pap; }

void Schueler::sprich () {    cout << name << " : " << this; }

void Schueler::raschel () {    raschel (2); }

void Schueler::raschel (int laut) {
    sprich ();
    if (papier == 0)
        cout << " und habe kein Papier zum Rascheln." << endl;
    else {
        cout << " und raschel mit Papier: raschel.";
        if (laut > 10)
            cout << " Raschel Raschel";
        if (laut > 100)
            cout << " RASCHEL RASCHEL";
        cout << endl;
    }
}

int main () {
    cout << "Unterrichtsstunde..." << endl;
    Schueler hans ("hans");
    Schueler toni ("toni", 0);
    hans.raschel ();
    toni.raschel (11);
    hans.raschel (11);
}

```

```

Schueler *mike = new Schueler ("mike");
mike->raschel (3);
Schueler *jens = new Schueler ("jens");
jens->raschel (255);
hans.raschel ();
delete mike;
Schueler *zeno=new Schueler ("Zeno");
delete zeno;
toni.raschel (255);
}

```

```

***** Wed Jun  7 20:20:47 EDT 2006 ***** c++ start *****
Unterrichtsstunde...
hans : 0xbffffad0 und raschel mit Papier: raschel.
toni : 0xbffffac4 und habe kein Papier zum Rascheln.
hans : 0xbffffad0 und raschel mit Papier: raschel. Raschel Raschel
mike : 0x804d290 und raschel mit Papier: raschel.
jens : 0x804d2a0 und raschel mit Papier: raschel. Raschel Raschel RASCHEL R
hans : 0xbffffad0 und raschel mit Papier: raschel.
mike : 0x804d290 und wurde gerade destrukтуриert!
Zeno : 0x804d290 und wurde gerade destrukтуриert!
toni : 0xbffffac4 und habe kein Papier zum Rascheln.
toni : 0xbffffac4 und wurde gerade destrukтуриert!
hans : 0xbffffad0 und wurde gerade destrukтуриert!
***** Wed Jun  7 20:20:47 EDT 2006 ***** c++ end *****

```

Nehmen Sie zu den folgenden Thesen schriftlich Stellung:

1. Die Datenkapselung ist perfekt.
2. Die Prototypen sind überladen.
3. Einer der Prototypen ist keiner und möchte deshalb inline kompiliert werden.
4. new schafft ein neues Objekt, das sich verhält wie ein ganz normales Objekt.
5. new-Objekte werden auf dem Heap und nicht im Datensegment alloziert.
6. Der obige Code gehört in eine einzige Datei.
7. delete löscht objekte nicht.
8. gelöschte objekte sind schrecklich geschwätzig.

## 41 Projektarbeit

### 41.1 class Screen

Ein DOS-Bildschirm hat 80 Spalten x 25 Zeilen, die man als sehr grobe Pixel ansteuern kann. Sinnvollerweise erfindet man eine Klasse Screen, die den Soll-Bildschirm in einem 2-dimensionalen Feld aufbewahrt und folgende Methoden kann: `clrscr()`, `isSetPoint(int p, int q)`; `setPoint(int p, int q)`; `setPoint(Point p)`; `drawLine (point1, point2)`; `redraw()`.

Dabei ist `pointX1` eine Instanz der Klasse Point. Ein Point weiß seine x- und y-Koordinaten und kann sie über `point1.x` (ungekapselt) bzw. `point1.getX()` (gekapselt) aufsagen. `redraw()` wird aufgerufen, wenn der Puffer wirklich mittels `cout` auf den Bildschirm geschrieben werden soll.

25 Pixel Höhenauflösung sind zu wenig. GTSLEHRER schlägt deshalb vor, jede Buchstabenzeile in eine obere und eine untere Hälfte aufzuteilen:

Wenn keins/oben/unten/beide gesetzt ist, wird ein " "/"/n/H angezeigt.

So kommen wir zu 50 Pixeln in der Höhe, das ist ein Vorteil.

Wir müssen aber auch jedesmal beim Ausgeben 2 "virtuelle Pixel" verknüpfen, damit der richtige Buchstabe angezeigt wird. Das 2-dimensionale Feld hat dann 80x50 Einträge.

### 41.2 Lissajous

Schreiben Sie eine Klasse Screen und projizieren Sie darauf die folgende parametrisierte Funktion:

für  $t=0$  bis genug :  $x=30+30*\cos(n*t)$  :  $y=30+30*\sin(m*t)$  : `drawPoint(x,y)`

mit  $n$  und  $m$  als natürlichen Zahlen größer 0, d.h. zwischen 1 und 10 incl.

### 41.3 Starfield

Schreiben Sie eine Klasse Screen und darauf den Screensaver StarField.

### 41.4 Snake

Schreiben Sie eine Klasse Screen und realisieren Sie Snake.