

Objektorientiertes Programmieren in C++

Aufgaben (Version 5)

Detlef Wilkening
www.wilkening-online.de
© 2021

Aufgaben für die C++ Vorlesung
FH Aachen WS 2020/21

1	Allgemeines	4
2	C++	4
3	Einführung	4
4	Microsoft Visual Studio	4
5	Typen & Variablen	4
6	Operatoren	4
6.1	Grundrechen-Arten	4
6.2	Increment-Operator.....	5
7	Kontrollstrukturen	5
7.1	Schleifen-Varianten.....	5
7.2	Teilbar?.....	5
7.3	Lesbare Zahlen 1	5
7.4	Zahlen-Liste	5
7.5	Multiplikations-Matrix 1.....	6
8	Ein- und Ausgabe	6
8.1	Summe.....	6
8.2	Lesbare Zahlen 2	6
8.3	Mittelwert und Varianz	6
8.4	Multiplikations-Matrix 2	7
8.5	Quadratwurzel 1.....	7
9	Texte	8
9.1	Hallo <Person>	8
9.2	Leerzeichen zählen 1	8
9.3	String-Analyse 1	9
9.4	Zahlen-Palindrom.....	9
10	Container & Iteratoren	10
10.1	Hallo <Personen>	10
10.2	Lesbare Zahlen 3	10
10.3	Leerzeichen zählen 2	10
10.4	String-Analyse 2.....	11
10.5	Telefonbuch 1	11
11	Typ-System und mehr	12
12	Funktionen	12
12.1	swap.....	12
12.2	Setze String-Länge	12
12.3	Fakultät.....	13
12.4	Fibonacci.....	13
12.5	Rekursive Multiplikation.....	13
12.6	Quadratwurzel-Funktion (optional)	14

12.7	Zahlen-Wandlung.....	14
12.8	Formatierung (optional).....	14
12.9	Telefonbuch 2	16
12.10	Türme von Hanoi 1 (optional)	17
13	Weiteres aus der Standard-Bibliothek	18
13.1	File-Stream	18
13.2	String-Container	18
13.3	Platten-Platz (optional)	20
13.4	Datei-Suche (optional).....	21
13.5	Lotto-Programm	22
13.6	Pi simulieren (optional).....	22
13.7	Telefonbuch 3	23
14	Klassen.....	23
14.1	Klasse date	23
14.2	Ringzähler	23
14.3	Telefonbuch 4	24
14.4	Türme von Hanoi 2 (optional).....	24
14.5	Gerüchteküche 1 (optional)	24
14.6	Tic-Tac-Toe 1 (optional).....	26
15	Präprozessor, Compiler, Linker,	26
15.1	Telefonbuch 5	26
15.2	Ring-Zähler Bibliothek	27
15.3	Türme von Hanoi 3 (optional).....	27
15.4	Gerüchteküche 2 (optional).....	27
15.5	Tic-Tac-Toe 2 (optional).....	27
16	Operator-Funktionen	27
16.1	Bruch-Klasse.....	27
16.2	Funktions-Objekte.....	27
17	Vererbung & Polymorphie.....	27
17.1	Obstkorb	27
17.2	Tic-Tac-Toe 3.....	28
17.3	Progress-Bar (optional)	28

Unter den Aufgaben befinden sich auch mehrere größere oder komplexere Programme. Im Normalfall würden wir ein Teil dieser Aufgaben als gemeinsame Übung abhalten. Sie sollen zeigen, dass Sie alle mit dem erlernten Wissen und einem Schritt-für-Schritt Vorgehen auch ohne Vorkenntnisse solche herausfordernden Aufgaben umsetzen können. Unter den aktuellen Umständen (Corona, Online-Vorlesung) sind diese Aufgaben in erster Linie für die Studenten gedacht, die entweder etwas Vorwissen haben und damit durch die normalen Aufgaben gelangweilt sind – oder für die Studenten, die ein bisschen Ehrgeiz mitbringen. Die entsprechenden Aufgaben sind als „optional“ gekennzeichnet.

1 Allgemeines

Keine Aufgaben

2 C++

Keine Aufgaben

3 Einführung

Dieses Kapitel enthält noch keine echten C++ Aufgaben.

Die Aufgaben dieses Kapitel haben mehr mit Ihrem System zu tun:

- Bringen Sie Ihre Entwicklungs-Umgebung ans Laufen.
- Erstellen Sie Projekte für alle Programme aus diesem Kapitel, compilieren Sie sie, und führen Sie sie aus.
- Erzeugen Sie Syntax-Fehler in Ihrem Code, und testen Sie, welche Fehler Ihr Compiler ausgibt.
- Untersuchen Sie, wie Ihre Entwicklungs-Umgebung die Daten auf dem Datei-System ablegt.
- Führen Sie alle Ihre erstellten Programme auch in der normalen Konsole aus.
- Wechseln Sie in Ihrer Entwicklungs-Umgebung zwischen Debug- und Release-Modus. Und testen Sie, was in Ihrer Entwicklungs-Umgebung im Debug- und Release-Modus passiert, wenn Sie das Programm mit dem fehlerhaften String-Zugriff ausführen.

4 Microsoft Visual Studio

Keine Aufgaben

5 Typen & Variablen

Keine Aufgaben

6 Operatoren

6.1 Grundrechen-Arten

Schreiben Sie ein kleines Programm, das am Anfang 2 Int-Konstanten definiert, dann in 5 Variablen das Ergebnis der Grundrechen-Arten („+“, „-“, „*“, „/“ und „%“) mit den Konstanten zwischenspeichert und am Ende die Ergebnisse mit einem passenden Text ausgibt.

6.2 Increment-Operator

Schreiben Sie ein kleines Programm, das anhand einer Int-Variablen und den entsprechenden Ausgaben den Unterschied zwischen Prä- und Post-Increment aufzeigt.

7 Kontrollstrukturen

7.1 Schleifen-Varianten

Schreiben Sie ein Programm, das dreimal hintereinander die Zahlen von 1 bis 5 aus gibt. Für die erste „1..5“ Ausgabe soll eine For-Schleife benutzt werden, für die Zweite eine While-Schleife, und für die Dritte eine Do-Schleife. Jede „1..5“ Ausgabe soll in einer eigenen Zeile stehen, und die Zahlen sollen untereinander mit einem Leerzeichen getrennt sein.

```
Ausgabe
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

7.2 Teilbar?

Schreiben Sie ein Programm, das hintereinander die Zahlen von 1 bis 10 und jeweils die Meldung ob die Zahl durch 2 bzw. durch 3 teilbar ist ausgibt. Benutzen Sie, um die Zahlen zu durchlaufen, eine For-Schleife.

```
Mögliche Ausgabe:
1 nicht-durch-2-teilbar nicht-durch-3-teilbar
2 durch-2-teilbar nicht-durch-3-teilbar
3 nicht-durch-2-teilbar durch-3-teilbar
4 durch-2-teilbar nicht-durch-3-teilbar
5 nicht-durch-2-teilbar nicht-durch-3-teilbar
...
```

7.3 Lesbare Zahlen 1

Schreiben Sie ein Programm, das hintereinander die Zahlen von 1 bis 5 als lesbaren Text aus gibt. Benutzen Sie, um die Zahlen zu durchlaufen, eine For-Schleife. Jede Zahl-Ausgabe soll in einer eigenen Zeile stehen.

```
Ausgabe:
1 -> eins
2 -> zwei
3 -> drei
4 -> vier
5 -> fuenf
```

7.4 Zahlen-Liste

Schreiben Sie ein Programm, das hintereinander die Zahlen von 1 bis 5 ausgibt. Die Zahlen sollen dabei durch einen Bindestrich, '-', getrennt werden, und die gesamte Zahlen-Liste soll

von runden Klammern umgeben sein. Benutzen Sie, um die Zahlen zu durchlaufen, eine For-Schleife.

```
Ausgabe:  
(1-2-3-4-5)
```

7.5 Multiplikations-Matrix 1

Schreiben Sie ein Programm, das zwei positive Integer-Zahlen initialisiert, und für diese (von bis inkl.) das Ein-mal-eins als Matrix ausgibt.

```
Beispiel (unter der Annahme dass eine Zahl 2 und die andere 4 ist):  
  | 2 3 4  
---+-----  
2 | 4 6 8  
3 | 6 9 12  
4 | 8 12 16
```

Hinweis – eine schönere Formatierung wäre wünschenswert, und wird in späteren Kapiteln – mit dann mehr Wissen – nachgeholt.

8 Ein- und Ausgabe

8.1 Summe

Schreiben Sie ein Programm, das zwei Zahlen einliest, und ihre Summe wieder ausgibt. Fangen Sie falsche Eingaben ab, indem Sie den Eingabe-Stream nach der Eingabe überprüfen, und im Falle eines Fehlers das Programm mit einer Fehlermeldung beenden.

8.2 Lesbare Zahlen 2

Im Prinzip eine ähnliche Aufgabe wie 7.3: schreiben Sie ein Programm, dass Benutzer-Eingaben der Ziffern 0 bis 9 als lesbaren Text ausgibt. Die Eingabe eines beliebigen anderen Zeichens soll das Programm beenden.

```
Bitte geben sie eine Ziffer ein: 2  
'2' -> zwei  
Bitte geben sie eine Ziffer ein: 7  
'7' -> sieben  
Bitte geben sie eine Ziffer ein: x  
'x' -> Ende
```

8.3 Mittelwert und Varianz

Schreiben Sie ein Programm, das den Mittelwert und die Varianz berechnet. Der Benutzer gibt die Werte nacheinander ein. Gibt er eine 0 ein, so werden Mittelwert und Varianz ausgegeben und das Programm beendet. Der Mittelwert ist die Summe der Werte geteilt durch die Anzahl. Die Varianz ist die Summe der Quadrate der Werte geteilt durch die um eins reduzierte Anzahl. Denken Sie auch bei diesem Programm an die Fehlerbehandlung.

8.4 Multiplikations-Matrix 2

Schreiben Sie ein Programm, das zwei positive Integer-Zahlen einliest (bitte an die Fehlerbehandlung denken), und für diese (von bis inkl.) das Ein-mal-eins als Matrix ausgibt.

Sorgen Sie diesmal – im Gegensatz zu Aufgabe 7.5 – für eine schöne Formatierung, z.B. wie im unteren Beispiel. Und fangen Sie zu große Eingabe-Bereiche ab, da diese nicht mehr sinnvoll am Bildschirm darstellbar sind.

Mögliche Ein- und Ausgabe:

```
Untere Grenze: 1
Obere Grenze: 100
Zu grosse Matrix
```

Mögliche Ein- und Ausgabe:

```
Untere Grenze: 2
Obere Grenze: 12
```

	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	36	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

8.5 Quadratwurzel 1

Schreiben Sie ein Programm, das die Quadratwurzel einer Fließkomma-Zahl größer-gleich 1.0 berechnet. Benutzen Sie hierbei nur die Grundrechenarten und Vergleiche, indem Sie sich der Lösung durch Intervall-Schachtelung Schritt für Schritt annähern.

Das Konzept der Intervall-Schachtelung arbeitet folgendermaßen:

- Beginnen Sie mit zwei Zahlen, von denen Sie ausgehen können, daß sie kleiner-gleich bzw. größer-gleich der Quadratwurzel sind.
- Berechnen Sie den Mittelwert der beiden Zahlen, und quadrieren Sie diesen.
- Weicht der quadrierte Mittelwert um weniger als einen Toleranzwert *epsilon* (z.B. 1.0e-5) von der Eingabe-Zahl ab, so verwenden Sie den Mittelwert als Ergebnis der Berechnung. D.h. Sie haben die Quadratwurzel gefunden.
- Ist das Quadrat des Mittelwertes größer als die Eingabe-Zahl, so verwenden Sie das untere Intervall für die nächste Iteration.
- Ist das Quadrat des Mittelwertes kleiner als die Eingabe-Zahl, so verwenden Sie das obere Intervall für die nächste Iteration.
- Führen Sie die Iteration so lange durch, bis sich der quadrierte Mittelwert bis auf den

Toleranzwert *epsilon* an die Eingabe-Zahl angenähert hat.

Geben Sie am Anfang der Berechnung den Toleranzwert *epsilon* aus.

Geben Sie vor jeder Iteration die Nummer der Iteration und das aktuelle Intervall aus.

Und geben Sie am Ende natürlich auch die Wurzel aus.

Mögliche Ein- und Ausgabe

```
Wurzel-Berechnung fuer Zahlen groesser-gleich 1.0
Zahl: 4
Berechne Wurzel aus 4 mit einem Epsilon von 0.0001
- Durchlauf 1: 1 < sqrt(4) < 4
- Durchlauf 2: 1 < sqrt(4) < 2.5
- Durchlauf 3: 1.75 < sqrt(4) < 2.5
- Durchlauf 4: 1.75 < sqrt(4) < 2.125
- Durchlauf 5: 1.9375 < sqrt(4) < 2.125
- Durchlauf 6: 1.9375 < sqrt(4) < 2.03125
- Durchlauf 7: 1.98438 < sqrt(4) < 2.03125
- Durchlauf 8: 1.98438 < sqrt(4) < 2.00781
- Durchlauf 9: 1.99609 < sqrt(4) < 2.00781
- Durchlauf 10: 1.99609 < sqrt(4) < 2.00195
- Durchlauf 11: 1.99902 < sqrt(4) < 2.00195
- Durchlauf 12: 1.99902 < sqrt(4) < 2.00049
- Durchlauf 13: 1.99976 < sqrt(4) < 2.00049
- Durchlauf 14: 1.99976 < sqrt(4) < 2.00012
- Durchlauf 15: 1.99994 < sqrt(4) < 2.00012
- Durchlauf 16: 1.99994 < sqrt(4) < 2.00003
Wurzel: 1.99998
```

9 Texte

9.1 Hallo <Person>

Schreiben Sie ein Programm, das einen kompletten Namen einliest, und diesen Namen mit Sternchen und Leerzeichen umrandet als Anrede mit „Hallo“ und „!“ ausgibt. Beispiel: der eingebene Name ist „Detlef Wilkening“, dann soll folgende Ausgabe erzeugt werden:

```
*****
*                                     *
*  Hallo Detlef Wilkening!           *
*                                     *
*****
```

9.2 Leerzeichen zählen 1

Schreiben Sie ein Programm, das einen String einliest, und die Anzahl an Leerzeichen zählt und ausgibt.

Mögliche Ein- und Ausgabe:

```
Eingabe: H all o We lt
Text "H all o We lt" -> 7 Leerzeichen
```

Hinweis – dieses Programm lässt sich noch leichter mit String-Iteratoren und Algorithmen implementieren. Darum gibt es diese Aufgabe noch einmal in einem späteren Kapitel, wenn uns Iteratoren und Algorithmen bekannt sind.

9.3 String-Analyse 1

Schreiben Sie ein Programm, das eine Zeile einliest, und ausgibt wie oft welches Zeichen in der Zeile vorkommt. Erzeugen Sie eine Ausgabe der Zeichen in der Reihenfolge ihres ersten Vorkommens im String.

```
Mögliche Ein- und Ausgabe:  
Eingabe: Hallo Welt: 123332  
'H': 1  
'a': 1  
'l': 3  
'o': 1  
' ': 2  
'W': 1  
'e': 1  
't': 1  
'.': 1  
'1': 1  
'2': 2  
'3': 3
```

Hinweis – dieses Programm lässt sich viel leichter und effizienter implementieren, wenn wir Container zur Verfügung haben, um Informationen strukturiert ablegen zu können. Da wir diese aber erst später kennen lernen werden, müssen wir mit den bis hierhin bekannten Sprach- und Bibliotheks-Elementen auskommen – aber es geht auch so. Im späteren Kapitel wird die Aufgabe dann aber erneut aufgegriffen.

9.4 Zahlen-Palindrom

Ein Zahlen-Palindrom ist eine Zahl, die von rückwärts gelesen die gleiche Zahl ergibt wie vorwärts gelesen – Beispiele: „151“, „26462“ oder „3443“.

Die Aufgabe ist, ein Programm zu schreiben, das alle Zahlen-Palindrome (und ihre Faktoren) ausgibt, die sich als Produkt zweier Zahlen im Bereich von „70“ bis „99“ (inkl.) erzeugen lassen. Hierbei soll der erste Faktor immer kleiner-gleich dem zweiten Faktor sein. Daher das Zahlen-Palindrom „7007“ soll nur als Produkt von „77*91“ auftauchen, und nicht noch zusätzlich als Produkt von „91*77“.

```
Ausgabe:  
-> 72*88 = 6336  
-> 73*99 = 7227  
-> 75*77 = 5775  
-> 77*78 = 6006  
-> 77*88 = 6776  
-> 77*91 = 7007  
-> 82*99 = 8118  
-> 88*91 = 8008  
-> 88*96 = 8448  
-> 91*99 = 9009
```

10 Container & Iteratoren

10.1 Hallo <Personen>

Schreiben Sie ein Programm, das beliebig viele komplette Namen einliest. Wird eine leere Eingabe gemacht, so gelten damit alle Namen als eingegeben. Danach sollen alle diese Namen mit Anrede und umrandet mit Sternchen und Leerzeichen ausgegeben werden. Hierbei soll sich die Umrandung am längsten Namen orientieren. Bsp: die eingegebenen Namen seien „Max“, „Johannes Wolfgang“ und „Werner“, dann soll folgende Ausgabe erzeugt werden:

```
*****
*
* Hallo Max!
* Hallo Johannes Wolfgang!
* Hallo Werner!
*
*****
```

In der ersten Version sollen die Namen in der Reihenfolge der Eingabe in der Begrüßungs-Box stehen. Schreiben Sie danach eine Version, bei der die Namen sortiert in der Begrüßungs-Box stehen.

Die Version mit sortierter Ausgabe gibt es auch noch mal als spätere Aufgabe. Nur soll dort dann kein implizit-sortierender Container genutzt werden, sondern eine explizite Sortierung durchgeführt werden.

10.2 Lesbare Zahlen 3

Im Prinzip die gleiche Aufgabe wie 8.2: schreiben Sie ein Programm, das Benutzer-Eingaben von Zahlen von 1 bis 9 als lesbaren Text ausgibt. Die Eingabe einer beliebigen anderen Zahl oder eine Fehleingabe soll das Programm beenden.

```
Bitte geben Sie eine Zahl ein: 2
-> zwei
Bitte geben Sie eine Zahl ein: 7
-> sieben
Bitte geben Sie eine Zahl ein: x
-> Ende
```

Als Unterschied zur früheren Aufgabe sollen Sie hier keine Kontrollstruktur (switch oder if) zur Abbildung der Zahlen auf Texte verwenden, sondern einen Container. Machen Sie sich klar, dass diese Lösung kürzer, performanter, und auch noch besser lesbar ist.

10.3 Leerzeichen zählen 2

Schreiben Sie ein Programm, das einen String einliest, und die Anzahl an Leerzeichen zählt und ausgibt. Im Gegensatz zur Aufgabe „Leerzeichen zählen 1“ sollen Sie hier aber Iteratoren und Algorithmen verwenden.

```
Mögliche Ein- und Ausgabe:  
Eingabe: H all o We lt  
Text "H all o We lt" -> 7 Leerzeichen
```

10.4 String-Analyse 2

Die gleiche Aufgabe wie schon früher: schreiben Sie ein Programm das eine Zeile einliest, und ausgibt wie oft welches Zeichen in der Zeile vorkommt. Erzeugen Sie eine Ausgabe der Zeichen in der Reihenfolge ihres ersten Vorkommens im String.

```
Mögliche Ein- und Ausgabe:  
Eingabe: Hallo Welt: 123332  
'H': 1  
'a': 1  
'l': 3  
'o': 1  
' ': 2  
'W': 1  
'e': 1  
't': 1  
'.': 1  
'1': 1  
'2': 2  
'3': 3
```

Im Unterschied zur früheren Aufgabe soll die Aufgabe hier mit Hilfe der STL gelöst werden. Diskutieren Sie außerdem am Schluß, welche Lösung vorzuziehen ist.

10.5 Telefonbuch 1

Schreiben Sie ein erstes ganz ganz einfaches Telefonbuch.

Das Telefonbuch soll Namen und Telefonnummern speichern – am Anfang begnügen wir uns mit einer Nummer pro Name, und es soll kein Name doppelt vorkommen (d.h. jeder Name im Telefonbuch ist ein Unikat).

Die Einträge im Telefonbuch sollen im Augenblick noch fest im Programm stehen, d.h. es können im Augenblick keine Einträge hinzugefügt, gelöscht, oder editiert werden.

Der Benutzer kann entweder einen der Befehle „end“ bzw. „list“ oder einen Namen eingeben.

- Bei „end“ soll das Programm beendet werden.
- Bei „list“ sollen alle Einträge im Telefonbuch ausgegeben werden – zuerst der Name, dann die Nummer (eine Zeile pro Eintrag). Die Einträge sollen sortiert nach dem Namen (nicht lexikalisch sortiert, sondern nach Zeichenkodierung) ausgegeben werden.
- Alle anderen Benutzer-Eingaben werden als Namen interpretiert, und im Telefonbuch gesucht. Wird der Name gefunden, so wird die Nummer ausgegeben. Wird der Name nicht gefunden, so wird eine entsprechende Meldung ausgegeben. Achtung – der Name muss exakt übereinstimmen, und es werden auch Groß- und Klein-Schreibung unterschieden.

Nach der Bearbeitung des Befehls „list“ oder eines Namens kann der Benutzer die nächste

Eingabe machen.

```
Mögliche Ein- und Ausgabe
Telefonbuch 1

Eingabe: list

6 Eintraege:
- Bernd => 0555 / 55 55 55
- Detlef => 0123 / 12 21 1
- Dietmar => 0321 / 888 222 99
- Edgar => 0111 / 11 11 1
- Edmund => 0222 / 33 22 11
- Karl => 0111 / 11 22 33

Eingabe: Max

Name "Max" ist nicht im Telefonbuch vorhanden.

Eingabe: Detlef

Detlef => 0123 / 12 21 1

Eingabe: end

Programmende
```

11 Typ-System und mehr

Keine Aufgaben

12 Funktionen

12.1 swap

Schreiben Sie eine „swap“ Funktion für int-Variablen. Die „swap“ Funktion ist die Funktion, die den Inhalt zweier übergebener Variablen austauscht, und zwar so, dass die Original-Variablen getauscht werden.

Schreiben Sie auch ein Beispiel-Programm mit sinnvoll initialisierten Variablen und einer Ausgabe die zeigt, dass die Funktion korrekt arbeitet. Und schreiben Sie sie so, dass sie mit verschiedenen Typen umgehen kann.

Hinweis – normalerweise muss man sich keine eigene Swap-Funktion schreiben, da die Standard-Bibliothek eine solche zur Verfügung stellt.

12.2 Setze String-Länge

Schreiben Sie eine Funktion, die einen String auf eine bestimmte Länge bringt.

D.h. die Funktion bekommt einen String, einen `string::size_type`, und einen `char` übergeben.

- Ist der String länger als die im `string::size_type` übergebene Länge, so wird er gekürzt,

d.h. die zuviel vorhandenen Zeichen werden abgeschnitten.

- Hat der String die korrekte Länge, passiert nichts.
- Ist der String zu kurz, so wird er um die entsprechende Anzahl von Zeichen verlängert.
Dazu wird das übergebene char n-mal an den String angehängt.

Die Änderung am String sollen sich natürlich auf die Original-Variable beziehen, und sollen daher nach dem Verlassen der Fkt noch vorhanden sein.

12.3 Fakultät

Schreiben Sie zwei Versionen der Fakultäts-Funktion. Die Fakultät ist das Produkt der natürlichen Zahlen von 1 bis n. Mathematisch sieht die Definition folgendermassen aus:

- $fak(1) ::= 1$
- $fak(n) ::= n * fak(n-1)$

Version 1 soll das Ergebnis iterativ berechnen – d.h. mit einer Schleife.

Version 2 soll das Ergebnis rekursiv berechnen – d.h. durch Aufruf von sich selber.

Schreiben Sie natürlich auch ein kleines Haupt-Programm, dass beide Funktionen für einige Int-Werte benutzt. Achtung – bedenken sie, dass die Fakultät sehr schnell wächst, und schon „14!“ nicht mehr in einen vorzeichenbehafteten 32-Bit Integer-Wert passt.

12.4 Fibonacci

Schreiben Sie eine Funktion, die die sogenannte Fibonacci-Folge berechnet. Diese ist wie folgt definiert:

- $fib[0] := 1$
- $fib[1] := 1$
- $fib[n+1] = fib[n] + fib[n-1]$

Schreiben Sie ein Programm, dass mit dieser Funktion die Fibonacci Zahlen von 0 bis zu einer einzugebenen positiven Integer-Zahl „n“ ausgibt.

Welche grundsätzlichen Wege kann man bei der Implementierung der Fibonacci-Funktion beschreiten? Welche ist für die Praxis sinnvoller?

12.5 Rekursive Multiplikation

Schreiben Sie eine Funktion „int mul(int, int)“, die zwei positive Int-Zahlen miteinander multipliziert, und das Ergebnis zurückgibt. Aber: die Funktion soll rekursiv implementiert sein, und intern nur die Addition von Int-Zahlen verwenden!

Haben Sie eine Idee, wie sie die Lösung bzgl. Performance optimieren können?

12.6 Quadratwurzel-Funktion (optional)

1. Implementieren Sie die Aufgabe „Berechnung der Quadratwurzel via Intervall-Schachtelung“ ohne Rückmeldung jeder Iteration als Funktion für den Fließkomma-Typ „double“.
2. Wenn Sie die Funktion für alle drei Fließkomma-Typen „float“, „double“ und „long double“ zur Verfügung stellen wollen – wie machen Sie das? Wie stellen Sie sicher, dass der Benutzer ohne viel Nachdenken immer die korrekte Variante nimmt?
3. Können Sie die Funktion jetzt auch anders implementieren? Wenn ja, wie? Implementieren Sie alternativ auch diese Lösung.

12.7 Zahlen-Wandlung

Schreiben Sie eine Funktion „as_string“, die eine „unsigned int“ Zahl in einen String zu einer übergebenen Basis (2-36) (Typ auch „unsigned int“) wandelt, und diesen String zurückgibt. Defaultmässig soll für die Wandlung die Basis „10“ genommen werden.

Fragen:

- Wie bieten Sie dem Benutzer ihre Funktion an?
- Wie verhält sich die Funktion bei fehlerhaften Parametern?

Schreiben Sie neben der Funktion auch ein kleines Haupt-Programm, das die Funktion nutzt und das Ergebnis ausgibt.

Natürlich soll Ihre Lösung ohne fertige Funktionen oder String-Streams. Hier sollen Sie selbst Hand anlegen, und eine eigene Konvertierungs-Funktion implementieren. Selbst wenn man in der Praxis natürlich eine der fertigen Lösungen nehmen würde.

Hinweise:

- Sie dürfen bei der Lösung die interne binäre Darstellung der Integer nicht nutzen, da sie vom C++ Standard nicht festgelegt ist.
- Die Aufgabe legt den Typ der zu wandelnden Zahl als „unsigned int“ fest, d.h. als Integer ohne Vorzeichen (immer ≥ 0), um Vorzeichen-Wandlungs Problemen aus dem Weg zu gehen.

12.8 Formatierung (optional)

Eine gute Vorgehensweise in der Programmier-Praxis ist, daß Sie Texte nicht direkt in den Quelltext als Literale eintragen:

```
// Schlechte Loesung, da sich so die Texte nur schwer aendern lassen.  
cout << "Ihr Filter " << filter << " liefert " << count << " Treffer" << endl;
```

Aber auch die einfache Nutzung von Konstanten macht die Situation nicht wirklich besser:

```
// Besser, aber immer noch nicht gut
const string cFilterText1("Ihr Filter ");
const string cFilterText2(" liefert ");
const string cFilterText3(" Treffer");

cout << cFilterText1 << filter << cFilterText2 << count << cFilterText3 << endl;
```

Warum – nun sind die Texte doch gut separiert und lassen sich relativ einfach ändern oder z.B. in andere Sprachen übersetzen? Nein, nur manche Änderungen lassen sich leicht durchführen – andere nicht:

```
// Oh, jetzt haben wir ein Problem - die Aenderung bekommen wir so nicht abgedeckt
const string cFilterTrefferText1("Anzahl Treffer ");
const string cFilterTrefferText2(" fuer Filter ");
const string cFilterTrefferText3(""); // Unschoener Versuch einer Loesung

cout << cFilterText1 << filter << cFilterText2 << count << cFilterText3 << endl;
```

Moegliche Ausgabe

```
Anzahl Treffer *.cpp fuer Filter 7
```

Wenn sich Texte ändern, passiert es schnell, dass die injizierten Elemente (im Beispiel „count“ und „filter“) in einer anderen Reihenfolge vorkommen, wegfallen, oder sogar mehrfach auftauchen. Gerade die Reihenfolge-Änderung ist bei Übersetzungen ganz normal.

In dieser Aufgabe wollen wir uns einem etwas einfacheren Problem zuwenden, und nicht Variablen-Werte sondern Texte in einen String injizieren. Schreiben Sie also eine Funktion, die folgende Ersetzungen in einem String durchführt:

- Die zu ersetzenden Texte (die Keys) werden durch ein „\$“ eingeleitet und beendet.
- Ein Wörterbuch (Dictionary) liefert zu jedem Key (ohne „\$“) den Ersatztext.
- Ist ein Key nicht im Wörterbuch vorhanden – so soll folgende Meldung „\$Key "key" ist unbekannt\$“ mit dem Original-Key als Ersatztext genutzt werden.
- Ist zu einem einleitenden „\$“ kein abschließendes „\$“ vorhanden, so soll folgende Meldung „Herr \$Key "key" hat kein Ende“ mit dem restlichen Original-Text als Ersatztext genutzt werden.
- „\$“ in Ersatz-Texten sollen nicht behandelt werden, sondern gelten als normale Zeichen.
- Soll ein „\$“ im Text vorkommen, so muss es durch ein weiteres „\$“ entwertet werden (so, wie wir das mit Backslashes in Strings kennen).

Das Wörterbuch soll folgende Einträge enthalten:

- anrede => Herr
- name=> Mustermann
- stadt => Musterstadt

Damit würden sich z.B. folgende String-Ersetzungen ergeben:

```
"$anrede$ $name$ wohnt in $stadt$" => "Herr Mustermann wohnt in Musterstadt"
"$anrede$ $name$ hat 23$$" => "Herr Mustermann hat 23$"
"Ein $anrede$ ist ein $anrede$ ist ein $anrede$" => "Ein Herr ist ein Herr ist ein Herr"
"$anrede$ $vorname$ $name$" => "Herr $Key "vorname" ist unbekannt$ Mustermann"
```

```
| "$anrede$ $name" => "Herr $Key "name" hat kein Ende$"
```

12.9 Telefonbuch 2

Erweitern Sie das Telefonbuch 1, das eine einfache Abbildung von Namen auf Telefonnummern macht.

Das Programm soll nun folgende Features unterstützen:

- Ein einfaches zeichenbasiertes Menü, um die einzelnen Aktionen anzutriggern:
 - Programm-Ende.
 - Sortierte Ausgabe aller Einträge (nicht lexikalisch sortiert).
 - Suchen, d.h. Eingabe eines Suchmusters und Ausgabe der passenden Namen mit ihren Telefonnummer, bzw. einer Fehlermeldung bei keinem gefundenen Eintrag. Es sollen alle Namen mit passendem Anfang zum Suchmuster ausgegeben werden. D.h. wird als Suchmuster „E“ eingegeben, so würden sowohl „Edgar“, „Edmund“ und auch „Erwin“ gefunden werden, bei Eingabe von „Ed“ dann „Edgar“ und „Edmund“. Groß- und Kleinschreibung werden bei der Suche weiterhin unterschieden.
 - Eingabe von neuen Einträgen (Name und Nummer). Bedenken Sie, dass die Namen im Telefonbuch immer noch Unikate sein müssen – schon vorhandene Namen sollen nicht überschrieben werden. Weder Name noch Nummer dürfen Leerstrings sein, da solche Einträge keinen Sinn machen.

Außerdem sollen Sie natürlich Funktionen nutzen, um den Quelltext im Verhältnis zur ersten Lösung übersichtlicher und strukturierter zu gestalten.

Der Programm-Ablauf soll ungefähr folgendermaßen sein:

```
Mögliche Ausgabe
Telefonbuch 2
-----

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> 1

3 Eintraege:
* Detlef => 1234
* Marvin => 555
* Max => 359

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> n

Neuer Eintrag:
Name: Michael
Nr.: 987

Bitte waehlen sie eine Aktion:
```



```

- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> l

3 Eintraege:
* Detlef => 1234
* Marvin => 555
* Max => 359
* Michael => 987

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> s

Suchen nach: Ma

Gefunden:
* Marvin => 555
* Max => 359

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> e

Auf wiedersehen
    
```

12.10 Türme von Hanoi 1 (optional)

Simulieren Sie die „Türme von Hanoi“.

Die „Türme von Hanoi“ ist ein Spiel, bei dem am Anfang alle Scheiben auf einem Stab liegen, und dann zu einem anderen Stab transportiert werden sollen. Für den Transport gelten zwei Regeln:

- Es darf immer nur eine Scheibe bewegt werden.
- Es darf immer nur eine kleinere Scheibe auf einer größeren liegen, nie umgekehrt.

Mögliche Ein- und Ausgabe

Mit wie vielen Scheiben sollen die Tuerme von Hanoi gespielt werden: 3

```

1.
  -|-      |      |
  --|--    |      |
  ---|---  |      |
  *****

2.
  --|--    |      |
  ---|---  |      |-|
  *****

3.
  --|--    |      |
  ---|---  |      |-|
    
```

```

*****
4.
  |           |           |
  |           -|-         |
  ---|---  ---|---
*****

5.
  |           |           |
  |           -|-         |
  ---|---  ---|---
*****

6.
  |           |           |
  -|-         ---|---
  ---|---  ---|---
*****

7.
  |           |           -|-
  -|-         ---|---
  ---|---  ---|---
*****

8.
  |           |           -|-
  |           -|-         -|-
  ---|---  ---|---
*****

```

13 Weiteres aus der Standard-Bibliothek

13.1 File-Stream

Erzeugen Sie einen Ausgabe-File-Stream und schieben Sie einige Elemente mit dem Ausgabe-Operator (<<) hinein (Zeichen und Zahlen). Öffnen Sie die Datei mit einem normalen Editor (z.B. „notepad“ oder „vi“) und überprüfen Sie, dass der Dateiinhalt wie die äquivalente Konsolen-Ausgabe aussieht, d.h. die Ausgaben alle text-orientiert sind.

13.2 String-Container

Schreiben Sie ein Programm, das folgende Dinge kann:

- Einlesen eines Strings von der Tastatur und hinten an einen String-Container anhängen.
- Ausgabe aller Strings des String-Containers (in der Reihenfolge der Eingabe).
- Abspeichern aller Strings des String-Containers auf Festplatte (in der Reihenfolge der Eingabe). Der Name der Datei ist fest im Programm codiert.
- Laden der Strings von Festplatte in der Original-Reihenfolge.

Das Programm soll dabei ein Menü anbieten, mit dem sich die verschiedenen Aktionen antriggern lassen.

- e : Programmende
- i : Eingabe eines neuen Strings

- a : Ausgabe aller Strings
- s : Abspeichern der Einträge in die Datei
- l : Laden der Datei

13.2.1 Telefonbuch 3

Erweitern Sie das „Telefonbuch 2“. Folgende Features sollen hinzukommen:

- Pro Name sollen jetzt mehrere Telefon-Nummern unterstützt werden – es sind auch keine Nummern erlaubt. Alle Nummern müssen beim Anlegen des Eintrags eingegeben werden.
- Es wird noch keine Bearbeitung eines Eintrags, z.B. um den Namen zu ändern oder Nummern zu ändern, hinzuzufügen oder zu löschen, unterstützt.
- Automatisches Laden des Telefonbuchs aus einer Datei bei Programmstart mit Rückmeldung – siehe Beispiel Ausgabe unten.
- Automatisches Speichern des Telefonbuchs in die Datei bei Programmende – geht das Speichern schief, so soll eine entsprechende Fehlermeldung ausgegeben werden, und das Programm wird nicht beendet.
- Der Datei-Name soll fest im Telefonbuch als Konstante verankert sein, und im aktuellen Arbeitsverzeichnis liegen – d.h. sie kann ohne Pfad-Angaben genutzt werden.

Mögliche Ein- und Ausgabe

```
Telefonbuch 3
=====

Lade Daten aus der Datei "telefonbuch3.dat"...
- 7 Eintraege geladen.

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> l

7 Eintraege:
* Detlef => (0124 / 35 53) (0258 / 147 258) (23982)
* Edgar => (0111 / 11 11 1) (0111 / 11 11 2)
* Edmund => (01240 / 89 89 80)
* Erwin => (010 / 123 45) (0160 123 456)
* Iris => (4567) (789) (147) (3698) (2569) (129547)
* Nils =>
* Willi => (01240 / 99 88 77) (0123 456 789)

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> s

Suchen nach: Ed

Gefunden:
* Edgar => (0111 / 11 11 1) (0111 / 11 11 2)
* Edmund => (01240 / 89 89 80)

Bitte waehlen sie eine Aktion:
- e : Programmende
```

```
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> s

Suchen nach: A

Kein Name im Telefonbuch beginnt mit "A".

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> n

Neuer Eintrag
> Name: Jan
> Nr.: 123 897
> Nr.: 01240 / 45 54
> Nr.:

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> l

8 Eintraege:
* Detlef => (0124 / 35 53) (0258 / 147 258) (23982)
* Edgar => (0111 / 11 11 1) (0111 / 11 11 2)
* Edmund => (01240 / 89 89 80)
* Erwin => (010 / 123 45) (0160 123 456)
* Iris => (4567) (789) (147) (3698) (2569) (129547)
* Jan => (123 897) (01240 / 45 54)
* Nils =>
* Willi => (01240 / 99 88 77) (0123 456 789)

Bitte waehlen sie eine Aktion:
- e : Programmende
- l : Alle Eintraege auflisten
- s : Nach Eintrag suchen
- n : Neuen Eintrag eingeben
> e

Sichere Daten in die Datei "telefonbuch3.dat"...
- fertig

Auf wiedersehen
```

13.3 Platten-Platz (optional)

In der Vorlesung haben wir eine einfache Form der rekursiven Datei-Suche kennengelernt. Nun sollen Sie das Programm etwas erweitern.

- Ihr Programm soll berechnen, wieviel Platten-Platz alle Dateien eines Typs (einer Extension) innerhalb und unterhalb eines Verzeichnisses verbrauchen.
- Zusätzlich soll der Benutzer das Verzeichnis und die Such-Extension (ohne „.“) auf der Kommando-Zeile eingeben können.
- Die Verzeichnis-Eingabe soll sich so oft wiederholen, bis der Benutzer ein gültiges Verzeichnis eingegeben hat.
- Wird keine Extension eingegeben, wird der Platten-Platz Verbrauch aller Dateien im

Verzeichnis berechnet.

- Vergleichbar zur rekursiven Datei-Suche aus der Vorlesung ignorieren wir Probleme, die sich aus User-Rechten und fehlenden Zugriffs-Rechten resultieren.

Beispiele, unter der Berücksichtigung der Datei-System-Struktur aus der Vorlesung, und entsprechender Datei-Größen:

Mögliche Ein- und Ausgabe

Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: `C:\DateiSystemBeispiele`

Geben Sie bitte die Extension ein: `dat`

Gesamtverbrauch: 3472 Bytes

Mögliche Ein- und Ausgabe

Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: `C:\DateiSystemBeispiele`

Geben Sie bitte die Extension ein: `txt`

Gesamtverbrauch: 9301 Bytes

Mögliche Ein- und Ausgabe

Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: `C:\DateiSystemBeispiele`

Geben Sie bitte die Extension ein:

Gesamtverbrauch: 12773 Bytes

Mögliche Ein- und Ausgabe

Platten-Platz Verbrauch

Geben Sie bitte das Verzeichnis ein: `kein-verzeichnis`

"kein-verzeichnis" ist kein Verzeichnis

Geben Sie bitte das Verzeichnis ein: `immer noch nicht`

"immer noch nicht" ist kein Verzeichnis

Geben Sie bitte das Verzeichnis ein: `C:\DateiSystemBeispiele`

Geben Sie bitte die Extension ein: `dat`

Gesamtverbrauch: 3472 Bytes

13.4 Datei-Suche (optional)

Die nächste Aufgabe erweitert die einfache Datei-Suche aus der Vorlesung.

- Ihr Programm soll alle Dateien innerhalb eines Verzeichnisses suchen, die einem von mehreren Namen entsprechen.
- Der Benutzer soll das Such-Verzeichnis und die Such-Datei-Namen auf der Kommando-Zeile eingeben können.
- Die Such-Verzeichnis-Eingabe soll sich so oft wiederholen, bis der Benutzer ein gültiges Verzeichnis eingegeben hat.
- Die Such-Datei-Namen-Eingabe soll sich so oft wiederholen, bis der Benutzer einen Leerstring eingibt. Alle Namen bis dahin gelten als zu Suchen.
- Die gefundenen Dateien mit ihren Pfaden sollen sortiert nach den Datei-Namen

ausgegeben werden. Jeder Ausgabe-Block soll mit dem Datei-Namen beginnen, und danach eine eingerückte Aufzählung der Verzeichnisse (inkl. Datei-Namen).

- Wurde keine Datei zu einem Datei-Such-Namen gefunden, so wird nur der Datei-Such-Name ausgegeben.
- Vergleichbar zur rekursiven Datei-Suche aus der Vorlesung ignorieren wir Probleme, die sich aus User-Rechten und fehlenden Zugriffs-Rechten resultieren.

Beispiele, unter der Berücksichtigung der Datei-System-Struktur aus der Vorlesung:

Mögliche Ein- und Ausgabe

Datei-Suche

Geben Sie bitte das Such-Verzeichnis ein: **C:\DateiSystemBeispiele**

Geben Sie bitte die Such-Namen ein: **find.txt**

Geben Sie bitte die Such-Namen ein: **search.dat**

Geben Sie bitte die Such-Namen ein:

find.txt

-> C:\DateiSystemBeispiele\verzeichnis1\verzeichnis3\verzeichnis4\find.txt

-> C:\DateiSystemBeispiele\verzeichnis1\verzeichnis5\find.txt

-> C:\DateiSystemBeispiele\verzeichnis2\find.txt

search.dat

-> C:\DateiSystemBeispiele\search.dat

-> C:\DateiSystemBeispiele\verzeichnis1\verzeichnis3\verzeichnis4\search.dat

Mögliche Ein- und Ausgabe

Datei-Suche

Geben Sie bitte das Such-Verzeichnis ein: **kein Verzeichnis**

"kein Verzeichnis" ist kein Verzeichnis

Geben Sie bitte das Such-Verzeichnis ein: **C:\DateiSystemBeispiele**

Geben Sie bitte die Such-Namen ein: **search.dat**

Geben Sie bitte die Such-Namen ein: **search.txt**

Geben Sie bitte die Such-Namen ein: **search.png**

Geben Sie bitte die Such-Namen ein:

search.dat

-> C:\DateiSystemBeispiele\search.dat

-> C:\DateiSystemBeispiele\verzeichnis1\verzeichnis3\verzeichnis4\search.dat

search.png

search.txt

13.5 Lotto-Programm

Schreiben Sie ein Lotto-Programm. D.h. ein Programm, das sechs Zufallszahlen zwischen 1 und 49 inkl. ausgibt, ohne dass dabei eine Zahl doppelt vorkommt. Die Ausgabe der Zahlen soll dabei sortiert sein.

13.6 Pi simulieren (optional)

Schreiben Sie ein Programm, das via Zufalls-Simulation den Wert von Pi zu bestimmen versucht. Stellen Sie sich dazu eine Kanone vor, die sehr stark streut, und deren Schüsse immer ein beliebiges quadratisches Feld der Größe „2*x“ treffen. Das quadratische Feld wird

immer getroffen, d.h. kein Schuß geht nebenher, aber die Schüsse verteilen sich auch gleichmäßig über das Feld – d.h. es gibt z.B. kein Maximum in der Mitte des Feldes.

Liegen die Treffer nun in einem Kreis mit Radius „ x “ um den Mittelpunkt des Feldes, so zählen sie zu Pi. Liegen sie außerhalb des Kreises im Feld, so zählen sie nicht zu Pi. Aus dem Verhältnis der Treffer im Kreis und außerhalb des Kreises lässt sich Pi näherungsweise *berechnen*.

Je mehr Schüsse Ihre Kanone im Programm abgibt, desto genauer wird Pi simuliert. Bei ungefähr 10.000 Schüssen bekommt man typischerweise schon eine auf zwei Stellen genaue Näherung von Pi. Aber probieren Sie auch ruhig mal 100.000 oder 1.000.000 Schüsse aus.

Noch ein Hinweis – lassen Sie sich von der Abstraktion mit der Kanone in der Aufgabenstellung nicht verwirren. Die Aufgabe ist eigentlich ganz einfach, und benötigt nur einfachste Geometrie- und Rechen-Kenntnisse aus der 8-ten Klasse. Das „*Bild*“ mit der Kanone macht die Aufgabe nur *lebendiger* und irgendwie *schöner*.

13.7 Telefonbuch 3

Erweitern Sie das Telefonbuch 2 um die Möglichkeit, die Daten in einer Datei zu speichern, und von dort zu laden. Entweder implementieren Sie dafür eigene Befehle im Menü, oder Sie laden die Daten einfach immer beim Start des Programms, und Speichern Sie beim Beenden des Programms.

Passen Sie auf, die Daten so in einer Datei zu speichern, dass Sie sie in jedem Fall wieder lesen können.

14 Klassen

14.1 Klasse date

- Implementieren Sie die komplette Klasse „date“ aus der Vorlesung.
- Erweitern Sie sie um Funktionen für Vergleich auf Gleichheit („eq“), Ungleichheit („ne“), kleiner („lt“), kleiner-gleich („le“), größer („gt“) und größer-gleich („ge“).
- Erweitern Sie sie um eine Eingabe-Funktion.
- Schreiben Sie ein kleines Programm, dass alle diese Funktion nutzt.

14.2 Ringzähler

Schreiben Sie einen Ringzähler. Ein Ringzähler ist ein Zähler, der von einem Startwert an hoch (oder runter) zählt (Delta muss nicht 1 sein) und nach Überschreiten des Endwertes wieder beim Startwert anfängt. Ein solcher Zähler wird in der Praxis häufig gebraucht, z.B.

als Index in ein Array, als Verweis auf ein Spielfeld, usw.

Kümmern Sie sich auch um widersprüchliche Initialisierungen.

14.3 Telefonbuch 4

Schreiben Sie das „Telefonbuch 3“ auf Ihr neues Wissen um: Nutzen Sie die Möglichkeit mit Klassen zusammengehörige Dinge kapseln zu können, und dadurch das Programm besser zu strukturieren.

Nachdem Sie den Umstieg gemacht haben, bauen Sie ihr Programm so um, dass bei der Sortierung und der Suche Groß- und Kleinschreibung ignoriert werden. Hinweis – möglicherweise (je nach Implementierung) kann es sein, dass Sie Elemente (z.B. den Namen) doppelt halten müssen. Nehmen Sie solche Unlänglichkeiten erstmal hin, und kümmern Sie sich um die primären Themen dieser Wochen, die da sind: Klassen und alles was dazu gehört. Wir werden später Sprache-Features kennen lernen, die uns erlauben, doppelte Datenhaltung zu vermeiden.

Der dritte Teil des Umbaus von Telefonbuch 4 sind dann folgende neuen Features:

- Erweitern Sie den Eintrag um einen Wohnort (z.B. einfach nur Strasse und Ort)
- Bieten Sie eine Möglichkeit, Einträge zu editieren.
- Bieten Sie eine Möglichkeit, Einträge zu löschen.

14.4 Türme von Hanoi 2 (optional)

Zumindest in der Muster-Lösung zur Aufgabe „Türme von Hanoi 1“, wurden globale Variablen genutzt, damit alle Funktionen problemlos Zugriff auf die Turm-Variablen, den Zug-Zähler und die Count-Variable hatten. Aber wir haben gelernt, dass globale Variablen zur dunklen Seite der Programmierung – nicht nur in C++ – gehören. Mit Klassen haben wir jetzt die Sprach-Mittel, um zusammenhängende Dinge zu gruppieren und den allgemeinen Zugriff zu verbieten. Damit können wir globale Variablen ersetzen.

Schreiben Sie also die „Türme von Hanoi“ neu, diesmal ohne globale Variablen, dafür aber mit dem Einsatz von Klassen.

14.5 Gerüchteküche 1 (optional)

Kleine Dörfer sind Gerüchteküchen erster Klasse. Jeder kennt jeden, und jeder redet über jeden, bzw. schweigt wissend. All das wollen wir in einem kleinen Programm nachbilden.

Simulieren Sie ein Dorf mit n Einwohnern. Am Anfang kennt nur ein beliebiger Einwohner (nehmen Sie einfach den ersten) das Gerücht.

Danach simulieren Sie das Geschehen, indem Sie maximal m Durchläufe starten. In jedem

Durchlauf treffen sich zwei zufällige Einwohner des Dorfes und reden miteinander. Was hierbei passiert, ist davon abhängig, welchen Status bzgl. des Gerüchts die beiden Einwohner haben.

Einwohner 1	Einwohner 2	=>	Einwohner 1	Einwohner 2
unwissend	unwissend		unwissend	unwissend
unwissend	erzählend		erzählend	erzählend
unwissend	schweigsam		unwissend	schweigsam
erzählend	unwissend		erzählend	erzählend
erzählend	erzählend		schweigsam	schweigsam
erzählend	schweigsam		schweigsam	schweigsam
schweigsam	unwissend		schweigsam	unwissend
schweigsam	erzählend		schweigsam	schweigsam
schweigsam	schweigsam		schweigsam	schweigsam

Das Programm ist damit im Prinzip beschrieben. Hier noch einige Hinweise:

- Der Benutzer soll am Anfang eingeben, wieviele Bewohner das Dorf hat, und wieviele Durchläufe maximal gewünscht sind.
- Gibt es stabile Zustände? D.h. Zustände, bei denen es keine Änderung der Stati der Einwohner mehr gibt? Wenn ja, welche? Erreicht das Programm einen solchen stabilen Zustand, soll die Simulation mit einer entsprechenden Meldung beendet werden.
- Wird das Programm durch die max. Anzahl an Durchläufen beendet, soll die Simulation auch hier mit einer entsprechenden Meldung beendet werden.
- Das Programm soll nach einer Status-Änderung folgende Informationen in einer Zeile ausgeben:
 - Nummer des Durchlaufs (n-stellig in Abhängigkeit der max. Durchläufe, rechtsbündig, führende Leerzeichen)
 - Anzahl an unwissenden Einwohnern in der Form „U(x)“ (n-stellig in Abhängigkeit der max. Durchläufe, rechtsbündig, führende Underscores)
 - Anzahl an erzählenden Einwohnern – Form analog zu den Unwissenden
 - Anzahl an schweigsamen Einwohnern – Form analog zu den Unwissenden
 - Ein Zeichen für jeden Einwohner:
 - Unwissend => „_“
 - Erzählend => „|“
 - Schweigsam => „*“

Eine Simulation könnte also folgendermaßen aussehen:

```

Mögliche Ein- und Ausgabe:
Geruechtekueche
- Anzahl Personen: 30
- Max Durchlaeufer: 400

0: U(29) E( 1) S( 0) | _____
3: U(28) E( 2) S( 0) | _____
5: U(27) E( 3) S( 0) | _____
6: U(26) E( 4) S( 0) | _____
8: U(26) E( 2) S( 2) | _____*_____
9: U(25) E( 3) S( 2) | _____|_____*_____
19: U(25) E( 2) S( 3) | _____*_____*_____
26: U(24) E( 3) S( 3) | _____*_____*_____
    
```

```

36: U(23) E( 4) S( 3) |  _ _ _ * | * _ |  _ _ _ |  _ *
41: U(22) E( 5) S( 3) |  _ _ _ * | * _ |  _ _ _ |  _ *
45: U(22) E( 4) S( 4) * |  _ _ _ * | * _ |  _ _ _ |  _ *
47: U(22) E( 2) S( 6) * |  _ _ _ * | * * |  _ _ _ |  _ *
62: U(21) E( 3) S( 6) * |  _ _ _ * | * * |  _ _ _ |  _ *
109: U(21) E( 2) S( 7) * |  _ _ _ * | * * |  _ _ _ |  _ *
116: U(21) E( 1) S( 8) * |  _ _ _ * | * * |  _ _ _ |  _ *
136: U(20) E( 2) S( 8) * |  _ _ _ * | * * |  _ _ _ |  _ *
139: U(19) E( 3) S( 8) * |  _ _ _ * | * * |  _ _ _ |  _ *
159: U(19) E( 1) S(10) * |  _ _ _ * | * * |  _ _ _ |  _ *
182: U(19) E( 0) S(11) * |  _ _ _ * | * * |  _ _ _ |  _ *

```

Ende wegen stabilem Zustand

14.6 Tic-Tac-Toe 1 (optional)

Schreiben Sie ein Tic-Tac-Toe Spiel. Tic-Tac-Toe wird auf einem 3x3 Brett gespielt. Die Spieler setzen immer abwechselnd. Pro Feld darf nur ein Spielstein liegen. Hat ein Spieler drei Steine in einer Reihe (horizontal, vertikal oder diagonal), so hat er gewonnen und das Spiel ist instantan vorbei. Spätestens nach neun Zügen ist das Spielbrett voll, und entweder hat ein Spieler gewonnen, oder es ist Remis ausgegangen. Hinweis – spielt ein Spieler fehlerfrei, so kann er nicht verlieren. Daraus folgt, dass wenn beide Spieler optimal spielen, das Spiel immer Remis endet.

Achtung – gehen Sie das Programm im ersten Schritt sehr pragmatisch an, d.h. versuchen Sie es erstmal vollständig (im Sinne von man kann Spielen) zum Laufen zu bringen, aber sparen Sie die Details aus: beschränken Sie sich im ersten Wurf auf eine einfache Ausgabe, auf eine einfache Eingabe (Sie könnten die Felder z.B. durchnummerieren und der Benutzer gibt einfach eine 1-9 ein). Legen Sie einfach fest, dass z.B. immer der Mensch anfängt und der Computer der zweite Spieler ist. Und vor allem fangen Sie mit einem ganz dummen Computerspieler an (im einfachsten Fall wählt er einfach das erste freie Feld aus, oder er zieht per Zufallsgenerator). Eine richtig gute Strategie zu implementieren – eine die nie verliert – ist sicher interessant und Sie sollten das auch auf Dauer machen – aber erstmal sollten Sie das Spiel zum Laufen bringen.

Fangen Sie auch nicht einfach blind an zu Implementieren – dazu ist diese Aufgabe wahrscheinlich schon zu groß. Setzen Sie sich hin, und überlegen Sie, welche Teile sie zuerst, warum, und wie implementieren. Welche Teile hängen von einander ab, usw. In dem sich sich diese Fragen stellen und beantworten, sollte die Struktur ihres Programmes fast von ganz alleine sinnvoll wachsen.

15 Präprozessor, Compiler, Linker, ...

15.1 Telefonbuch 5

Verteilen Sie den Code Ihrer Lösung der Aufgabe „Telefonbuch 4“ auf mehrere Dateien.

15.2 Ring-Zähler Bibliothek

Erzeugen Sie mit Ihrem Compiler aus Ihrer Ring-Zähler Klasse eine Bibliothek, die Sie dann in späteren Projekten nutzen können. Schreiben Sie ein kleines Beispiel-Projekt, das Ihre erstellte Bibliothek verwendet.

15.3 Türme von Hanoi 3 (optional)

Verteilen Sie den Code Ihrer Lösung der Aufgabe „Türme von Hanoi 2“ auf mehrere Dateien.

15.4 Gerüchteküche 2 (optional)

Verteilen Sie den Code Ihrer Lösung der Aufgabe „Gerüchteküche 1“ auf mehrere Dateien.

15.5 Tic-Tac-Toe 2 (optional)

Verteilen Sie den Code Ihrer Lösung der Aufgabe „Tic-Tac-Toe 1“ auf mehrere Dateien.

16 Operator-Funktionen

16.1 Bruch-Klasse

Schreiben Sie eine Klasse „rational“ für Brüche, und implementieren Sie für diese die Operatoren für Multiplikation, die multiplikative Zuweisung, und den Ausgabe-Operator. Schreiben Sie außerdem ein kleines Beispiel-Programm, das alle diese Funktionen nutzt.

16.2 Funktions-Objekte

Schreiben Sie eine einfache Klasse, die den Funktions-Aufrufs-Operator überladen hat. Zeigen Sie, dass Objekte dieser Klasse *syntaktisch quasi* wie Funktionen benutzt werden können.

17 Vererbung & Polymorphie

17.1 Obstkorb

Implementieren Sie das Programm Obstkorb aus der Vorlesung, möglicherweise mit weiteren Änderungen, die sich aus den restlichen Kapiteln ergeben haben. Fügen Sie als weitere Obstsorte noch Bananen (engl. „banana“) hinzu. Welche Auswirkungen hat dies auf Ihr Programm?

17.2 Tic-Tac-Toe 3

Erweitern Sie das Tic-Tac-Toe 2.

- Entwickeln Sie einen weiteren Computer-Spieler. Wenn sie damals einen entwickelt haben der per Zufall zieht, schreiben sie jetzt einen der immer das erste Feld nimmt. Oder umgekehrt.
- Wenn Sie Lust haben (d.h. zu lösen außerhalb des Praktikums) entwickeln sie einen Computer-Spieler, der strategisch spielt. Oder einen, der eine tiefe Analyse der Spiel-Situation durchführt und immer optimal zieht.
- Mit dem Repertoire von jetzt mindestens drei Spielern: schreiben Sie das Spiel um, so dass der Benutzer am Anfang wählen kann, wer gegen wen spielt, und wer anfängt. Jede beliebige Kombination soll möglich sein.

17.3 Progress-Bar (optional)

Ein typisches Problem der „Modul-Entkopplung“ ist eine Fortschritts-Anzeige („Progress-Bar). In der BL Schicht findet eine Verarbeitung statt, die längere Zeit braucht. Damit der Benutzer eine Rückkopplung bekommt, soll eine Fortschritts-Anzeige aufgeblendet werden. Nur: die BL-Schicht kennt keine UI-Schicht und weiss auch nicht, was für eine Fortschritts-Anzeige im jeweiligen UI-Kontext sinnvoll ist. Daher bietet sich eine Entkopplung über eine Basis-Klasse an.

- Denken sie sich eine einfache Verarbeitung in der BL-Schicht aus, die etwas Zeit kostet. Z.B. eine einfache Schleife.
- Schreiben sie ein erstes kleines Programm ohne Fortschritts-Anzeige mit BL- und UI-Schicht.
- Definieren sie ein Interface, mit dem eine Fortschritts-Anzeige betrieben werden kann.
- Implementieren sie eine Fortschritts-Anzeige.
- Integrieren sie Interface und Fortschritts-Anzeige in ihr Programm.
- Implementieren sie eine weitere Fortschritts-Anzeige. Zeigen sie durch einen einfachen Austausch der Fortschritts-Anzeigen dass die BL-Schicht mit beiden betrieben werden kann, ohne dass die BL-Schicht betroffen ist (d.h. geändert werden muss).
- Mögliche Fortschritts-Anzeigen auf Konsolen-Ebene wären z.B.:
 - Ausgabe, die von „0 %“ bis „100 %“ hochzählt.
 - Liniengrafik mit z.B. dem Zeichen „|“.