C++
Aufgabensammlung

Verfasser: Livio Marti
Überarbeitung: Thomas Kälin

Version 1: 21.07.2006
Version 2: 26.08.2006

Inhaltsverzeichnis

Übungsserie 2

Aufgabe 1 – Zeichen zählen (distance, count)

```
#include <iostream>
#include <iterator>
#include <vector>
#include <algorithm>
using namespace std;

int main (int argc, char * const argv[]) {
	typedef istreambuf_iterator<char> input;	//istreambuf liest auch Whitespaces
	input eof;
	vector<char> zeichen(input(cin), eof);
	cout << "anzahl zeichen: " << distance(zeichen.begin(),zeichen.end()) << endl;
	cout << "anzahl zeilen: " << count(zeichen.begin(),zeichen.end(),'\n') << endl;
	return 0;

	/* Alternative von Sommerlad
	char c;
	while (cin.get(c)) {
		++chars;
	}
	*/
}
```

Aufgabe 2 – Summe (accumulate)

```
#include <iostream>
#include <iterator>
#include <numeric>  //Accumulate braucht <numeric>
using namespace std;

int main (int argc, char * const argv[]) {
	istream_iterator<int> eof;
	int n = accumulate(istream_iterator<int>(cin),eof,0);
	cout << "Summe: " << n << endl;
}
```

Aufgabe 3 – Mittelwert (copy)

```
#include <iostream>
#include <vector>
#include <iterator>
#include <numeric>
#include <algorithm>	//Copy
using namespace std;

int main (int argc, char * const argv[]) {
	istream_iterator<int> eof;
	vector<int> zahlen;
	copy(istream_iterator<int>(cin), eof, back_inserter(zahlen));	//Am Ende einfügen
	copy(zahlen.begin(), zahlen.end(), ostream_iterator<int>(cout, "\n"));
	cout << "Mittelwert: " <<
accumulate(zahlen.begin(),zahlen.end(),0)/distance(zahlen.begin(),zahlen.end()) << endl;
}
```

Aufgabe 4 – Rechentrainer 1

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

int main (int argc, char * const argv[]) {
	float erste = 0;
	float zweite = 0;

	cout << "Geben Sie zwei Zahlen ein: " << flush;
	cin >> erste >> zweite;
	cout << erste << " + " << zweite << " = " << erste+zweite << endl;
	cout << erste << " - " << zweite << " = " << erste-zweite << endl;
	cout << erste << " * " << zweite << " = " << erste*zweite << endl;
	if (zweite != 0) {
		cout << erste << " / " << zweite << " = " << erste/zweite << endl;
	}
	else {
		cout << erste << " / " << zweite << " = DIVIS DURCH 0 VERB!!!" << endl;
	}
}
```

Aufgabe 5 – Rechentrainer 2

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

int main (int argc, char * const argv[]) {
	int erste;
	int zweite;
	int resultat;

	cout << "Geben Sie zwei Zahlen ein: " << flush;
	cin >> erste >> zweite;
	cout << erste << " + " << zweite << " = " << flush;
	cin >> resultat;
	while(resultat != erste+zweite){
		cout << erste << " + " << zweite << " = " << flush;
		cin >> resultat;
	}
	cout << "RICHTIG!" << endl;
}
```

Aufgabe 6 – Klassenliste (Zeilenweise einlesen)

```
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
using namespace std;
int main (int argc, char * const argv[]) {
	string line;
	vector<string> v;
	while(getline(cin, line)){	//Ganze Zeile einlesen, std::getline
		v.push_back(line);
	}
	sort(v.begin(),v.end());	//Sortieren des Vektors
	copy(v.begin(),v.end(),ostream_iterator<string>(cout, "\n"));
}
```

Übungsserie 3

Aufgabe 1 – ‚A' zählen (count)

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

int main (int argc, char * const argv[]) {
	istream_iterator<char> eof;
	cout << count(istream_iterator<char>(cin),eof,'a') << endl;
}
```

Aufgabe 2 – Vokale zählen (std::map)

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

int main (int argc, char * const argv[]) {
	map<char,int> refMap;
	refMap['a'] = 0;
	refMap['e'] = 0;
	refMap['i'] = 0;
	refMap['o'] = 0;
	refMap['u'] = 0;

	cout << "Geben Sie bitte einen Text ein:" << endl;

	char chrInput;
	while (cin.get(chrInput)) {
		if (refMap.find(chrInput) != refMap.end()) {
			refMap[chrInput]++;
		}
	}

	for (map<char,int>::iterator itRun = refMap.begin();itRun != refMap.end(); ++itRun) {
		//Zugriff auf Key mittels first, auf Value mittels second
		cout << itRun->first << "\t" << itRun->second << endl;
	}
}
```

Aufgabe 3 – 7-Segment-Anzeige (std::map)

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

int main (int argc, char * const argv[]) {
	map<char, map<int, string> > refNumbers;	//Map von Maps mit <int, string>

	map<int, string> num0;
	num0[0] = " _ ";
	num0[1] = "| |";
	num0[2] = "|_|";
	refNumbers['0'] = num0;

	map<int, string> num1;
	num1[0] = "   ";
	num1[1] = "  |";
	num1[2] = "  |";
	refNumbers['1'] = num1;
	//Jetz folgen noch viele weitere Zahlen… ☺

	cout << "Geben Sie bitte eine ganze Zahl ein: " << endl;
	string strInput = "0";
	getline(cin,strInput);

	char chrZeichen = '0';
	for (unsigned int i=0; i<=2 ; ++i) {
		for (unsigned int j=0; j<strInput.length() ; ++j) {
			chrZeichen = strInput[j]; //Indexzugriff auf String möglich
			cout << refNumbers[chrZeichen][i];
		}
		cout << endl;
	}
}
```

Aufgabe 4 – Wortliste (std::set)

```
#include <iostream>
#include <iterator>
#include <algorithm>
#include <set>
using namespace std;

int main (int argc, char * const argv[]) {
	istream_iterator<string> eof;
	set<string> wlist(istream_iterator<string>(cin),eof);	//Set ist autom. sortiert
	copy(wlist.begin(),wlist.end(),ostream_iterator<string>(cout,", "));
}
```

Aufgabe 5 – Wortstatistik (std::map)

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

int main (int argc, char * const argv[]) {
	map<string,int> wstat;

	string strWord;
	while (cin >> strWord) {
		if (wstat.count(strWord) == 0) {
			wstat[strWord] = 1;
		} else {
			wstat[strWord]++;
		}
	}

	/* Variante Livio
	istream_iterator<string> beg(cin);
	istream_iterator<string> eof;
	for(istream_iterator<string> it = beg; it != eof; ++it){
		if(wstat.find(*it) != wstat.end()) { wstat[*it] = wstat[*it] + 1; }
		else { wstat[*it] = 1; }
	}
	*/


	for(map<string,int>::iterator it = wstat.begin(); it != wstat.end(); ++it){
		cout << it->first << "\t\t\t" << it->second << endl;
	}
}
```

Übungsserie 4

Aufgabe 1 - boost/assign

```
#include <iostream>
#include <set>
#include <iterator>
#include <boost/assign.hpp>
#include <boost/assign/list_of.hpp>
using namespace std;
using namespace boost::assign;

int main (int argc, char * const argv[]) {

	set<int> zahlenListe1;
	zahlenListe1.insert(5);
	zahlenListe1.insert(3);
	//Es folgen viele weitere Inserts..

	set<int> zahlenListe2;
	zahlenListe2 += 5, 3, 77, 36, 363, 1;  //funktioniert dank boost_assign!

	set<int> zahlenListe3 = list_of(5)(3)(77)(36)(353)(2); //funktioniert mit list_of.hpp

	copy(zahlenListe3.begin(), zahlenListe3.end(), ostream_iterator<int> (cout, "\n"));
}
```

Aufgabe 5 – 7-Segment-Anzeige (mit Klassen)

digits.h:

```
#ifndef DIGITS_H
#define DIGITS_H
#include <vector>
#include <string>
#include <stdexcept>      //für std::out_of_range

class digits{
      public:
            digits();
            std::string getDigitLine(int intNumber, int intRow) throw (std::out of range);
      private:
            std::vector< std::vector<std::string> > vecZiffern;
};

#endif //DIGITS_H
```

digits.cpp:

```
#include "digits.h"
#include <boost/assign.hpp>
#include <boost/assign/list_of.hpp>
using namespace std;
using namespace boost::assign;

digits::digits(){
      vecZiffern +=       list_of(" _ ")("| |")("|_|"),    //0
                          list_of("   ")("  |")("  |"),    //1
                          list_of(" _ ")(" _|")("|_ "),    //2
                          list_of(" _ ")(" _|")(" _|"),    //3
                          list_of("   ")("|_|")("  |"),    //4
                          list_of(" _ ")("|_ ")(" _|"),    //5
                          list_of("   ")("|_ ")("|_|"),    //6
                          list_of(" _ ")("  |")("  |"),    //7
                          list_of(" _ ")("|_|")("|_|"),    //8
                          list_of(" _ ")("|_|")(" _|");    //9
}

string digits::getDigitLine(int intNumber, int intRow) throw (out_of_range){
      if(intNumber >= 0 && intNumber <= 9 && intRow >= 0 && intRow <= 2) {
            return digitsvec[a][b];
      } else {
            throw out of range("attributes out of bounds");      //Exception werfen
      }
}
```

DigitLine.h:

```
#ifndef DIGIT_LINE_H
#define DIGIT_LINE_H
#include <iostream>
#include "digits.h"

class DigitLine{
      public:
            DigitLine(std::string strInput);
            void print();
      private:
            std::string strLine;
            digits objSegment;
};
#endif //DIGIT_LINE_H
```

DigitLine.cpp:

```
#include "DigitLine.h"
using namespace std;

DigitLine::DigitLine(std::string strInput) : strLine(strInput) {} //Konstruktor, Initialisier.

void DigitLine::print() {
	for(unsigned int i = 0; i <= 2; ++i){
		for(unsigned int j=0; j < strNumber.length(); ++j){
			cout << objSegment.getDigitLine(atoi(&chrZeichen), i) << flush;
			//cout << objSegment.getDigitLine(chrZeichen-'0', i) << flush;
			//Alternative wäre auch noch boost::lexical_cast!
		}
		cout << endl;
	}
}
```

main.cpp:

```
#include <iostream>
#include "DigitLine.h"
using namespace std;

int main (int argc, char * const argv[]) {
	try {
		string strInput;
		cout << "Ziffernfolge eingeben: ";
		cin >> strInput;
		DigitLine theDigits(strInput);
		theDigits.print();
	}
	catch(exception& e) {
		cout << e.what() << endl; //Exception ausgeben
	}
	return 0;
}
```

Übungsserie 5

Aufgabe 1 - boost/tokenizer - boost/lexical_cast

```
#include <iostream>
#include <vector>
#include <iterator>
#include <boost/tokenizer.hpp>
#include <boost/lexical_cast.hpp>
#include <stdexcept>
using namespace std;
using namespace boost;

int main (int argc, char * const argv[]) {
	string strInput;
	string strOperations = "+-*/";

	cout << "Bitte geben Sie eine Rechung in Form von: 435 * 234 ein:" << endl;
	getline(cin,strInput);

	//\x0D ist Zeilenumbruch, \x20 ist Leerzeichen, Alternative: „ \n\r\t"
	char_separator<char> chrSeperator("\x0D\x20");	//Trennzeichen definieren
	tokenizer< char_separator<char> > refTok(strInput, chrSeperator); //trennen

	int intCount = 1;
	for(tokenizer< char_separator<char> >::iterator it = refTok.begin();it != refTok.end();
	++it) {
		switch (intCount) {
			case 1: //Double
			case 3:
				try {
					cout << lexical_cast<double>(*it) << " ";
				} catch (bad_lexical_cast &) {
					cout << "(falscher Input) ";
				}
			break;
			case 2: //Operationszeichen +-*/
				if(strOperations.find(*it,0) == strOperations.npos) {
					cout << "(falscher Operator) ";
				} else {
					cout << *it << " ";
				}
			break;
		}
		++intCount;
	}
}
```

Aufgabe 2 - ANSI-Colour (enum, static, const)

Ansi.h:

```
#ifndef ANSIKONSOLE_HPP_
#define ANSIKONSOLE_HPP_
#include <string>

class AnsiKonsole {
	private:
		const std::string strEsc;
	public:
		AnsiKonsole();
		enum enumColors {BLACK=0, RED, GREEN, YELLOW, BLUE, DARKRED, LIGHTBLUE, WHITE=7};

		static std::string clear() const;
		static std::string bold() const;
```

```
            static std::string pos(int intX,int intY) const;
            static std::string home() const;
            static std::string foreColor(enumColors eColor) const;
            static std::string backColor(enumColors eColor) const;
            static std::string attrOff() const;
            static std::string up(int intY) const;
            static std::string down(int intY) const;
            static std::string left(int intX) const;
            static std::string right(int intX) const;
};
#endif /*ANSIKONSOLE_HPP_*/
```

Ansi.cpp:

```
#include <iostream>
#include <string>
#include <boost/lexical_cast.hpp>
#include "Ansi.h"

using namespace boost;

const std::string Ansi::strEsc = "\x1B[";

std::string AnsiKonsole::clear() const {      //const für Methoden die nichts an Obj ändern!
      return strEsc + "2J";
}

std::string AnsiKonsole::bold() const {
      return strEsc + "1m";
}

std::string AnsiKonsole::pos(int intX,int intY) const {
      return strEsc + lexical_cast<std::string>(intX) + ";" + lexical_cast<std::string>(intY)
+ "H";
}

std::string AnsiKonsole::home() const {
      return strEsc + "0;0H";
}

std::string AnsiKonsole::foreColor(Ansi::enumColors eColor) const {
      return strEsc + lexical_cast<std::string>(eColor+30) + "m";
}

std::string AnsiKonsole::backColor(Ansi::enumColors eColor) const {
      return strEsc + lexical_cast<std::string>(eColor+40) + "m";
}

std::string AnsiKonsole::attrOff() const { //maybe buggy
      return strEsc + "0m";
}

std::string AnsiKonsole::up(int intY) const {
      return strEsc + lexical_cast<std::string>(intY) + "A";
}

std::string AnsiKonsole::down(int intY) const {
      return strEsc + lexical_cast<std::string>(intY) + "B";
}

std::string AnsiKonsole::right(int intX) const {
      return strEsc + lexical_cast<std::string>(intX) + "C";
}

std::string AnsiKonsole::left(int intX) const {
      return strEsc + lexical_cast<std::string>(intX) + "D";
}
```

main.cpp:

```
#include <iostream>
#include "Ansi.h"
using namespace std;

int main (int argc, char * const argv[]) {
	//Keine Objekterzeugung wg. Statischen Funktionen nötig!
	cout << Ansi::clear() << Ansi::home() << "Alles wurde geloescht!" << endl;
	cout << Ansi::bold() << "Dies ist in FETT geschrieben!" << endl;
	cout << Ansi::attrOff() << "Alle Attribute deaktiviert!" << endl;
	cout << Ansi::pos(15, 20) << "Springe zu Position 15/20!" << flush;
	cout << Ansi::clear() << Ansi::home() << "Loeschen und nach Home!" << endl;
	cout << Ansi::foreColor(Ansi::green) << "Alles in Gruen!" << endl;
	cout << Ansi::backColor(Ansi::red) << "Hintergrund in Rot!" << endl;
	cout << Ansi::down(17) << "17 nach unten!" << flush;
	cout << Ansi::right(30) << "30 nach rechts!" << flush;
	cout << Ansi::left(15) << Ansi::up(10) << "10 nach oben!" << flush;
	cout << Ansi::left(40) << "40 nach links!" << endl;
	cout << Ansi::home() << Ansi::attrOff() << Ansi::down(22) << endl;
}
```

Übungsserie 6

Aufgabe 2 - Primzahlen generieren (Prädikat)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>

using namespace std;

int intNumbers=1;
int generateFunc() { //Function für Generator
	return intNumbers++;
}

bool isNotPrime(int intNumber) { //Prädikat
	int intCount = 0;
	for (int i=1; i <= intNumber; i++) {
		if (intNumber % i == 0) {
			intCount++;
		}
		if (intCount > 2) {
			return true; //keine Primzahl, abbrechen
		}
	}
	return false;  //schleife komplett, ist primzahl
}

int main() {
	vector<int> refVec(100);
	generate(refVec.begin(), refVec.end(), generateFunc); //Vector füllen
	//Alternative: generate_n(back_inserter(refFec), 100, generateFunc)

	cout << "Anzahl Primzahlen: " << refVec.size() - count_if(refVec.begin(), refVec.end(),
	isNotPrime) << endl;

	remove_copy_if(refVec.begin(), refVec.end(), ostream_iterator<int>(cout, ", "),
	isNotPrime);
}
```

Aufgabe 3 - wordcount (operator<, operator>>, operator<<)

Wort.h:

```
#ifndef WORT_H_
#define WORT_H_

#include <string>
#include <iosfwd>

class Wort {
	public:
		void read(std::istream &isIn);
		void print(std::ostream &osOut) const;
		bool operator<(const Wort &refWort) const;    //Benötigt für std::set
	private:
		std::string strValue;
};

std::istream& operator>>(std::istream &isIn, Wort& refWort);
std::ostream& operator<<(std::ostream& osOut, const Wort& refWort);
#endif /*WORT_H_*/
```

Wort.cpp:

```
#include "Wort.h"
#include <iostream>
#include <string>
#include <cctype>

void Wort::read(std::istream &isIn) {
	std::string strTemp;
	isIn >> strTemp;

	if (strTemp.size() > 0) {
		int i=0;
		while (!isalpha((int) strTemp[i])) { //wortanfang suchen
			++i;
		}

		int intStart = i;
		while (isalpha((int) strTemp[i])) { //length ermitteln
			++i;
		}
		int intLength = i-intStart;
		strValue = strTemp.substr(intStart, intLength);

		/* Alternative für Kleinschreibung
		for (i = intStart; i < intStart + intLength;i++) {
			strValue += (char) tolower(strTemp[i]);
		}
		*/
	}
}

std::istream& operator>>(std::istream &isIn, Wort& refWort) {
	refWort.read(isIn);
	return isIn;
}

void Wort::print(std::ostream &osOut) const {
	osOut << strValue;
}

std::ostream& operator<<(std::ostream& osOut, const Wort& refWort) {
    refWort.print(osOut);
    return osOut;
 }

bool Wort::operator<(const Wort &refWort) const {
	return strValue < refWort.strValue;
}
```

Main.cpp

```
#include "Wort.h"
#include <iostream>
#include <iterator>
#include <vector>
#include <string>

using namespace std;

int main() {
	cout << "Geben Sie bitte Wörter ein:" << endl;
	istream_iterator<Wort> itEof;
	istream_iterator<Wort> itInp (cin);
	vector<Wort> refVec(itInp,itEof);
	cout << "Anzahl Wörter: " << refVec.size() << endl;
	copy(refVec.begin(), refVec.end(), ostream_iterator<Wort>(cout ,", "));
}
```

Aufgabe 4 – wordcount – nur verschiedene Wörter (set)

```
#include "Wort.h"
#include <iostream>
#include <iterator>
#include <set>
#include <string>

using namespace std;

int main() {
	cout << "Geben Sie bitte Wörter ein:" << endl;
	istream_iterator<Wort> itEof;
	istream_iterator<Wort> itInp (cin);
	set<Wort> refSet(itInp, itEof);
	cout << "Anzahl verschiedener Wörter: " << refSet.size() << endl;
	copy(refSet.begin(), refSet.end(), ostream_iterator<Wort>(cout, ", "));
	cout << endl << "Anzahl Woerter: " << s.size() << endl;
}
```

Aufgabe 6 – wordstat (map, pair)

```
#include <iostream>
#include <iterator>
#include <map>
#include <string>
#include <algorithm>
#include "Wort.h"
using namespace std;

void print_pair(pair<Wort,int> wpair){
	cout << wpair.first << ":    " << wpair.second << endl;
}

int main (int argc, char * const argv[]) {
	typedef istream_iterator<Wort> input;
	Wort wort;
	map<Wort,int> m;
	while(cin >> wort){
		if (m.count(wort) == 0) {
			m[wort] = 1;
		} else {
			m[wort]++;
		}
	}
	for_each(m.begin(),m.end(),print_pair);
	cout << endl << "Anzahl verschiedene Woerter: " << m.size() << endl;

	/* Alternative ohne Funktion print_pair
	for (map<Wort,int>::iterator itR = m.begin(); itR != m.end(); ++itR) {
		cout << (*itR).first << ": " << (*itR).second << endl;
	}
	*/
}
```

Übungsserie 7

Aufgabe 1 - boost/bind

```
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <boost/bind.hpp>
using namespace std;

int main (int argc, char * const argv[]) {
	vector<int> v1, v2;
	copy(istream_iterator<int>(cin), istream_iterator<int>(), back_inserter(v1));
	//remove_copy_if = ENTFERNEN (=> nicht kopieren), wenn Prädikat erfüllt ist!
	remove_copy_if(v1.begin(),v1.end(),back_inserter(v2),boost::bind(modulus<int>(), 1,3));
	cout << endl << "Die durch 3 teilbaren Zahlen: " << flush;
	copy(v2.begin(),v2.end(),ostream_iterator<int>(cout, ", "));
	transform(v2.begin(),v2.end(),v2.begin(),boost::bind(divides<int>(), 1,3));
	cout << endl << "Die durch 3 geteilten Zahlen:  " << flush;
	copy(v2.begin(),v2.end(),ostream_iterator<int>(cout, ", "));
}
```

Aufgabe 1 - Variante mit Funktoren (bzw. Prädikaten)

```
#include <iostream>
#include <vector>
#include <iterator>

using namespace std;

struct isNotDivideableByThree {  //Überladen des operator() -> Funktor (Prädikat)
	bool operator()(int intNumber) const {
		return (intNumber % 3 != 0) ? true : false;
	}
};

struct divideByThree {    //Überladen des operator() -> Funktor (Prädikat)
	int& operator()(int& intNumber) {
		intNumber = intNumber / 3;
		return intNumber;
	}
};

int main() {
	istream_iterator<int> itEof;
	istream_iterator<int> itInp(cin);
	vector<int> v1(itInp,itEof);
	vector<int> v2;
	//remove_copy_if = ENTFERNEN (=> nicht kopieren), wenn Prädikat erfüllt ist!
	remove_copy_if(v1.begin(),v1.end(),back_inserter(v2),isNotDivideableByThree());
	copy(v2.begin(),v2.end(),ostream_iterator<int>(cout,", "));
	for_each(v2.begin(),v2.end(),divideByThree());
	copy(v2.begin(),v2.end(),ostream_iterator<int>(cout,", "));
}
```

Theorie 1 - limits

Studieren Sie den Standard-Header <limits>. Wo finden Sie den auf Ihrem System?

Mac: /Developer/SDKs/MacOSX10.4u.sdk/usr/include/gcc/darwin/4.0/c++/
Windows: C:\Programme\Cygwin\usr\include\c++\3.3.3
Unix: /usr/lib/gcc/i686-pc-cygwin/3.4.4/include/c++/limits

Welche sinnvollen Fähigkeiten bieten ihnen die die Funktionen min(), max()?

Die Funktionen min() und max() geben die minimal bzw. maximal möglichen Werte eines Datentyps zurück. Nachfolgend ein Beispiel:

```
#include <iostream>
#include <limits>
using namespace std;
int main() {
      cout << numeric_limits<int>::min() << endl;   //-2147483648
      cout << numeric_limits<int>::max();           // 2147483647
}
```

Wie können Sie damit verhindern, dass eine Typkonvertierung von unsigned int nach int ein ungewöhnliches Ergebnis liefert?

```
unsigned int x = 65535;
signed int y = 0;
if(x < std::numeric_limits<signed int>::max()) { y = x; }
else { y = std::numeric_limits<signed int>::max(); }
```

Theorie 2 – typedefs konstruieren

Konstruieren Sie anhand der folgenden verbalen Beschreibungen ein typedef für den entsprechenden Typ. Versuchen Sie es zuerst ohne Unterstützung durch den Compiler und prüfen Sie ihre Antwort anschliessend.

Ein Zeiger auf einen double Wert, wobei dieser double Wert nicht geändert werden kann.

- `typedef double const *zeigerAufDouble;`
- `typedef const double *zeigerAufDouble; //identisch`

Ein Zeiger, der nicht geändert werden kann auf ein Objekt der Klasse Rational.

- `typedef Rational * const zeigerAufRational;`

Ein Funktionszeiger auf eine Funktion, die einen Zeiger auf konstante char liefert und eine konstante Referenz auf einen std::string als ersten Parameter und einen int-Wert als zweiten Parameter hat. (mittelschwer)

- `typedef char const * (*funktionsZeiger)(std::string const &,int);`
- `typedef char const * (*funktionsZeiger)(const std::string &,int); //identisch`

Ein std::vector mit Zeigern auf Funktionen, die int liefern und einen double Parameter per const Referenz haben (schwer)

- `typedef std::vector<int (*)(double const &)> vectorMitFunktionszeigern;`
- `typedef int (*fktzeiger)(double const &); //zweistufige Alternative`
- `typedef std::vector<fktzeiger> vectorMitFunktionszeigern;`

Ein Funktionszeiger auf eine Funktion, die keine Parameter hat und keinen Wert zurückliefert.

- `typedef void(*fktzeiger)();`

Theorie 3 – Deklarationen/Definitionen verbal beschreiben

Beschreiben Sie verbal die folgenden Deklarationen/Definitionen und geben Sie an, ob es sich dabei um eine Deklaration oder eine Definition handelt.

Deklaration:

Man gibt dem Compiler an, das ein bestimmter Typ (class/struct) oder eine bestimmte Funktion (Signatur) existiert, ohne Details der Implementierung anzugeben. Für Typen kann man damit schon gültige Zeiger und Referenzen definieren. Funktionen kann man aufrufen (jede Funktion muss vor ihrer Benutzung deklariert oder definiert sein).

Definition

Jede Definition ist auch eine Deklaration. *Eine Definition gibt für einen Namen vollständig an, wie dieser definiert ist. Der Name kann einen Typ, Variable, Funktion beschreiben. Es gilt das One-Definition-Rule: für Typen, Variablen und Funktionen gilt, dass sie nur genau einmal definiert sein können. Bei typedef sind mehrfache identische Definitionen möglich (unterschiedliche sind ein Fehler).*

Erste (schwer):

```
struct Eins{
   const char *s1;
   char const* s2;
   char * const s3;
} eins;
```

Definition des Datentyps Eins und der Variablen eins vom Typ Eins.

Zweite:

```
typedef int const * const zwei;
```

typedefs sind Definitionen (Fehler auf einer der Vorlesungsfolien). zwei ist eine Zeigerkonstante auf konstante ints.

Dritte:

```
double drei(const double &,int, bool &);
```

Deklaration der Funktion drei, die irgendwo anders implementiert sein kann. Sie muss implementiert sein, wenn drei auch aufgerufen wird. Der Linker sucht die Funktion drei dann in allen angegebenen Objekt-Dateien und Bibliotheken.

Vierte:

```
int * const * vier = 0;
```

Definition der Zeigervariablen vier, die auf Zeigerkonstanten auf variable ints verweist.

Fuenfte:

```
char const **argv;
```

Definition der Variablen argv, die als Zeiger auf Zeiger auf konstante chars definiert ist.
Als Parameter wäre diese Definition äquivalent zu
Void foo(char const * argv[])

Sechste:

```
int main(int argc,char *argv[]);
```

Deklaration der main Funktion. In Zeigerschreibweise wäre folgendes äquivalent wegen der Degenerierung eines Array-Parameters zum Zeiger:
int main(int argc,char ** argv);

Aufgabe 4 – operator double(), operator+, operator<<

Zeit.h:

```
#ifndef ZEIT_H
#define ZEIT_H
#include <iostream>

class Zeit {
	public:
		Zeit(int h, int m, int s);
		int getStunden() const;
		int getMinuten() const;
		int getSekunden() const;
		Zeit operator+(const Zeit& refRight) const;
		operator double() const;  //Cast-Operator überladen
	private:
		int stunden, minuten, sekunden;
};

std::ostream& operator<<(std::ostream& osOut, const Zeit& refZeit);	//Globale Funktion
#endif //ZEIT_H
```

Zeit.cpp:

```
#include "Zeit.h"
#include <iostream>
using namespace std;

Zeit::Zeit(int h, int m, int s) : stunden(h), minuten(m), sekunden(s) {}
int Zeit::getStunden() const {   return stunden; }
int Zeit::getMinuten() const {   return minuten; }
int Zeit::getSekunden() const {  return sekunden; }

Zeit Zeit::operator+(const Zeit& refRight) const {
	int intStunden = stunden + refRight.getStunden();
	int intMinuten = minuten + refRight.getMinuten();
	int intSekunden = sekunden + refRight.getSekunden();
	int intAdd = 0;

	if (intSekunden > 60) {
		intAdd = intSekunden / 60;
		intMinuten += intAdd;
		intSekunden %= 60;
	}

	if (intMinuten > 60) {
		intAdd = intMinuten / 60;
		intStunden += intAdd;
		intMinuten %= 60;
	}

	Zeit refReturn(intStunden, intMinuten, intSekunden);

	return refReturn;
}

Zeit::operator double() const {
	double dblReturn(stunden);
	dblReturn += (double(minuten)/60);
	dblReturn += (double(sekunden)/3600);
	return dblReturn;
}

ostream& operator<<(ostream& osOut, const Zeit& refZeit) {
	osOut << refZeit.getStunden() << ":" << refZeit.getMinuten() << ":" <<
	refZeit.getSekunden();
	return osOut;
}
```

main.cpp:

```
#include <iostream>
#include "Zeit.h"
using namespace std;

int main (int argc, char * const argv[]) {
	Zeit z1(10, 2, 5);
	Zeit z2(27, 59, 37);
	cout << z1 << endl;    // Ausgabe: 10:2:5
	cout << z1 + z2 << endl; // Ausgabe: 38:1:42
	cout << "Zeit in Stunden: " << (double)z1 << endl; // Ausgabe: 10.0347
}
```

Übungsserie 8

Aufgabe 0-1 – Fileread/-write (SharedPtr, Argc, Argv)

```
#include <fstream>
#include <iostream>
#include <boost/shared_ptr.hpp>
using namespace std;
typedef boost::shared_ptr<ostream> FstrPtr;

FstrPtr createFile(string filename){
	FstrPtr file( new ofstream(filename.c_str(), ios::app) );
	if (!*file) throw "file unwritable";
	// don't return unusable file
	return file;
}

int main(int argc, char* argv[]){
	try {
		string argument;
		if(argc > 1)
		{
			argument = argv[1];
		}
		else
		{
			argument = "dudu.txt";
		}
		FstrPtr file = createFile(argument);
		(*file) << "Hello world\n";
	}
	catch (char const *&e) {
		cerr << e << std::endl;
	}
 }
```

Aufgabe 1 - Objekt-Initialisierung (Konstruktor/Destruktor)

```
#include <iostream>

class Objekt{
	public:
		Objekt();
		~Objekt();
};
Objekt::Objekt(){   std::cout << "Initialisierung" << std::endl; }
Objekt::~Objekt(){  std::cout << "Aufraeumen" << std::endl; }

Objekt objekt;      //Globale Objekte werden vor main() erstellt, nach main() augegräumt!

int main (int argc, char * const argv[]) {
    std::cout << "Hello, World!\n";
}
```

Aufgabe 2 - Rational (operator double(), operator+(int))

Main.cpp

```
#include <iostream>
#include "Rational.h"
int main() {
	Rational r1(7,3);
	Rational r2(19,2);

	double d = r1 + 2;
	int i = r1 + 0.7;

	std::cout << d << std::endl;
	std::cout << i << std::endl;
}
```

Rational.h

```
#ifndef RATIONAL_H_
#define RATIONAL_H_
#include <iosfwd>
class Rational {
	public:
		Rational(long lngDefZaehler,long lngDefNenner);
		void print(std::ostream &osOut) const;
		void read(std::istream &osInt);
		Rational operator+(Rational refSecond);
		Rational& operator+=(const Rational& refSecond);
		Rational operator+(int intSecond);      //Rational + Int
		Rational operator+(double dblSecond);   //Rational + Double
		Rational operator-(Rational refSecond);
		Rational& operator-=(const Rational& refSecond);
		Rational operator*(Rational refSecond);
		Rational& operator*=(const Rational& refSecond);
		Rational operator/(Rational refSecond);
		Rational& operator/=(const Rational& refSecond);
		operator double() const;   //Cast Operator für double
		operator float() const;    //Cast Operator für float
		operator int() const;      //Cast Operator für int
	private:
		long ggt(long lngZahl1, long lngZahl2);
		void normalize(Rational& refNumber);
		long lngZaehler;
		long lngNenner;
};
//Output Operator muss globale Funktion sein, da ostream nicht kopiert werden kann
std::ostream& operator<<(std::ostream &osOut, const Rational& refRational);
#endif /*RATIONAL_H_*/
```

Rational.cpp:

```
#include <iostream>
#include <cmath>
#include <cctype>
#include "Rational.h"

Rational::Rational(long lngDefZaehler=1,long lngDefNenner=1) :
	lngZaehler(lngDefZaehler), lngNenner(lngDefNenner) {
	try {
		if (lngNenner == 0) {
			throw "Division durch 0";
		}
	} catch (std::string &refEx) {
		lngNenner = 1;
		std::cout << refEx << std::endl;
	}
	normalize(*this);
}

void Rational::print(std::ostream &osOut) const {
	osOut << lngZaehler << '/' << lngNenner;
}

std::ostream& operator<<(std::ostream &osOut, const Rational& refRational) {
	refRational.print(osOut);
	return osOut;
}

void Rational::normalize(Rational& refNumber) {
	//calculate ggt first
	long lngGGT = ggt(refNumber.lngZaehler,refNumber.lngNenner);
	refNumber.lngZaehler /= lngGGT;
	refNumber.lngNenner /= lngGGT;
}

long Rational::ggt(long lngZahl1, long lngZahl2) {
	//Found on: http://www.datasource.de/programmierung/toolbox_cpp_03.html
	long lngTemp = 0;
	lngZahl1 = abs(lngZahl1);
	lngZahl2 = abs(lngZahl2);

	if (lngZahl1 < lngZahl2) {
		lngTemp = lngZahl1;
		lngZahl1 = lngZahl2;
		lngZahl2 = lngTemp;
	}

	while (lngZahl2 > 0) {
	lngTemp = lngZahl2;
	lngZahl2 = lngZahl1 % lngZahl2;
	lngZahl1 = lngTemp;
	}

	return lngZahl1;
}

Rational Rational::operator+ (Rational refSecond) {
	//Memberfunktion, this ist implizit erstes Argument
	normalize(refSecond += *this);
	return refSecond;
}

Rational& Rational::operator+=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngNenner) + (refSecond.lngZaehler * lngNenner);
	lngNenner *= refSecond.lngNenner;
	normalize(*this);
	return *this;//Bei Zuweisungen aktuelles Objekt zurückliefern
}
```

```
Rational Rational::operator+(int intSecond) {
	Rational refReturn(intSecond);
	normalize(refReturn += *this);
	return refReturn;
}

Rational Rational::operator+(double dblSecond) {
	Rational refReturn((long) dblSecond);
	normalize(refReturn += *this);
	return refReturn;
}

Rational Rational::operator-(Rational refSecond) {
	normalize(refSecond -= *this);
	return refSecond;
}

Rational& Rational::operator-=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngNenner) - (refSecond.lngZaehler * lngNenner);
	lngNenner *= refSecond.lngNenner;
	normalize(*this);
	return *this;
}

Rational Rational::operator*(Rational refSecond) {
	normalize(refSecond *= *this);
	return refSecond;
}

Rational& Rational::operator*=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngZaehler);
	lngNenner *= refSecond.lngNenner;
	normalize(*this);
	return *this;
}

Rational Rational::operator/(Rational refSecond) {
	normalize(refSecond /= *this);
	return refSecond;
}

Rational& Rational::operator/=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngNenner);
	lngNenner *= refSecond.lngZaehler;
	normalize(*this);
	return *this;
}

Rational::operator double() const {
	return (double) this->lngZaehler / this->lngNenner;
}

Rational::operator float() const {
	return (float) this->lngZaehler / this->lngNenner;
}

Rational::operator int() const {
	return (int) this->lngZaehler / this->lngNenner;
}
```

Aufgabe 3 - shared_ptr example

```
#include <vector>
#include <set>
#include <iostream>
#include <algorithm>
#include <boost/shared_ptr.hpp>

//  The application will produce a series of objects of type Foo which later must be
//  accessed both by occurrence (std::vector) and by ordering relationship (std::set).

struct Foo {
      Foo( int _x ) : x(_x) {}
      ~Foo() { std::cout << "Destructing a Foo with x=" << x << "\n"; }
      int x;
      /* ... */
};

typedef boost::shared_ptr<Foo> FooPtr;

struct FooPtrOps {
      bool operator()( const FooPtr & a, const FooPtr & b ) { return a->x > b->x; }
      void operator()( const FooPtr & a ) { std::cout << a->x << ", "; }
};

int main() {
      std::vector<FooPtr> foo_vector;
      std::set<FooPtr,FooPtrOps> foo_set; // NOT multiset!

      FooPtr foo_ptr( new Foo(2) );
      foo_vector.push_back( foo_ptr );
      foo_set.insert( foo_ptr );

      foo_ptr.reset( new Foo(1) );     //Referenzierung von foo_ptr löschen (und neu setzen)
      foo_vector.push_back( foo_ptr );
      foo_set.insert( foo_ptr );

      foo_ptr.reset( new Foo(3) );
      foo_vector.push_back( foo_ptr );
      foo_set.insert( foo_ptr );

      foo_ptr.reset ( new Foo(2) );
      foo_vector.push_back( foo_ptr );
      foo_set.insert( foo_ptr );

      std::cout << "foo_vector:\n";
      std::for_each( foo_vector.begin(), foo_vector.end(), FooPtrOps() );

      std::cout << "\nfoo_set:\n";
      std::for_each( foo_set.begin(), foo_set.end(), FooPtrOps() );
      std::cout << "\n";

      //   Expected output:
      //
      //   foo_vector:
      //   2, 1, 3, 2,
      //
      //   foo_set:
      //   3, 2, 1,
      //
      //   Destructing a Foo with x=2
      //   Destructing a Foo with x=1
      //   Destructing a Foo with x=3
      //   Destructing a Foo with x=2

      return 0;
}
```

Übungsserie 9

Aufgabe 1 - Der Zoo (Vererbung, Virtual, PIMPL)

Main.cpp

```
#include "Mensch.h"
#include "Tiergruppe.h"
#include "Katze.h"
#include "Hund.h"
#include "Schlange.h"
#include "Papagei.h"
#include <iostream>

using namespace std;

int main() {
	Mensch ich("Ich",22,'M');
	Mensch tochter("Tochter",20,'W');
	Mensch sohn("Sohn",19,'M');

	Tiergruppe familie("Haustiere",&ich);

	Tiergruppe exoten("Exoten",&tochter);
	familie.addiereTier(&exoten);

	Tiergruppe normale("Normale",&sohn);
	familie.addiereTier(&normale);

	Hund Max("Max",&ich,"Labrador");
	familie.addiereTier(&Max);

	Schlange Jacky("Jacky",0,200);
	exoten.addiereTier(&Jacky);

	Katze Che("Che",0,"Braun");
	normale.addiereTier(&Che);

	familie.gibLaut();
	cout << "Der Besitzer von " << Max.getName() << " ist " << *Max.getBesitzer();
	cout << "Der Besitzer von " << Jacky.getName() << " ist " << *Jacky.getBesitzer();
	cout << "Der Besitzer von " << Che.getName() << " ist " << *Che.getBesitzer() << endl;

	cout << "Lösche Katze Che…" << endl;
	normale.loeseTier("Che");
	cout << endl << "Gebt laut, ihr Tiere...! :-)" << endl;
	familie.gibLaut();
	cout << endl;

	Papagei Polly("Polly",&sohn,"Nymphensittich");
	exoten.addiereTier(&Polly);
	familie.gibLaut();
	cout << "Der Besitzer von " << Polly.getName() << " ist " << *Polly.getBesitzer();
}
```

Mensch.h:

```
#ifndef MENSCH_H_
#define MENSCH_H_
#include <string>
#include <iosfwd>

class Mensch {
	public:
		Mensch(std::string _strName, int _intAlter, char _chrGeschlecht);
		void print(std::ostream & osOut) const;
		std::string getName() const;
		int getAlter() const;
		char getGeschlecht() const;
	private:
		std::string strName;
		int intAlter;
		char chrGeschlecht;
};
std::ostream & operator<<(std::ostream & osOut, Mensch const & refObj);
#endif /*MENSCH_H_*/
```

Mensch.cpp:

```
#include "Mensch.h"
#include <string>
#include <iostream>

Mensch::Mensch(std::string _strName, int _intAlter, char _chrGeschlecht) :
	strName(_strName), intAlter(_intAlter), chrGeschlecht(_chrGeschlecht) {}

void Mensch::print(std::ostream & osOut) const {
	osOut << getName() << " (" << getGeschlecht() << ", " << getAlter() << " Jahre)";
}

std::string Mensch::getName() const {
	return this->strName;
}

int Mensch::getAlter() const {
	return this->intAlter;
}

char Mensch::getGeschlecht() const {
	return this->chrGeschlecht;
}

std::ostream & operator<<(std::ostream & osOut, Mensch const & refMensch) {
	refMensch.print(osOut);
	return osOut;
}
```

HaustierBasis.h:

```
#ifndef HAUSTIERBASIS_H_
#define HAUSTIERBASIS_H_
#include "Mensch.h"
#include <string>
class HaustierBasis {
	public:
		HaustierBasis();
		HaustierBasis(std::string _strName, Mensch * _pBesitzer);
		virtual ~HaustierBasis(); //virtueller Destruktor wegen virtueller Methode!
		virtual void gibLaut() = 0; //Rein virtuelle Methode, kein Impl. in Basisklasse
		void setName(std::string const & strName);
		std::string getName() const;
		void setBesitzer(Mensch * pBesitzer);
		Mensch * getBesitzer() const;
	protected:
		std::string strName;
	private:
		Mensch * pBesitzer;
		HaustierBasis(HaustierBasis const & refHaustier);
		//Kopierkonstruktor: Private verhindert „klonen"
		HaustierBasis & operator=(HaustierBasis const &);
		//Zuweisungsoperator: Private verhindert „klonen"
};
#endif /*HAUSTIERBASIS_H_*/
```

HaustierBasis.cpp:

```
#include "HaustierBasis.h"
#include "Mensch.h"
#include <iostream>
#include <string>

HaustierBasis::HaustierBasis() : strName("unbekannt"), pBesitzer(0) {}
HaustierBasis::HaustierBasis(std::string _strName, Mensch * _pBesitzer) :
                          strName(_strName), pBesitzer(_pBesitzer) {}

HaustierBasis::~HaustierBasis() {
	//std::cout << this->getName() << " ist gestorben." << std::endl;
}

void HaustierBasis::setName(std::string const & strName) {
	this->strName = strName;
}

std::string HaustierBasis::getName() const {
	return this->strName;
}

void HaustierBasis::setBesitzer(Mensch * pBesitzer) {
	this->pBesitzer = pBesitzer;
}

Mensch * HaustierBasis::getBesitzer() const {
	return this->pBesitzer;
}
```

Tiergruppe.h:

```
#ifndef TIERGRUPPE_H_
#define TIERGRUPPE_H_
#include "HaustierBasis.h"
#include "Mensch.h"
#include <map>
#include <string>

class Tiergruppe : public HaustierBasis {
	public:
		Tiergruppe();
		Tiergruppe(std::string _strName, Mensch * _pBesitzer);
		virtual ~Tiergruppe(); //Virtuelle Methode = Virtueller Konstruktor
		virtual void gibLaut();
		void addiereTier(HaustierBasis * pHaustierBasis);
		HaustierBasis * loeseTier(std::string const & strName);
		HaustierBasis * getTier(std::string const & strName);
	private:
		HaustierBasis * findeTier(std::string const & strName);
		std::map<std::string, HaustierBasis *> mapTiere;
		Tiergruppe(Tiergruppe const & refTiergruppe);
		//Kopierkonstruktor: Private verhindert „klonen"
		Tiergruppe & operator=(Tiergruppe const &);
		//Zuweisungsoperator: Private verhindert „klonen"
};
#endif /*TIERGRUPPE_H_*/
```

Tiergruppe.cpp:

```
#include "Tiergruppe.h"
#include "HaustierBasis.h"
#include "Mensch.h"
#include <map>
#include <string>
#include <iostream>

Tiergruppe::Tiergruppe() : HaustierBasis("unbekannt",0) {}
Tiergruppe::Tiergruppe(std::string _strName, Mensch * _pBesitzer) :
	HaustierBasis(_strName,_pBesitzer) {}

Tiergruppe::~Tiergruppe() {}

void Tiergruppe::gibLaut() {
	std::map<std::string, HaustierBasis *>::iterator itRun;
	for (itRun = this->mapTiere.begin(); itRun != this->mapTiere.end(); itRun++) {
		itRun->second->gibLaut();
	}
}

void Tiergruppe::addiereTier(HaustierBasis * pHaustierBasis) {
	if (!pHaustierBasis->getBesitzer()) {
		pHaustierBasis->setBesitzer( this->getBesitzer() );
	}
	this->mapTiere[pHaustierBasis->getName()] = pHaustierBasis;
}

HaustierBasis * Tiergruppe::loeseTier(std::string const & strName) {
	std::map<std::string,HaustierBasis *>::iterator itRun = this->mapTiere.find(strName);
	HaustierBasis * pReturn = itRun->second;
	this->mapTiere.erase(strName);
	return pReturn;
	return 0;
}

HaustierBasis * Tiergruppe::getTier(std::string const & strName) {
	return this->mapTiere[strName];
}
```

Katze.h:

```
#ifndef KATZE_H_
#define KATZE_H_
#include "HaustierBasis.h"
#include "Mensch.h"
#include <string>

class Katze : public HaustierBasis {
	public:
		Katze(std::string _strName, Mensch * _pBesitzer, std::string _strFarbe);
		virtual ~Katze();
		virtual void gibLaut();
		std::string getFarbe() const;
	private:
		std::string strFarbe;
		Katze(Katze const &);
		//Kopierkonstruktor: Private verhindert klonen
		Katze & operator=(Katze const &);
		//Zuweisungsoperator: Private verhindert klonen
};
#endif /*KATZE_H_*/
```

Katze.cpp:

```
#include "Katze.h"
#include "Mensch.h"
#include "HaustierBasis.h"
#include <string>
#include <iostream>

Katze::Katze(std::string _strName, Mensch * _pBesitzer, std::string _strFarbe) :
			HaustierBasis(_strName,_pBesitzer), strFarbe(_strFarbe) {}

Katze::~Katze() {}

void Katze::gibLaut() {
	std::cout << "Miau" << std::endl;
}

std::string Katze::getFarbe() const {
	return this->strFarbe;
}
```

Hund.h:

```
#ifndef HUND_H_
#define HUND_H_
#include "HaustierBasis.h"
#include "Mensch.h"
#include <string>

class Hund : public HaustierBasis {
	public:
		Hund(std::string _strName, Mensch * _pBesitzer, std::string _strRasse);
		virtual ~Hund();
		virtual void gibLaut();
		std::string getRasse() const;
	private:
		std::string strRasse;
		Hund(Hund const & refHund);
		//Kopierkonstruktor: Private verhindert klonen
		Hund & operator=(Hund const &);
		//Zuweisungsoperator: Private verhindert klonen
};
#endif /*HUND_H_*/
```

Hund.cpp:

```
#include "Hund.h"
#include "HaustierBasis.h"
#include <string>
#include <iostream>

Hund::Hund(std::string _strName, Mensch * _pBesitzer, std::string _strRasse) :
                    HaustierBasis(_strName, _pBesitzer), strRasse(_strRasse) {}

Hund::~Hund() {}

void Hund::gibLaut() {
      std::cout << "Wuff" << std::endl;
}

std::string Hund::getRasse() const {
      return this->strRasse;
}
```

Schlange.h:

```
#ifndef SCHLANGE_H_
#define SCHLANGE_H_
#include "HaustierBasis.h"
#include "Mensch.h"
#include <string>

class Schlange : public HaustierBasis {
      public:
            Schlange(std::string _strName, Mensch * _pBesitzer, double _dblLaenge);
            virtual ~Schlange();
            virtual void gibLaut();
            double getLaenge() const;
      private:
            double dblLaenge;
            Schlange(Schlange const &);
            //Kopierkonstruktor: Private verhindert klonen
            Schlange & operator=(Schlange const &);
            //Zuweisungsoperator: Private verhindert klonen
};
#endif /*SCHLANGE_H_*/
```

Schlange.cpp:

```
#include "Schlange.h"
#include "HaustierBasis.h"
#include <iostream>
#include <string>

Schlange::Schlange(std::string _strName, Mensch * _pBesitzer, double _dblLaenge) :
                                  HaustierBasis(_strName,_pBesitzer), dblLaenge(_dblLaenge) {}

Schlange::~Schlange() {}

void Schlange::gibLaut() {
      std::cout << "Zisch" << std::endl;
}

double Schlange::getLaenge() const {
      return this->dblLaenge;
}
```

Papagei.h:

```
#ifndef PAPAGEI_H_
#define PAPAGEI_H_
#include "Mensch.h"
#include "HaustierBasis.h"
#include <string>
#include <boost/shared_ptr.hpp>

class PapageiImpl;

class Papagei : public HaustierBasis {
	public:
		Papagei(std::string _strName, Mensch * _pBesitzer, std::string _strArt);
		virtual ~Papagei();
		virtual void gibLaut();
		std::string getName() const;
		std::string getArt() const;
	private:
		boost::shared_ptr<PapageiImpl> pPapagei; //PIMPL-Idiom mit shared_ptr
		Papagei(Papagei const & refPapagei);
		//Kopierkonstruktor: Private verhindert klonen
		Papagei & operator=(Papagei const &);
		//Zuweisungsoperator: Private verhindert klonen
};
#endif /*PAPAGEI_H_*/
```

Papagei.cpp:

```
#include "Papagei.h"
#include <iostream>
#include <string>
using namespace std;

// PapageiImpl ------------------------------------------------------------------------------
class PapageiImpl {
	public:
		PapageiImpl(std::string _strName, Mensch * _pBesitzer, std::string _strArt);
		~PapageiImpl();
		void gibLaut();
		std::string getArt() const;
	private:
		std::string strArt;
};
PapageiImpl::PapageiImpl(std::string _strName, Mensch * _pBesitzer, std::string _strArt)
		:	HaustierBasis(_strName, _pBesitzer), strArt(_strArt);
PapageiImpl::~PapageiImpl() {}
void PapageiImpl::gibLaut() { std::cout << "Kraaa! C++ sucks! Kraaa!" << std::endl; }
std::string PapageiImpl::getArt() const { return this->strArt; }
// PapageiImpl ------------------------------------------------------------------------------

Papagei::Papagei(std::string _strName, Mensch * _pBesitzer, std::string _strArt) :
	pPapagei ( new PapageiImpl( strName, pBesitzer, strArt) ) { }

Papagei::~Papagei() {}

void Papagei::gibLaut() {
	pPapagei->gibLaut(); //Propagieren ins PapageiImpl
}

std::string Papagei::getName() const {
	return pPapagei->getName(); //Propagieren ins PapageiImpl
}

std::string Papagei::getArt() const {
	return pPapagei->getArt(); //Propagieren ins PapageiImpl
}
```

Übungsserie 11

Aufgabe 1 - Formatierung (iomanip, setfill, setw, left, …)

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

using namespace std;

int main() {
	int intInteger = 12345;
	double dblDouble = -1234.56789;
	string strString = "abcdefghiklmnop";

	cout << "Standardformat" << endl;
	cout << intInteger << endl;
	cout << dblDouble << endl;
	cout << strString << endl << endl;

	cout << "Breite 12 Zeichen, Füllzeichen '_'" << endl;
	cout << setfill(' ');
	cout << setw(12) << intInteger << endl;
	cout << setw(12) << dblDouble << endl;
	cout << setw(12) << strString << endl << endl;

	cout << "dito, aber linksbündig" << endl;
	cout << setfill('_') << left;
	cout << setw(12) << intInteger << endl;
	cout << setw(12) << dblDouble << endl;
	cout << setw(12) << strString << endl << endl;

	cout << "12 stellige Genauigkeit, Breite 20 Zeichen" << endl;
	cout << setfill(' ') << right << fixed << setprecision(12);
	cout << setw(20) << intInteger << endl;
	cout << setw(20) << dblDouble << endl;
	cout << setw(20) << strString << endl << endl;

	cout << "dito, aber wissenschaftliches Format" << endl;
	cout << scientific;
	cout << setw(20) << intInteger << endl;
	cout << setw(20) << dblDouble << endl;
	cout << setw(20) << strString << endl << endl;

	cout << "dito, aber nur noch 3 stellige Genauigkeit" << endl;
	cout << setprecision(3);
	cout << setw(20) << intInteger << endl;
	cout << setw(20) << dblDouble << endl;
	cout << setw(20) << strString << endl << endl;

	cout << "dito, intern bündig" << endl;
	cout << internal;
	cout << setw(20) << intInteger << endl;
	cout << setw(20) << dblDouble << endl;
	cout << setw(20) << strString << endl << endl;

	cout << "dito, aber Hex mit Präfix, und 0 als Füllzeichen" << endl;
	cout << hex << showbase << setfill('0');
	cout << setw(20) << intInteger << endl;
	cout << setw(20) << dblDouble << endl;
	cout << setw(20) << strString << endl << endl;
}
```

Aufgabe 2 – Binärwerte erzeugen (iomanip, lexical_cast)

Bin.h:

```
#ifndef BIN_H_
#define BIN_H_
#include <string>

class Bin {
	public:
		Bin(int intInitial);
		std::string toBin();
	private:
		int intValue;
};

#endif /*BIN_H_*/
```

Bin.cpp:

```
#include "Bin.h"
#include <iostream>
#include <iomanip>
#include <string>
#include "boost/lexical_cast.hpp"

Bin::Bin(int intInitial) : intValue(intInitial) {}

std::string Bin::toBin() {
	std::cout << std::setw(32);
	std::cout << std::setfill('0');

	std::string strReturn = "";
	int intTemp = this->intValue;

	while(true) {
		strReturn.append( boost::lexical_cast<std::string>(intTemp % 2) );
		if (intTemp == 0) {
			return strReturn;
		} else {
			intTemp = intTemp / 2;
		}
	}
}
```

main.cpp:

```
#include "Bin.h"
#include <iostream>

using namespace std;

int main() {
	Bin theBinaryString(1234);
	cout << theBinaryString.toBin() << endl;
	//00000000000000000000010011010010
	return 0;
}
```

Aufgabe 3 - File Parsing (lexical_cast, tokenizer, iomanip)

Content.h:

```
#ifndef CONTENT_H_
#define CONTENT_H_
#include <string>
#include <iosfwd>
class Content {
            public:
                  Content(bool _isDir, std::string _strDate, std::string _strTime,
                              std::string _strFileSize, std::string _strFileName);
                  bool isDirectory() const;
                  std::string getDirOrFile() const;
                  std::string getDate() const;
                  std::string getTime() const;
                  std::string getFileSize() const;
                  std::string getFileName() const;
                  void print(std::ostream& osOut) const;
                  void read(std::istream& isIn);
            private:
                  bool boolDir;
                  std::string strDate;
                  std::string strTime;
                  std::string strFileSize;
                  std::string strFileName;
                  std::string cleanFileSize(std::string strInput) const;

};
std::ostream& operator<<(std::ostream& osOut, const Content& refContent);
std::istream& operator>>(std::istream& isIn, Content& refContent);
#endif /*CONTENT_H_*/
```

Content.cpp:

```
#include "Content.h"
#include <string>
#include <iostream>
#include <iomanip>
#include <cmath>
#include "boost/lexical_cast.hpp"
#include "boost/tokenizer.hpp"

Content::Content(bool _isDir, std::string _strDate, std::string _strTime, std::string
      _strFileSize, std::string _strFileName) :
      boolDir(_isDir), strDate(_strDate), strTime(_strTime),
      strFileSize(cleanFileSize(_strFileSize)), strFileName(_strFileName){}

std::string Content::cleanFileSize(std::string strInput) const {
      std::string strTemp;

      for (unsigned int i=0; i < strInput.length(); ++i) {
            if (strInput[i] != '\'') {
                  strTemp.append( boost::lexical_cast<std::string>(strInput[i]) );
            }
      }

      double dblTemp = boost::lexical_cast<double>(strTemp);
      dblTemp /= 1024;
      dblTemp = ceil(dblTemp);
      strTemp = boost::lexical_cast<std::string>(dblTemp);
      return strTemp;
}

bool Content::isDirectory() const {
      if (this->boolDir) { return true; }
      else { return false; }
}
```

```
std::string Content::getDirOrFile() const {
	if (this->isDirectory()) { return "Verzeichnis"; }
	else { return "Datei"; }
}

std::string Content::getDate() const { return this->strDate; }

std::string Content::getTime() const { return this->strTime; }

std::string Content::getFileSize() const { return this->strFileSize + " kB"; }

std::string Content::getFileName() const {
	std::string strReturn;
	strReturn = this->strFileName;
	if (strReturn.length() > 20) {
		strReturn = strReturn.substr(0,17);
		strReturn.append("...");
	}
	return strReturn;
}

void Content::print(std::ostream& osOut) const {
	osOut << std::left << std::setw(20) << this->getFileName() << "\t";
	osOut << std::right << std::setw(10) << this->getFileSize() << "\t";
	osOut << std::left << std::setw(11) << this->getDirOrFile() << std::right;
}

void Content::read(std::istream& isIn) {
	using namespace boost;

	std::string strRead;
	char_separator<char> chrSeperator("\x0D\x20");	//Trennzeichen
	std::getline(isIn,strRead);
	tokenizer< char_separator<char> > refTok(strRead,chrSeperator);

	std::vector<string> vecValues(5);
	int intCount = 0;
	for(tokenizer< char_separator<char> >::iterator it=refTok.begin(); it != refTok.end();
	++it) {
		vecValues[intCount] = *it;
		++intCount;
	}

	if (vecValues[2] == "<DIR>") {
		this->boolDir = true;
		this->strDate = arrValues[0];
		this->strTime = arrValues[1];
		this->strFileSize = "0";
		this->strFileName = arrValues[3];
	} else {
		this->boolDir = false;
		this->strDate = arrValues[0];
		this->strTime = arrValues[1];
		this->strFileSize = this->cleanFileSize(arrValues[2]);
		this->strFileName = arrValues[3];
	}
}

std::ostream& operator<<(std::ostream& osOut, const Content& refContent) {
	refContent.print(osOut);
	return osOut;
}

std::istream& operator>>(std::istream& isIn, Content& refContent) {
	refContent.read(isIn);
	return isIn;
}
```

FileOperator.h

```
#ifndef FILEOPERATOR_H_
#define FILEOPERATOR_H_
#include "Content.h"
#include <string>
#include <vector>

class FileOperator {
	public:
		FileOperator(std::string _strFileName);
		void printAll() const;
		void printRow(unsigned int intRow) const;
		void writeFile() const;
	private:
		std::string strFileName;
		std::vector<Content> vecFiles;
		void readFile();
};

//Funktor zum sortieren nach Name
struct NameCompare : public std::binary_function<Content &,Content &,bool> {
	bool operator()(Content & refLeft, Content & refRight) {
		return refLeft.isDirectory() < refRight.isDirectory();
	}
};


#endif /*FILEOPERATOR_H_*/
```

FileOperator.cpp:

```
#include "FileOperator.h"
#include "Content.h"
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
#include <iterator>
#include <fstream>
#include "boost/tokenizer.hpp"

FileOperator::FileOperator(std::string _strFileName) : strFileName(_strFileName) {
	this->readFile();
}

void FileOperator::readFile() {
	using namespace boost;

	std::ifstream inpFile( this->strFileName.c_str() );

	std::string strRead;
	char_separator<char> chrSeperator("\x0D\x20");	//Trennzeichen

	while (std::getline(inpFile,strRead)) {
		tokenizer< char_separator<char> > refTok(strRead,chrSeperator);

		std::vector<string> vecValues[5];
		int intCount = 0;
		for(tokenizer< char_separator<char> >::iterator it=refTok.begin(); it !=
		refTok.end(); ++it) {
			vecValues[intCount] = *it;
			++intCount;
		}

		if (arrValues[2] == "<DIR>") {
			this->vecFiles.push_back(
				Content(true,arrValues[0],arrValues[1],"0",arrValues[3]));
		} else {
```

```
                    this->vecFiles.push_back(
                          Content(false,arrValues[0],arrValues[1],arrValues[2],arrValues[3]));
             }
       }
}



void FileOperator::printAll() const {
       std::ostream_iterator<Content> out(std::cout,"\n");
       copy(this->vecFiles.begin(), this->vecFiles.end(), out);
       std::cout << std::endl;
}

void FileOperator::printRow(unsigned int intRow) const {
       if (intRow < this->vecFiles.size()) {
             std::cout << this->vecFiles[intRow-1] << std::endl;
             std::cout << std::endl;
       } else {
             std::cout << "Ungueltiger Index." << std::endl;
             std::cout << std::endl;
       }
}

void FileOperator::writeFile() const {
       std::ofstream outFile("output.txt");
       sort(this->vecFiles.begin(), this->vecFiles.end(), NameCompare());
}
```

main.cpp:

```
#include "FileOperator.h"
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <iterator>

using namespace std;

int main() {
       FileOperator objFile("Mydir2.txt");
       objFile.printAll();
       objFile.printRow(7);
       objFile.printRow(17);
       objFile.printRow(38);
}
```

Theorie 1 - shared_ptr

Folgende Situation: Ein erster shared ptr zeigt auf einen int. Ein zweiter und ein dritter shared ptr zeigen ebenfalls auf denselben int.

Wie hoch ist der Zählerstand nun bei jedem der drei shared_ptrs?

Es ist wichtig, dass zum Initialisieren des zweiten und des dritten shared_ptr das erste shared_ptr Objekt verwendet wird! Wird statdessen ein normaler Pointer auf das selbe Objekt verwendet, so wird beim beenden des Programmes eine "double free" exception auftreten! Der Zaehlerstand (use_count()) ist bei allen shared_ptr 3.

Wie können Sie den Zählerstand abfragen?

```
boost::shared_ptr<int> ptr2(ptr1); ptr2.use_count();
```

Warum referenzieren Sie die Methode um den Zählerstand abzufragen mit . und nicht mit ->, obwohl es sich um einen Pointer handelt?

Der Punkt muss verwendet werden, weil ansonsten die methode use_count auf das Objekt (in diesem Falle int) ausgefuehrt wird, auf welches der shared_ptr zeigt. use_count() ist aber eine funktion der shared_ptr class.

Jemand setzt den ersten shared_ptr zurück (Methode reset()). Wie lauten nun die Zählerstand aller drei shared_ptrs?

shared_ptr1:0 , shared_ptr2:2 , shared_ptr3:2

Warum?

shared_ptr verwenden Referenzzaehlung, um selber festzustellen wenn kein anderer Pointer mehr auf das selbe Objekt zeigt. Sobald dies der Fall ist und der letzte Pointer auf das Objekt entfernt wird, loescht dann der letzte Pointer automatisch das ganze Objekt.

Theorie 2 - bitset

Bei einem 32 Bit breiten ganzzahligen Wert ist das Bit an der Stelle 17 (von rechts gezählt, Beginn bei 0) auf 1 gesetzt, alle anderen sind auf 0. Schreiben Sie den Wert als 8-stellige Hex Zahl (mit Präfix 0x) auf

2^17 = 131072 = 0x00020000

Wie können Sie bei einem beliebigen vorzeichenlosen, ganzahligen Wert von 32 Bit Breite in C++ einfach feststellen, ob genau dieses Bit gesetzt ist oder nicht? Kurze Erklärung oder Code (nehmen Sie als Zahlenbeispiele 546123 und 677195 dezimal).

```
#include <iostream>
#include <bitset>
int main() {
  int value = 1234;
  std::cout << std::bitset<10>(value);
}
//Erzeugt ein bitset-Objekt mit 10 Stellen und dem Wert 1234. Der Wert kann nun ganz einfach
binär ausgegeben werden.

std::bitset<10> bin = std::bitset<10>(16);     // 0000010110
std::cout << bin.test(2) << std::endl;         // true
```

Mit der Methode test(int) des bitset-Objekts ist es nun möglich ein bestimmtes Bitzeichen abzufragen. Gezählt wird von Rechts!

Übungsserie 12

Aufgabe 1 - Mittelwert (Funktionstemplate)

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

template<typename T> inline T const & mitte(T const & eins, T const & zwei, T const & drei) {
	if((zwei < eins && eins < drei) || (drei < eins && eins < zwei)) return eins;
	if((eins < zwei && zwei < drei) || (drei < zwei && zwei < eins)) return zwei;
	if((eins < drei && drei < zwei) || (zwei < drei && drei < eins)) return drei;
};

int main (int argc, char * const argv[]) {
	if (mitte(7.0, 7.1, 7.05) == 7.05) { cout << "OK" << endl; }

	string eins("eins"), zwei("zwei"), drei("drei");
	if (mitte(eins, zwei, drei) == eins) { cout << "OK" << endl; }

	if (mitte<string>("vier", "fuenf", "sechs") == "sechs") { cout << "OK" << endl; }
}
```

Aufgabe 2 - Dreitausch (Funktionstemplate)

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

template <typename T> inline void dreitausch(T& eins, T& zwei, T& drei) {
	T tmp = eins;
	eins = zwei;
	zwei = drei;
	drei = tmp;
};

//Alternative Sommerlad für Vektoren
template<> void dreitausch2(vector<int>& t1,vector<int>& t2,vector<int>& t3) {
	swap(t1,t2); //t1 = t2
	swap(t1,t3); //t2 = t3
 }

int main (int argc, char * const argv[]) {
	int eins(1), zwei(2), drei(3);
	dreitausch(eins,zwei,drei);
	cout << eins << zwei << drei << endl; //--> 231
}
```

Aufgabe 3 – Eigene Vektorklasse (Klassentemplate, Iterator)

MyVector.h:

```
#ifndef MYVECTOR_H_
#define MYVECTOR_H_
#include <vector>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <cmath>

template <typename T>
class MyVector {
	private:
		typedef std::vector<T> vecType;
		typedef typename vecType::size_type size_type;
		typedef typename vecType::iterator iterator;
		vecType vecContainer;
	public:
		MyVector() {
			std::cout << "Bitte einige Werte eingeben.." << std::endl;
			std::istream_iterator<T> itEof;
			std::istream_iterator<T> itInp(std::cin);
			vecType vecContainer(itInp, itEof);
			this->vecContainer = vecContainer;

		}

		void print() const {
			copy(this->vecContainer.begin(), this->vecContainer.end(),
				std::ostream_iterator<T>(std::cout,", "));
		}

		void push_back(T value) {
			this->vecContainer.push_back(value); //Propagieren
		}

		T at(int const intIndex) const {
			int maxIndex = this->vecContainer.size() - 1;

			if (maxIndex == -1 || abs(intIndex) > maxIndex) {
				std::cerr << "Ungueltiger Index" << std::endl;
				return 0;
			} else {
				if (intIndex < 0) {
					return	this->vecContainer.at(this->vecContainer.size()-
						abs(intIndex));
				} else {
					return this->vecContainer.at(intIndex);
				}
			}
		}

		T operator[](int const intIndex) const {
			return this->at(intIndex);
		}

		iterator begin() {
			return this->vecContainer.begin();
		}

		iterator end() {
			return this->vecContainer.end();
		}
};
#endif /*MYVECTOR_H_*/
```

main.cpp:

```
#include "MyVector.h"
#include <iostream>
#include <algorithm>
#include <string>

int main() {
	MyVector<int> refVector;
	refVector.print();
	std::cout << "Zweitletztes Element ist: " << refVector.at(-2) << std::endl;
	std::cout << "Zweitletztes Element ist: " << refVector[-2] << std::endl;
	copy(refVector.begin(), refVector.end(), std::ostream_iterator<int>(std::cout,", "));
}
```

Übungsserie 13

Aufgabe 1 - Zufallstexte (lexical_cast, bind, Prädikat)

```
#include <iostream>
#include <boost/lexical_cast.hpp>
#include <vector>
#include <iterator>
#include <stdexcept>
#include <boost/bind.hpp>
using namespace std;

bool startswith(string search, string pattern){            //Prädikat
	return search.substr(0,pattern.size()) == pattern;
}

int main (int argc, char * const argv[]) {
	int groupsize = 3;
	int howmuchchars = 300;
	string textbegin = "ut";
	string templine = "";
	string inputlines = "";
	vector<string> parts;
	string finaltext = textbegin;
	if(argc > 1){
		try{
			groupsize = boost::lexical_cast<int>(argv[1]);
		} catch(boost::bad_lexical_cast &e){
			cout << "Bitte geben Sie eine Zahl als Argument ein!" << endl;
			return -1;
		}
	}
	cout << "Geben Sie einen Text ein und bestaetigen Sie mit <ENTER> CTRL-X
                                                    CTRL-D <ENTER>:" << endl;
	while(getline(cin, templine)){
		inputlines.append(templine);
	}
	try{
		for(int i = 0; i < inputlines.size()-1; ++i){
			parts.push_back(inputlines.substr(i, groupsize));
		}
	} catch(exception &e){
		cout << "Bitte geben Sie einen Text ein!" << endl;
		return -1;
	}

	//copy(parts.begin(),parts.end(),ostream_iterator<string>(cout,"\n"));
	while(finaltext.size() != howmuchchars){
		int howmany = count_if(parts.begin(), parts.end(),
				boost::bind(startswith,_1,finaltext.substr(finaltext.size()-2,2)));
		if(!howmany) { break; }
		int where = (rand() % howmany)+1;
		vector<string>::iterator found = parts.begin();
		vector<string>::iterator tmpit = parts.begin();
			for(int i = 0;i != where; ++i){
				found = find_if(tmpit, parts.end(),
				boost::bind(startswith,_1,finaltext.substr(finaltext.size()-2,2)));
				tmpit = found;
				++tmpit;
			}
		finaltext.append(found->substr(found->size()-1,1));
	}
	cout << "\n----------------------\nThe generated Text:\n\n" << finaltext << endl;
}
```

Aufgabe 2 – Permutation (next_permuation)

```
#include <iostream>
#include <iterator>
#include <vector>
#include <algorithm>

int main() {
	std::vector<int> vecNumbers;
	vecNumbers.push_back(1);
	vecNumbers.push_back(2);
	vecNumbers.push_back(3);
	vecNumbers.push_back(4);
	vecNumbers.push_back(5);

	int intCount = 1;
	std::cout << intCount << ": 1 2 3 4 5" << std::endl;
	while (std::next_permutation( vecNumbers.begin(), vecNumbers.end())) {
		++intCount;
		std::cout << intCount << ": ";
		std::ostream_iterator<int> out(std::cout, " ");
		std::copy(vecNumbers.begin(), vecNumbers.end(), out);
		std::cout << std::endl;
	}
}
```

Aufgabe 3 – Palindrom (copy_if, Prädikat)

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

template<typename In, typename Out, typename Pred>
Out copy_if(In anf, In end, Out res, Pred pred){
	for(;anf != end; ++anf){
		if(pred(*anf)) *res++ = *anf;
	}
	return res;
}

bool isPalindrom(string word){ //Prädikat
	transform(word.begin(),word.end(),word.begin(),::tolower);
	string rword = word;
	reverse(rword.begin(),rword.end());
	return word == rword;
}

int main (int argc, char * const argv[]) {
	copy_if(istream_iterator<string>(cin), istream_iterator<string>(),
	ostream_iterator<string>(cout,"\n"),isPalindrom);
}
```

Aufgabe 4 - Rotate Words (istringstream, rotate_copy)

```
#include <iostream>
#include <iterator>
#include <sstream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
        vector<string> lines;

        string input;
        while (getline(cin, input)) {
                istringstream line(input);
                istream_iterator<string> eol;
                vector<string> words(istream_iterator<string>(line), eol);
                for (vector<string>::size_type i = 0; i < words.size(); ++i) {
                        ostringstream rotated;
                        rotate_copy(words.begin(), words.begin() + i, words.end(),
                                    ostream_iterator<string>(rotated, " "));
                        lines.push_back(rotated.str());
                }
        }

        sort(lines.begin(), lines.end());
        copy(lines.begin(), lines.end(), ostream_iterator<string>(cout, "\n"));
}
```

Übungsserie 14

Aufgabe 1 – Eigener Primzahlen-Iterator

CountingIterator.h

```
#ifndef COUNTINGITERATOR_H_
#define COUNTINGITERATOR_H_
#include <iostream>
#include <iterator>

class CountingIterator : public std::iterator<std::input_iterator_tag, int> {
	public:
		CountingIterator(int intInit);
		int operator*() const;
		CountingIterator& operator++();		//Prefix
		CountingIterator operator++(int);	//Postfix
		bool operator==(CountingIterator const &other) const;
		bool operator!=(CountingIterator const &other) const;
	private:
		int intCounter;
};
#endif /*COUNTINGITERATOR_H_*/
```

CountingIterator.cpp

```
#include "CountingIterator.h"
#include <iostream>

CountingIterator::CountingIterator(int intInit) : intCounter(intInit) {}

int CountingIterator::operator*() const {
	return this->intCounter;
}

CountingIterator& CountingIterator::operator++(){		//Prefix
	++this->intCounter;
	return *this;
}

CountingIterator CountingIterator::operator++(int){		//Postfix
	CountingIterator result(*this);
	++this->intCounter;
	return result;
}

bool CountingIterator::operator==(CountingIterator const &other) const{
	return this->intCounter == other.intCounter;
}

bool CountingIterator::operator!=(CountingIterator const &other) const{
	return !(*this == other);
}
```

main.cpp

```
#include "CountingIterator.h"
#include <iostream>
#include <iterator>
#include <algorithm>

bool isNotPrime(int intNumber) {
	int intCount = 0;
	for (int i=1; i <= intNumber; i++) {
		if (intNumber % i == 0) {
			intCount++;
		}
		if (intCount > 2) {
			//keine Primzahl, abbrechen
			return true;
		}
	}
	//schleife komplett, ist primzahl
	return false;
}

int main() {
	CountingIterator itStart(1);
	CountingIterator itEnde(100);
	std::ostream_iterator<int> output(std::cout, ", ");
	//remove_copy_if = ENTFERNEN (=> nicht kopieren), wenn Prädikat erfüllt ist!
	std::remove_copy_if(itStart, itEnde, output, isNotPrime);
}
```

Aufgabe 2 - Rational (boost/operators)

Rational.h

```
#ifndef RATIONAL_H_
#define RATIONAL_H_
#include <iosfwd>
#include <boost/operators.hpp>

class Rational : boost::arithmetic<Rational>, boost::less_than_comparable<Rational> {
	public:
		Rational(long lngDefZaehler,long lngDefNenner);
		void print(std::ostream &osOut) const;
		void read(std::istream &osInt);

		Rational& operator+=(const Rational& refSecond);	// += erzeugt +
		Rational& operator-=(const Rational& refSecond);	// -= erzeugt -
		Rational& operator*=(const Rational& refSecond);	// *= erzeugt *
		Rational& operator/=(const Rational& refSecond);	// /= erzeugt /
		bool operator<(const Rational& refSecond) const;	// < erzeugt <= > >=
	private:
		long ggt(long lngZahl1, long lngZahl2);
		void normalize(Rational& refNumber);
		long lngZaehler;
		long lngNenner;
};
//Output Operator muss freie Funktion sein, da ostream nicht kopiert werden kann
std::ostream& operator<<(std::ostream &osOut, const Rational& refRational);
#endif /*RATIONAL_H_*/
```

Rational.cpp:

```
#include <iostream>
#include <cmath>
#include <cctype>
#include <boost/operators.hpp>
#include "Rational.h"
#include "DivByZero.h"

//Folgende Klasse beschränkt sich nur aufs wesentliche!

Rational& Rational::operator+=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngNenner) + (refSecond.lngZaehler * lngNenner);
	lngNenner *= refSecond.lngNenner;
	normalize(*this);
	return *this;//Bei Zuweisungen aktuelles Objekt zurückliefern
}

Rational& Rational::operator-=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngNenner) - (refSecond.lngZaehler * lngNenner);
	lngNenner *= refSecond.lngNenner;
	normalize(*this);
	return *this;
}

Rational& Rational::operator*=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngZaehler);
	lngNenner *= refSecond.lngNenner;
	normalize(*this);
	return *this;
}

Rational& Rational::operator/=(const Rational& refSecond) {
	lngZaehler = (lngZaehler * refSecond.lngNenner);
	lngNenner *= refSecond.lngZaehler;
	normalize(*this);
	return *this;
}

bool Rational::operator<(const Rational& refSecond) const {
	long lngZahl1 = this->lngZaehler * refSecond.lngNenner;
	long lngZahl2 = refSecond.lngZaehler * this->lngNenner;

	return lngZahl1 < lngZahl2;
}
```

Aufgabe 4 - getline-Iterator

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

class line_iterator : public iterator<forward_iterator_tag, string> {
public:
	line_iterator() : m_in(NULL) {}
	line_iterator(istream * in) : m_in(in) { ++(*this); }
	~line_iterator() {}

	bool operator!=(const line_iterator & other) const {
		return other.m_in || (bool)(*m_in);
	}

	line_iterator & operator++() {
		getline(cin, m_line);
		return *this;
	}

	string operator*() const {
		return m_line;
```

```
        }
private:
        istream * m_in;
        string m_line;
};

int main()
{
        line_iterator eof;
        line_iterator in(&cin);
        remove_copy_if(in, eof, ostream_iterator<string>(cout, "\n"),
                                                    mem_fun_ref(&string::empty));
}
```