| | |
|---|---|
| Name | Matr.-Nr. |

# Klausur Programmieren III – TI3

SS 12

Prof. Dr. K. Baer

Hilfsmittel: 1 Blatt DIN A4, beidseitig beschr.

Bearbeitungszeit: 90 Min.

1. Bitte tragen Sie zuerst Name und Matrikelnummer ein!
2. Kontrollieren Sie die Vollständigkeit der Aufgabenblätter.
3. Die Klausur besteht aus 6 Aufgaben. Verschaffen sie sich einen kurzen Überblick über die Aufgaben und beginnen Sie am besten mit der Aufgabe, die Ihnen am ehesten ein Erfolgserlebnis bringt.
4. Lesen Sie die Aufgabenstellung aufmerksam durch, bevor Sie eine Aufgabe lösen!
5. Verwenden Sie zur Beantwortung der Fragen den vorgesehenen Platz auf den Aufgabenblättern.
6. Schreiben Sie leserlich. Nicht lesbare Teile werden mit 0 Punkten bewertet!

Viel Erfolg!

| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | Summe |
|---|---|---|---|---|---|---|---|
| Punkte | 20 | 6 | 14 | 20 | 6 | 24 | 90 |
| erreicht | | | | | | | |

## Aufgabe 1 (20 Punkte = 2+11+2+2+1+ 2)

Gegeben Sie folgende Implementierung:

```
class Bauteil {
protected:
	string name;
public:
	Bauteil(){ name = "???"; cout << "Bauteil erzeugt" << endl; }
	~Bauteil(){ cout << "Bauteil zerstoert" << endl;}
	void zeigeName(){ cout << "Name: " << name << endl; }
};

class A : public Bauteil {
public:
	A() { name = "A"; cout << name << " erstellt" << endl; }
	~A(){ cout << name << " zerstoert" << endl; }
};

class B : public Bauteil {
public:
	B() { name = "B"; cout << name << " erstellt" << endl; }
	~B(){ cout << name << " zerstoert" << endl; }
	void zeigeName(){ cout << "B-Teile" << endl; Bauteil::zeigeName(); }
};


class Baugruppe {
private:
	set<Bauteil*> gruppe;
	typedef set<Bauteil*>::iterator Iter;
public:
	Baugruppe(){ cout << "Baugruppe erstellt" << endl; }
	Baugruppe(const Baugruppe& other){ cout << "Baugruppe kopiert" << endl;
		for(Iter i=other.gruppe.begin(); i != other.gruppe.end(); ++i){
			gruppe.insert(*i);
		}
	}
	virtual ~Baugruppe(){ cout << "Baugruppe zerstoert" << endl; }
	void hinzufuegen(Bauteil* teil){ gruppe.insert(teil); }
	void entfernen( Bauteil* teil){ gruppe.erase(teil); }
	void zeigeElemente(){
		cout << endl << "-------------------------";
		cout << "\nBaugruppe bestueckt mit: " << endl;
		for(Iter i = gruppe.begin(); i != gruppe.end(); ++i){
			(*i)->zeigeName();
		}
		cout << "-------------------------" << endl;
	}
};
```

a) Erläutern Sie den Begriff „Späte Bindung“!

- Gegenstück zu statischer Bindung, d.h. Methodenbindung zur Compilezeit
- Relevant in polymorphen Klassenhierarchien
- Es wird zur Laufzeit anhand des tatsächlichen Objekttyps bestimmt, welche Methode zur Ausführung kommt.
- Beispiel:

```
class A {
    virtual void m1(){ // tu was ...};
}
class B : public A {
    void m1(){ // tu was anderes … }
}

void test(){
        A* ptr_A = new B;
        ptr_A->m1();   // es wird m1() der Klasse B ausgeführt
}
```

b) Was gibt folgendes Testprogramm aus?
Geben Sie bitte zur Verdeutlichung für jede Ausgabezeile die **Zeilennummer** in der Methode test() an, die diese Ausgabe erzeugt.

```
1   #include "bauteil.h"
2
3   void anzeigen(Baugruppe gruppe){
4    gruppe.zeigeElemente();
5   }
6
7   void test(){
8    Baugruppe gruppe1;
9
10   A* a1 = new A();
11   B* b1 = new B();
12   Bauteil* b2 = new B();
13
14   gruppe1.hinzufuegen(a1);
15   gruppe1.hinzufuegen(b1);
16   gruppe1.hinzufuegen(b2);
17
18   anzeigen(gruppe1);
19
20   b1->zeigeName();
21
22   Baugruppe gruppe2 = gruppe1;
23   A a2;
24   gruppe2.hinzufuegen(&a2);
25   gruppe2.zeigeElemente();
26   delete b1;
27   delete b2;
28  }
29
30  int main(){
31   test();
32   cin.sync();cin.get();
33  }
```

| | | | |
|---|---|---|---|
| 8 | Baugruppe erstellt | | |
| 10 | Bauteil erzeugt | | |
| | A erstellt | | |
| 11 | Bauteil erzeugt | | |
| | B erstellt | | |
| 12 | Bauteil erzeugt | | |
| | B erstellt | 4 x 0,5 = | 2 |
| 18 | Baugruppe kopiert | | 1 |
| | | | |
| 18 | ------------------------ | | 1 |
| | Baugruppe bestueckt mit: | | |
| | Name: A | | |
| | Name: B | | |
| | Name: B | | |
| | ------------------------ | | |
| 18 | Baugruppe zerstoert | | 1 |
| 20 | B-Teile | | 1 |
| | Name: B | | |
| 22 | Baugruppe kopiert | | 1 |
| 23 | Bauteil erzeugt | | 0,5 |
| | A erstellt | | |
| | | | |
| 25 | ------------------------ | | 1 |
| | Baugruppe bestueckt mit: | | |
| | Name: A | | |
| | Name: A | | |
| | Name: B | | |
| | Name: B | | |
| | ------------------------ | | |
| 26 | B zerstoert | | 0,5 |
| | Bauteil zerstoert | | |
| 27 | Bauteil zerstoert | | 0,5 |
| 28 | A zerstoert | | 0,5 |
| | Bauteil zerstoert | | |
| | Baugruppe zerstoert | | 0,5 |
| | Baugruppe zerstoert | | 0,5 |

c) Das Bauteil b1 wird an verschiedenen Stellen ausgegeben. Erscheint es immer gleich? Begründen Sie Ihre Antwort.

Nein.
Zugriff teils über Basisklassenzeiger, teils über Objekt.
ZeigeName nicht virtual deklariert, aber überschrieben.
d.h. Unterschiedliches Verhalten:
- via Basisklassezeiger: Methode der Basisklassen
- via Objekt direkt: überschriebene Methoden

d) Erklären Sie Ihre Ausgaben der Zeilen 26-28.

estruktor in Basisklasse nicht virtual!
26: delete mit Objektzeiger vom Typ B, d.h. Es wird der Destruktor von b aufgerufen und nachfolgend der Destruktor der Basisklasse

27: delete mit Basisklassenzeiger. Es wird der Destruktor der Basisklasse aufgerufen, da Destruktor nicht virtual deklariert!

28: die Methode test() wird beendet, d.h alle lokalen Variablen / Objekte werden zerstoert, beginnend mit dem zuletzt erzeugten: a2, g2,g1.

In den Gruppen werden nur Pointer verwaltet, d.h. Die Objekte auf dem Heap werden nicht abgeräumt → Speicherleck

e) Erläutern Sie detailliert, welche Funktionen in der Zeile 22 aufgerufen werden!

Kopierkonstruktor

Neues Objekt wird aus bereits bestehendem erzeugt

f) Was ist zur Implementierung der Funktion

```
Baugruppe(const Baugruppe& other)
```

anzumerken?

Es wird keine Kopie von Objekten, auf die die Pointer zeigen, angelegt

## Aufgabe 2 (6 Punkte)

Wie geht man sinnigerweise bei der Operatorüberladung vor? Erläutern Sie – etwa am Beispiel einer Klasse Bruch – welche Probleme typischerweise auftauchen und wie man am besten damit umgeht. Wie erreicht man bspw. im Falle der Klasse Bruch, dass die Implementierung das Kommutativgesetz erfüllt?

- Kurzformoperatoren als Elementfunktion in der Klasse
- Rückgabe Bruch&
- normale, binäre Operatoren (+,-,*/) als globale Funktionen, Rückgabe Bruch

- damit wird Vertauschen der Operanden ermöglicht
- nach Abwägung de Vor- und Nachteile gewährt man den globalen Funktionen evtl. per friend-Deklaration Zugriff auf private Datenelemente der Klasse (nur wenn deutliche Performanzvorteile entstehen, ansonsten Zugriff via Getter/Setter).
- Zurückführen auf Kurzformoperatoren um Redundanz zu vermeiden
- keine Konvertierungsfunktion wg. möglicher Mehrdeutigkeiten, besser „asZieltyp()“

## Aufgabe 3 (14 Punkte=7+7)

a) Erläutern Sie das friend-Konzept in C++!
(Zweck, Verwendung, Beispiel)

Eine Klasse kann anderen Klassen oder Funktionen oder Methoden außerhalb der Klasse (sog. "Freunden") den Zugriff auf private Datenelemente und Methoden erlauben
Nur die Klasse selbst kann bestimmen, wen sie als Freund haben möchte!

```
class A {
  ...
  friend void someGlobalFunc(A * aPtr);
  friend int  B::someMethod(const A& aRef);
  ...
};
```

Aufweichung der Datenkapselung!!
Akzeptabel, falls ansonsten benötigte Datenzugriffe sehr aufwendig werden.

```
class Complex {
  private:
    double re, im;
  public:
    ...
    friend operator==(double d1, Complex c2)
};
bool operator== (const Complex& c1, double d2){
return  ( ( abs( (c1.re - d2 ) / c1.re) < FEHLERSCHRANKE )  &&
  ( c1.im < FEHLERSCHRANKE ) );
```

b) Was ist ein Binder in der STL?
(Zweck, Verwendung, Beispiel)

```
Was: → spezieller Funktor
Zweck: Verminderung der Zahl benötigter Funktoren
     bei zweiwertigen Funktoren wird einer der Operanden
     durch Binder gebunden. Damit sind nur 2-wertige Funktoren
     & 2 Binder für Binden des 1. Operanden und Binden des 2.
     Operanden nowendig.

template<class Operation, class T>
class Binder {
public:
     Binder(const Operation& o, const T& t) : op(o),y(t) { }
     bool operator()(const T& x) const { return op(x, y); }
private:
     const Operation op;
     const T y;
};

Beispiel:
int f[] = { 3, 1, 4, 5, 7, 0, 8, 2, 6 };
transform(f, f + 9, f, bind2nd(plus<int>(), 1));
transform(f, f + 9, f, ostream_iterator<int>(cout, "  "),
```

## Aufgabe 4 (20 Punkte)

Eine Matrix kann aufgefasst werden als ein Feld von Feldern.

Bei Verwendung der STL kann man eine Template-Klasse für eine Matrix auf einfache Weise entwickeln, indem man eine Template-Klasse `Matrix` implementiert, die öffentlich von `vector< vector<T> >` abgeleitet wird.

Entwickeln Sie eine solche Template-Klasse mit folgenden öffentlichen Methoden:

- ein 2-stelliger Konstruktor zur Angabe der Matrix-Dimension
- den Funktionen `Rows()` bzw. `Columns()` zur Emittlung der Dimensionen
- eine Funktion `init()`, die erlaubt, die Matrix-Elemente mit einem angegebenen Wert zu initialisieren
- eine Funktion `I()`, die die Einheitsmatrix zurückgibt (überall 0, nur Diagonale mit 1 besetzt)
- den Ausgabeoperator operator<<, mit dem die Matrix in der Form wie folgende Einheitsmatrix ausgegeben wird (D.h., Zeilennr: Werte durch Leerzeichen getrennt)

0: 1 0 0 0
1: 0 1 0 0
2: 0 0 1 0

```
template<class T>
class Matrix : public vector< vector<T> >{
public:
        typedef typename vector<T>::size_type size_type;

        Matrix(size_type x = 0,size_type y = 0) : vector< vector<T> >( x,
vector<T>(y) ), rows(x), columns(y) {}                          // 4

        size_type Rows() const { return rows; }                 // 1
        size_type Columns() const { return columns; } // 1

        void init (const T& value ){                            // 3
                for(size_type i = 0; i < rows; ++i )
                        for(size_type j = 0; j < columns; ++j)
                                (*this)[i][j] = value;
        }

        Matrix<T>& I(){
                for(size_type i = 0; i < rows; ++i)
                        for(size_type j = 0; j < columns; ++j)
                                (*this)[i][j] = (i==j)?T(1):T(0);
```

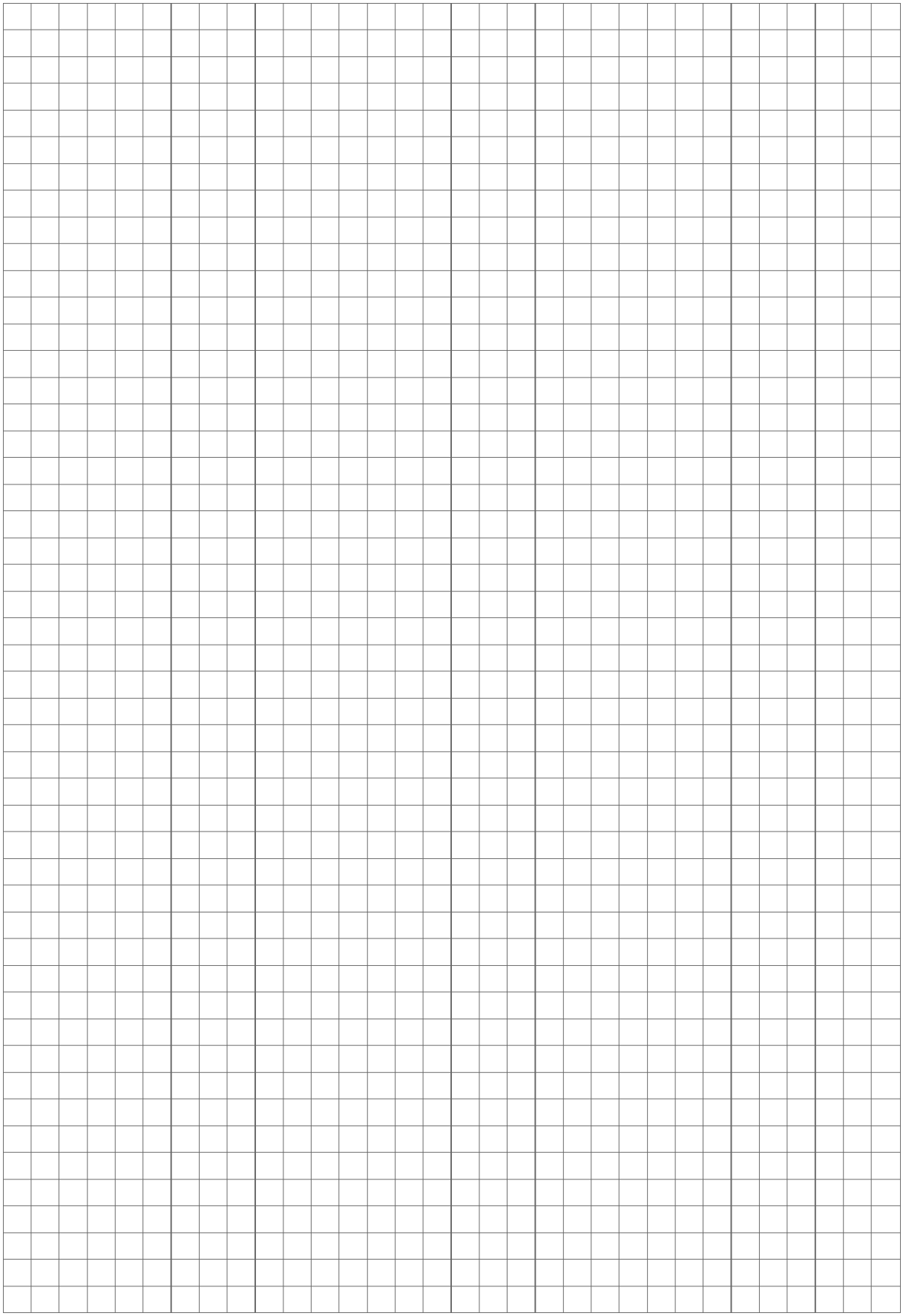
```
            return *this;
        }                                   // 3

    protected:
        size_type rows;
        size_type columns;                  // 2

    };// Klasse Matrix


    template<class T>
    inline ostream& operator<<(ostream& s, const Matrix<T>& m){
        typedef typename Matrix<T>::size_type size_type;

        for(size_type i = 0; i < m.Rows(); ++i){
            s << endl << i << ": ";
            for(size_type j = 0; j < m.Columns(); ++j)
                s << m[i][j] << " ";
        }
        s << endl;
        return s;
    }                                       // 6
```



## Aufgabe 5 (6 Punkte )

Erläutern Sie, was man unter einem downcast versteht.

Stellen Sie dar, in welchen Situationen Downcasts problematisch sind.

Stellen Sie dar, wie man korrekt damit umgeht.

Geben Sie ein Beispiel.

Down-Cast
Basisklassenzeiger wird durch Cast in Zeiger auf abgeleitete Klasse konvertiert
Vorsicht: Down-Casts sind eventuell unsicher und gefährlich!!!
unproblematisch:
Basisklassenzeiger zeigt zur Laufzeit tatsächlich auf Objekttyp, auf den gecastet wird
problematisch
Basisklassenzeiger zeigt zur Laufzeit auf Elternobjekt, auf das gecastet wird.

Korrekter Umgang:
dynamic-Cast liefert NULL-Pointer falls yp nicht passt, Darauf kann abgeprüft werden und ggf. eine Fehlerbehandlung stattfinden.

```
Kfz * kfzPtr = new Kfz;
Pkw * pkwPtr = NULL;

pkwPtr = dynamic_cast<Pkw*>(kfzPtr); // liefert Null-Zeiger!
if (pkwPtr == 0)
        cout << "Fehler!!" << endl;
else {
        pkwPtr->display();
        pkwPtr->einsteigen(3);
}
```

## Aufgabe 6 (24 Punkte = 2+1+6+4+11)

Entwickeln Sie eine Klasse „PlzOrtMapper“, die das Mapping von Postleitzahlen und Orten unterstützt.

Gegeben sei eine Textdatei nach untenstehendem Muster, in der pro Zeile eine Zuordnung PLZ zu Ort angegeben ist:

```
89075 Ulm
89076 Ulm
89077 Ulm
89077 Ulm-Soeflingen
89078 Ulm
89079 Goegglingen
89079 Ulm
89079 Ulm-Donautal
89079 Ulm-Eggingen
89079 Ulm-Einsingen
89079 Ulm-Goegglingen
89079 Ulm-Unterweiler
89079 Ulm-Wiblingen
89080 Ulm
89081 Ulm
89081 Ulm-Jungingen
89081 Ulm-Lehr
89081 Ulm-Maehringen
89081 Ulm-Seligweiler
89081 Ulm-Soeflingen
89081 Ulm-Ermingen
89081 Ulm-Jungingen
```

Daraus ist zu erkennen, dass ein Ort mehrere PLZ haben kann (z.B. Ulm). Genauso kann allerdings eine PLZ auch für mehrere Orte gültig sein (z.B. 89079)

a) Erläutern Sie die Vorteile einer MultiMap.
Aus welchen Elementen ist eine Map / Multimap aufgebaut?

Effizienter Zugriff via Schlüssel (Logarithmisch)
sortiert
mehrere einträge / Schlüssel möglich
pair<const Key, T>

b) Entwickeln Sie die Klasse „PlzOrtMapper“. Sie soll folgende Funktionen unterstützen:

```
bool einlesen(string dateiname) throw (FileError);
int zaehleVorkommen(string ort);
string listPlzOrt( … );
```

- Überlegen Sie sich zunächst eine geeignete Datenstruktur, mit der Sie PLZ/Ort-Paarungen am einfachsten innerhalb Ihrer Klasse verwalten können.
- Die Methode „einlesen“ soll – falls die Datei nicht geöffnet werden kann – ein entsprechendes Fehlerobjekt „FileError“zur Ausnahmebehandlung erzeugen. Der Dateiname soll an der aufrufenden Stelle dem Fehlerobjekt entnommen werden können.
- Die Methode zaehleVorkommen() liefert die Anzahl PLZ/Ort-Paarungen zu einem gegebenen Ort.
- Die Methode listPlzOrt() soll PLZ/Ort-Paarungen als string liefern. Es soll möglich sein, die Funktion so aufzurufen, dass
  - alle Paarungen mit dem genauen Ortsnamen,
  - alle Paarungen, in denen der Ortsname den gegebenen Suchstring enthält,
  - alle Paarungen, die zu einer PLZ gehören,

  aufgelistet werden.

Hinweise zur STL:

- Multimap bietet eine Elementfunktion „equal_range“, mit der die Grenzen eines Bereichs in der Map ermittelt werden können, in dem der Schlüssel einen bestimmten Wert hat:

```
pair<iterator,iterator> equal_range ( const key_type& x );
```

- Der Abstand zweier Iteratoren kann mit Hilfe der globalen Funktion distance ermittelt werden:

```
size_t = distance( iterator1, iterator2 );
```

- Mit Hilfe der globalen Function find_if kann in einem Container nach Werten gesucht werden, die eine bestimmte Bedingung (ein Prädikat) erfüllen. Dazu benötigt die Funktion den Bereich im Container, in dem gesucht werden soll sowie die Vergleichsfunktion. Sie liefert die Position des ersten gefundenen Elements (oder *last*, falls nichts gefunden wurde).

```
template <class InputIterator, class Predicate>
InputIterator find_if ( InputIterator first,
                        InputIterator last,
                        Predicate pred );
```

b1 ) Deklarieren Sie die interne **Datenstruktur** zur Aufnahme und Verwaltung der PLZ/Ort-Paarungen.

```
multimap< string, int > m_plz;
```

b2 ) Implementieren Sie die Methode **einlesen** zusammen mit der Fehlerklasse „FileError“

```
bool einlesen(string dateiname) throw (FileError){
  string    ort;
   int        plz;

   ifstream datei(dateiname); // Input-Stream auf Datei
   if (!datei) {
          throw FileError("Couldn't open file " + dateiname);
  }

   while (!datei.eof()) {
         if (datei >> plz) {
               datei >> ort;
                m_plz.insert( pair<string,int>( ort,plz) );
          }

  }
   return true;
}

class FileError {
private:
       string m_msg;
public:
       FileError(string msg){m_msg = msg;}
       string what(){return m_msg;}
};
```

b3 ) Implementieren Sie die Methode **zaehleVorkommen**

```
int zaehleVorkommen(string ort){
        pair< multimap< string, int >::iterator, multimap< string, int >::iterator
> p;

        p = m_plz.equal_range( ort );
        return distance(p.first, p.second);
}
```

b4 ) Implementieren die Methode listPlzOrt.

**Hilfestellung**:

- Die unterschiedlichen Suchen können Sie über unterschiedliche Funktoren abbilden

- Einmal wird nach Plz (→ int), die anderen Male nach Ort (→ string) gesucht. Wenn man listPlzOrt als template implementiert, kann man entsprechend Implmentierungsaufwand sparen.

- In stringstream-Objekte kann mit dem üblichen Ausgabe-Operator << wie auf die Konsole gesschrieben werden. Stringstream-Objekte besitzen eine Methdoe str() zur Konvertierung in einen string.

- Strings bieten die Funktion

```
size_t find ( const string& str, size_t pos = 0 ) const;
```

die die Position des ersten Auftretens des Suchstrings liefert.

Ein Testprogramm sollte Output nach folgendem Muster liefern:

```
Liste Postleitzahl(en) zu Ort: Ulm
89070  Ulm
89073  Ulm
89074  Ulm
89075  Ulm
… (usw.)

Liste PLZ zu Orten, die den Namensbestandteil enthalten: Soef
89077  Ulm-Soeflingen
89081  Ulm-Soeflingen

Liste alle Orte, die zu einer Postleitzahl gehören: 89079
89079  Goegglingen
89079  Ulm
89079  Ulm-Donautal
89079  Ulm-Eggingen
89079  Ulm-Einsingen
89079  Ulm-Goegglingen
89079  Ulm-Unterweiler
89079  Ulm-Wiblingen
```

```
.template <class T, class U>
class KeyContains {
public:
	KeyContains(T suche){ m_such = suche; }

	bool operator()(pair<T, U> p){
		T key = p.first;
		int pos = 0;
		pos = key.find(m_such);

		return ( pos > -1 );
	}
private:
	T m_such;
};


template <class T, class U>
class KeyEquals {
public:
	KeyEquals(T suche){ m_such = suche; }

	bool operator()(pair<T, U> p){
		return ( p.first == m_such );
	}
private:
	T m_such;
};



template <class T, class U>
class ValueEquals {
public:
	ValueEquals(U suche){ m_such = suche; }

	bool operator()(pair<T, U> p){
		return ( p.second == m_such );
	}
private:
	U m_such;
};



	template < class Comp>
```

```
string listPlzOrt( Comp cmpFunc){
  stringstream buffer;

  multimap< T, U >::iterator i = m_plz.begin();

  while ( i != m_plz.end() ){
       i = find_if(i, m_plz.end(), cmpFunc);

       if ( i != m_plz.end() ){
            buffer << i->second << "  " << i->first << endl;
            ++i;
       }
}

  return buffer.str();
}
```