Name | Matr.-Nr.

Klausur Programmieren III – TI3

WS 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	19	12	15	18	6	20	90
erreicht							

Aufgabe 1 (19 Punkte = 12+5+2)

Gegeben Sie folgende Implementierung:

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



class Part{
public:
  Part()            { cout << " cPart"; }
  Part(Part& b)     { cout << " copyPart"; }
  ~Part()           { cout << " ~Part"; }
};


class Top {
public:
  Top()           { cout << " cTop"; }
  Top(Top& a){ cout << " copyTop"; }
  ~Top()     { cout << " ~Top"; }
};

class Middle : public Top {
protected:
      Part* p1;
      Part p2;
public:
  Middle()            { cout << " cMiddle"; }
  Middle(Middle& m)   { cout << " copyMiddle"; }
  ~Middle()           { cout << " ~Middle"; }
  void useTop(Top t)  { cout << " usingTop"; }
};

class Bottom : public Middle {
public:
  Bottom()            { cout << " cBottom"; }
  Bottom(Bottom& b)   { cout << " copyBottom"; }
  ~Bottom()           { cout << " ~Bottom"; }
};
```

a) Was gibt folgendes Testprogramm aus?

```
void test(){
      Middle m1;
      cout << endl;

      Bottom* b1 = new Bottom();
      cout << endl;

      Top* t1 = new Bottom();
      cout << endl;

      Bottom b2(*b1);
      cout << endl;

      m1 = *b1;
      cout << endl;

      m1.useTop(*b1);
      cout << endl;

      b1->useTop(m1);
      cout << endl;

      delete b1;
      cout << endl;

      delete t1;
      cout << endl;

}

int main(){
      test();
      cin.sync(); cin.get();
}
```

cTop cPart cMiddle

cTop cPart cMiddle cBottom

cTop cPart cMiddle cBottom

cTop cPart cMiddle copyBottom

copyTop usingTop ~Top

copyTop usingTop ~Top

~Bottom ~Middle ~Part ~Top

~Top

~Bottom ~Middle ~Part ~Top ~Middle ~Part ~Top

b) Was versteht man unter „Slicing“?
In welchen Zeilen tritt Slicing in obigem Code auf?

Ein Objekt einer abgeleiteten Klasse kann an ein Objekt einer direkten oder indirekten
Basisklasse zugewiesen werden (aber nicht umgekehrt!).
Bei der Zuweisung werden nur die Elemente der Basisklassekomponentenweise kopiert

Bei Zeigern analog:
- Objekt wird bei Zugriff über die Basisklassenreferenz wie ein Objekt der Basisklasse behandelt!
z.B. bei redefinierten Methoden in der abgeleiteten Klasse wird Basisklassen-Version aufgerufen!

```
m1 = *b1;
Objekt vom Typ Bottom wird an Objekt vom Typ middle zugewiesen

m1.useTop(*b1);
useTop erwartet Objekt vom Typ Top, erhält aber Bottom

b1->useTop(m1);
analog mit Middle
```

c) Würde sich in diesem Beispiel etwas ändern, wenn man den Destruktor von Top virtual deklarieren würde? Begründen Sie Ihre Antwort!

T1 ist Basisklassenzeiger, zeigt aber auf Bottom Objekt.
Die Zeile

delete t1;

bewirkt den Aufruf des Destruktors von Top.
Falls Destruktor virtual deklariert würde korrekter Destruktor von Bottom aufgerufen
Output wäre dann in der zweitletzten Zeile:nicht nur

~top

sondern
~Bottom ~Middle ~Part ~Top

Aufgabe 2 (12 Punkte=4+4+4)

a) Achten Sie in den folgenden Deklarationen darauf, dass u.U. Initialwerte benötigt werden. Dafür sei gegeben:

```
int i = 5;
```

Definieren Sie einen Zeiger p1 auf den Typ int:

```
int *p1;
```

Definieren Sie einen Zeiger p2 auf den Typ konstanter int:

```
const int *p2;
```

Definieren Sie einen konstanten Zeiger p3 auf den Typ int.

```
int * const p3 = &i;
```

Definieren Sie einen konstanten Zeiger p4 auf den Typ konstanter int:

```
const int * const p4 = &i;
```

// 4 Punkte

b) Welche der in a.) deklarierten Zeiger kann man in die folgenden Funktionen einsetzen? (Bitte in unten stehender Tabelle entsprechend ankreuzen).

```
void f1(int *);
void f2(const int * const);
```

	p1	p2	p3	p4
f1	x		x	
f2	x	x	x	x

c) Gegeben seien folgende Funktionsdeklarationen:

```
void f(double d, int i);      // Funktion 1
void f(long k, long l);       // Funktion 2
void f(long l, double d);     // Funktion 3
```

und folgende Variablendeklarationen:

```
int i, j;
long k,l;
double x,y;
```

Welche Funktionsaufrufe sind möglich und welche Funktion wird dann aufgerufen? Geben Sie die Nummer der Funktion an.

	Nr.	falsch
`f(x,l);`		x
`f(i,j);`	1	
`f(k,l);`	2	
`f(x,y);`		x

Aufgabe 3 (15 Punkte=3+4+5+3)

a) Welche grundlegenden Template-Klassen bietet die STL für sequentielle Container an?

Vector
list
deque

b) Wie unterscheiden sich diese Klassen hinsichtlich Elementzugriff und Einfügen/Entfernen von Elementen (jeweils Vorne, Mitte, Hinten). Geben Sie die Komplexität der ausgeführten Operationen an.

	vector	deque	list
Einfügen und Entfernen			
Vorne	O(n)	O(1)	O(1)
Hinten	O(1)	O(1)	O(1)
Mitte	O(n)	O(n)	O(1)
Zugriff			
erstes	O(1)	O(1)	O(1)
letztes	O(1)	O(1)	O(1)
mittleres	O(1)	O(1)	O(n)
Iterator	RandAcc	RandAcc	BiDir

c) Was versteht man unter Iteratoren und wozu dienen sie in der STL? Welche Iterator-Kategorien unterscheidet man in der STL und was sind wesentliche Unterschiede?

Iteratoren sind eine Verallgemeinerung des Zeigerkonzepts
Typische Verwendung von C-Pointern

```
char buf[20];
... // Wertzuweisung
for (char* i = buf; *i != '\0'; ++i)
*i = toupper(*i);
```

Typische Verwendung von Iteratoren

```
vector < char > buf;
... // Wertzuweisung
for (vector<char>::iterator i=buf.begin(); i!=buf.end(); ++i)
*i = toupper(*i);
```

vector<> kann auch mit Standard-C-Pointern verwendet werden
Iteratoren sind das Interface, das die generischen Algorithmen der STL nutzen, um mit Containern zusammenzuarbeiten

Input Output Forward Bidirectional Random access

Menge an unterstützten Operatoren. Bis Forward nur ++ (neben *, ==, != →)
Ab BiDir auch –, RA volles Set

d) Wie ist der Zusammenhang zwischen Iterator und dem zugehörigen Reverse-Iterator? Weshalb? Welche Iterator-Kategorien erlauben die Bereitstellung eines Reverse-Iterator?

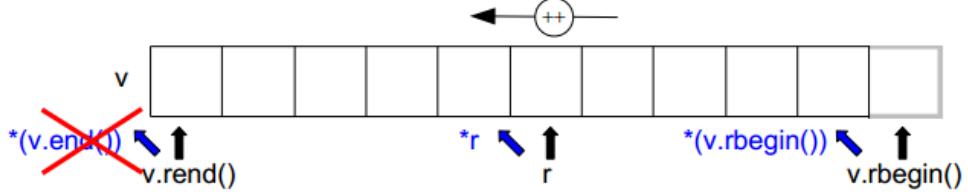
Reverse-Iterator hat dieselbe Schnittstelle wie der zugrunde liegende Bidirectional- oder Random-Access-Iterator.

Achtung

- i sei ein Iterator. Dann gilt für den zugehörigen reverse_iterator

```
&*(reverse_iterator(i)) == &*(i-1)
```

- Problem
  - es gibt i.a. keine gültige Adresse vor dem ersten Element.
  - reverse_iterator liefert beim Dereferenzieren das vorhergehende Element.



Aufgabe 4 (18 Punkte=10+8)

Zu entwickeln ist 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) .

Als interne Datenstruktur zur Aufnahme und Verwaltung der PLZ/Ort-Paarungen könnte eine multimap Anwendung finden:

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

Für die Klasse „PlzOrtMapper“ sollen folgende Funktionen entwickelt werden:

```
string listPlz(string ort);
int zaehleVorkommen(string ort);
string listPlzPredicate(string suche);
```

- Die Methode listPlz() soll für einen gegebenen Ort alle PLZ / Ort – Paarungen listen.
- Die Methode zaehleVorkommen() liefert die Anzahl PLZ/Ort-Paarungen

zu einem gegebenen Ort.

- Die Methode listPlzPedicate() liefert alle PLZ / Ort-Paarungen, in denen der Ortsname den gegebenen Suchstring enthält.

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 );
```

Implementieren Sie die drei Methoden.

Implementieren Sie die notwendige Vergleichsfunktion in der Funktion listPlzPedicate als Funktor.

Hinweis: Strings bieten die Funktion

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

die die Position des ersten Auftretens des Suchstrings liefert.

```
string listPlz(string ort){
        stringstream buffer;

        pair< multimap< string, int >::iterator, multimap< string, int >::iterator > p;

        p = m_plz.equal_range( ort );

        if( p.first != m_plz.end() ) {

                multimap< string, int >::iterator i = p.first;
                for(i; i != p.second; ++i){
                     buffer << i->first << " " << i->second << "\n";
                }
                buffer << endl << zaehleVorkommen( ort ) << " Vorkommen";
        }
        else {
                buffer << "Nichts gefunden zu " << ort << endl;
        }

        return buffer.str();
}
// 5 P


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);
}
// 4 P
```

```
string listPlzPredicate(string suche){
        stringstream buffer;


        multimap< string, int >::iterator i = m_plz.begin();


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

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

      return buffer.str();
}
// 5 P




class Enthaelt {
public:
      Enthaelt(string suchstring){ m_such = suchstring; }

      bool operator()(pair<string, int> p){
             string ort = p.first;
             int pos = 0;
             pos = ort.find(m_such);

             return ( pos > -1 );
      }
private:
      string m_such;
};
// 4P
```

Aufgabe 5 (6 Punkte)

Entwickeln Sie eine Templatefunktion, die in einer Map die Summe der Objekte an 2. Stelle berechnet und zurück gibt. Der Operator+ sei für für diese Objekte definiert.

Folgendes Programmstück soll z.B. mit Ihrer Funktion ausführbar sein und im gezeigten Beispiel den Wert 240 ausgeben:

```
int main() {
	map<string, int> klausurergebnis;

	klausurergebnis["Mayer"] = 90;
	klausurergebnis["Müller"] = 80;
	klausurergebnis["Schmidt"] = 70;


	cout <<"Sum = "<< sum(klausurergebnis) << endl;

	cin.sync();cin.get();

	return 0;
}
```

```
template <class A, class B>
B sum(map<A, B> &m) {
	map<A,B>::iterator i = m.begin();

	B sum = 0;

	if ( ! (i == m.end()) ) {
		//throw "add ist ja leer ....";
		sum = i->second;
		for (i++; i != m.end(); ++i) {
			sum = sum + i->second;
		}
	}
	return sum;
}
```

Aufgabe 6 (20 Punkte)

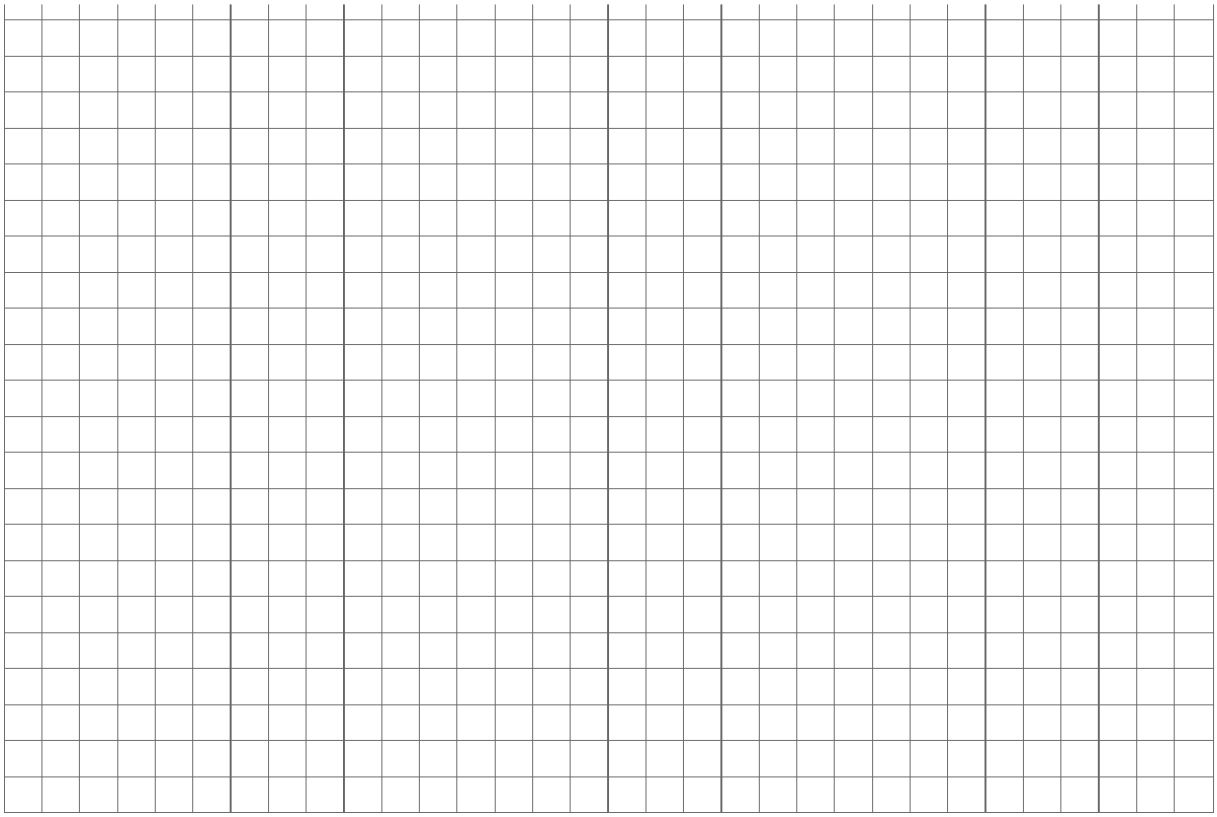
Eine Matrix kann aufgefasst werden als ein Feld von Feldern.

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
```