Name

Matr. no.

Exam Programming III - TI3

WS 12

Prof Dr K. Baer

Aids: 1 sheet DIN A4, labelled on both sides.

Processing time: 90 min.

1. Please enter your name and matriculation number first!
2. Check that the task sheets are complete.
3. The exam consists of 6 tasks. Get a brief overview of the tasks and start with the task that is most likely to give you a sense of achievement.
4. Read the task carefully before solving it!
5. Use the space provided on the task sheets to answer the questions.
6. Write legibly. Illegible parts will be awarded 0 points!

Good luck!

Task	1	2	3	4	5	6	Total
Points	19	12	15	18	6	20	90
achieved							

Task 1 (19 points = 12+5+2)

Given the following implementation:

```
#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) What does the following test programme output?

```
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) What is meant by "slicing"?
In which lines does slicing occur in the above code?

An object of a derived class can be linked to an object of a direct or indirect class. base class (but not vice versa!).
During assignment, only the elements of the base class are copied component by component

Analogue for hands:
- Object is treated like an object of the base class when accessed via the base class reference!
e.g. for redefined methods in the derived class, the base class version is called!

m1 = *b1;
Object of type bottom is assigned to object of type middle

m1.useTop(*b1);
useTop expects object of type Top, but receives Bottom

b1->useTop(m1);
analogue with
Middle

c) Would anything change in this example if you were to declare the destructor of Top virtual? Give reasons for your answer!

T1 is the base class pointer, but points to the bottom object.

The line

```
delete t1;
```

causes the destructor of Top to be called.

If the destructor is declared virtual, the correct destructor would be called from bottom

Output would then be in the second last line:not only

~top

but

~Bottom ~Middle ~Part ~Top

Task 2 (12 points=4+4+4)

a) Please note that initial values may be required in the following declarations. This is given:

```
int i = 5;
```

Define a pointer p1 to the type int:

```
int *p1;
```

Define a pointer p2 to the type constant int:

```
const int *p2;
```

Define a constant pointer p3 to the type int.

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

Define a constant pointer p4 to the type constant int:

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

// 4 points
```

b) Which of the pointers declared in a.) can be used in the following functions? (Please tick accordingly in the table below).

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

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

c) The following function declarations are given:

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

and the following variable declarations:

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

Which function calls are possible and which function is then called? Enter the number of the function.

	No.	wrong
`f(x,l);`		x
`f(i,j);`	1	
`f(k,l);`	2	
`f(x,y);`		x

Task 3 (15 points=3+4+5+3)

a) What basic template classes does the STL offer for sequential containers?

Vector
list
deque

b) How do these classes differ in terms of element access and insertion/removal of elements (front, centre, back)? Indicate the complexity of the operations performed.

	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) What are iterators and what are they used for in the STL? Which iterator categories are distinguished in the STL and what are the main differences?

Iterators are a generalisation of the pointer concept Typical use of C pointers

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

Typical use of iterators vector < char > buf;

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

vector<> can also be used with standard C-pointers
Iterators are the interface that the generic algorithms of the STL use to work together with containers

Input Output Forward Bidirectional Random access

Set of supported operators. Until Forward only ++ (next to *, ==, != →)
From BiDir also -, RA full set

d) What is the relationship between the iterator and the corresponding reverse iterator? Why? Which iterator categories allow the provision of a reverse iterator?

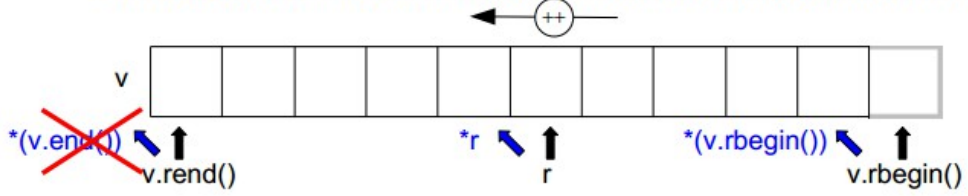
Reverse-Iterator hat dieselbe Schnittstelle wie der zugrunde liegende Bidirectional- oder Random-Access-Iterator.

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.



Task 4 (18 points=10+8)

A "PlzOrtMapper" class is to be developed that supports the mapping of postcodes and towns.

Given is a text file according to the sample below, in which an assignment of postcode to city is specified for each line:

```
89075 Ulm
89076 Ulm
89077 Ulm
89077 Ulm-
      Soeflingen
89078 Ulm
89079 Goegglingen
89079 Ulm
89079 Ulm-Danube
      Valley
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
```

This shows that a city can have several postcodes (e.g. Ulm). However, a postcode can also be valid for several towns (e.g. 89079).

A multimap could be used as an internal data structure for recording and managing postcode/city pairings:

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

The following functions are to be developed for the "PlzOrtMapper" class:

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

- The method listPlz() should list all postcode / city pairs for a given city.
- The zaehleVorkommen() method returns the number of postcode/location pairings

to a given location.

- The listPlzPedicate() method returns all postcode / city pairings in which the city name contains the given search string.

Notes on the STL:

- Multimap offers an element function "equal_range", which can be used to determine the limits of an area in the map in which the key has a certain value:

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

- The distance between two iterators can be determined using the global function distance:

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

- The global function find_if can be used to search for values in a container that fulfil a specific condition (a predicate). To do this, the function requires the area in the container to be searched and the comparison function. It returns the position of the first element found (or last, if nothing was found).

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

Implement the three methods.

Implement the necessary comparison function in the listPlzPedicate function as a functor.

Note: Strings offer the function

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

which returns the position of the first occurrence of the search string.

```
string listPlz(string city){
        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 ) << " Occurrence"; buffer
        }
        else {
                << "Nothing found for " << 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 search){
        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 Contains {
public:
    Contains(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
```

Task 5 (6 points)

D e v e l o p a template function that displays the sum of the objects in a map. 2nd place is calculated and returned. The+ operator is defined for these objects.

The following programme section, for example, should be executable with your function and output the value 240 in the example shown:

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

        exam result["Mayer"] = 90;
        exam result["Müller"] = 80;
        exam result["Schmidt"] = 70;


        cout <<"Sum = "<< sum(exam result) << 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 is empty.....";
            sum = i->second;
            for (i++; i != m.end(); ++i) {
                  sum = sum + i->second;
            }
      }
      return sum;
}
```

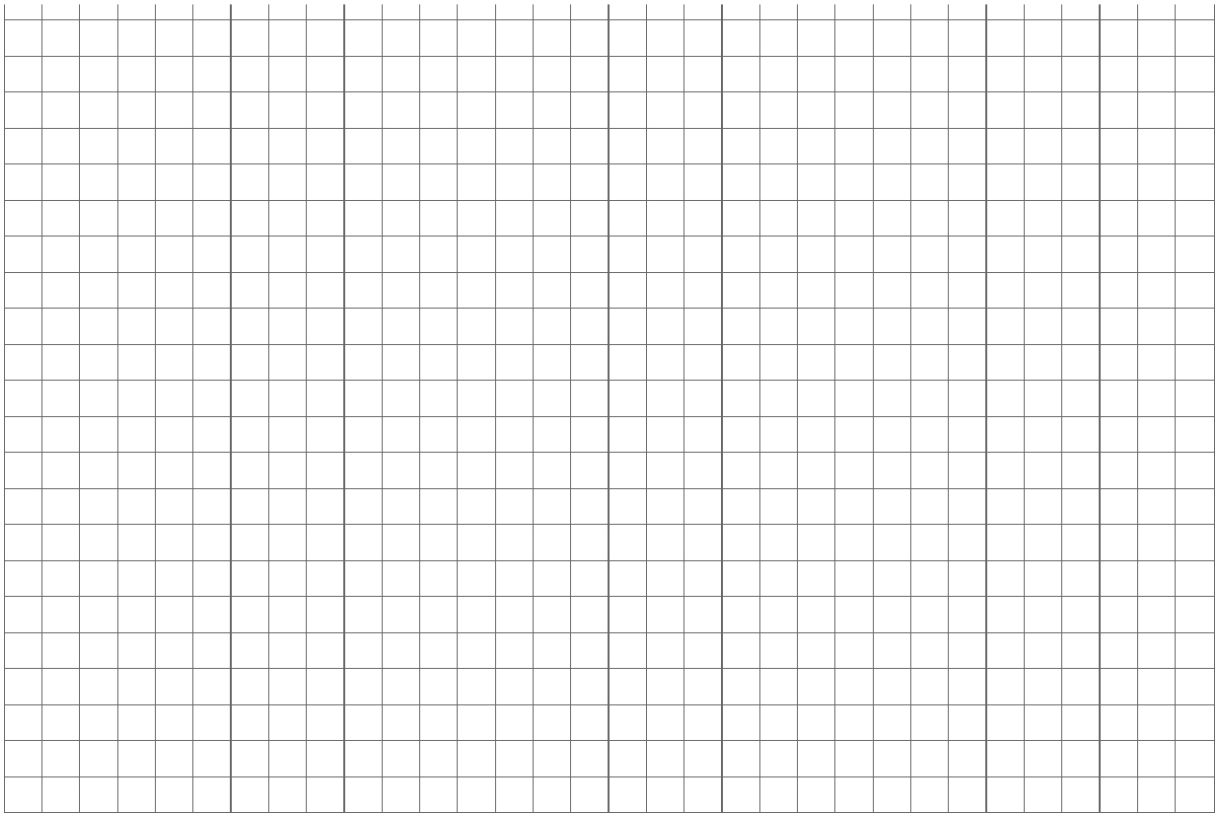
Task 6 (20 points)

A matrix can be understood as an array of fields.

When using the STL, you can easily develop a template class for a matrix by creating a template class
`matrix`, which is publicly derived from `vector< vector<T> >`. Develop such a template class with the following public methods:

- a 2-digit constructor for specifying the matrix dimension
- the `Rows()` and `Columns()` functions for determining the dimensions
- an `init()` function that allows the matrix elements to be initialised with a specified value
- a function `I()` that returns the unit matrix (0 everywhere, only diagonal occupied by 1)
- the output operator operator<<, with which the matrix is output in the form of the following unit matrix (i.e., row no.: values separated by spaces)

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> >(,
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

};// Class 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
```