Name

Matr. no.

Exam Programming III - TI3

SS 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	20	6	14	20	6	24	90
achieved							

Task 1 (20 points = 2+11+2+2+1+2)

Given the following implementation:

```
class Component
{ protected:
        string name;
public:
        Component(){ name = "???"; cout << "Component created" << endl; }
        ~Component(){ cout << "Component destroyed" <<
        endl;} void showName(){ cout << "Name: " << name <<
        endl; }
};

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

class B : public Bauteil {
public:
        B() { name = "B"; cout << name << " created" << endl; }
        ~B(){ cout << name << " destroyed" << endl; }
        void showName(){ cout << "B-Parts" << endl; Component::showName(); }
};


class Assembly {
private:
        set<component*> group;
        typedef set<component*>::iterator
Iter; public:
        Assembly(){ cout << "Assembly created" << endl; }
        assembly(const assembly& other){ cout << "assembly copied" << endl; for(Iter
                i=other.group.begin(); i  != other.group.end(); ++i){
                        group.insert(*i);
                }
        }
        virtual ~assembly(){ cout << "Assembly destroyed" << endl; } void
        hinzufuegen(Bauteil* teil){ gruppe.insert(teil); }
        void remove( component* part){
        group.erase(part); } void showElements(){----------
                cout << endl << "                                        ";
                cout << "\nAssembly group populated with: " <<
                endl; for(Iter i = group.begin(); i != group.end();
                ++i){
                        (*i)->showName();----------
                }
                cout << "                                        " << endl;
        }
};
```

a) Explain the term "late bonding"!

- Counterpart to static binding, i.e. method binding at compile time
- Relevant in polymorphic class hierarchies
- At runtime, the actual object type is used to determine which method is to be executed.
- Example

```
: class A
{
     virtual void m1(){ // do what ...};
}
class B : public A {
    void m1(){ // do something else ... }
}

void test(){
          A* ptr_A = new B;
          ptr_A->m1();    // m1() of class B is executed
}
```

b) What does the following test programme output?
For clarification, please enter the **line number** for each output line in the test() method that generates this output.

```
1   #include "bauteil.h"
2
3   void display(assembly group)
4    group.showElements();
5   }
6
7   void test(){
8    Assembly group group1;
9
10     A* a1 = new  A();
11     B* b1 = new  B();
12  Component*  = new B();
              b2
13
14   group1.add(a1);
15   group1.add(b1);
16   group1.add(b2);
17
18   display(group1);
19
20   b1->showName();
21
22   Assembly group group2 = group1;
23   A a2;
24   group2.add(&a2);
25   group2.showElements();
26   delete b1;
27   delete b2;
28  }
29
30  int main(){
31   test();
32   cin.sync();cin.get();
33  }
```

```
8Assembly group created
10Component
created A
created
{11Component created
B created
Component generated
B created
Module copied
-------------------------
Assembly equipped with:
Name: A
Name: B

Name: B
-------------------------
 Assembly destroyed
 B-parts
 Name: B
 Module copied
 Component generated
 A created

-------------------------
Assembly equipped with:
Name: A
Name: A
Name: B
Name: B
-------------------------
 B destroyed
 Component destroyed
 Component destroyed
 A destroyed
 Component destroyed
 Assembly destroyed
 Assembly destroyed
```

	4 x 0,5 =	2
18		1
18		1
18		1
20		1
22		1
23		0,5
25		1
26		0,5
27		0,5
28		0,5
		0,5
		0,5

c) Component b1 is output at different locations. Does it always appear the same? Give reasons for your answer.

No.
Access partly via base class pointer, partly via object.
ShowName not declared virtual, but overwritten.
i.e. different behaviour:
- via base class pointer: Method of the base classes
- via object directly: overwritten methods

d) Explain your expenditure on lines 26-28.

estructor in base class not virtual!
26: delete with object pointer of type B, i.e. the destructor of b is called and then the destructor of the base class

27: delete with base class pointer. The destructor of the base class is called, as the destructor is not declared virtual!

28: the method test() is terminated, i.e. all local variables / objects are destroyed, starting with the last one created: a2, g2,g1.

Only pointers are managed in the groups, i.e. the objects on the heap are not cleared → Memory leak

e) Explain in detail which functions are called in line 22!

Copy constructor

New object is created from an existing one

f) What is required to implement the function

```
Assembly(const assembly& other)
```

to note?

No copy of objects to which the pointers point is created

Task 2 (6 points)

What is the sensible approach to operator overloading? Explain - using the example of a fraction class - which problems typically occur and how best to deal with them. For example, in the case of the class Fraction, how do you ensure that the implementation fulfils the commutative law?

- Short form operators as element function in the class
- Return breakage&
- normal, binary operators (+,-,*/) as global functions, return break

- This enables the operands to be swapped
- After weighing up the pros and cons, the global functions may be granted access to private data elements of the class via friend declaration (only if there are clear performance advantages, otherwise access via getter/setter).
- Reduce to short form operators to avoid redundancy
- no conversion function due to possible ambiguities, better "asTargetType()"

Task 3 (14 points=7+7)

a) Explain the friend concept in C++!
(purpose, use, example)

A class can allow other classes or functions or methods outside the class (so-called "friends") to access private data elements and methods
Only the class itself can decide who it wants to have as a friend!

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

Softening of data encapsulation!!!
Acceptable if otherwise required data access is very time-consuming.

```
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) < ERROR SHORTAGE ) &&
  ( c1.im < ERROR SHORTAGE ) );
```

b) What is a binder in the STL?
(purpose, use, example)

What: → special functor
Purpose: Reduction in the number of required functors
For two-valued functors, one of the operands
bound by binders. This means that only 2-valued functors
& 2 binders are required for binding the 1st operand and
binding the 2nd operand.

```
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;
};
```

Example:

```
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, " "),
```

Task 4 (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
```

Task 5 (6 points)

Explain what is meant by a downcast.

Describe the situations in which downcasts are problematic. Explain how to handle them correctly.

Give an example.

Down-Cast
Base class pointer is converted to pointer to derived class by cast
Caution: Down-casts may be unsafe and dangerous!!! unproblematic:
At runtime, the base class pointer actually points to the object type that is being cast to
problematic
At runtime, the base class pointer points to the parent object that is cast to. becomes.

Correct handling:
dynamic-Cast returns a NULL pointer if yp does not match, this can be checked and error handling can take place if necessary.

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

pkwPtr = dynamic_cast<Pkw*>(kfzPtr); // returns null
pointer! if (pkwPtr == 0)
        cout << "Error!!!" << endl;
else {
        pkwPtr->display();
        pkwPtr->get in(3);
}
```

Task 6 (24 points = 2+1+6+4+11)

Develop a "PlzOrtMapper" class that supports the mapping of postcodes and cities.

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) Explain the advantages of a MultiMap.
What elements does a map / multimap consist of?

Efficient access via key (logarithmically) sorted
several entries / keys possible
pair<const Key, T>

b) Develop the "PlzOrtMapper" class. It should support the following functions:

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

- First of all, think about a suitable data structure with which you can most easily manage postcode/location pairings within your class.
- If the file cannot be opened, the "read in" method should generate a corresponding "FileError" error object for exception handling. The file name should be taken from the error object at the calling point.
- The method zaehleVorkommen() returns the number of postcode/location pairings for a given location.
- The method listPlzOrt() should return postcode/city pairings as a string. It should be possible to call the function in such a way that
  - all pairings with the exact place name,
  - all pairings in which the city name contains the given search string,
  - all pairings that belong to a postcode are

  listed.

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 );
```

b1) Declare the internal **data structure** for recording and managing postcode/city pairings.

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

b2) Implement the method **read in** together with the error class "FileError"

```
bool read(string filename) throw (FileError){
string    location;
    int          plz;

    ifstream file(filename); // Input stream to file if (!file)
    {
          throw FileError("Couldn't open file " + filename);
  }

    while (!file.eof()) { if
          (file >> plz) {
                file >> location;
                 m_plz.insert( pair<string,int>( city,postcode) );
          }

  }
    return true;
}

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

b3) Implement the method **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) Implement the method listPlzOrt.

Assistance:

- You can map the different searches using different functors

- Once the search is for the postcode (→ int), the other times for the city (→ string). If you implement listPlzOrt as a template, you can save the corresponding implementation effort.

- The usual output operator << can be used to write to stringstream objects as on the console. Stringstream objects have a methdoe str() for conversion to a string.

- 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. A test programme should provide output according to the

following pattern:

```
List postcode(s) for location: Ulm
89070 Ulm
89073 Ulm
89074 Ulm
89075 Ulm
... (etc.)

List postcodes for places containing the name element: Soef 89077
Ulm-Soeflingen
89081 Ulm-Soeflingen

List of all places belonging to the same postcode: 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 search){ m_search = search; }

        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 search){ m_search = search; }

        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 search){ m_search = search; }

        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();
 }
```