BERGISCHE UNIVERSITÄT
WUPPERTAL
GAUSS-STRASSE 20
42119 WUPPERTAL
TELEFAX (0202) 439-2901
TELEFON (0202) 439-0
WWW www.uni-wuppertal.de

Fachbereich C

MATHEMATIK UND NATURWISSENSCHAFTEN
Fachgruppe Mathematik und Informatik
Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

E-Mail: Hans-Juergen.Buhl@math.uni-wuppertal.de
WWW: http://www.math.uni-wuppertal.de/∼buhl

Generische Programmierung
(Spezielle Kapitel der praktischen Informatik)

WS 2010/2011 – Übungsblatt 9

20. Dezember 2010
Abgabe: bis 10. Januar 2011 an *c.seepold@uni-wuppertal.de*

Aufgabe 1. *EqualTypes*

Testen Sie mit Hilfe des Templates

```
template< typename T1, typename T2 >
struct EqualTypes {
enum { result = false };
};
template< typename T >
struct EqualTypes<T,T> {
enum { result = true };
};
```

in wieweit durch typedefs erklärte Typnamen von C++ als identisch zu ihren Ursprungstypen aufgefasst werden (bei selbstdefinierten Klassen, enum's, ...).

Aufgabe 2. *gfilt*

Besorgen Sie sich von http://www.bdsoft.com/tools/stlfilt.html das Programm `gfilt` und installieren Sie es in ∼`/bin`.

Vergleichen Sie die Ausgabe von

```
g++ rtmap.cpp -o rtmap
```

mit derjenigen von

```
gfilt rtmap.cpp -o rtmap
```

(bei Benutzung der Datei `rtmap.cpp` mit dem Inhalt:)

```
#include <map>
#include <algorithm>
#include <cmath>

const int values[] = { 1,2,3,4,5 };
const int NVALS = sizeof values / sizeof (int);

int main()
{
        using namespace std;

        typedef map<int, double> valmap;

        valmap m;

        for (int i = 0; i < NVALS; i++)
            m.insert(make_pair(values[i], pow(values[i], .5)));

        valmap::iterator it = 100;              // error
        valmap::iterator it2(100);              // error
        m.insert(1,2);                          // error

        return 0;
}
```

Interpretieren Sie die Brauchbarkeit der Fehlermeldungen!

Lesen Sie

„How can any human hope to understand these overly verbose template-based error messages?“

(http://www.parashift.com/c++-faq-lite/templates.html#faq-35.17).

Aufgabe 3. *Inheritance check*

Welche Ausgabe produziert das Programm:

```
template <class Derived, class Base>
class Check
{
class Nope {};
class Yep {char Dummy[3];};
static Yep Test(Base*);
static Nope Test(...);
public:
enum {
IsDerived = sizeof(Test(static_cast<Derived*>(0)))
== sizeof(Yep)
};
};
```

```
class X {};
class Y : public X {};

int main()
{
cout << Check<Y, X>::IsDerived << endl;
cout << Check<int, string>::IsDerived << endl;
return 0;
}
```

Warum?

Schreiben Sie eine kurze Anwendungsdokumentation für die Metafunktion `Check<.,.>::IsDerived`

Aufgabe 4. *boost concept check*

Statten Sie die Templatefunktion `arithAverage(T1 x, T2 y)` (Aufgabe 1/Blatt 6) mit eigenen Boost-Concepts aus und benutzen Sie das Makro `BOOST_CONCEPT_REQUIRE()` zur Erzwingung der Benutzung nur geeigneter generischer Parameter bei der Instantiierung dieser Funktion. Vergessen Sie nicht die Konstruktion und Testcompilation geeigneter Archetypen sowie die Bereitstellung geeigneter assoziierter Typen.