BERGISCHE
UNIVERSITÄT
WUPPERTAL

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik/Numerik

Fachbereich C
Mathematik und Naturwissenschaften,
Mathematik und Informatik

E-MAIL buhl@math.uni-wuppertal.de
WWW www.math.uni-wuppertal.de/~buhl

DATUM 15. Januar 2014

# generische Programmierung

**WS 2013/2014 – Übungsblatt 11**
**Ausgabe: 14. Januar 2014**
**Abgabe bis 22. Januar 2014 an: *kheidsch@studs.math.uni-wuppertal.de***

**Aufgabe 1.** *Fibonacci-Zahlen*

Testen Sie die aktuelle Version (C++11) der Metafunktion `fib` von

http://ideone.com/7exywB

und vergleichen Sie mit

http://stackoverflow.com/questions/908256/getting-template-metaprogramming-compile-time-constants-at-runtime.

Welche Unterschiede stellen Sie fest? Warum wurden ander Sprachkonstrukte benutzt?

**Aufgabe 2.** *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 3.** *Vorbedingungen in Templates*

Demonstrieren Sie die Verfahrensweisen von Abschnitt 2.7, um die Template-Metafunktion `fact` (Aufgabe 1 von Übungsblatt 9) vor dem Aufruf mit einem negativen Templateparameter-Wert zu schützen. Testen Sie!

Warum nennt man `fact` in diesem Zusammenhang eine Metafunktion und nicht einfach eine Funktion?

**Aufgabe 4.** *Vorbedingungen in Templates: Fortsetzung*

Demonstrieren Sie analog, wie Sie die Template-Metafunktion `template<int n, int m> struct cpower::result` gegen unsinnige Parameter absichern können.

**Aufgabe 5.** *Vorbedingungen in Templates: Fortsetzung 2*

Demonstrieren Sie anlog, wie Sie die Template-Metafunktion `template <unsigned long N> struct binary::value` gegen einen unsinnigen Parameter absichern.