



BERGISCHE
UNIVERSITÄT
WUPPERTAL

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

Fakultät für
Mathematik und Naturwissenschaften,
Mathematik und Informatik

E-MAIL buhl@math.uni-wuppertal.de

WWW www.math.uni-wuppertal.de/~buhl

DATUM 11. Januar 2018

generische Programmierung

WS 2017/2018 – Übungsblatt 10

Ausgabe: 11. Januar 2018

Abgabe bis 18. Januar 2018 an: <mailto:gregor.hildebrand@uni-wuppertal.de>

Aufgabe 1. C++11 und zu erwartende Konzepte

Lesen Sie das 2008 auf Konzepte hin geänderte Kapitel 26

[Draft N2736](#)

des neuen C++-Standards.

Welche Konzepte wurden in `numeric_concepts` bereitgestellt? Wo wird `ArithmeticLike`, wo `FloatingPointType` benutzt? Warum?

Wie sehen die Requirements von `accumulate()` in 26.6, wie diejenigen von `inner_product()` aus?

Aufgabe 2. Template binary

Testen Sie:

```
template <unsigned long N>
struct binary
{
    static unsigned const value
        = binary<N/10>::value * 2 // prepend higher bits
        + N%10; // to lowest bit
};

template <> // specialization
struct binary<0> // terminates recursion
{
    static unsigned const value = 0;
};
```

Was wird hier berechnet? Lassen Sie eine Tabelle berechneter Werte ausdrucken. Welche ähnlichen Anwendungen von Templates erscheinen Ihnen nützlich?

Aufgabe 3. *Metaprogrammierung*

Lesen Sie

http://www.boost.org/doc/libs/1_31_0/libs/mpl/doc/paper/html/intro.html

und testen Sie das `all_permutations()`-Beispiel.

Welche Einsatzgebiete sieht der Autor für Metaprogrammierung?

Welche Einsatzgebiete sehen Sie?

Aufgabe 4. *Fibonacci-Zahlen*

Testen Sie die aktuelle Version 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 andere Sprachkonstrukte benutzt?

Aufgabe 5. *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, ...).