



**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 8. Dezember 2014

generische Programmierung

WS 2014/2015 – Übungsblatt 7

Ausgabe: 8. Dezember 2014

Abgabe bis 15. Dezember 2014 an: <mailto:125319@uni-wuppertal.de>

Aufgabe 1. *promote_trait*

Benutzen Sie das Template `promote_trait` in einer Templatefunktion `my_min` und testen Sie es für mindestens 10 unterschiedliche Typpaare.

Schreiben Sie ein ähnliches Template `arithAverage_trait` zur Nutzung in einer Templatefunktion `arithAverage(T x, T y)` zweier numerischer skalarer Parameter. Der Ergebnistyp für `T = int` soll dabei jedoch `double` sein (warum?). Welche Konzepte sollte `T` modellieren?

Aufgabe 2. *geomMittel*

Modifizieren Sie `geomMittel2(const T1&, const T2&)` (Aufgabe 1 von Übungsblatt 1) zu einer Funktion

```
template <class InputIterator, class T>  
T geomMittel(InputIterator first, InputIterator last, T init);
```

zur Berechnung des geometrischen Mittels der Elemente des Arguments im Bereich `[first, last)`. Mit welchem dritten Argument sollte `geomMittel` aufgerufen werden?

Schreiben Sie eine Dokumentation analog zur STL-Dokumentation <http://www.sgi.com/tech/stl/accumulate.html>.

Aufgabe 3. *geomMittel Fortsetzung*

Ergänzen Sie Ihre Lösung von Aufgabe 2 um eine Überprüfung des Generischen Parameters `InputIterator` auf eben diese Eigenschaft (vgl. 1.19.1 der Materialsammlung) und des generischen Parameters `T` auf das Requirement `is_arithmetic` (nennen Sie `T` auch geeignet um).

Aufgabe 4. „errechnete“ Funktionsergebnistypen

Lesen Sie Abschnitt 1.19.4 der Materialsammlung und beschreiben Sie, wie sie in C++11 den Ergebnistyp von `geomMittel2` automatisch aus T1 und T2 bestimmen lassen können, statt ihn fest als `double` anzunehmen. Ändern Sie Ihren Code entsprechend.

Vergleichen Sie dazu auch Seite 378 von

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf>.

Was ändert sich an der „Function return type deduction“ in C++14 (<http://en.wikipedia.org/wiki/C%2B%2B14>)?

Aufgabe 5. *non-type template-Parameter*

Warum funktioniert:

```
#include <iostream>
using namespace std;

template <bool k> void print ()
{
    if(k==true) // oder auch: if(k)
        cout << "true" << endl;
    else
        cout << "false" << endl;
    return;
}

int main()
{
    print <7>();
    print <0>();

    return 0;
}
```

Wie viele und welche Inkarnationen der Templatefunktion `print()` werden automatisch erzeugt?

Welche Regeln gelten für die Typ-Transformation von *non-type* Funktionstemplate-Parametern über diejenigen der *type* Funktionstemplate-Parameter hinaus?

Ergänzen Sie in `main()` Aufrufe von

```
print<8>(); print<9>(); ...
```

Was erwarten Sie? Überprüfen Sie Ihre Erwartungen durch Benutzung von `nm`.

Was ändert sich in diesem Umfeld in C++17?