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 6

29. November 2010
Abgabe: bis 6. Dezember 2010 an *c.seepold@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. *SunStudio CC*

Welche Vorteile bei der C++-Entwicklung bietet der SunStudio-Compiler

http://www.pcwelt.de/it-profi/business-ticker/593705/sun_studio_12_auf_multicore_cpus_getrimmt/?

Wie unterscheidet sich das C++-Namemangling dieses Compilers von dem des GNU-Compilers `g++`?

Was sind die Auswirkungen dieses Unterschieds?

Lesen Sie dazu

http://docs.sun.com/app/docs/doc/819-5267

und

http://developers.sun.com/solaris/articles/studio_qs.html.

Aufgabe 3. *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`.

Aufgabe 4. *Template-Funktion mit array-Parameter*

Vergleichen Sie

```
#include <iostream>
#include <numeric>

template <typename Type>
Type sum(Type *tp, size_t n)
{
    return std::accumulate(tp, tp+n, Type());
}

int main()
{
    int x[10];
    for(int l = 0; l < 10; l++)
        x[l] = l;
```

```
    std::cout << sum(x, 10) << std::endl;

    return(0);
}
```

mit

```
#include <iostream>
#include <numeric>

template <typename Type, size_t n>
Type sum(const Type (&tp)[n])
{
    return std::accumulate(tp, tp+n, Type());
}

int main()
{
    int x[10];
    for(int l = 0; l < 10; l++)
        x[l] = l;

    std::cout << sum(x) << std::endl;

    return(0);
}
```

Welche Vor- und Nachteile bietet die erste Variante gegenüber der zweiten? Schreiben Sie eine Template-Funktion `unsigned int get_dimension()`, die die Anzahl der Komponenten eines übergebenen Arrays als Funktionsergebnis liefert.