kurze Wiederholung „class templates“

- Ein „class template“ ist ein Template, dass mit einem oder mehreren „typename“-Parametern implementiert wird.
- Um solch ein Template zu benutzen, übergibt man dem Template die Typen als Argumente. Daraufhin ist das „class template“ für diese Typen instanziiert (und auch compiliert).
- Nur diejenigen „member functions“, die auch wirklich benutzt werden, sind zur Laufzeit instanziiert.
- „class templates“ lassen sich auf bestimmte Typen spezialisieren.
- Es ist ausserdem möglich Standardwerte für die Template-Parameter zu definieren.
    - Diese können sich auf vorher benutzte Template-Parameter beziehen.

Nontype Function Template Parameters

- Template Parameter müssen nicht zwangsweise Types sein
- Es sind auch gewöhnliche Werte erlaubt

- „nontype parameter“funktionieren auch für funktionale Templates:

```
template <typename T, int VAL>
T addValue (T const& x)
{
return x + VAL;
}
```

- Nützlich wenn Funktionen bzw. Operationen als Parameter genutzt werden
- z.B. mit der Standart Template Library (STL) kann eine Instanz dieser Funktion übergeben werden um einen Wert zu jedem Element einer „collection“ hinzuzufügen:

Nontype Function Template Parameters

```
1 std::transform (source.begin(), source.end(),  //start and end of source
2 dest.begin(),  //start of destination
3 addValue<int,5>);  //operation
```

- bei obenstehender Lösung gibt es ein Problem
- addValue (int,5) ist ein Funktionstemplate - kann keine Template Parameter ableiten
- deshalb müssen wir den genauen Typ des Templates angeben:

```
1 std::transform (source.begin(), source.end(),  //start and end of source
2 dest.begin(),  //start of destination
3 (int(*)(int const&)) addValue<int,5>);  //operation
```

Restrictions for Nontype Template Parameters

- „nontype template„-Parameter haben Einschränkungen:
- erlaubt: „integral values„ (z.B. „enums“), Objektpointer
- nicht erlaubt: „floating-point numbers“ und „class-type“-Objekte

```
template <double VAT>   //ERROR: floating−point values are not
double process (double v)   //allowed as template parameters
{
return v * VAT;
}

template <std::string name>   //ERROR: class−type objects are not
class MyClass{   // allowed as template parameters
...
};
```

- Anmerkung: „floating-point numbers“ sind aus Historischen Gründen nicht erlaubt, allerdings ist davon auszugehen, dass dies in zukünftigen C++ Versionen möglich sein wird.

Restrictions for Nontype Template Parameters

- Im Gegensatz zu Objektpointern, haben „string literals“ eine interne Verknüpfung (zwei „string literals“ mit dem gleichem Wert, aber in zwei verschiedenen Modulen sind verschiedene Objekte). Aus diesem Grund kann man „string literals“ auch nicht als Argument für Templates benutzen:

```
template <char const* name>
class MyClass {
...
};

MyClass<"hello"> x;   //ERROR: string literal "hello" not allowed
```

Restrictions for Nontype Template Parameters

- globale Pointer dürfen auch nicht verwendet werden:

```
template <char const* name>
class MyClass {
...
};

char const* s = "hello";

MyClass<s> x;   //ERROR: s is pointer to object with internal linkage
```

Restrictions for Nontype Template Parameters

- folgendes funktioniert aber trotzdem:

```
template <char const* name>
class MyClass {
...
};

extern char const s[] = "hello";

MyClass<s> x;   //OK
```

- Das globale Array s wird mit der Zeichefolge „hello“initialisiert, deshalb ist s ein Objekt mit externer „linkage“.

Summary

- Templates können Templateparameter haben, die eher Werte sind als Typen
- Man kann keine „floating-point numbers“, „class-type objects“ oder „objects with internal linkage“ als Argumente für „nontype template“-Parameter übergeben.

Tricky Basics

- typename keyword
- Memberfunktionen definieren und Klassen als Templates benutzen

typename keyword

- wird in C++ benutzt um einen „identifier“ in einem Template als Typ festzulegen.

```
template <typename T>
class MyClass {
typename T::SubType * ptr;
...
};
```

- das zweite „typename“ definiert SubType als ein Typ, innerhalb von der Klasse T (ohne das „typename“ würde SubType als „static member“ angesehen.

typename keyword

- Eine übliche Anwendung für den typename ist der Zugriff auf Iteratoren auf „STL containers“ in einem Template:

```
#include <iostream>

// print elements of an STL container
template <typename T>
void printcoll (T const& coll)
{
    typename T::const_iterator pos;  // iterator to iterate over coll
    typename T::const_iterator end(coll.end());  // end position

    for (pos=coll.begin(); pos!=end; ++pos) {
        std::cout << *pos << ' ';
    }
    std::cout << std::endl;
}
```

typename keyword

- Funktionstemplate mit einem STL contatiner als Aufrufparameter „T“
- um diesen abzuzählen, wird ein Zähler („iterator“) benutzt, der als type const iterator innerhalb jeder „STL container“-Klasse deklariert wird:

```
class stlcontainer {
...
typedef ... iterator;    //iterator for read/write access
typedef ... const_iterator;    //iterator for read access
...
};
```

- Um auf den „type const iterator“ (mit dem Template-Typ T) zuzugreifen, muss dieser mit einem „typename“ definiert werden:

```
typename T::const_iterator pos;
```