

# MATERIALSAMMLUNG - GENERISCHE PROGRAMMIERUNG

Prof. Dr. Hans-Jürgen Buhl



Wintersemester 2013/2014

Fachgruppe Mathematik und Informatik  
Fachbereich C — Mathematik und Naturwissenschaften  
Bergische Universität Wuppertal

Praktische Informatik  
PIBUW - WS13/14  
Oktober 2013  
4. Auflage, 2013



# Inhaltsverzeichnis

<b>1. Generische Programmierung</b>	<b>7</b>
1.1. Was ist generische Programmierung?	7
1.2. Beispiel einer generischen Funktion mit einem generischen Parameter	7
1.3. Benutzung von <code>std::swap()</code>	8
1.4. Einsatzgebiete und Beispielrepositorien für generische Konstrukte: die STL, ...	9
1.5. Instanzen generischer Objekte	10
1.5.1. Objekt-Dateien: wo sind welche Instanzen meiner generischen Objekte (nm und c++filt)?	
1.5.2. Erstellen und Benutzen von statischen Bibliotheken	22
1.5.3. Erstellen und Benutzen einer „shared object“- Bibliothek	25
1.5.4. Bibliotheksmanagement insbesondere unter verschiedenen Betriebssystemen	26
1.6. STL-Templatequellen unter SuSE-Linux fürs zeilenweise Debuggen auch innerhalb der STL-Routi	
1.7. Automatisch überprüfte Requirements an Template-Parameter	28
1.7.1. mit Hilfe von <code>BOOST_STATIC_ASSERT()</code>	30
1.7.2. mit Hilfe des c++11-Modus des g++	31
1.8. „horrible error messages“ bei STL-Nutzung	32
1.8.1. C++11 <code>type_traits</code>	34
1.8.2. BOOST <code>type_traits</code>	35
1.8.3. <code>numeric_limits</code> als Typ-Abbildung	37
1.9. Rückblick: typsichere Funktionsbenutzung	38
1.10. Concepts und zielführende knappe Fehlermeldungen bei der Benutzung fehlerhafter aktueller gene	
1.11. Zielgerichtete Fehlermeldungen bei Nutzung einer C++-Standardbibliothek mit Konzepten	42
1.12. Die Boost-Bibliotheken	43
1.13. Orte, wo statische Zusicherungen benutzt werden	43
1.14. statische Zusicherunge in C++11	43
1.15. Fehlermeldungen bei uneingeschränkter Generizität (Fortsetzung von 1.8)	44
1.16. Verbesserte Fehlermeldungen bei Nutzung von <code>StaticAssert</code>	47
1.16.1. <code>RandomAccessIterator</code>	47
1.16.2. Nicht instanziiierbare Klassen	50
1.16.3. Erzwingung gleicher Typen	50
1.16.4. Funktionen mit „(int/float/...) type promotion“-Returntyp	51
1.16.5. Auf Unterklassen eingeschränkte Generizität	52
1.16.6. g++ <code>type_traits Compiler Extensions</code>	52
1.16.7. Type Traits in D	52
1.16.8. C++ <code>has_member</code> fehlt	53
1.16.9. SFINAE	54
1.16.10.C++11: Traits mit <code>decltype</code> statt <code>sizeof()</code> -Tricks	55

1.16.11.Überladene Templatefunktionen . . . . .	55
1.16.11.1.enable_if-Funktionen . . . . .	55
1.16.11.2.Konflikt beim enable_if-Funktionsüberladen . . . . .	56
1.16.11.3.bedingte „template class specializations“ . . . . .	56
1.17. Template-Deklarationen zur Erzeugung von Objektdateien mit einer Ansammlung von 7	
1.18. Wo ist die Template-Instanz? . . . . .	60
1.19. C++11 extern template . . . . .	60
1.20. Generic Programming . . . . .	61
1.21. Generic Programming in ConceptC++ . . . . .	61
1.22. Concepts, concept_maps, axioms . . . . .	62
1.23. ConceptC++-Tutorial . . . . .	64
1.24. C++11: was aus C++03 ist nicht mehr da? . . . . .	70
1.25. aktueller Workaround: Nutzung von Typetraits statt von Concepts . . . . .	70
1.26. Assoziierte Typen, Tags, Tag-Dispatching . . . . .	72
1.27. Archetypen . . . . .	74
1.28. Generic Programming Techniques of the BOOST Libraries . . . . .	74
1.29. POD-Typen und trait-fallweises Überladen . . . . .	75
1.30. Eigene Klassen-Tags und Tag-Dispatching oder fallweise Spezialisierung . . . . .	75
1.31. Für die Template-Programmierung interessante Neuerungen in C++11 . . . . .	76
1.32. Curiously recurring template pattern . . . . .	77
1.33. Objektorientierte Entwurfsmuster . . . . .	77
1.34. Sprechweisen bei C++(11)-Programmkonstrukten . . . . .	77
1.35. Type Generator . . . . .	78
1.36. Object Generator . . . . .	78
1.37. Parameterized Base Class . . . . .	78
1.38. Barton-Nackman Trick (Restricted Template Expansion) . . . . .	78
1.39. Die Boost Concept Check Library (BCCL) . . . . .	79
1.39.1. Difference of Draft C++ Cocepts and BCCL . . . . .	80
1.39.2. In der Boost-Bibliothek vordefinierte (STL-)Konzepte . . . . .	81
1.40. Ein Blick zurück (2003..2008) — und vorwärts 2017? Usage-Pattern oder Pseudosignatur	
1.41. Creating Own Boost Concept Checking Classes . . . . .	87
1.41.1. Ein Beispiel . . . . .	87
1.41.2. Erstellung eines zugehörigen Archetypes . . . . .	88
1.42. Programmieren mit Konzepten . . . . .	89
<b>2. Metaprogrammierung . . . . .</b>	<b>91</b>
2.1. Metafunktionen . . . . .	91
2.2. Metafunktionen in /usr/include/c++/4.7/type_traits und Feldlängen . . . . .	93
2.3. Factorial, Combinations, IF, id, add und die Rekursion statt der Schleife . . . . .	96
2.4. Rechnende Compiler: . . . . .	98
2.5. Typfunktionen: längerer Datentyp, IfThenElse-Werte . . . . .	99
2.6. Template Nontype Parameter . . . . .	100
2.7. Compilzeit-Fehlermeldungen in constexpr-Metafunktionen . . . . .	101
2.8. Vor- und Nachteile der Metaprogrammierung . . . . .	103

2.9.	C++11 Metaprogramming Examples . . . . .	104
2.10.	Fortgeschrittene Metaprogrammierung . . . . .	109
2.10.1.	Domain specific language extensions: C++11 Compile-time rational arithmetic	109
2.10.2.	Unrolled Loops: Durch Rekursion wegoptimierte Schleifen . . . . .	111
2.10.3.	Expression templates . . . . .	114
2.11.	Nachteile der Metaprogrammierung (Forts.) . . . . .	116
2.12.	Die BOOST Metaprogramming Library MPL . . . . .	116
2.13.	Metaprogramme für die Manipulation von Typen in C++ . . . . .	116
2.14.	Spracherweiterung Maßeinheiten . . . . .	118
2.14.1.	Eine Softwarekatastrophe und ihr Einfluß auf neue Programmiersprachen	118
2.14.2.	DSLs . . . . .	121
2.14.3.	Ausflug in die Domain des technische-wissenschaftlichen Rechnens: Units and Measure in C++	121
2.14.4.	SI-Einheitssystem . . . . .	122
2.14.5.	Boost.Units . . . . .	122
2.14.6.	Erweiterung des C++-Typsystems um Units . . . . .	132
2.14.7.	Nachteile von DSLs . . . . .	135
2.15.	Literaturhinweise zum Metaprogrammieren . . . . .	135
<b>3.</b>	<b>Template template-Parameter, Policy-basiertes Klassendesign</b>	<b>137</b>
3.1.	Templates als Template-Parameter . . . . .	137
3.2.	Policies . . . . .	138
3.3.	Entwurfsmuster Strategie . . . . .	141
3.4.	Policies als Template Template-Parameter . . . . .	142
3.5.	Orthogonale Policy-Dimensionen . . . . .	143
3.6.	Policies (Fortsetzung) . . . . .	143
3.7.	Loki . . . . .	143
3.8.	A Policy-Based flex_string Implementation . . . . .	144
<b>4.</b>	<b>Aspektorientiertes Programmieren in komplexen Unternehmensanwendungen</b>	<b>145</b>
<b>5.</b>	<b>Unternehmensanwendungen und Frameworks</b>	<b>155</b>
5.1.	Frameworks . . . . .	155
5.2.	Testframeworks . . . . .	158
5.2.1.	Testumgebungen in dyn. Programmiersprachen: pydev und „Stand der Technik“-Testumgebung	158
5.2.2.	cxxtest (mit minimalen Systemanforderungen, Ubuntu) . . . . .	158
5.2.3.	Gleitkomma-Maschinengenauigkeit . . . . .	158
5.2.4.	Tests auf annähernde Gleichheit: Boost.Test, ... . . . .	159
5.2.5.	UnitTest++ . . . . .	159
5.2.6.	CppUnit (OpenSUSE) . . . . .	159
5.2.7.	Vergleich UnitTest++ mit CppUnit . . . . .	160
<b>6.</b>	<b>C++-Contract Programming with AOP</b>	<b>161</b>
6.1.	Contract enforcement with AOP . . . . .	162

<b>7. Aspects in AspectJ Cookbook</b>	<b>163</b>
<b>A. Ausblick</b>	<b>165</b>

# Abbildungsverzeichnis

1.1. Die Phasen der Compilation . . . . .	10
---	----





# Tabellenverzeichnis

0.1. C++ Requirements DefaultConstructible, ... . . . . . 5



# Vorbemerkungen:

## Literatur

- B. Stroustrup: Einführung in die Programmierung mit C++, Pearson 2010, München, Kapitel 19.3 ff.
- R. Grimm: C++11, Der Leitfaden zum neuen Standard, Pearson 2011
- D. Vandervoorde, N. M. Josuttis: C++ Templates — The Complete Guide, Pearson 2003, Boston
- Scott Meyers: Effective STL, Addison-Wesley 2001, Indianapolis,
- D. Abrahams, A. Gurtovoy: C++ Template Metaprogramming, Addison Wesley 2005
- Björn Karlsson: Beyond the C++ Standard Library — An Introduction to Boost, Pearson 2006, Boston
- B. Schäling: The Boost C++ Libraries, XML Press 2011
- A. Alexandrescu: Modern C++ Design — Generic Programming and Design Patterns Applied, Pearson 2001, Indianapolis
- Sumant Tambe: [More C++ Idioms](#), WikiBooks 2009

## Einordnung in die Programmierparadigmen

imperativ, funktional, objektbasiert, objektorientiert, Automaten-basiert, Ereignis-getrieben, generisch, Policy-basiert, aspektorientiert, deklarativ (logische Programmierung, regelbasiert), ...

imperativ (strukturiert prozedural): Sequenz von Anweisungen, die den Programmstatus direkt ändern; Variablen, Wertzuweisungen, Iterationen, Schleifen, Fallunterscheidungen, Unterprogramme, Modularisierung, ...

Beispiel: C

**funktional:** Komposition von Komposition von ... von Komposition von nebeneffekt-freien Funktionen; keine Variablen (änderbare Datenfelder) , keine Schleifen, dafür Rekursion, Lambdaausdrücke, ...

Beispiel: (reines) LISP, Haskell, C++ Template-Metaprogrammierung, ...

**objektorientiert:** Recordfelder (Klassenfelder) als lokale Daten (Attribute), die durch im Record definierte Funktionen (Methoden) bearbeitet werden; Vererbung an Unterrecords, Polymorphismus, Überschreiben von Methoden in Unterrecords, ...  
Beispiel: C++, Java, Python, ...

objektbasiert: objektorientiert ohne Vererbung oder Polymorphismus oder mit vielen eingebauten Datenwerten, die keine Objekte sind.  
Beispiel: VisualBasic

Automatenbasiert: (endlicher) Automat mit Status-Enumeration und imperativer ereignisgetriebener Statusübergangsschleife.  
Beispiel: zelluläre Automaten, LEX/YACC, ...

Ereignis-getrieben: asynchrone Hauptschleife mit Ereignis-Handlern, Callback-Funktionen, ...  
Beispiel: qt, GUI-Systeme

**generisch:** (objektorientierte) Programmiermethoden mit Typen als Parametern  
Beispiel: C++ Templates (Schablonen)

**Policy-basiert:** Compiletime-Version des Policy-Designpattern, um verschiedene Implementierungsvarianten eines generischen Konstrukts durch einen generischen Policy-Parameter bei der Instanziierung auswählen zu können.  
Beispiel: Template-Templateparameter mit Policy-Bedeutung

**aspektorientiert:** objektorientierte Programmierung, um generische Funktionalitäten über mehrere Klassen hinweg zu verwenden (Cross-Cutting Concern). Logische Aspekte eines Anwendungsprogramms werden dabei von der eigentlichen Geschäftslogik getrennt. Typische Anwendungsbeispiele sind Transaktionsverwaltung, Auditfähigkeit und Loggingverhalten. (siehe: [http://de.wikipedia.org/wiki/Aspektorientierte\\_Programmierung](http://de.wikipedia.org/wiki/Aspektorientierte_Programmierung))  
Beispiel: AspectJ, AspectC++

deklarativ, logische Programmierung, regelbasiert: Bearbeitungsregeln, die das was, nicht das wie der Bearbeitungsschritte definieren; Laufzeitsystem bearbeitet Eingaben gemäß des Regelsatzes durch automatische Inferenzmaschine.  
Beispiel: Prolog, SQL, reguläre Ausdrücke

## Die Entwicklung der Aussagekraft der formalen generischen Parameternamen

- von einfallslosen Parameternamen wie `class T1`, `class T2`, ...  
vergleiche <http://www.cplusplus.com/doc/tutorial/templates/>
- über semantisch inhaltvolle Parameternamen wie `typename InputIterator1`, `typename InputIterator2`, `typename NumericT`, ...  
vergleiche <http://www.iue.tuwien.ac.at/phd/heinzl/node32.html#SECTION01022300000000000000>.

## Ein Typsystem für generische Parameter

- hin zur Nennung der Requirements an die zur Instantiierung benutzbaren aktuellen Parameter wie `T` shall meet the requirements of `CopyConstructible` and `CopyAssignable` types  
(Seite 972 von <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf>) mit der Erläuterung:

Table 33 — `DefaultConstructible` requirements [`defaultconstructible`]

Expression	Post-condition
<code>T t;</code>	object <code>t</code> is default-initialized
<code>T u{}</code> ;	object <code>u</code> is value-initialized
<code>T()</code>	a temporary object of type <code>T</code> is value-initialized
<code>T{}</code>	

Table 34 — `MoveConstructible` requirements [`moveconstructible`]

Expression	Post-condition
<code>T u(rv);</code>	<code>u</code> is equivalent to the value of <code>rv</code> before the construction
<code>T(rv)</code>	<code>T(rv)</code> is equivalent to the value of <code>rv</code> before the construction
[ <i>Note</i> : <code>rv</code> remains a valid object. Its state is unspecified. — <i>end note</i> ]	

Table 35 — `CopyConstructible` requirements (in addition to `MoveConstructible`) [`copyconstructible`]

Expression	Post-condition
<code>T u(v);</code>	the value of <code>v</code> is unchanged and is equivalent to <code>u</code>
<code>T(v)</code>	the value of <code>v</code> is unchanged and is equivalent to <code>T(v)</code>

Table 36 — `MoveAssignable` requirements [`moveassignable`]

Expression	Return type	Return value	Post-condition
<code>t = rv</code>	<code>T&amp;</code>	<code>t</code>	<code>t</code> is equivalent to the value of <code>rv</code> before the assignment
[ <i>Note</i> : <code>rv</code> remains a valid object. Its state is unspecified. — <i>end note</i> ]			

Tabelle 0.1.: C++ Requirements `DefaultConstructible`, ...

(Seite 431f. des Drafts)

<http://www.heise.de/newsticker/meldung/C-11-einstimmig-als-Standard-angenommen-1322726.html>

C++11 ohne „Concepts“

<http://en.wikipedia.org/wiki/C%2B%2B11>

... aber dokumentatorisch z.B. im STL-Manual benutzt:

`accumulate`

Was hätten Concepts gebracht?(automatische überprüfte Requirements der aktuellen generischen Parameter)

[http://en.wikipedia.org/wiki/Concepts\\_\(C%2B%2B\)](http://en.wikipedia.org/wiki/Concepts_(C%2B%2B))

## TR1 - ein „Zwischenstandard“ für die C++-Standardbibliothek

TR1

[http://en.wikipedia.org/wiki/Technical\\_Report\\_2#Mathematical\\_special\\_functions](http://en.wikipedia.org/wiki/Technical_Report_2#Mathematical_special_functions)

## TR2 call for proposals

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1810.html>

## Ziele des Draft-Designs C++0x (C++11)

<http://www.artima.com/cppsource/cpp0x.html>

## Welche Eigenschaften von C++11 beherrscht g++ schon?

C++0x/C++11 Support in GCC

## Typsicherheit ... bei generischen Konstrukten

<http://de.wikipedia.org/wiki/Typsicherheit>

[http://en.wikipedia.org/wiki/Type\\_safety](http://en.wikipedia.org/wiki/Type_safety)

Typsichere generische Typen in C++

Irreführende Monster-Fehlermeldungen bei Instanziierung generischer C++-Konstrukte

Hinweise zur Fehlermeldungsqualität der C++ Template-Programmierung

## „Generic Programming“-Kochbuch

Glossar des generischen Programmierens

<http://www.generic-programming.org/>

<http://www.generic-programming.org/about/intro/>

Verallgemeinerung/Lifting konkreter Implementierungen ähnlicher Algorithmen

## neue C++11 Standard Library

C++ standard library changes

C++ Reference

Standard Template Library Manual

Generic Programming: Libraries

## Metaprogramming

Metaprogramming

*Factorial < 4 >:: value*

# 1. Generische Programmierung

## 1.1. Was ist generische Programmierung?

<http://pdfcast.org/pdf/the-java-generic-programming-system>

[http://www.boost.org/community/generic\\_programming.html](http://www.boost.org/community/generic_programming.html)

[http://en.wikipedia.org/wiki/Generic\\_programming](http://en.wikipedia.org/wiki/Generic_programming)

## 1.2. Beispiel einer generischen Funktion mit einem generischen Parameter

```
#include <iostream>
template <typename T>
/*
 * Requirements: T muss einen Kopierkonstruktor haben,
 *               T muss einen Zuweisungsoperator zu T haben.
 */
void swap(T& a, T& b)
{
    T  old_a(a);

    a = b;
    b = old_a;
}
int main() {
    int k(1);
    int l(5);
    std::cout << k << " " << l << std::endl;
    swap(k, l);
    std::cout << k << " " << l << std::endl;
    double d1(3.1415);
    double d2(15.1055);
    std::cout << d1 << " " << d2 << std::endl;
    swap(d1, d2);
    std::cout << d1 << " " << d2 << std::endl;
}
```

## 1.3. Benutzung von `std::swap()`

```
#include <iostream>
using std::cout;

int main() {

    int i = 5;
    int j = 6;
    cout << "i = " << i << "; j = " << j << std::endl;
    std::swap(i, j);
    cout << "i = " << i << "; j = " << j << std::endl;

    int a[2] = {2, 3};
    int b[2] = {12, 13};
    cout << a[0] << " " << a[1] << std::endl;
    cout << b[0] << " " << b[1] << std::endl;
    std::swap(a, b);
    cout << a[0] << " " << a[1] << std::endl;
    cout << b[0] << " " << b[1] << std::endl;
}
```

unter Benutzung der „general utilities library“ (Kapitel 20 des C++-Drafts):

### 20.2.2 swap

[utility.swap]

```
template<class T> void swap(T& a, T& b) noexcept(see below);
```

- 1 *Remark:* The expression inside `noexcept` is equivalent to:  
`is_nothrow_move_constructible<T>::value &&`  
`is_nothrow_move_assignable<T>::value`
- 2 *Requires:* Type `T` shall be `MoveConstructible` (Table 20) and `MoveAssignable` (Table 22).
- 3 *Effects:* Exchanges values stored in two locations.

Zu den Requirements an den generischen Parameter siehe [utility.arg.requirements]:



Table 20 — MoveConstructible requirements [moveconstructible]

Expression	Post-condition
<code>T u = rv;</code>	<code>u</code> is equivalent to the value of <code>rv</code> before the construction
<code>T(rv)</code>	<code>T(rv)</code> is equivalent to the value of <code>rv</code> before the construction
<code>rv</code> 's state is unspecified. [Note: <code>rv</code> must still meet the requirements of the library component that is using it. The operations listed in those requirements must work as specified whether <code>rv</code> has been moved from or not. — end note]	

Table 21 — CopyConstructible requirements (in addition to MoveConstructible) [copyconstructible]

Expression	Post-condition
<code>T u = v;</code>	the value of <code>v</code> is unchanged and is equivalent to <code>u</code>
<code>T(v)</code>	the value of <code>v</code> is unchanged and is equivalent to <code>T(v)</code>

Table 22 — MoveAssignable requirements [moveassignable]

Expression	Return type	Return value	Post-condition
<code>t = rv</code>	<code>T&amp;</code>	<code>t</code>	<code>t</code> is equivalent to the value of <code>rv</code> before the assignment
<code>rv</code> 's state is unspecified. [Note: <code>rv</code> must still meet the requirements of the library component that is using it. The operations listed in those requirements must work as specified whether <code>rv</code> has been moved from or not. — end note]			

Table 23 — CopyAssignable requirements (in addition to MoveAssignable) [copyassignable]

Expression	Return type	Return value	Post-condition
<code>t = v</code>	<code>T&amp;</code>	<code>t</code>	<code>t</code> is equivalent to <code>v</code> , the value of <code>v</code> is unchanged

Die swappable.requirements findet man in Abschnitt 20.2.2 des Drafts. Fassen Sie sie in eigenen Worten zusammen.

SGI-Manual: swap-Parameter Assignable  
 automatically generated C++98-Class-members

## 1.4. Einsatzgebiete und Beispielrepositorien für generische Konstrukte: die STL, ...

<http://www.sgi.com/tech/stl/>  
 generische Java-Datentypen  
 Die Boost C++-Bibliotheken

## 1.5. Instanzen generischer Objekte

### 1.5.1. Objekt-Dateien: wo sind welche Instanzen meiner generischen Objekte (nm und c++filt)?

What goes into an object file?

[http://en.wikipedia.org/wiki/Executable\\_and\\_Linkable\\_Format](http://en.wikipedia.org/wiki/Executable_and_Linkable_Format)

<http://www.ibm.com/developerworks/aix/library/au-unixtools/index.html>

GCC and Make: 1.4 GCC Compilation Process:

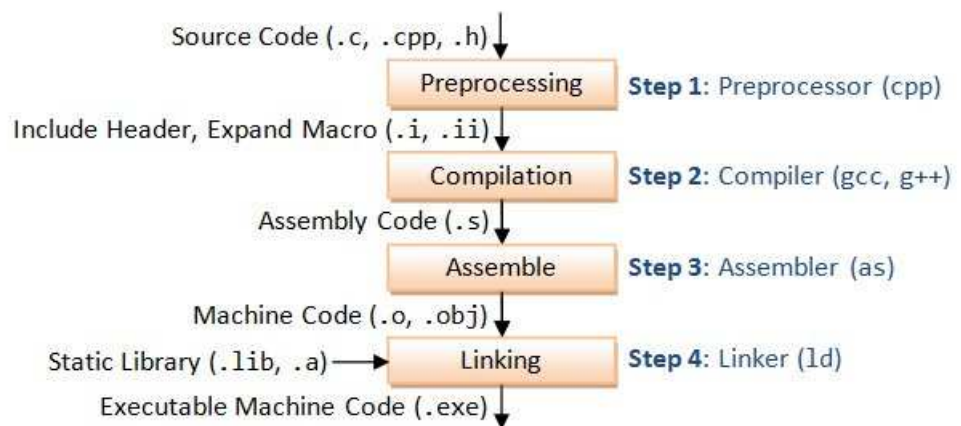


Abbildung 1.1.: Die Phasen der Compilation

The C++ compilation process

Template Compilation Models  
extern template in C++11

ldd und was es zeigt:

```
> ls  
swap1.cpp  
> cat swap1.cpp
```

```
#include <iostream>  
template <typename T>  
/*  
 * Requirements: T muss einen Kopierkonstruktor haben,
```

```

*           T muss einen Zuweisungsoperator zu T haben.
*/
void swap(T& a, T& b)
{
    T  old_a(a);

    a = b;
    b = old_a;
}
int main(){
...
    int k(1);
    int l(5);
    swap(k,l);
...
}

```

```

> make swap1
g++ swap1.cpp -o swap1

```

```

> ldd ./swap1
linux-vdso.so.1 => (0x00007fffe1b0d000)
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007fa6005e9000)
libm.so.6 => /lib64/libm.so.6 (0x00007fa600392000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007fa60017c000)
libc.so.6 => /lib64/libc.so.6 (0x00007fa5ffe1c000)
/lib64/ld-linux-x86-64.so.2 (0x00007fa6008f3000)

```

Beim Programmmlauf werden nacheinander die „shared object“-Bibliotheken geöffnet und nötige Teile in das auszuführende Binary eingebunden:

```

> strace ./swap1
execve("./swap1", ["/swap1"], [/* 66 vars */]) = 0
brk(0) = 0x602000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e3146000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=335735, ...}) = 0
mmap(NULL, 335735, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f17e30f4000
close(3) = 0
open("/usr/lib64/libstdc++.so.6", O_RDONLY) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\305\5\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1003544, ...}) = 0
mmap(NULL, 3182936, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f17e2c1f000
fadvise64(3, 0, 3182936, POSIX_FADV_WILLNEED) = 0
mprotect(0x7f17e2d0b000, 2093056, PROT_NONE) = 0
mmap(0x7f17e2f0a000, 40960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xeb000) = 0x7f17e2f0a000
mmap(0x7f17e2f14000, 82264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f17e2f14000
close(3) = 0
open("/lib64/libm.so.6", O_RDONLY) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0'\>\0\0\0\0\0"... , 832) = 832

```

```

fstat(3, {st_mode=S_IFREG|0755, st_size=391908, ...}) = 0
mmap(NULL, 2449592, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f17e29c8000
fadvise64(3, 0, 2449592, POSIX_FADV_WILLNEED) = 0
mprotect(0x7f17e2a1e000, 2093056, PROT_NONE) = 0
mmap(0x7f17e2c1d000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x55000) = 0x7f17e2c1d000
close(3) = 0
open("/lib64/libgcc_s.so.1", O_RDONLY) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0'-\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=88544, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e30f3000
mmap(NULL, 2184184, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f17e27b2000
fadvise64(3, 0, 2184184, POSIX_FADV_WILLNEED) = 0
mprotect(0x7f17e27c7000, 2093056, PROT_NONE) = 0
mmap(0x7f17e29c6000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x14000) = 0x7f17e29c6000
close(3) = 0
open("/lib64/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\354\1\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1670469, ...}) = 0
mmap(NULL, 3537800, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f17e2452000
fadvise64(3, 0, 3537800, POSIX_FADV_WILLNEED) = 0
mprotect(0x7f17e25a8000, 2097152, PROT_NONE) = 0
mmap(0x7f17e27a8000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x156000) = 0x7f17e27a8000
mmap(0x7f17e27ad000, 19336, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f17e27ad000
close(3) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e30f2000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e30f0000
arch_prctl(ARCH_SET_FS, 0x7f17e30f0720) = 0
mprotect(0x7f17e27a8000, 16384, PROT_READ) = 0
mprotect(0x7f17e29c6000, 4096, PROT_READ) = 0
mprotect(0x7f17e2c1d000, 4096, PROT_READ) = 0
mprotect(0x7f17e2f0a000, 32768, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ) = 0
mprotect(0x7f17e3147000, 4096, PROT_READ) = 0
munmap(0x7f17e30f4000, 335735) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e3145000
write(1, "1 5\n", 4) = 4
write(1, "5 1\n", 4) = 4
write(1, "3.1415 15.1055\n", 15) = 15
write(1, "15.1055 3.1415\n", 15) = 15
exit_group(0) = ?

```

Neben gcc, g++, as, ld, gprof und gdb/ddd sind die folgenden Tools von Interesse.

### Die GNU-Binutils:

- nm
- objdump
- objcopy
- readelf
- strip
- size

- `c++filt`
- `ar`
- `ranlib`

```

> file ./swap1
./swap1: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped
> nm ./swap1
0000000000600e20 d _DYNAMIC
0000000000600fe8 d _GLOBAL_OFFSET_TABLE_
0000000000400a75 t _GLOBAL__I_main
0000000000400bd8 R _IO_stdin_used
w _Jv_RegisterClasses
0000000000400a35 t _Z41__static_initialization_and_destruction_0ii
0000000000400ab6 W _Z4swapIdEvRT_S1_
0000000000400a8a W _Z4swapIiEvRT_S1_
U _ZNSolsEPFRSoS_E@@GLIBCXX_3.4
U _ZNSolsEd@@GLIBCXX_3.4
U _ZNSolsEi@@GLIBCXX_3.4
U _ZNSt8ios_base4InitC1Ev@@GLIBCXX_3.4
U _ZNSt8ios_base4InitD1Ev@@GLIBCXX_3.4
0000000000601060 B _ZSt4cout@@GLIBCXX_3.4
U _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_@@GLIBCXX_3.4
0000000000601180 b _ZStL8__ioinit
U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@@GLIBCXX_3.4
0000000000600e00 d __CTOR_END__
0000000000600df0 d __CTOR_LIST__
0000000000600e10 D __DTOR_END__
0000000000600e08 d __DTOR_LIST__
0000000000400d40 r __FRAME_END__
0000000000600e18 d __JCR_END__
0000000000600e18 d __JCR_LIST__
0000000000601060 A __bss_start
U __cxa_atexit@@GLIBC_2.2.5
0000000000601050 D __data_start
0000000000400b90 t __do_global_ctors_aux
0000000000400850 t __do_global_dtors_aux
0000000000601058 D __dso_handle
w __gmon_start__
U __gxx_personality_v0@@CXXABI_1.3
0000000000600dec d __init_array_end
0000000000600dec d __init_array_start
0000000000400af0 T __libc_csu_fini

```

```

0000000000400b00 T __libc_csu_init
                    U __libc_start_main@@GLIBC_2.2.5
0000000000601060 A _edata
0000000000601188 A _end
0000000000400bc8 T _fini
0000000000400730 T _init
0000000000400800 T _start
000000000040082c t call_gmon_start
0000000000601170 b completed.7424
0000000000601050 W data_start
0000000000601178 b dtor_idx.7426
00000000004008c0 t frame_dummy
00000000004008e4 T main

```

... und mit demangled Symbolen:

```

nm ./swap1 | c++filt
0000000000600e20 d _DYNAMIC
0000000000600fe8 d _GLOBAL_OFFSET_TABLE_
0000000000400a75 t global constructors keyed to main
0000000000400bd8 R _IO_stdin_used
                    w _Jv_RegisterClasses
0000000000400a35 t __static_initialization_and_destruction_0(int, int)
0000000000400ab6 W void swap<double>(double&, double&)
0000000000400a8a W void swap<int>(int&, int&)
                    U std::basic_ostream<char, std::char_traits<char> >::operator<<(
                    U std::basic_ostream<char, std::char_traits<char> >::operator<<(
                    U std::basic_ostream<char, std::char_traits<char> >::operator<<(
                    U std::ios_base::Init::Init()@@GLIBCXX_3.4
                    U std::ios_base::Init::~Init()@@GLIBCXX_3.4
0000000000601060 B std::cout@@GLIBCXX_3.4
                    U std::basic_ostream<char, std::char_traits<char> >& std::endl<
0000000000601180 b std::__ioinit
                    U std::basic_ostream<char, std::char_traits<char> >& std::operat
0000000000600e00 d __CTOR_END__
0000000000600df0 d __CTOR_LIST__
0000000000600e10 D __DTOR_END__
0000000000600e08 d __DTOR_LIST__
0000000000400d40 r __FRAME_END__
0000000000600e18 d __JCR_END__
0000000000600e18 d __JCR_LIST__
0000000000601060 A __bss_start
                    U __cxa_atexit@@GLIBC_2.2.5
0000000000601050 D __data_start

```

```
0000000000400b90 t __do_global_ctors_aux
0000000000400850 t __do_global_dtors_aux
0000000000601058 D __dso_handle
      w __gmon_start__
      U __gxx_personality_v0@@CXXABI_1.3
0000000000600dec d __init_array_end
0000000000600dec d __init_array_start
0000000000400af0 T __libc_csu_fini
0000000000400b00 T __libc_csu_init
      U __libc_start_main@@GLIBC_2.2.5
0000000000601060 A _edata
0000000000601188 A _end
0000000000400bc8 T _fini
0000000000400730 T _init
0000000000400800 T _start
000000000040082c t call_gmon_start
0000000000601170 b completed.7424
0000000000601050 W data_start
0000000000601178 b dtor_idx.7426
00000000004008c0 t frame_dummy
00000000004008e4 T main
```

```
> nm ./swap1 | c++filt -n _Z4swapIiEvRT_S1_
void swap<int>(int&, int&)
```

Vergleiche:

[C++ name mangling](#)

[name mangling in Java](#)

[Getting the best from g++](#)

```
> objdump -x swap1 | c++filt
```

```
swap1:      file format elf64-x86-64
swap1
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000004007e0

Program Header:
  PHDR off   0x0000000000000040 vaddr 0x000000000400040 paddr 0x000000000400040 align 2**3
        filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off 0x0000000000000238 vaddr 0x000000000400238 paddr 0x000000000400238 align 2**0
        filesz 0x000000000000001c memsz 0x000000000000001c flags r--
...
Dynamic Section:
NEEDED          libstdc++.so.6
NEEDED          libm.so.6
NEEDED          libgcc_s.so.1
NEEDED          libc.so.6
INIT            0x000000000400720
FINI            0x000000000400ba8
...
Version References:
required from libc.so.6:
  0x09691a75 0x00 03 GLIBC_2.2.5
required from libstdc++.so.6:
  0x08922974 0x00 02 GLIBCXX_3.4
Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .interp        0000001c 0000000000400238 0000000000400238 00000238 2**0
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag  00000020 0000000000400254 0000000000400254 00000254 2**2
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.SuSE     00000018 0000000000400274 0000000000400274 00000274 2**2
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .note.gnu.build-id 00000024 000000000040028c 000000000040028c 0000028c 2**2
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .hash          00000048 00000000004002b0 00000000004002b0 000002b0 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .gnu.hash      00000030 00000000004002f8 00000000004002f8 000002f8 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .dynsym        00000138 0000000000400328 0000000000400328 00000328 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .dynstr        0000015e 0000000000400460 0000000000400460 00000460 2**0
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .gnu.version  0000001a 00000000004005be 00000000004005be 000005be 2**1
                CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .gnu.version_r 00000040 00000000004005d8 00000000004005d8 000005d8 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
10 .rela.dyn      00000030 0000000000400618 0000000000400618 00000618 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
11 .rela.plt     000000d8 0000000000400648 0000000000400648 00000648 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
12 .init         00000018 0000000000400720 0000000000400720 00000720 2**2
                CONTENTS, ALLOC, LOAD, READONLY, CODE
```



```

13 .plt      000000a0 0000000000400738 0000000000400738 00000738 2**2
CONTENTS, ALLOC, LOAD, READONLY, CODE
14 .text    000003c8 00000000004007e0 00000000004007e0 000007e0 2**4
CONTENTS, ALLOC, LOAD, READONLY, CODE
15 .fini    0000000e 0000000000400ba8 0000000000400ba8 00000ba8 2**2
CONTENTS, ALLOC, LOAD, READONLY, CODE
16 .rodata  00000006 0000000000400bb8 0000000000400bb8 00000bb8 2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA
17 .eh_frame_hdr 00000044 0000000000400bc0 0000000000400bc0 00000bc0 2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA
18 .eh_frame 00000104 0000000000400c08 0000000000400c08 00000c08 2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA
19 .ctors   00000018 0000000000600de0 0000000000600de0 00000de0 2**3
CONTENTS, ALLOC, LOAD, DATA
20 .dtors   00000010 0000000000600df8 0000000000600df8 00000df8 2**3
CONTENTS, ALLOC, LOAD, DATA
...
25 .data    00000010 0000000000601048 0000000000601048 00001048 2**3
CONTENTS, ALLOC, LOAD, DATA
26 .bss     00000128 0000000000601060 0000000000601060 00001058 2**5
ALLOC
...

```

SYMBOL TABLE:

```

0000000000400238 l d .interp      0000000000000000 .interp
0000000000400254 l d .note.ABI-tag 0000000000000000 .note.ABI-tag
0000000000400274 l d .note.SuSE    0000000000000000 .note.SuSE
000000000040028c l d .note.gnu.build-id 0000000000000000 .note.gnu.build-id
00000000004002b0 l d .hash         0000000000000000 .hash
00000000004002f8 l d .gnu.hash     0000000000000000 .gnu.hash
0000000000400328 l d .dynsym      0000000000000000 .dynsym
0000000000400460 l d .dynstr      0000000000000000 .dynstr
00000000004005be l d .gnu.version 0000000000000000 .gnu.version
00000000004005d8 l d .gnu.version_r 0000000000000000 .gnu.version_r
0000000000400618 l d .rela.dyn    0000000000000000 .rela.dyn
0000000000400648 l d .rela.plt    0000000000000000 .rela.plt
0000000000400720 l d .init        0000000000000000 .init
0000000000400738 l d .plt         0000000000000000 .plt
00000000004007e0 l d .text       0000000000000000 .text
0000000000400ba8 l d .fini       0000000000000000 .fini
0000000000400bb8 l d .rodata     0000000000000000 .rodata
...
0000000000400a96 w F .text        0000000000000032 void swap<double>(double&, double&)
...
0000000000400a6a w F .text        000000000000002c void swap<int>(int&, int&)
0000000000601058 g *ABS*         0000000000000000 _edata
00000000004008c4 g F .text        0000000000000151 main
0000000000400720 g F .init        0000000000000000 _init

```

> objdump -t swap1 | c++filt

swap1: file format elf64-x86-64

SYMBOL TABLE:

```

0000000000400238 l d .interp      0000000000000000 .interp
0000000000400254 l d .note.ABI-tag 0000000000000000 .note.ABI-tag
0000000000400274 l d .note.SuSE    0000000000000000 .note.SuSE
000000000040028c l d .note.gnu.build-id 0000000000000000 .note.gnu.build-id
00000000004002b0 l d .hash         0000000000000000 .hash
00000000004002f8 l d .gnu.hash     0000000000000000 .gnu.hash
0000000000400328 l d .dynsym      0000000000000000 .dynsym
0000000000400460 l d .dynstr      0000000000000000 .dynstr
00000000004005be l d .gnu.version 0000000000000000 .gnu.version
00000000004005d8 l d .gnu.version_r 0000000000000000 .gnu.version_r
0000000000400618 l d .rela.dyn    0000000000000000 .rela.dyn
0000000000400648 l d .rela.plt    0000000000000000 .rela.plt
0000000000400720 l d .init        0000000000000000 .init
0000000000400738 l d .plt         0000000000000000 .plt
00000000004007e0 l d .text       0000000000000000 .text

```

```

0000000000400ba8 1 d .fini 0000000000000000 .fini
0000000000400bb8 1 d .rodata 0000000000000000 .rodata
0000000000400bc0 1 d .eh_frame_hdr 0000000000000000 .eh_frame_hdr
0000000000400c08 1 d .eh_frame 0000000000000000 .eh_frame
0000000000600de0 1 d .ctors 0000000000000000 .ctors
0000000000600df8 1 d .dtors 0000000000000000 .dtors
0000000000600e08 1 d .jcr 0000000000000000 .jcr
0000000000600e10 1 d .dynamic 0000000000000000 .dynamic
0000000000600fe0 1 d .got 0000000000000000 .got
0000000000600fe8 1 d .got.plt 0000000000000000 .got.plt
0000000000601048 1 d .data 0000000000000000 .data
0000000000601060 1 d .bss 0000000000000000 .bss
0000000000000000 1 d .comment.SUSE.OPTs 0000000000000000 .comment.SUSE.OPTs
0000000000000000 1 d .comment 0000000000000000 .comment
0000000000000000 1 d .debug_aranges 0000000000000000 .debug_aranges
0000000000000000 1 d .debug_pubnames 0000000000000000 .debug_pubnames
0000000000000000 1 d .debug_info 0000000000000000 .debug_info
0000000000000000 1 d .debug_abbrev 0000000000000000 .debug_abbrev
0000000000000000 1 d .debug_line 0000000000000000 .debug_line
0000000000000000 1 d .debug_str 0000000000000000 .debug_str
0000000000000000 1 d .debug_loc 0000000000000000 .debug_loc
0000000000000000 1 d .debug_pubtypes 0000000000000000 .debug_pubtypes
0000000000000000 1 d .debug_ranges 0000000000000000 .debug_ranges
0000000000000000 1 df *ABS* 0000000000000000 init.c
0000000000000000 1 df *ABS* 0000000000000000 initfini.c
000000000040080c 1 F .text 0000000000000000 call_gmon_start
0000000000000000 1 df *ABS* 0000000000000000 crtstuff.c
0000000000600de0 1 O .ctors 0000000000000000 __CTOR_LIST__
0000000000600df8 1 O .dtors 0000000000000000 __DTOR_LIST__
0000000000600e08 1 O .jcr 0000000000000000 __JCR_LIST__
0000000000400830 1 F .text 0000000000000000 __do_global_dtors_aux
0000000000601170 1 O .bss 0000000000000001 completed.5939
0000000000601178 1 O .bss 0000000000000008 dtor_idx.5941
00000000004008a0 1 F .text 0000000000000000 frame_dummy
0000000000000000 1 df *ABS* 0000000000000000 crtstuff.c
0000000000600df0 1 O .ctors 0000000000000000 __CTOR_END__
0000000000400d08 1 O .eh_frame 0000000000000000 __FRAME_END__
0000000000600e08 1 O .jcr 0000000000000000 __JCR_END__
0000000000400b70 1 F .text 0000000000000000 __do_global_ctors_aux
0000000000000000 1 df *ABS* 0000000000000000 initfini.c
0000000000000000 1 df *ABS* 0000000000000000 swapl.cpp
0000000000601180 1 O .bss 0000000000000001 std::_ioinit
0000000000400a15 1 F .text 0000000000000040 __static_initialization_and_destruction_0(int, int)
0000000000400a55 1 F .text 0000000000000015 global constructors keyed to main
0000000000000000 1 df *ABS* 0000000000000000 elf-init.c
0000000000600fe8 1 O .got.plt 0000000000000000 .hidden _GLOBAL_OFFSET_TABLE_
0000000000600ddc 1 .ctors 0000000000000000 .hidden __init_array_end
0000000000600ddc 1 .ctors 0000000000000000 .hidden __init_array_start
0000000000600e10 1 O .dynamic 0000000000000000 .hidden _DYNAMIC
0000000000601048 w .data 0000000000000000 data_start
0000000000000000 F *UND* 0000000000000000 std::basic_ostream<char, std::char_traits<char> >::oper
0000000000000000 F *UND* 0000000000000000 std::basic_ostream<char, std::char_traits<char> >::oper
0000000000400b60 g F .text 0000000000000002 __libc_csu_fini
00000000004007e0 g F .text 0000000000000000 _start
0000000000000000 w *UND* 0000000000000000 __gmon_start__
0000000000000000 w *UND* 0000000000000000 _Jv_RegisterClasses
0000000000400ba8 g F .fini 0000000000000000 _fini
0000000000000000 F *UND* 0000000000000000 std::ios_base::Init::Init()@GLIBCXX_3.4
0000000000000000 F *UND* 0000000000000000 __libc_start_main@GLIBC_2.2.5
0000000000000000 F *UND* 0000000000000000 __cxa_atexit@GLIBC_2.2.5
0000000000400798 F *UND* 0000000000000000 std::ios_base::Init::~Init()@GLIBCXX_3.4
0000000000000000 F *UND* 0000000000000000 std::basic_ostream<char, std::char_traits<char> >& std:
0000000000400bb8 g O .rodata 0000000000000004 _IO_stdin_used
0000000000601048 g .data 0000000000000000 __data_start
0000000000400a96 w F .text 0000000000000032 void swap<double>(double&, double&)
0000000000601060 g O .bss 00000000000000110 std::cout@GLIBCXX_3.4

```

0000000000601050	g	0	.data	0000000000000000	.hidden __dso_handle
0000000000600e00	g	0	.dtors	0000000000000000	.hidden __DTOR_END__
0000000000400ad0	g	F	.text	0000000000000089	__libc_csu_init
0000000000601058	g		*ABS*	0000000000000000	__bss_start
0000000000601188	g		*ABS*	0000000000000000	_end
0000000000000000		F	*UND*	0000000000000000	std::basic_ostream<char, std::char_traits<char> >::operator<<(std::
00000000004007c8		F	*UND*	0000000000000000	std::basic_ostream<char, std::char_traits<char> >& std::endl<char,
0000000000400a6a	w	F	.text	000000000000002c	void swap<int>(int&, int&)
0000000000601058	g		*ABS*	0000000000000000	_edata
00000000004008c4	g	F	.text	0000000000000151	main
0000000000400720	g	F	.init	0000000000000000	_init

> readelf -s swap1 | c++filt

Symbol table '.dynsym' contains 13 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	std::basic_ostream<char, std::char_traits<char> >::operator<<(double)@GLIBCXX_3.4 (2)
2:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	std::basic_ostream<char, std::char_traits<char> >::operator<<(int)@GLIBCXX_3.4 (2)
3:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__
4:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	_Jv_RegisterClasses
5:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	std::ios_base::Init::Init()@GLIBCXX_3.4 (2)
6:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_2.2.5 (3)
7:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__cxa_atexit@GLIBC_2.2.5 (3)
8:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	_ZStlsIStI1char_traitsIcE@GLIBCXX_3.4 (2)
9:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char, std::char_
10:	0000000004007c8	0	FUNC	GLOBAL	DEFAULT	UND	_ZSt4endlIcSt11char_trait@GLIBCXX_3.4 (2)
11:	000000000400798	0	FUNC	GLOBAL	DEFAULT	UND	std::ios_base::Init::~Init()@GLIBCXX_3.4 (2)
12:	000000000601060	272	OBJECT	GLOBAL	DEFAULT	27	std::cout@GLIBCXX_3.4 (2)

Symbol table '.symtab' contains 93 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	000000000400238	0	SECTION	LOCAL	DEFAULT	1	
2:	000000000400254	0	SECTION	LOCAL	DEFAULT	2	
3:	000000000400274	0	SECTION	LOCAL	DEFAULT	3	
4:	00000000040028c	0	SECTION	LOCAL	DEFAULT	4	
5:	0000000004002b0	0	SECTION	LOCAL	DEFAULT	5	
6:	0000000004002f8	0	SECTION	LOCAL	DEFAULT	6	
7:	000000000400328	0	SECTION	LOCAL	DEFAULT	7	
8:	000000000400460	0	SECTION	LOCAL	DEFAULT	8	
9:	0000000004005be	0	SECTION	LOCAL	DEFAULT	9	
10:	0000000004005d8	0	SECTION	LOCAL	DEFAULT	10	
11:	000000000400618	0	SECTION	LOCAL	DEFAULT	11	
12:	000000000400648	0	SECTION	LOCAL	DEFAULT	12	
13:	000000000400720	0	SECTION	LOCAL	DEFAULT	13	
14:	000000000400738	0	SECTION	LOCAL	DEFAULT	14	
15:	0000000004007e0	0	SECTION	LOCAL	DEFAULT	15	
16:	000000000400ba8	0	SECTION	LOCAL	DEFAULT	16	
17:	000000000400bb8	0	SECTION	LOCAL	DEFAULT	17	
18:	000000000400bc0	0	SECTION	LOCAL	DEFAULT	18	
19:	000000000400c08	0	SECTION	LOCAL	DEFAULT	19	
20:	000000000600de0	0	SECTION	LOCAL	DEFAULT	20	
21:	000000000600df8	0	SECTION	LOCAL	DEFAULT	21	
22:	000000000600e08	0	SECTION	LOCAL	DEFAULT	22	
23:	000000000600e10	0	SECTION	LOCAL	DEFAULT	23	
24:	000000000600fe0	0	SECTION	LOCAL	DEFAULT	24	
25:	000000000600fe8	0	SECTION	LOCAL	DEFAULT	25	
26:	000000000601048	0	SECTION	LOCAL	DEFAULT	26	
27:	000000000601060	0	SECTION	LOCAL	DEFAULT	27	
28:	0000000000000000	0	SECTION	LOCAL	DEFAULT	28	
29:	0000000000000000	0	SECTION	LOCAL	DEFAULT	29	
30:	0000000000000000	0	SECTION	LOCAL	DEFAULT	30	
31:	0000000000000000	0	SECTION	LOCAL	DEFAULT	31	
32:	0000000000000000	0	SECTION	LOCAL	DEFAULT	32	
33:	0000000000000000	0	SECTION	LOCAL	DEFAULT	33	
34:	0000000000000000	0	SECTION	LOCAL	DEFAULT	34	
35:	0000000000000000	0	SECTION	LOCAL	DEFAULT	35	
36:	0000000000000000	0	SECTION	LOCAL	DEFAULT	36	
37:	0000000000000000	0	SECTION	LOCAL	DEFAULT	37	
38:	0000000000000000	0	SECTION	LOCAL	DEFAULT	38	
39:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	init.c
40:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	initfini.c
41:	00000000040080c	0	FUNC	LOCAL	DEFAULT	15	call_gmon_start
42:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	crtstuff.c
43:	000000000600de0	0	OBJECT	LOCAL	DEFAULT	20	__CTOR_LIST__
44:	000000000600df8	0	OBJECT	LOCAL	DEFAULT	21	__DTOR_LIST__
45:	000000000600e08	0	OBJECT	LOCAL	DEFAULT	22	__JCR_LIST__
46:	000000000400830	0	FUNC	LOCAL	DEFAULT	15	__do_global_dtors_aux
47:	000000000601170	1	OBJECT	LOCAL	DEFAULT	27	completed.5939
48:	000000000601178	8	OBJECT	LOCAL	DEFAULT	27	dtor_idx.5941
49:	0000000004008a0	0	FUNC	LOCAL	DEFAULT	15	frame_dummy
50:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	crtstuff.c
51:	000000000600df0	0	OBJECT	LOCAL	DEFAULT	20	__CTOR_END__
52:	000000000400d08	0	OBJECT	LOCAL	DEFAULT	19	__FRAME_END__
53:	000000000600e08	0	OBJECT	LOCAL	DEFAULT	22	__JCR_END__
54:	000000000400b70	0	FUNC	LOCAL	DEFAULT	15	__do_global_ctors_aux
55:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	initfini.c
56:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	swap1.cpp
57:	000000000601180	1	OBJECT	LOCAL	DEFAULT	27	std::_ioinit
58:	000000000400a15	64	FUNC	LOCAL	DEFAULT	15	_Z41__static_initializati
59:	000000000400a55	21	FUNC	LOCAL	DEFAULT	15	global constructors keyed to main
60:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	elf-init.c
61:	000000000600fe8	0	OBJECT	LOCAL	HIDDEN	25	_GLOBAL_OFFSET_TABLE_
62:	000000000600ddc	0	NOTYPE	LOCAL	HIDDEN	20	__init_array_end
63:	000000000600ddc	0	NOTYPE	LOCAL	HIDDEN	20	__init_array_start
64:	000000000600e10	0	OBJECT	LOCAL	HIDDEN	23	_DYNAMIC
65:	000000000601048	0	NOTYPE	WEAK	DEFAULT	26	data_start
66:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	std::basic_ostream<char, std::char_traits<char> >::operator<<(double)@GLIBCXX_3.4
67:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	std::basic_ostream<char, std::char_traits<char> >::operator<<(int)@GLIBCXX_3.4
68:	000000000400b60	2	FUNC	GLOBAL	DEFAULT	15	__libc_csu_fini
69:	0000000004007e0	0	FUNC	GLOBAL	DEFAULT	15	_start

```

70: 0000000000000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__
71: 0000000000000000 0 NOTYPE WEAK DEFAULT UND _Jv_RegisterClasses
72: 0000000000400ba8 0 FUNC GLOBAL DEFAULT 16 _fini
73: 0000000000000000 0 FUNC GLOBAL DEFAULT UND std::ios_base::Init::Init()@@
74: 0000000000000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_
75: 0000000000000000 0 FUNC GLOBAL DEFAULT UND __cxa_atexit@@GLIBC_2.2.5
76: 0000000000400798 0 FUNC GLOBAL DEFAULT UND std::ios_base::Init::~Init()@@
77: 0000000000000000 0 FUNC GLOBAL DEFAULT UND _ZStlsISt11char_traitsIcE
78: 0000000000400bb8 4 OBJECT GLOBAL DEFAULT 17 _IO_stdin_used
79: 0000000000601048 0 NOTYPE GLOBAL DEFAULT 26 __data_start
80: 0000000000400a96 50 FUNC WEAK DEFAULT 15 void swap<double>(double&, double&)
81: 0000000000601060 272 OBJECT GLOBAL DEFAULT 27 std::cout@@GLIBCXX_3.4
82: 0000000000601050 0 OBJECT GLOBAL HIDDEN 26 __dso_handle
83: 0000000000600e00 0 OBJECT GLOBAL HIDDEN 21 __DTOR_END__
84: 0000000000400ad0 137 FUNC GLOBAL DEFAULT 15 __libc_csu_init
85: 0000000000601058 0 NOTYPE GLOBAL DEFAULT ABS __bss_start
86: 0000000000601188 0 NOTYPE GLOBAL DEFAULT ABS _end
87: 0000000000000000 0 FUNC GLOBAL DEFAULT UND std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char, std::char_traits<char> >&
88: 00000000004007c8 0 FUNC GLOBAL DEFAULT UND _ZSt4endlIcSt11char_trait
89: 0000000000400a6a 44 FUNC WEAK DEFAULT 15 void swap<int>(int&, int&)
90: 0000000000601058 0 NOTYPE GLOBAL DEFAULT ABS _edata
91: 00000000004008c4 337 FUNC GLOBAL DEFAULT 15 main
92: 0000000000400720 0 FUNC GLOBAL DEFAULT 13 _init

```

Die Sektionstypen von Objektdateien:

text, data und bss

```

> size ./swap1
  text    data    bss     dec     hex filename
  3245    704     280    4229    1085 ./swap1

```

Hinweis zu verfügbaren Softwareentwicklungssystemen:

GNU g++ für Linux

cygwin für Windows

DreamSpark Premium (früher MSDNAA): VisualStudio 201x für Windows

## 1.5.2. Erstellen und Benutzen von statischen Bibliotheken

\*.a-Bibliotheken als Sammlungen von Objektdateien

Static library

Erzeugen statischer Bibliotheken

ar Manualpage

Wo werden einmal statisch gelinkte Binaries positioniert?

Binaries nur noch in `/usr/bin`

```
> cat swap1.cpp
```

```
#include <iostream>
template <typename T>
/*
 * Requirements: T muss einen Kopierkonstruktor haben,
 *               T muss einen Zuweisungsoperator zu T haben.
 */
void swap(T& a, T& b)
{
    T old_a(a);

    a = b;
    b = old_a;
}

int main() {
    int k(1);
    int l(5);
    std::cout << k << " " << l << std::endl;
    swap(k, l);
    std::cout << k << " " << l << std::endl;

    double d1(3.1415);
    double d2(15.1055);
    std::cout << d1 << " " << d2 << std::endl;
    swap(d1, d2);
    std::cout << d1 << " " << d2 << std::endl;
}

> make CXXFLAGS=-g swap1
```

```

g++ -g swap1.cpp -o swap1
> nm swap1 | grep swap | c++filt
0000000000400a96 W void swap<double>(double&, double&)
0000000000400a6a W void swap<int>(int&, int&)

> g++ -c swap1.cpp
> ls -al swap1.o
-rw-r--r-- 1 user1 users 4464  9. Nov 13:59 swap1.o

> ar rc libswap.a swap1.o
> ls -al libswap.a
-rw-r--r-- 1 user1 users 4650  9. Nov 14:02 libswap.a

> nm libswap.a | c++filt
0000000000000000 W void swap<double>(double&, double&)
0000000000000000 W void swap<int>(int&, int&)
                U std::basic_ostream<char, std::char_traits<char> >::operator<<(double)
                U std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
0000000000000000 T main

```

oder ein vollständiges Beispiel:

```

> cat person.h

/*
 * person.h
 */
class Person
{
public:
    Person() {};
    ~Person() {};

    void speak(const char * sentence);
};

> cat person.cpp

#include "person.h"
#include <iostream>

void Person::speak(const char * sentence)
{
    std::cout << sentence << std::endl;
}

```

```

> cat main.cpp

/*
 * main.cpp
 */

#include "person.h"
#include <iostream>

int main()
{
    Person person;
    person.speak("Hello world!");

    return 0;
}

> g++ -c person.cpp
> g++ -c main.cpp
> ar rc libperson.a person.o
> g++ -o main main.o -L. -lperson
> ldd main
        linux-vdso.so.1 => (0x00007ffffdfbff000)
        libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007f1f48ef2000)
        libm.so.6 => /lib64/libm.so.6 (0x00007f1f48c9b000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f1f48a85000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f1f48725000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f1f491fc000)

> ls -al main
-rwxr-xr-x 1 user1 users 13015  9. Nov 14:13 main

```

Bei den impliziten make-Regeln benutzte Environment-Variablen



### 1.5.3. Erstellen und Benutzen einer „shared object“- Bibliothek

```
> g++ -fPIC -c person.cpp
> g++ -shared -o libperson.so person.o
> g++ -o main main.o -L. -lperson
> ls -al main
-rwxr-xr-x 1 user1 users 12570  9. Nov 16:27 main

> ldd main
    linux-vdso.so.1 => (0x00007ffffc85fa000)
    libperson.so => not found
    libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007f990e302000)
    libm.so.6 => /lib64/libm.so.6 (0x00007f990e0ab000)
    libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f990de95000)
    libc.so.6 => /lib64/libc.so.6 (0x00007f990db35000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f990e60c000)

> ./main
./main: error while loading shared libraries: libperson.so: cannot
  open shared object file: No such file or directory

> export LD_LIBRARY_PATH=.
> ldd main
    linux-vdso.so.1 => (0x00007ffff6ebff000)
    libperson.so => ./libperson.so (0x00007f76ec213000)
    libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007f76ebf09000)
    libm.so.6 => /lib64/libm.so.6 (0x00007f76ebcb2000)
    libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f76eba9c000)
    libc.so.6 => /lib64/libc.so.6 (0x00007f76eb73c000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f76ec415000)

> ./main
Hello world!
```

Oder besser (mit Versionsinformationen):

[shared library HOWTO](#)

[YoLinux tutorial: libraries](#)

[Im Linuxumfeld genutzte Versionsnummern](#)

[.so Versionsnummern und Kompaibilität \(im Apache-Projekt\)](#)

[Why LD\\_LIBRARY\\_PATH is bad](#)

[When should I set LD\\_LIBRARY\\_PATH?](#)

[ldconfig](#)

[ldconfig\(8\)](#)

[Verwaltung von Shared Libraries](#)

Creating shared object libraries  
Anatomy of Linux dynamic libraries

Workarount für fehlende .so  
fix shared library load problems

#### **1.5.4. Bibliotheksmanagement insbesondere unter verschiedenen Betriebssystemen**

Using static and shared libraries across platforms  
Writing and Using Libraries (and plugins)

Dynamically Loaded (DL) Libraries  
plugin  
dynamic loading

## 1.6. STL-Templatequellen unter SuSE-Linux fürs zeilenweise Debuggen auch innerhalb der STL-Routinen

<input checked="" type="checkbox"/>	<b>gcc47-c++</b> Der GNU C++-Compiler	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>gcc47-c++-debuginfo</b> Debug information for package <b>gcc47-c++</b>	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>gcc47-debuginfo</b> Debug information for package <b>gcc47</b>	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>gcc47-debugsource</b> Debug sources for package <b>gcc47</b>	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>libstdc++47</b> Dynamische C++-Bibliotheken	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>libstdc++47-32bit</b> Dynamische C++-Bibliotheken	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>libstdc++47-debuginfo</b> Debug information for package <b>libstdc++47</b>	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>libstdc++47-devel</b> Enthält Dateien und Bibliotheken, die zum Programmieren benötigt werden	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>libstdc++47-devel-32bit</b> Enthält Dateien und Bibliotheken, die zum Programmieren benötigt werden	4.7.1_2...3-1.1.1
<input checked="" type="checkbox"/>	<b>glibc</b> Die Standard Shared Libraries (aus der GNU C-Bibliothek)	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-32bit</b> Die Standard Shared Libraries (aus der GNU C-Bibliothek)	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-debuginfo</b> Debug information for package <b>glibc</b>	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-debuginfo-32bit</b> Debug information for package <b>glibc</b>	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-debugsource</b> Debug sources for package <b>glibc</b>	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-devel</b> Include Files and Libraries Mandatory for Development	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-devel-32bit</b> Include Files and Libraries Mandatory for Development	2.15-22.6.4
<input checked="" type="checkbox"/>	<b>glibc-devel-debuginfo</b> Debug information for package <b>glibc-devel</b>	2.15-22.6.4

## 1.7. Automatisch überprüfte Requirements an Template-Parameter

Ein traditionelles Template-Beispiel:

```
#include <iostream>
#include <cmath>
#include <limits>

using namespace std;

template <typename T1, typename T2>
double geomMittel2(const T1& a, const T2& b)
{
    return sqrt(abs(a*b));
}

int main()
{
    cout << geomMittel2(3.0, 300.0) << endl;
    cout << geomMittel2(3, 300.0) << endl;
    cout << geomMittel2(-3, 300.0) << endl;
    cout << geomMittel2(-3, 300) << endl;
    cout << geomMittel2(3.0, 'c') << endl;
    cout << geomMittel2(3.0, "c") << endl;

    return 0;
}
```

und die Fehlermeldung für den letzten Testfall:

```
In Funktion >>double geomMittel2(const T1&, const T2&) [with T1 = double, T2 = char [2]]<<:
geomMittel2-0.cpp:20:33: instantiated from here
geomMittel2-0.cpp:10:25: Fehler: ungültige Operanden der Typen >>const double<< und >>const char [2]<< für binäres >>operator*<<
```

Nach einer verbesserten Bezeichnerwahl:

```
#include <iostream>
#include <cmath>
#include <limits>

using namespace std;

template <typename ArithmeticLike1, typename ArithmeticLike2>
double geomMittel2(const ArithmeticLike1& a, const
    ArithmeticLike2& b)
{
    return sqrt(abs(a*b));
}

int main()
{
    cout << geomMittel2(3.0, 300.0) << endl;
    cout << geomMittel2(3, 300.0) << endl;
    cout << geomMittel2(-3, 300.0) << endl;
    cout << geomMittel2(-3, 300) << endl;
    cout << geomMittel2(3.0, 'c') << endl;
    cout << geomMittel2(3.0, "c") << endl;

    return 0;
}
```

**Aufgabe:** Wie sieht die Fehlermeldung nun aus?

### 1.7.1. mit Hilfe von BOOST\_STATIC\_ASSERT()

```
#include <iostream>
#include <cmath>
#include <limits>
#include <boost/static_assert.hpp>
#include <boost/type_traits.hpp>

using namespace std;

BOOST_STATIC_ASSERT(std::numeric_limits<int>::digits >= 32);

template <typename ArithmeticLike1, typename ArithmeticLike2>
double geomMittel2(const ArithmeticLike1& a, const
    ArithmeticLike2& b)
{
    BOOST_STATIC_ASSERT(::boost::is_arithmetic<ArithmeticLike1>::
        value);
    BOOST_STATIC_ASSERT(::boost::is_arithmetic<ArithmeticLike2>::
        value);

    return sqrt(abs(a*b));
}

int main()
{
    cout << geomMittel2(3.0, 300.0) << endl;
    cout << geomMittel2(3, 300.0) << endl;
    cout << geomMittel2(-3, 300.0) << endl;
    cout << geomMittel2(-3, 300) << endl;
    cout << geomMittel2(3.0, 'c') << endl;
    cout << geomMittel2(3.0, "c") << endl;

    return 0;
}
```

mit der Compiler-Fehlermeldung:

```
geomMittel2.cpp: In instantiation of 'double geomMittel2(const ArithmeticLike1&, const ArithmeticLike2&) [with ArithmeticLike1 = double;
    ArithmeticLike2 = char [2]]':
geomMittel2.cpp:28:33:   required from here
geomMittel2.cpp:15:1: error: invalid application of 'sizeof' to incomplete type 'boost::STATIC_ASSERTION_FAILURE<false>'
geomMittel2.cpp:17:25: error: invalid operands of types 'const double' and 'const char [2]' to binary 'operator*'
```

Vergleiche `is_arithmetic` und `type_traits` der Boost

oder

## 1.7.2. mit Hilfe des c++11-Modus des g++

```
#include <iostream>
#include <cmath>
#include <limits>
#include <type_traits>

using namespace std;

static_assert(std::numeric_limits<int>::digits >= 32, "int not
    enough digits");

template <typename ArithmeticLike1, typename ArithmeticLike2>
double geomMittel2(const ArithmeticLike1& a, const
    ArithmeticLike2& b)
{
    static_assert(std::is_arithmetic<ArithmeticLike1>::value,
        "ArithmeticLike1 is not arithmetic");
    static_assert(std::is_arithmetic<ArithmeticLike2>::value,
        "ArithmeticLike2 is not arithmetic");

    return sqrt(abs(a*b));
}

// uebersetze mit -std=c++11
// oder make CXXFLAGS="-std=c++11" ...

int main()
{
    cout << geomMittel2(3.0, 300.0) << endl;
    cout << geomMittel2(3, 300.0) << endl;
    cout << geomMittel2(-3, 300.0) << endl;
    cout << geomMittel2(-3, 300) << endl;
    cout << geomMittel2(3.0, 'c') << endl;
    cout << geomMittel2(3.0, "c") << endl;

    return 0;
}
```

mit der Compiler-Fehlermeldung:

```
geomMittel-sa2.cpp: In instantiation of 'double geomMittel2(const ArithmeticLike1&, const ArithmeticLike2&) [with ArithmeticLike1 = double; ArithmeticLike2 = char [2]]':
geomMittel-sa2.cpp:29:33:   required from here
geomMittel-sa2.cpp:14:1: error: static assertion failed: ArithmeticLike2 is not arithmetic
geomMittel-sa2.cpp:16:25: error: invalid operands of types 'const double' and 'const char [2]' to binary 'operator*'
```

Vergleiche `static_assert`.

## 1.8. „horrible error messages“ bei STL-Nutzung

```
testmm.cpp:30: error: cannot convert ‘std::_Rb_tree_iterator<
  std::pair<const std::basic_string<char, std::char_traits<
  char>, std::allocator<char> >, Widget> > to ‘int in
  initialization
testmm.cpp:36: error: no matching function for call to ‘std::
  multimap<std::basic_string<char, std::char_traits<char>, std
  ::allocator<char> >, Widget, std::less<std::basic_string<
  char, std::char_traits<char>, std::allocator<char> > >, std
  ::allocator<std::pair<const std::basic_string<char, std::
  char_traits<char>, std::allocator<char> >, Widget> > >::
  insert(int)
/usr/include/c++/4.2.1/bits/stl_multimap.h:339: note:
  candidates are: typename std::_Rb_tree<_Key, std::pair<const
  _Key, _Tp>, std::_Select1st<std::pair<const _Key, _Tp> >,
  _Compare, typename _Alloc::rebind<std::pair<const _Key, _Tp>
  >::other>::iterator std::multimap<_Key, _Tp, _Compare,
  _Alloc>::insert(const std::pair<const _Key, _Tp>&) [with
  _Key = std::basic_string<char, std::char_traits<char>, std::
  allocator<char> >, _Tp = Widget, _Compare = std::less<std::
  basic_string<char, std::char_traits<char>, std::allocator<
  char> > >, _Alloc = std::allocator<std::pair<const std::
  basic_string<char, std::char_traits<char>, std::allocator<
  char> >, Widget> >]
/usr/include/c++/4.2.1/bits/stl_multimap.h:363: note:
  typename std::_Rb_tree<_Key, std::pair<const
  _Key, _Tp>, std::_Select1st<std::pair<const _Key, _Tp> >,
  _Compare, typename _Alloc::rebind<std::pair<const _Key, _Tp>
  >::other>::iterator std::multimap<_Key, _Tp, _Compare,
  _Alloc>::insert(typename std::_Rb_tree<_Key, std::pair<const
  _Key, _Tp>, std::_Select1st<std::pair<const _Key, _Tp> >,
  _Compare, typename _Alloc::rebind<std::pair<const _Key, _Tp>
  >::other>::iterator, const std::pair<const _Key, _Tp>&) [
  with _Key = std::basic_string<char, std::char_traits<char>,
  std::allocator<char> >, _Tp = Widget, _Compare = std::less<
  std::basic_string<char, std::char_traits<char>, std::
  allocator<char> > >, _Alloc = std::allocator<std::pair<const
  std::basic_string<char, std::char_traits<char>, std::
  allocator<char> >, Widget> >]
testmm.cpp:38: error: no matching function for call to ‘std::
  multimap<int, int, intComp, std::allocator<std::pair<const
  int, int> > >::insert(int)
```



```

/usr/include/c++/4.2.1/bits/stl_multimap.h:339: note:
  candidates are: typename std::_Rb_tree<_Key, std::pair<const
    _Key, _Tp>, std::_Select1st<std::pair<const _Key, _Tp> >,
    _Compare, typename _Alloc::rebind<std::pair<const _Key, _Tp>
      >::other>::iterator std::multimap<_Key, _Tp, _Compare,
    _Alloc>::insert(const std::pair<const _Key, _Tp>&) [with
    _Key = int, _Tp = int, _Compare = intComp, _Alloc = std::
    allocator<std::pair<const int, int> >]
/usr/include/c++/4.2.1/bits/stl_multimap.h:363: note:
    typename std::_Rb_tree<_Key, std::pair<const
      _Key, _Tp>, std::_Select1st<std::pair<const _Key, _Tp> >,
    _Compare, typename _Alloc::rebind<std::pair<const _Key, _Tp>
      >::other>::iterator std::multimap<_Key, _Tp, _Compare,
    _Alloc>::insert(typename std::_Rb_tree<_Key, std::pair<const
      _Key, _Tp>, std::_Select1st<std::pair<const _Key, _Tp> >,
    _Compare, typename _Alloc::rebind<std::pair<const _Key, _Tp>
      >::other>::iterator, const std::pair<const _Key, _Tp>&) [
    with _Key = int, _Tp = int, _Compare = intComp, _Alloc = std
    :: allocator<std::pair<const int, int> >]

```

### Workaround: Postprozessing

oder besser statische Requirementsüberprüfung zur Compilezeit.

Sehr leicht können bei unpassenden Typparametern und anderen Problemen komplizierte und unverständliche Compiler-Meldungen entstehen, was einfach mit der Tatsache zusammenhängt, dass die konkreten Anforderungen an die Typparameter unbekannt sind. Die Arbeit mit C++-Templates erfordert deshalb eine lückenlose Dokumentation der Anforderungen an einen Typparameter. Durch Template-Metaprogrammierung können die meisten Anforderungen (Basisklasse, Vorhandensein von Methoden, Kopierbarkeit, Zuweisbarkeit etc.) auch in speziellen Konstrukten abgefragt werden, wodurch sich lesbarere Fehlermeldungen ergeben. Obgleich sie standardkonform sind, werden diese Konstrukte jedoch nicht von allen Compilern unterstützt. (Siehe [http://de.wikipedia.org/wiki/Generische\\_Programmierung\\_in\\_Java](http://de.wikipedia.org/wiki/Generische_Programmierung_in_Java) Abschnitt „Das Konzept“ )

### Why do templates produce such horrible error messages?

In current C++, template errors are detected when a certain type argument does not support a certain operation (often expressed as the inability to convert one type to another or a failure of type deduction). It often happens deep into the instantiation tree. You need an equivalent of the stack trace to figure out where the root cause of the error is. It's called an "instantiation stack" and is dumped by the compiler upon a template error. It often spans several pages and contains unfamiliar names and implementation details of some library code.

Siehe <http://bartoszmilewski.wordpress.com/2010/06/24/c-concepts-a-postmortem/>: Absatz „Error Reporting“.

## 1.8.1. C++11 `type_traits`

Abschnitt 20.9.4ff.:

```
template <class T> struct is_void;
template <class T> struct is_integral;
template <class T> struct is_floating_point;
template <class T> struct is_array;
template <class T> struct is_pointer;
template <class T> struct is_lvalue_reference;
template <class T> struct is_rvalue_reference;
template <class T> struct is_member_object_pointer;
template <class T> struct is_member_function_pointer;
template <class T> struct is_enum;
template <class T> struct is_union;
template <class T> struct is_class;
template <class T> struct is_function;

template <class T> struct is_reference;
template <class T> struct is_arithmetic;
template <class T> struct is_fundamental;
template <class T> struct is_object;
template <class T> struct is_scalar;
template <class T> struct is_compound;
template <class T> struct is_member_pointer;

template <class T> struct is_const;
template <class T> struct is_volatile;
template <class T> struct is_trivial;
template <class T> struct is_trivially_copyable;
template <class T> struct is_standard_layout;
template <class T> struct is_pod;
template <class T> struct is_literal_type;
template <class T> struct is_empty;
template <class T> struct is_polymorphic;
template <class T> struct is_abstract;
template <class T> struct is_signed;
template <class T> struct is_unsigned;
template <class T> struct is_constructible;
template <class T> struct is_default_constructible;
template <class T> struct is_copy_constructible;
template <class T> struct is_move_constructible;
template <class T, class U> struct is_assignable;
template <class T> struct is_copy_assignable;
```

```

template <class T> struct is_move_assignable;
template <class T> struct is_destructible;
...
template <class T, class U> struct is_same;
template <class Base, class Derived> struct is_base_of;
template <class From, class To> struct is_convertible;
template <class From, class To> class is_explicitly_convertible;

```

## 1.8.2. BOOST type\_traits

### Boost: Type Traits

```

has_bit_and
has_bit_and_assign
has_bit_or
has_bit_or_assign
has_bit_xor
has_bit_xor_assign
has_complement
has_dereference
has_divides
has_divides_assign
has_equal_to
has_greater
has_greater_equal
has_left_shift
has_left_shift_assign
has_less
has_less_equal
has_logical_and
has_logical_not
has_logical_or
has_minus
has_minus_assign
has_modulus
has_modulus_assign
has_multiplies
has_multiplies_assign
has_negate
has_new_operator
has_not_equal_to
has_nothrow_assign
has_nothrow_constructor
has_nothrow_copy

```

has\_nothrow\_copy\_constructor  
has\_nothrow\_default\_constructor  
has\_plus  
has\_plus\_assign  
has\_post\_decrement  
has\_post\_increment  
has\_pre\_decrement  
has\_pre\_increment  
has\_right\_shift  
has\_right\_shift\_assign  
has\_trivial\_assign  
has\_trivial\_constructor  
has\_trivial\_copy  
has\_trivial\_copy\_constructor  
has\_trivial\_default\_constructor  
has\_trivial\_destructor  
has\_unary\_minus  
has\_unary\_plus  
has\_virtual\_destructor  
integral\_constant  
integral\_promotion  
is\_abstract  
is\_arithmetic  
is\_array  
is\_base\_of  
is\_class  
is\_complex  
is\_compound  
is\_const  
is\_convertible  
is\_empty  
is\_enum  
is\_floating\_point  
is\_function  
is\_fundamental  
is\_integral  
is\_lvalue\_reference  
is\_member\_function\_pointer  
is\_member\_object\_pointer  
is\_member\_pointer  
is\_object  
is\_pod  
is\_pointer  
is\_polymorphic

```
is_reference
is_rvalue_reference
is_same
is_scalar
is_signed
is_stateless
is_union
is_unsigned
is_virtual_base_of
is_void
is_volatile
```

### 1.8.3. numeric\_limits als Typ-Abbildung

#### numeric\_limits als generische Klasse

mit traits-ähnlichem Charakter für die Benutzung zum Beispiel für Requirements von Template-Parametern:

```
#include <limits>

template <class UnsignedInt>
class myclass
{
private:
    static_assert(std::numeric_limits<UnsignedInt>::digits >= 16,
        "UnsignedInt isn't long enough");
    static_assert(std::numeric_limits<UnsignedInt>::is_specialized,
        "UnsignedInt isn't specialized");
    static_assert(std::numeric_limits<UnsignedInt>::is_integer,
        "UnsignedInt isn't integer");
    static_assert(!std::numeric_limits<UnsignedInt>::is_signed,
        "UnsignedInt isn't unsigned");

public:
    /* details here */
};
myclass<unsigned> m1;
//myclass<int> m2;
myclass<unsigned char> m3;

int main()
{
    return 0;
}
```

18.3.2.3: allgemeines Template mit is\_specialized==false

18.3.2.7: Spezialisierung für float, bool

numeric\_limis: 18.3.2.2: alle bereitgestellten Spezialisierungen

## 1.9. Rückblick: typsichere Funktionsbenutzung

Der Prototypen (Signaturen) von Funktionen:

```
const double& max(const double& a, const double& b) throw();  
inline static constexpr float max() noexcept { return 3.40282347E+38F; }  
int myfunction (int param) throw(); // no exceptions allowed  
int myfunction (int param);        // all exceptions allowed ...
```

### Prototypen in Headerdateien

„Wichtig: Bei Prototypen unterscheidet C zwischen einer leeren Parameterliste und einer Parameterliste mit void . Ist die Parameterliste leer, so bedeutet dies, dass die Funktion eine nicht definierte Anzahl an Parametern besitzt. Das Schlüsselwort void gibt an, dass der Funktion keine Werte übergeben werden dürfen.“

(Funktionsprototypen in C)

### C functions without prototypes

## 1.10. Concepts und zielführende knappe Fehlermeldungen bei der Benutzung fehlerhafter aktueller generischer Parameter: 2017 oder wann?

C++11: Features originally planned but removed or not included

Concepts (C++)

[http:// www.generic-programming.org/languages/conceptcpp/tutorial/](http://www.generic-programming.org/languages/conceptcpp/tutorial/)

```
template<std:: CopyConstructible T>  
requires Addable<T>  
T sum(T array[], int n)  
{  
    T result = 0;  
    for (int i = 0; i < n; ++i)  
        result = result + array[i];  
    return result;  
}
```

nur für Klassen T mit:

```
auto concept CopyConstructible<typename T> {  
    T::T(T const&);  
    T::~~T();  
};
```

```

auto concept Addable<typename T, typename U = T> {
    typename result_type;
    result_type operator+(T, U);
};

```

Statt einer buchstabengetreuen Überprüfung auf Einhaltung der Konzepteigenschaften mittels der `auto`-Konzeptdefinition kann man durch `concept_maps` auch eine mittels „Übersetzung“ einzelner Operationen/Typen erreichbare Erfüllung der Konzepteigenschaften definieren:

```

concept Stack<typename X> {
    typename value_type;
    void push(X&, const value_type&);
    void pop(X&);
    value_type top(const X&);
    bool empty(const X&);
};

template<typename T> concept_map Stack<std::vector<T>> {
    typedef T value_type;
    void push(std::vector<T>& v, const T& x) { v.push_back(x); }
    void pop(std::vector<T>& v) { v.pop_back(); }
    T top(const std::vector<T>& v) { return v.back(); }
    bool empty(const std::vector<T>& v) { return v.empty(); }
};

```

(siehe [Concept\\_maps](#), retroaktive Modellierung)

Zu weiteren Concepts vergleiche:

[http://www.generic-programming.org/languages/conceptcpp/concept\\_web.php](http://www.generic-programming.org/languages/conceptcpp/concept_web.php)

Eine praktische Anwendung:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <concepts>

using namespace std;

auto concept HasAbs<typename T> {
    typename result_type;
    result_type abs(const T&);
}
auto concept HasPower<typename T>{
    typename result_type;
    result_type pow(const T&, int);
}
auto concept HasPowerd<typename T>{
    requires FloatingPointLike<T>;
    double pow(const T&, const T&);
}

template <int p = 2, InputIterator InputIter, FloatingPointLike T>
requires True<p >= 1>,
    HasAbs<InputIter::value_type>,
    HasPowerd<T>,
    HasPower<HasAbs<InputIter::value_type>::result_type>,
    HasPlusAssign<T,
        HasPower<HasAbs<InputIter::value_type>::result_type>::result_type>
    T pNorm(InputIter first, InputIter last, T init)
{
    for (; first != last; first++)
    {
        init += pow(abs(*first), p);
    };
    return pow((init), (1.0/p));
}

int main()
{
    vector<double> TD (2);
    TD[0] = 200.0;
    TD[1] = 0.0;

    double res = pNorm<3>(TD.begin(), TD.end(), 0.0f);
    cout << res << " sizeof: "
        << sizeof(pNorm(TD.begin(), TD.end(), 0.0f))
        << endl;

    double TestData [] = {110.0, 10.0, 10.0};
    cout << pNorm(TestData, TestData + 3, 0.0)
        << " sizeof: " << sizeof(pNorm(TestData, TestData + 3, 0.0))
        << endl;
}
```



```

    double TestData2[] = {10.0, 10.0, 10.0};
    cout << pNorm<1>(TestData2, TestData2 + 3, 0.01)
         << " sizeof: " << sizeof(pNorm<1>(TestData2, TestData2 +
         3, 0.01
))
    << endl;

    return 0;
}

```

Dabei wird aus das Konzept `HasPlusAssign<T, U>` der C++0x Standard Library benutzt:

```

auto concept HasPlusAssign<typename T, typename U = T> {
typename result_type;
result_type operator+=(T&, const U&);
}

```

(vergleiche [Foundational Concepts for the C++0x Standard Library \(Revision 5\)](#),  
[Proposed Wording for Concepts \(Revision 3\)](#),  
[Containers](#),  
[Algorithms \(Revision 2\)](#),  
[Iterators \(Revision 3\)](#),  
[Numerics \(Revision 3\)](#))

Link zu `conceptg++`:

<http://www.generic-programming.org/software/ConceptGCC/download.php>

compiler can type-check templates when they are defined, so mistakes show up earlier. Real support for Generic Programming also means that many of the template tricks that are needed in standard C++ are no longer necessary, and, yes, it provides much-improved error messages than we get with C++ compilers today.“

C++ Concepts: a Postmortem

Nächster C++-Standard soll 2017 kommen

... oder 2014?

## 1.11. Zielgerichtete Fehlermeldungen bei Nutzung einer C++-Standardbibliothek mit Konzepten

Fehlerhafte Benutzung einer generischen Funktion:

```
#include <list>
#include <algorithm>

int main() {
    std::list<int> l;
    std::sort(l.begin(), l.end());
}
```

g++ ohne eingeschränkte Generizität:

```
/usr/lib/gcc/i686-pc-linux-gnu/4.1.2/include/g++-v4/bits/stl_algo.h: In -
function 'void std::sort(_RandomAccessIterator, _RandomAccessIterator) [ -
with _RandomAccessIterator = std::_List_iterator<int>]':
list-sort.c++:6: instantiated from here
/usr/lib/gcc/i686-pc-linux-gnu/4.1.2/include/g++-v4/bits/stl_algo.h:2713: -
error: no match for 'operator-' in '__last - __first'
/usr/lib/gcc/i686-pc-linux-gnu/4.1.2/include/g++-v4/bits/stl_algo.h: In -
function 'void std::_final_insertion_sort(_RandomAccessIterator, -
_RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator< -
int>]':
...
```

conceptg++ mit Konzepte benutzender Standardbibliothek:

```
list-sort.c++: In function 'int main()':
list-sort.c++:6: error: no matching function for call to 'sort(std:: -
_List_iterator<int>, std::_List_iterator<int>)'
/usr/local/lib/gcc/i686-pc-linux-gnu/4.3.0/../../../../include/c++/4.3.0/ -
bits/stl_algo.h:2872: note: candidates are: void std::sort(_Iter, _Iter) -
[with _Iter = std::_List_iterator<int>] <where clause>
list-sort.c++:6: note: no concept map for requirement 'std:: -
MutableRandomAccessIterator<std::_List_iterator<int> >'
```

**Aufgabe:** Warum ist `list<...>::iterator` kein `MutableRandomAccessIterator`?

Eine Sammlung von Konzepten findet man unter [ConceptC++ Specification](#).

Zitat: „ConceptC++ makes programming with C++ templates easier, ...

## 1.12. Die Boost-Bibliotheken

Boost.StaticAssert mit RandomAccessIterator, UnsignedInt, ...

The Boost Concept Check Library (BCCL)

enable\_if für property based template overloading

Boost.Foreach (siehe auch for\_each() in 1.14)

Math Special Functions

## 1.13. Orte, wo statische Zusicherungen benutzt werden

Boost.StaticAssert

Use at namespace scope

Use at function scope

Use at class scope

Use in templates

## 1.14. statische Zusicherungen in C++11

static assertions

Range-based for-loop

25.2.2 ff.: all\_of(), any\_of(), none\_of(), for\_each()

### 25.2.2 Any of

[alg.any\_of]

```
template <class InputIterator, class Predicate>
bool any_of(InputIterator first, InputIterator last, Predicate pred);
```

1 *Returns:* false if [first,last) is empty or if there is no iterator *i* in the range [first,last) such that pred(\*i) is true, and true otherwise.

2 *Complexity:* At most last - first applications of the predicate.

### 25.2.3 None of

[alg.none\_of]

```
template <class InputIterator, class Predicate>
bool none_of(InputIterator first, InputIterator last, Predicate pred);
```

1 *Returns:* true if [first,last) is empty or if pred(\*i) is false for every iterator *i* in the range [first,last), and false otherwise.

2 *Complexity:* At most last - first applications of the predicate.

## 1.15. Fehlermeldungen bei uneingeschränkter Generizität (Fortsetzung von 1.8)

```
#include <vector>
#include <complex>
#include <algorithm>
```

```
int main()
{
    std::vector<std::complex<float>> v;
    std::stable_sort(v.begin(), v.end());
}
```

und die Fehlermeldung:

```
In file included from /usr/include/c++/4.5/algorithm:63:0,
    from bad_error_eg.cpp:3:
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>void std::_insertion_sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator =
/usr/include/c++/4.5/bits/stl_algo.h:3358:4: instantiated from >>void std::_inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2103:4: Fehler: no match for >>operator<< in >>_i.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [wit
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>void std::_merge_without_buffer(_BidirectionalIterator, _BidirectionalIterator, _BidirectionalIterator,
/usr/include/c++/4.5/bits/stl_algo.h:3364:7: instantiated from >>void std::_inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2963:4: Fehler: no match for >>operator<< in >>_middle.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator*
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>void std::_unguarded_linear_insert(_RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__no
/usr/include/c++/4.5/bits/stl_algo.h:2111:6: instantiated from >>void std::_insertion_sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAcc
/usr/include/c++/4.5/bits/stl_algo.h:3358:4: instantiated from >>void std::_inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2064:7: Fehler: no match for >>operator<< in >>_val < _next.__gnu_cxx::__normal_iterator<_Iterator, _Container>::ope
In file included from /usr/include/c++/4.5/vector:61:0,
    from bad_error_eg.cpp:1:
/usr/include/c++/4.5/bits/stl_algobase.h: In Funktion >>_ForwardIterator std::lower_bound(_ForwardIterator, _ForwardIterator, const _Tp&) [with _ForwardIter
/usr/include/c++/4.5/bits/stl_algo.h:2975:4: instantiated from >>void std::_merge_without_buffer(_BidirectionalIterator, _BidirectionalIterator, _Bidirec
/usr/include/c++/4.5/bits/stl_algo.h:3364:7: instantiated from >>void std::_inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algobase.h:976:4: Fehler: no match for >>operator<< in >>_middle.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator
In file included from /usr/include/c++/4.5/algorithm:63:0,
    from bad_error_eg.cpp:3:
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>_FIter std::upper_bound(_FIter, _FIter, const _Tp&) [with _FIter = __gnu_cxx::__normal_iterator<std::com
/usr/include/c++/4.5/bits/stl_algo.h:2982:4: instantiated from >>void std::_merge_without_buffer(_BidirectionalIterator, _BidirectionalIterator, _Bidirec
/usr/include/c++/4.5/bits/stl_algo.h:3364:7: instantiated from >>void std::_inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2461:4: Fehler: no match for >>operator<< in >>_val < _middle.__gnu_cxx::__normal_iterator<_Iterator, _Container>::o
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _OIter) [with _IIter1 = std::complex<float>*, _IIt
/usr/include/c++/4.5/bits/stl_algo.h:2838:4: instantiated from >>void std::_merge_adaptive(_BidirectionalIterator, _BidirectionalIterator, _Bidirectional
/usr/include/c++/4.5/bits/stl_algo.h:3315:7: instantiated from >>void std::_stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:5299:4: Fehler: no match for >>operator<< in >>_first2.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator*
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>_BidirectionalIterator3 std::_merge_backward(_BidirectionalIterator1, _BidirectionalIterator1, _Bidirec
/usr/include/c++/4.5/bits/stl_algo.h:2847:4: instantiated from >>void std::_merge_adaptive(_BidirectionalIterator, _BidirectionalIterator, _Bidirectional
/usr/include/c++/4.5/bits/stl_algo.h:3315:7: instantiated from >>void std::_stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2740:4: Fehler: no match for >>operator<< in >>_last2 < _last1.__gnu_cxx::__normal_iterator<_Iterator, _Container>
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _OIter) [with _IIter1 = __gnu_cxx::__normal_iterat
/usr/include/c++/4.5/bits/stl_algo.h:3163:4: instantiated from >>void std::_merge_sort_loop(_RandomAccessIterator1, _RandomAccessIterator1, _RandomAccess
/usr/include/c++/4.5/bits/stl_algo.h:3261:4: instantiated from >>void std::_merge_sort_with_buffer(_RandomAccessIterator, _RandomAccessIterator, _Pointer
/usr/include/c++/4.5/bits/stl_algo.h:3312:4: instantiated from >>void std::_stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:5299:4: Fehler: no match for >>operator<< in >>_first2.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator*
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion >>_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _OIter) [with _IIter1 = std::complex<float>*, _IIt
/usr/include/c++/4.5/bits/stl_algo.h:3163:4: instantiated from >>void std::_merge_sort_loop(_RandomAccessIterator1, _RandomAccessIterator1, _RandomAccess
/usr/include/c++/4.5/bits/stl_algo.h:3263:4: instantiated from >>void std::_merge_sort_with_buffer(_RandomAccessIterator, _RandomAccessIterator, _Pointer
/usr/include/c++/4.5/bits/stl_algo.h:3312:4: instantiated from >>void std::_stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2: instantiated from >>void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41: instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:5299:4: Fehler: no match for >>operator<< in >>* _first2 < * _first1<
```

C++11:

25.4.1.2 `stable_sort`

[stable.sort]

```
template<class RandomAccessIterator>
void stable_sort(RandomAccessIterator first, RandomAccessIterator last);
```

```
template<class RandomAccessIterator, class Compare>
void stable_sort(RandomAccessIterator first, RandomAccessIterator last,
                 Compare comp);
```

- 1 *Effects:* Sorts the elements in the range `[first, last)`.
- 2 *Requires:* `RandomAccessIterator` shall satisfy the requirements of `ValueSwappable` (17.6.3.2). The type of `*first` shall satisfy the requirements of `MoveConstructible` (Table 20) and of `MoveAssignable` (Table 22).
- 3 *Complexity:* It does at most  $N \log^2(N)$  (where  $N == \text{last} - \text{first}$ ) comparisons; if enough extra memory is available, it is  $N \log(N)$ .
- 4 *Remarks:* Stable.

# SGL: stable\_sort

## Algorithms

Category: algorithms

## Function

Component type:

### Prototype

`stable_sort` is an overloaded name; there are actually two `stable_sort` functions.

```
template <class RandomAccessIterator>
void stable_sort(RandomAccessIterator first, RandomAccessIterator last);

template <class RandomAccessIterator, class StrictWeakOrdering>
void stable_sort(RandomAccessIterator first, RandomAccessIterator last,
                StrictWeakOrdering comp);
```

### Description

`stable_sort` is much like `sort`: it sorts the elements in `[first, last)` into ascending order, meaning that if `i` and `j` are any two valid iterators in `[first, last)` such that `i` precedes `j`, then `*i` is not less than `*j`. `stable_sort` differs from `sort` in two ways. First, `stable_sort` uses an algorithm that has different run-time complexity than `sort`. Second, as the name suggests, `stable_sort` is stable: it preserves the relative ordering of equivalent elements. That is, if `x` and `y` are elements in `[first, last)` such that `x` precedes `y`, and if the two elements are equivalent (neither `x < y` nor `y < x`) then a postcondition of `stable_sort` is that `x` still precedes `y`. [1]

The two versions of `stable_sort` differ in how they define whether one element is less than another. The first version compares objects using operators`<`, and the second compares objects using a [function object](#) `comp`.

### Definition

Defined in the standard header [algorithm](#), and in the nonstandard backward-compatibility header [algo.h](#).

### Requirements on types

For the first version, the one that takes two arguments:

- `RandomAccessIterator` is a model of [Random Access Iterator](#).
- `RandomAccessIterator` is mutable.
- `RandomAccessIterator`'s value type is [LessThan Comparable](#).
- The ordering relation on `RandomAccessIterator`'s value type is a *strict weak ordering*, as defined in the [LessThan Comparable](#) requirements.

[iterator\\_traits](#)  
[associated types](#)

## 1.16. Verbesserte Fehlermeldungen bei Nutzung von StaticAssert

### 1.16.1. RandomAccessIterator

Zunächst die uneingeschränkt generische Variante:

```
#include <iostream>
#include <vector>
#include <set>
#include <complex>
#include <algorithm>
#include <iterator>
#include <type_traits>

template <typename RandomAccessIterator >
RandomAccessIterator foo(RandomAccessIterator from,
                        RandomAccessIterator to)
{
    // this template should only be used with
    // random access iterators...
    //
    // ...
    reverse(from, to);
    return from;
};

int main()
{
    std::vector<float> v;
    // std::set<float> v;

    v.insert(v.end(), 3.14);
    v.insert(v.end(), 15.15);
    foo(v.begin(), v.end());
    copy(v.begin(), v.end(), std::ostream_iterator<float>(
        std::cout, " "));

    std::cout << std::endl;
}

```

wird einwandfrei übersetzt

```
> make CXXFLAGS="-std=c++11" ra2
g++ -std=c++11 ra2.cpp -o ra2

```

und funktioniert einwandfrei, für

```
int main()
{
    // std::vector<float> v;
    std::set<float> v;
    //...
```

erscheint jedoch keine vernünftige Fehlermeldung

```
> make CXXFLAGS="-std=c++11" ra2
/usr/include/c++/4.7/bits/stl_algobase.h: In instantiation of 'static void std::_iter_swap<_BidirectionalIterator, _BidirectionalIterator>(_BidirectionalIterator, _BidirectionalIterator) [with _BidirectionalIterator = std::set_iterator<float>]':
/usr/include/c++/4.7/bits/stl_algobase.h:139:7:   required from 'void std::iter_swap(_ForwardIterator, _ForwardIterator) [with _ForwardIterator = std::set_iterator<float>]'
/usr/include/c++/4.7/bits/stl_algo.h:1428:6:   required from 'void std::_reverse(_BidirectionalIterator, _BidirectionalIterator) [with _BidirectionalIterator = std::set_iterator<float>]'
/usr/include/c++/4.7/bits/stl_algo.h:1474:7:   required from 'void std::reverse(_BIter, _BIter) [with _BIter = std::set_iterator<float>]'
ra2.cpp:25:4:   required from 'RandomAccessIterator foo(RandomAccessIterator, RandomAccessIterator) [with RandomAccessIterator = std::set_iterator<float>]'
ra2.cpp:36:28:   required from here
/usr/include/c++/4.7/bits/stl_algobase.h:90:11: error: assignment of read-only location 'a'
/usr/include/c++/4.7/bits/stl_algobase.h:91:11: error: assignment of read-only location 'b'
```

## RandomAccessIterator

Durch ein geeignetes statisches Assert der Art

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <set>
#include <complex>
#include <type_traits>

template <typename RandomAccessIterator >
RandomAccessIterator foo(RandomAccessIterator from,
                        RandomAccessIterator to)
{
    // this template can only be used with
    // random access iterators...
    typedef typename std::iterator_traits<
        RandomAccessIterator >::iterator_category cat;
    static_assert(
        (std::is_convertible<
            cat,
            const std::random_access_iterator_tag >::value),
        "no random access iterator");
    //
    // ...
    reverse(from, to);
}
```



```

    return from;
};

int main()
{
    // std::vector<float> v;
    std::set<float> v;

    v.insert(v.begin(), 3.14);
    v.insert(v.begin(), 15.15);
    foo(v.begin(), v.end());
    copy(v.begin(), v.end(), std::ostream_iterator<float>(std
        ::cout, " "));
    std::cout << std::endl;
}

```

wird jedoch der falsche aktuelle generische Parametertyp gemeldet:

```

make CXXFLAGS="-std=c++11" ra2
g++ -std=c++11 -g ra2.cpp -o ra2
ra2.cpp: In instantiation of 'RandomAccessIterator foo(RandomAccessIterator, RandomAccessIterator) [with
ra2.cpp:36:28:   required from here
ra2.cpp:18:4: error: static assertion failed: no random access iterator
make: *** [ra2] Fehler 1

```

## 1.16.2. Nicht instanziierbare Klassen

```
#include <iostream>
#include <algorithm>
#include <type_traits>

template <typename T>
struct abstractClass
{
    static_assert(false, "This class may not be instantiated!");
    // ...
};

int main()
{
    abstractClass<int> ac;
    std::cout << std::endl;
}
```

Beachte eventuell: [http://www.boost.org/doc/libs/1\\_55\\_0/doc/html/boost\\_staticassert.html#boost\\_staticassert\\_templates](http://www.boost.org/doc/libs/1_55_0/doc/html/boost_staticassert.html#boost_staticassert_templates)

## 1.16.3. Erzwingung gleicher Typen

```
#include <limits>
#include <type_traits>

template <class UnsignedInt>
class myclass
{
private:
    static_assert(std::is_same<UnsignedInt, unsigned int>::value,
        "UnsignedInt isn't unsigned int");
public:
    /* details here */
};

//myclass<unsigned> m1;
//myclass<int> m2;
//myclass<unsigned char> m3;
myclass<unsigned long> m4;

int main()
{
    return 0;
}
```

siehe auch: [http://en.cppreference.com/w/cpp/types/is\\_same](http://en.cppreference.com/w/cpp/types/is_same)

## 1.16.4. Funktionen mit „(int/float/...) type promotion“-Returntyp

4.5 Integral promotions

4.6 Floating point promotion

5 Expressions, clause 10: usual arithmetic conversions

```
#include "promote.h"
template <typename T1, typename T2>
    typename promote_trait<T1,T2>::T_promote
    my_function(T1 x, T2 y)
{
    return (x + y)/2.0;
}
```

mit `promote.h` ähnlich wie (alle möglichen Typkombinationen spezialisiert):

```
template <typename T1, typename T2>
struct promote_trait{
    typedef T1 T_promote;
};

template<> struct promote_trait<char, char> {
public:
    typedef int T_promote;
};
//...
```

(Vergleiche: [promote.h](#))

In C++11 eleganter: [automatisch bestimmter return-Typ](#):

```
template <typename T1, typename T2>
auto my_function(T1 x, T2 y) -> decltype((x + y)/2.0)
{
    return (x + y)/2.0;
}
```

Siehe auch [BOOST promote](#)

## 1.16.5. Auf Unterklassen eingeschränkte Generizität

```
template <typename ListUnterklasse>
class MyList
{
    static_assert (std::is_base_of<List,
                    ListUnterklasse >::value,
                    "ListUnterklasse ist keine Kindklasse von List");
    // ...
}
```

Siehe [C++11 Abschnitt 20.9.6 Relationships between types](#).

Diese Typeigenschaften werden durch trickreichen Einsatz der Metaprogrammierung aus dem C++-Compiler „herausgekitzelt“: [How is\\_base\\_of works?](#) beziehungsweise [Alternate implementation of is\\_base\\_of](#) oder direkt durch die g++ type\_traits Compiler Extensions erfragt.

## 1.16.6. g++ type\_traits Compiler Extensions

... oder sie benutzen direkt die syntaktischen Erweiterungen der „GNU Compiler Collection“ [7.9 Type Traits](#):

```
__is_base_of (base_type, derived_type)
    If base_type is a base class of derived_type ([class.derived]) then the
    trait is true, otherwise it is false. Top-level cv qualifications of
    base_type and derived_type are ignored. For the purposes of this trait,
    a class type is considered its own base.
    Requires: if __is_class (base_type) and __is_class (derived_type) are
    true and base_type and derived_type are not the same type (disregarding
    cv-qualifiers), derived_type shall be a complete type. Diagnostic is
    produced if this requirement is not met.
```

## 1.16.7. Type Traits in D

Andere Programmiersprachen (hier z.B. D) haben ähnliche elegantere semantische Erweiterungen, um den Compiler direkt abzufragen:

```
// Returns true if instances of type T can be added
template isAddable(T)
{ // Works by attempting to add two instances of type T
    const isAddable = __traits(compiles, (T t) { return t + t;
    });
}

int Foo(T)(T t)
```

```

    if (isAddable!(T))
{
    return 3;
}
// ...

```

Deren Anwendung für ADTs (als Requirements-Ersatz):

```

template isStack(T)
{
    const isStack =
        --traits(compiles,
            (T t)
            {
                T.value_type v = top(t);
                push(t, v);
                pop(t);
                if (empty(t)) { }
            }
        );
}

```

```

template Foo(T)
    if (isStack!(T))
{
    ...
}

```

D Template Constraints

D Traits

### 1.16.8. C++ *has\_member* fehlt

C++ type traits to check if class has operator/member

How to decide if a template class has a certain member function?

C++-Pattern „Member Detector“

## 1.16.9. SFINAE

substitution failure is not an error

**SFINAE** Warum ein fehlgehender Instanzierungsversuch keinen Compilationsfehler verursacht:

```
struct Test {
    typedef int foo;
};

template <typename T>
void f(typename T::foo) {} // Definition #1

template <typename T>
void f(T) {} // Definition #2

int main() {
    f<Test>(10); // Call #1.
    f<int>(10); // Call #2. Without error (SFINAE)
}
```

### C++: SFINAE

SFINAE bei der nützlichen Trait-Arbeit:

```
template <typename T>
    struct has_iterator
    {
        template <typename U>
        static char test(typename U::iterator* x);

        template <typename U>
        static long test(U* x);

        static const bool value = sizeof(test<T>(0)) == 1;
    };
// ...
cout << has_iterator<int>::value << endl;
```

### C++ SFINAE examples: IsClassT

arguments. For each function template, if the argument deduction and checking succeeds, the *template-arguments* (deduced and/or explicit) are used to synthesize the declaration of a single function template specialization which is added to the candidate functions set to be used in overload resolution. If, for a given function template, argument deduction fails, no such function is added to the set of candidate functions for that template. The complete set of candidate functions includes all the synthesized declarations and all of

## 1.16.10. C++11: Traits mit decltype statt sizeof()-Tricks

Detect operator support with decltype/SFINAE

## 1.16.11. Überladene Templatefunktionen

Overload candidate function set

enable\_if

Bedingte Compilation mittels enable\_if und SFINAE:

```
template <bool B, class T =  
void> struct enable_if; // If B is true, the member typedef type  
// shall equal T; otherwise, there shall be  
// no member typedef type.
```

---

### 1.16.11.1. enable\_if-Funktionen

```
#include <iostream>  
#include <cmath>  
#include <limits>  
  
using namespace std;  
  
template <typename ArithmeticLike1, typename ArithmeticLike2>  
typename std::enable_if<std::is_arithmetic<ArithmeticLike1>::value,  
double>::type  
geomMittel2(const ArithmeticLike1& a, const ArithmeticLike2& b)  
{  
    return sqrt(abs(a*b));  
}  
  
// uebersetze mit -std=c++11  
// oder g++ CXXFLAGS="-std=c++11" ...  
  
int main()  
{  
    cout << geomMittel2(3.0, 300.0) << endl;  
    cout << geomMittel2(3, 300.0) << endl;  
    cout << geomMittel2(-3, 300.0) << endl;  
    cout << geomMittel2(-3, 300) << endl;  
    cout << geomMittel2(3.0, 'c') << endl;  
    // cout << geomMittel2(3.0, "c") << endl;  
  
    return 0;  
}
```

## 1.16.11.2. Konflikt beim enable\_if-Funktionsüberladen

### 3.2 Overlapping enabler conditions

```
template <class T>
typename enable_if<std::is_integral<T>::value, void>::type
foo(T t) {}
```

```
template <class T>
typename enable_if<std::is_arithmetic<T>::value, void>::type
foo(T t) {}
```

Beide foo()-Deklarationen schließen sich nicht gegenseitig aus, erzeugen also mehrdeutigen Code. Durch Kombination ähnlich ... possible to use std::enable\_if to ... kann das vermieden werden:

```
struct A
{
    template <typename T>
    typename std::enable_if<std::is_integral<T>::value, T>::type foo()
    {
        std::cout << "integral" << std::endl;
        return T();
    }

    template <typename T>
    typename std::enable_if<!std::is_integral<T>::value
        &&(std::is_arithmetic<T>::value), T>::type foo
        ()
    {
        std::cout << "arithmetic but not integral" << std::endl;
        return T();
    }
}
```

## 1.16.11.3. bedingte „template class specializations“

### 3.1 Enabling template class specializations

```
template <class T, class Enable = void>
class A { ... };
```

```
template <class T>
class A<T, typename enable_if<std::is_integral<T>::value>::type> { ... };
```

```
template <class T>
class A<T, typename enable_if<std::is_float<T>::value>::type> { ... };
```

Ein Beispiel:

```
#include <iostream>
#include <vector>
```



```

using namespace std;

template <typename T, typename Enable = void>
struct A { A(){ cout<< "is something other ..." << endl;} };

template <typename T>
struct A<T, typename enable_if<is_floating_point<T>::value >::type> {
A(){ cout<< "is floatingpoint" << endl;}
};

template <typename T>
struct A<T, typename enable_if<is_integral<T>::value >::type> {
A(){ cout<< "is integral" << endl;}
};

template <typename T>
struct A<T, typename enable_if<!is_pod<T>::value >::type> {
A(){ cout<< "is no pod" << endl;}
};

template <typename T>
struct A<T, typename enable_if<(is_enum<T>::value)&&(is_scalar<T>::value)
>::type> {
A(){ cout<< "is enum and scalar" << endl;}
};

int main(){
    typedef vector<string> vektortyp1;
    typedef int* IntPtrertype;
    enum Farbe {ROT, GELB, GRUEN, BLAU};

    A<char>();
    A<signed>();
    A<float>();
    A<vektortyp1>();
    A<Farbe>();
    A<IntPtrertype>();
}

```

und die Ergebnisse:

```

is integral
is integral
is floatingpoint
is no pod
is enum and scalar
is something other ...

```

Siehe auch: [std::conditional](#)

```

int main()
{
    typedef std::conditional<true, int, double>::type Typ1;
    ...
}

```

und seine Benutzung im C++11-Standarddokument

```
template <class T> typename conditional<  
    !is_nothrow_move_constructible<T>::value &&  
    is_copy_constructible<T>::value ,  
    const T&, T&&>::type move_if_noexcept(T& x) noexcept;
```

um die Signatur der Funktion `move_if_noexcept()` anzugeben.

## 1.17. Template-Deklarationen zur Erzeugung von Objektdateien mit einer Ansammlung von Template-Instanzen

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

template <typename ArithmeticLike1, typename ArithmeticLike2>
double geomMittel2(const ArithmeticLike1& a,
                  const ArithmeticLike2& b)
{
    return sqrt(abs(a*b));
}

template double geomMittel2<short, float>(const short&, const float&);
template double geomMittel2<int, float>(const int&, const float&);
template double geomMittel2<long, float>(const long&, const float&);
template double geomMittel2<float, float>(const float&, const float&);
template double geomMittel2<double, float>(const double&, const float&);
template double geomMittel2<long double, float>(const long double&, const
float&);
// ...
template double geomMittel2<short, double>(const short&, const double&);
template double geomMittel2<int, double>(const int&, const double&);
template double geomMittel2<long, double>(const long&, const double&);
template double geomMittel2<float, double>(const float&, const double&);
template double geomMittel2<double, double>(const double&, const double&);
template double geomMittel2<long double, double>(const long double&, const
double&);
// ...
template double geomMittel2<short, long double>(const short&, const long
double&);
template double geomMittel2<int, long double>(const int&, const long
double&);
template double geomMittel2<long, long double>(const long&, const long
double&);
template double geomMittel2<float, long double>(const float&, const long
double&);
template double geomMittel2<double, long double>(const double&, const long
double&);
template double geomMittel2<long double, long double>(const long double&,
const long double&);
```

**Besser:** Eine Sammlung von Objektdateien, die jeweils (nur) eine Instantiierung enthält, damit die erzeugte Bibliothek nur die benötigten Kompilationseinheiten einbinden läßt.

## 1.18. Wo ist die Template-Instanz?

Abschnitt 7.5: *Where's the template?*

g++-Compileroptionen mit Template-Relevanz:

```
-fno-implicit-templates  
-fno-implicit-inline-templates  
-fno-pretty-templates  
-frepo
```

(siehe *g++-Manual*, Kapitel 3).

## 1.19. C++11 extern template

C++11 extern template

N1448

## 1.20. Generic Programming

[www.generic-programming.org](http://www.generic-programming.org) mit:

- (viele/mehrerere/...) konkrete Implementierungen -> größtallgemeine Templateversion (Lifting), Requirements
- Muster immer wieder gemeinsam auftretender Requirements: Concepts
- hierarchische Sortierung der Concepts der Anwendungsdomain

problem domains of generic libraries

Example generic algorithms/concepts:

- generische Algorithmen mit „Requirements on Types“:  
`sort()`  
`power()`  
`accumulate()`  
...  
`fill()`
- Concepts:  
`Associative Container`  
`Assignable`  
...  
`Default Constructible`

## 1.21. Generic Programming in ConceptC++

<http://www.generic-programming.org/languages/conceptcpp/>

Concepts mit:

1. functions, operators, constructors, ...
2. associated types (`return_type`, `value_type`, `difference_type`, ...)

(Concepts können Verfeinerungen anderer Concepts sein (Vererbung))

`concept_maps` weisen Typen als konzeptmodellierend aus und beschreiben eine eventuell nichtbuchstabengetreue, per Abbildung vermittelte Konzepterfüllung (Morphismus). Sie ermöglichen retroaktives Modellieren.

Requirement-Klauseln ermöglichen vollständige zielgerichtete Fehlermeldungen bei Konzeptverletzung sowohl bei Instanzierungsversuch eines Templates mit einem falschen generischen Parameter als auch bei der Nutzung von mehr als der durch Requirement-Klauseln geforderten Eigenschaften in der Template-Implementierung.

Konzeptbasiertes Funktionsüberladen ohne `enable_if` möglich.

C++-Standardbibliothek mit Konzepten.

[ConceptGCC Download](#)

## 1.22. Concepts, concept\_maps, axioms

`void fill(ForwardIterator first, ForwardIterator last, const T& value)`

wie es in C++11 ist: 25.3.6

wie es hätte sein können: 25.2.6

im SGI STL-Manual

Siehe [Concepts, concept maps, axioms](#)

Aus der konzeptnutzenden STL:

```
template<ForwardIterator Iter , class V>
    requires Assignable<Iter::value_type,V>
    void fill(Iter first , Iter last , const V& v);
```

und die Benutzung:

```
fill(0, 9, 9.9);
// error: int is not a ForwardIterator

fill(&v[0], &v[9], 9.9);
// ok: int* is a ForwardIterator
```

Fehlermeldung bei der Implementierung mit unzureichender Requirementliste:

```
template<ForwardIterator Iter , class V>
    requires Assignable<Iter::value_type,V>
void fill(Iter first , Iter last , const V& v)
{
    while (first!=last) {
        *first = v;
        first=first+1; // error: + not defined
                       // for Forward_iterator
    }
}
```

**Concept based overloading:**

```
// iterator-based standard sort (with concepts):
template<Random_access_iterator Iter>
    requires Comparable<Iter::value_type>
void sort(Iter first , Iter last); // use the usual implementation

// container-based sort:
template<Container Cont>
    requires Comparable<Cont::value_type>
void sort(Cont& c)
{
    sort(c.begin(),c.end()); // simply call the iterator
                             version
}

void f(vector<int>& v)
{
    sort(v.begin(), v.end()); // one way
    sort(v);                  // another way
    // ...
}
```

### concept\_maps:

Zeiger sind Vorbild für Iteratoren, ihnen fehlt jedoch der assoziierte Typ `value_type`. Dem kann man mit einer `concept_map` abhelfen:

```
template<typename T>
concept_map ForwardIterator<T*> {
    typedef T value_type;
};
```

### Axioms:

```
concept Semigroup<typename Op, typename T> : CopyConstructible<T>
{
    T operator()(Op, T, T);
    axiom Associativity(Op op, T x, T y, T z) {
        op(x, op(y, z)) <=> op(op(x, y), z);
    }
}

concept Monoid<typename Op, typename T> : Semigroup<Op, T> {
    T identity_element(Op);
    axiom Identity(Op op, T x) {
        op(x, identity_element(op)) <=> x;
        op(identity_element(op), x) <=> x;
    }
}
// in concept TotalOrder:
axiom Transitivity(Op op, T x, T y, T z)
{
    if (op(x, y) && op(y, z)) op(x, z) <=> true;
    // conditional equivalence
}
```

## 1.23. ConceptC++-Tutorial

vergleiche <http://www.generic-programming.org/languages/conceptcpp/tutorial/>

Ausgangspunkt: ein oder mehrere konkrete Algorithmen

```
double sum(double array[], int n)
{
    double result = 0;
    for (int i = 0; i < n; ++i)
        result = result + array[i];
    return result;
}
```

**Erster Verallgemeinerungsschritt:**

```
template<typename T>
T sum(T array[], int n)
{
    T result = 0;
    for (int i = 0; i < n; ++i)
        result = result + array[i];
    return result;
}
```

führt beim ersten Instanziierungsversuch

```
struct pod
{
    int i;
};

int main()
{
    pod values[3] = { {1}, {2}, {3} };
    sum(values, 3);
    return 0;
}
```

zu Fehlermeldungen:

```
> conceptg++ array1-1.C
array1-1.cc: In function 'T sum(T*, int) [with T = pod]':
array1-1.cc:19:   instantiated from here
array1-1.cc:5: error: conversion from 'int' to non-scalar type 'pod' requested
array1-1.cc:7: error: no match for 'operator+' in 'result + *((+(((unsigned int)i) * 4u)) + array)'
```



Die Spezifikation eines notwendigen Konzepts:

```
#include <concepts>
template<std::CopyConstructible T>
T sum(T array[], int n)
{
    T result = 0;
    for (int i = 0; i < n; ++i)
        result = result + array[i];
    return result;
}
```

Jetzt lautet die Fehlermeldung:

```
> conceptg++ array2.C
array2.cpp: In function 'T sum(T*, int)':
array2.cpp:5: error: conversion from 'int' to non-scalar type 'T' requested
array2.cpp:7: error: no match for 'operator+' in 'result + array[i]'
```

Vervollständigung des Sets der nötigen Konzepte:

```
template<std::CopyConstructible T>
requires Addable<T>
T sum(T array[], int n)
{
    T result = 0;
    for (int i = 0; i < n; ++i)
        result = result + array[i];
    return result;
}
```

Fehlermeldung:

```
array3.cpp: In function 'T sum(T*, int)':
array3.cpp:10: error: conversion from 'int' to non-scalar type 'T' requested
array3.cpp:12: error: no match for 'operator=' in 'result = Addable<T>::operator+(result, array[i])'
```

Erneute Vervollständigung des Sets der benötigten Konzepte:

```
template<std::CopyConstructible T>
requires Addable<T>, Assignable<T>
T sum(T array[], int n)
{
    T result = 0;
    for (int i = 0; i < n; ++i)
        result = result + array[i];
    return result;
}
```

Immer noch Fehler:

```
array4.cpp: In function 'T sum(T*, int)':
array4.cpp:14: error: conversion from 'int' to non-scalar type 'T' requested
```

Letzte Erweiterung der benötigten Konzepte:

```
auto concept IntConstructible <typename T> {  
    T::T(int);  
};  
  
template<std::CopyConstructible T>  
    requires Addable<T>, Assignable<T>, IntConstructible<T>  
    T sum(T array[], int n)  
    {  
        T result = 0;  
        for (int i = 0; i < n; ++i)  
            result = result + array[i];  
        return result;  
    }
```

und fehlerlose Übersetzung. Wir haben das Konzept für die generische Funktion `sum()` zusammengestellt.

**Bemerkung:** Nach Muster der STL (siehe zum Beispiel `accumulate()`) wäre die folgende generische Version besser:

```
template<std::CopyConstructible T>  
    requires Addable<T>, Assignable<T>  
    T sum(T array[], int n, T result)  
    {  
        for (int i = 0; i < n; ++i)  
            result = result + array[i];  
        return result;  
    }
```

**Generische Programmierung ist die Programmierung mit Concepts!**

## Zweiter Verallgemeinerungsschritt: (Zeiger statt Felder)

```
template<std::CopyConstructible T>
requires Addable<T>, Assignable<T>
T sum(T* first, T* last, T result)
{
    for (; first != last; ++first)
        result = result + *first;
    return result;
}
```

Übersetzung ohne Fehlermeldung!

## Dritter Verallgemeinerungsschritt: (Iteratoren statt Zeiger)

```
template<ForwardIterator Iter>
requires Addable<Iter::value_type>, Assignable<Iter::value_type>
Iter::value_type sum(Iter first, Iter last, Iter::value_type result)
{
    for (; first != last; ++first)
        result = result + *first;
    return result;
}
```

mit dem Konzept

```
concept ForwardIterator<typename Iter> {
    typename value_type;

    Iter& operator++(Iter& x);

    value_type& operator*(Iter);

    bool operator==(Iter, Iter);
    bool operator!=(Iter, Iter);
};
```

der STL-Übersetzung der Implementierung erfolgreich.

Jedoch Instanziierung mittels

```
...
double TestData [] = { 110.0, 10.0, 10.0};
std::cout << sum(TestData, TestData + 3, 0.0f)
...
```

führt zur Fehlermeldung, dass Zeiger keine Forwarditeratoren sind. Die `concept_map`

```
concept_map ForwardIterator<double*> {
    typedef double value_type;
};
```

behebt diese Unkenntnis.

Um alle Zeiger zu Forwarditeratoren zu machen, kann man folgende Konzeptmap einführen:

```
template<typename T>
concept_map ForwardIterator<T*> {
    typedef T value_type;
};
```

#### **Retroaktive Konzepterfüllung:**

Soll Die Klasse Color

```
class Color {
public:
    Color ();
    Color mix(const Color& other) const;

    // ...
};
```

das Konzept Addable erfüllen, ist das durch

```
concept_map Addable<Color> {
    Color operator+(const Color& x, const Color& y) { return x.mix(y); }
};
```

erreichbar.

#### **Vierter Verallgemeinerungsschritt: sum() result type different to Iter::value\_type**

```
template<ForwardIterator Iter, Assignable T>
requires Addable<T>
T sum(Iter first, Iter last, T result)
{
    for (; first != last; ++first)
        result = result + *first;
    return result;
}
```

Fehlermeldung:

```
op1.cpp: In function 'T sum(Iter, Iter, T)':
op1.cpp:26: error: no match for 'operator+' in 'result + * first'
op1.cpp:3: note: candidates are: T Addable<T>::operator+(const T&, const T&)
```

Neue Konzeptmap:

```
auto concept Addable<typename T, typename U = T> {
    T operator+(T x, U y);
};
template<ForwardIterator Iter, Assignable T>
requires Addable<T, value_type>
T sum(Iter first, Iter last, T result)
{
    for (; first != last; ++first)
        result = result + *first;
    return result;
}
```

Fehlerfrei!

Aufgabe: Überlege eine allgemeinere `sum`-Version, die Container statt mittels `+` mit einer beliebigen zweiargumentigen Operation `op(,)` **verjüngt**.

ConceptC++ führt den Programmierer generischer Funktionen oder Klassen Schritt für Schritt zur ausreichenden Deklaration der benötigten Eigenschaften der formalen generischen Parameter. Nach einmaliger erfolgreicher Übersetzung sind später keine Überraschungen wegen fehlenden Operationen mehr zu erwarten! Das erreicht ConceptC++ dadurch, dass es eine minimale Klasse, die nur genau die in allen Concepts genannten Operationen enthält (einen Archetyp) erzeugt und für einen Testinstanzierungsversuch benutzt: Die Fehlermeldungen bei Nutzung von nicht in der Liste der Concepts genannte Operationen, die im Code der generischen Funktion/Klasse benutzt werden, sahen Sie oben.

Eine Sammlung von Konzepten findet man unter [ConceptC++ Specification](#):

```
concept ArithmeticLike<typename T>
: Regular<T>, LessThanComparable<T>, HasPlus<T>, HasMinus<T>,
  HasMultiply<T>, HasDivide<T>, HasUnaryPlus<T>, HasNegate<T> {

T::T(long long);

T& operator++(T&);
T operator++(T& t, int) { T tmp(t); ++t; return tmp; }
T& operator--(T&);
T operator--(T& t, int) { T tmp(t); --t; return tmp; }

requires Convertible<HasUnaryPlus<T>::result_type, T>
  && Convertible<HasNegate<T>::result_type, T>
  && Convertible<HasPlus<T>::result_type, T>
  && Convertible<HasMinus<T>::result_type, T>
  && Convertible<HasMultiply<T>::result_type, T>
  && Convertible<HasDivide<T>::result_type, T>;

T& operator*=(T&, T);
T& operator/=(T&, T);
T& operator+=(T&, T);
T& operator-=(T&, T);
}

auto concept Semiregular<typename T>
: CopyConstructible<T>, CopyAssignable<T> {
  requires SameType<CopyAssignable<T>::result_type, T&>;
}

auto concept Regular<typename T>
: Semiregular<T>, DefaultConstructible<T>, EqualityComparable<T>,
  HeapAllocatable<T> { }
```

## 1.24. C++11: was aus C++03 ist nicht mehr da?

Features removed or deprecated:

„sequence point“, export template, NNN() throw(std::bad\_alloc), auto\_ptr, unary\_function

## 1.25. aktueller Workaround: Nutzung von Typetraits statt von Concepts

### concept

A concept contains a set of requirements:

#### requirement

A requirement is part of a [concept](#) that describes the behavior of an abstraction. Requirements tend to be syntactic (e.g., all Input Iterators have a dereference operation), semantic (e.g., one can traverse the sequence of values returned from a Forward Iterator multiple times), or performance-related (e.g., incrementing an Input Iterator occurs in constant amortized time).

(aus: [Generic Programming Glossary](#))

### 20.9.2 Header <type\_traits> synopsis

[meta.type.synop]

```
namespace std {
    // 20.9.3, helper class:
    template <class T, T v> struct integral_constant;
    typedef integral_constant<bool, true> true_type;
    typedef integral_constant<bool, false> false_type;

    // 20.9.4.1, primary type categories:
    template <class T> struct is_void;
    template <class T> struct is_integral;
    template <class T> struct is_floating_point;
    template <class T> struct is_array;
```

`std::integral_constant`

Ein Workaround-Concept als Kombination mehrerer Traiteigenschaften an generische Parameter:

```
template<typename Iter , typename T>
struct is_summable :
    std::integral_constant<bool ,
        NNN::is_input_iterator<Iter >::value &&
        std::is_assignable<T,T>::value &&
        boost::has_plus_assign<
            T,
            iterator_traits<iter >::value_type ,
            T
        >::value
    >
{};
```

(Implementierungs-idee ähnlich /usr/include/c++/4.7/type\_traits)

Leider ist weder in BOOST noch in C++11 eine trait-Metafunktion `is_input_iterator` <> vordefiniert. In <http://calder.sdml.cs.kent.edu/svn/origin/old/sandbox/iterators/include/origin/iterator/traits.hpp> war sie noch vorhanden.

Also bleibt nur die Möglichkeit, gemäß 1.16.1 vorzugehen oder eine eigene Metafunktion nach den Ideen von 1.16.1 oder

<http://stackoverflow.com/questions/8751460/how-to-restrict-an-iterator-to-being-a-forward-iterator> zu schreiben:

```
template <typename It>
typename std::enable_if<
    is_same<typename std::iterator_traits<It>::iterator_category,
        std::forward_iterator_tag>::value,
    bool>::type ...
```

oder besser (warum?):

```
template <typename It>
typename std::enable_if<std::is_base_of<std::forward_iterator_tag,
    typename std::iterator_traits<It>::
        iterator_category>::value,
    bool>::type ...
```

(aus: <http://stackoverflow.com/questions/8751460/how-to-restrict-an-iterator-to-being-a-forward-iterator>)

## 1.26. Assoziierte Typen, Tags, Tag-Dispatching

associated type

An associated type is a type that is used to describe the requirements of a concept, but is not actually a parameter to the concept. For instance, the reference type returned when dereferencing an Input Iterator is expressed as an associated type. In languages that do not directly support associated types, type parameters can be used instead at some cost to brevity.

(aus: <http://www.generic-programming.org/about/glossary.php>)

```
<class Iterator> struct iterator_traits {
    typedef typename Iterator::difference_type difference_type;
    typedef typename Iterator::value_type value_type;
    typedef typename Iterator::pointer pointer;
    typedef typename Iterator::reference reference;
    typedef typename Iterator::iterator_category iterator_category;
}
```

### Associated Types

Beispiel eines eigenen assoziierten Typs:

```
template <typename T1, typename T2>
struct promote_trait{
    typedef T1 T_promote;
};
template<> struct promote_trait<char, unsigned char> {
public:
    typedef int T_promote;
};
template<> struct promote_trait<short int, long> {
public:
    typedef long T_promote;
};
// ...
```

Siehe: [promote\\_trait](#)

[value\\_type](#), [difference\\_type](#), ... in [iterator\\_traits](#), ...:

```
namespace std {
template<class Iterator> struct iterator_traits {
typedef typename Iterator::difference_type difference_type;
typedef typename Iterator::value_type value_type;
typedef typename Iterator::pointer pointer;
typedef typename Iterator::reference reference;
typedef typename Iterator::iterator_category iterator_category;
};
\\ ...
template<class T> struct iterator_traits<T*> {
typedef ptrdiff_t difference_type;
typedef T value_type;
typedef T* pointer;
typedef T& reference;
typedef random_access_iterator_tag iterator_category;
};
```



```
// ...
}
```

Fallweise je nach Iteratortyp unterschiedlicher Code:

Neben der SFINE-Lösung vom Ende des vorherigen Abschnitts:

```
struct input_iterator_tag { };
struct output_iterator_tag { };
struct forward_iterator_tag : public input_iterator_tag { };
struct bidirectional_iterator_tag : public forward_iterator_tag { };
struct random_access_iterator_tag : public bidirectional_iterator_tag { };
```

Eine Hierarchie von **Tags** und **Tag-Dispatching**:

```
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

template< class BDIter >
void alg(BDIter, BDIter, std::bidirectional_iterator_tag)
{
    std::cout << "alg() called for bidirectional iterator\n";
}

template <class RAIter>
void alg(RAIter, RAIter, std::random_access_iterator_tag)
{
    std::cout << "alg() called for random-access iterator\n";
}

template< class Iter >
void alg(Iter first, Iter last)
{
    alg(first, last,
        typename std::iterator_traits<Iter>::iterator_category());
}

int main()
{
    std::vector<int> v;
    alg(v.begin(), v.end());

    std::list<int> l;
    alg(l.begin(), l.end());

    // std::istreambuf_iterator<char> i1(std::cin), i2;
    // alg(i1, i2); // compile error: no matching function for call
}
```

## 1.27. Archetypen

<http://www.generic-programming.org/languages/cpp/techniques.php#archetypes>

Der Workaround „Nutzung von Traits statt Concepts“ versagt bei der Unterstützung des Autors generischer Konstrukte durch Nutzeffekte von Archetypen!

## 1.28. Generic Programming Techniques of the BOOST Libraries

Survey of some of the generic programming techniques used in the boost libraries

Anatomy of a Concept

Traits

C++ type\_traits

BOOST type\_traits

Tag dispatching

Adaptors

Type generators (type factory)

Metaprogramming

Object generators (object factory)

Policy classes

Policy-based design

Weitere C++ Template-Technologien:

Curiously Recurring Template Pattern

Restricted Template Expansion

Parameterized Base Class

## 1.29. POD-Typen und trait-fallweises Überladen

POD Types

C++11: POD = "trivial type" or "standard-layout"

Move-Semantik für pod-Daten/Copy-Semantik für non-pod-Daten:

```
template<typename T>
void copy(T const* source, T* dest, unsigned count)
{
    static_assert(std::is_pod<T>::value, "T must be a POD");
    memcpy(dest, source, count*sizeof(T));
}

// ... oder besser pod/non-pod sensitives copy():

template<typename T>
typename std::enable_if<std::is_pod<T>::value, void>::type
copy(T const* source, T* dest, unsigned count)
{
    memcpy(dest, source, count*sizeof(T));
}

template<typename T>
typename std::enable_if<!std::is_pod<T>::value, void>::type
copy(T const* source, T* dest, unsigned count)
{
    for (unsigned i=0; i<count; ++i)
    {
        *dest++=*source++;
    }
}
```

aus: pod/non-pod overloading in: [Checking Concepts without Concepts in C++](#)

## 1.30. Eigene Klassen-Tags und Tag-Dispatching oder fallweise Spezialisierung

Alternativ Tag-Dispatching:

```
template <bool> struct podness {};
typedef podness<true> pod_tag;
typedef podness<false> non_pod_tag;

template <typename T> void f2(T, pod_tag) { /* POD */ }
template <typename T> void f2(T, non_pod_tag) { /* non-POD */ }

template <typename T>
void f(T x)
{
    // Dispatch to f2 based on tag.
}
```

```

    f2(x, podness<std::is_pod<T>::value>());
}

```

oder fallweise spezialisierte Klassen:

```

template <typename T, bool> struct f2;

template <typename T>
struct f2<T, true> { static void f(T) { /* POD */ } };

template <typename T>
struct f2<T, false> { static void f(T) { /* non-POD */ } };

template <typename T>
void f(T x)
{
    // Select the correct partially specialised type.
    f2<T, std::is_pod<T>::value>::f(x);
}

```

(aus: [Tag dispatch versus static methods on partially specialised classes](#))

## 1.31. Für die Template-Programmierung interessante Neuerungen in C++11

- constexpr
- initialize lists
- type inference: auto, decltype
- range-based for-loop insbesondere bei der Existenz von Iteratoren:
  - foreach and custom iterator
  - Creating my own Iterators
  - How to implement an STL-style iterator and avoid common pitfalls
  - iterator library
  - Iterator categories
  - Custom Iterator in C++
  - How to realize a custom implementation of a std-like iterator?
- alternative function syntax
- override and final
- u8 string literals
- defaulted and deleted special member functions
- alias templates
- variadic templates
- C++11 - New features - Variadic templates

## 1.32. Curiously recurring template pattern

CRTP ist die Technologie, in der eine Kindklasse einer generischen Elternklasse den Zugriff auf sich selbst (mittel aktuellem Templateparameter) erlaubt.

```
template <typename T>
struct counter
{
    static int objects_created;
    static int objects_alive;

    counter()
    {
        ++objects_created;
        ++objects_alive;
    }
protected:
    ~counter() // objects should never be removed through pointers of this
               type
    {
        --objects_alive;
    }
};
template <typename T> int counter<T>::objects_created( 0 );
template <typename T> int counter<T>::objects_alive( 0 );

class X : counter<X>
{
    // ...
};

class Y : counter<Y>
{
    // ...
};
```

(aus [Curiously recurring template pattern](#), 16.3 The Curiously Recurring Template Pattern (CRTP) )

## 1.33. Objektorientierte Entwurfsmuster

[Design Patterns](#)  
[Entwurfsmuster](#)  
[Design Patterns](#)

## 1.34. Sprechweisen bei C++(11)-Programmkonstrukten

[Make C++ easy to teach and to learn without ...](#)  
[More C++ Idioms](#)

[Effective C++, Third Edition, 2005](#); [More Effective C++, 1996](#); [Effective STL, 2001](#)

## 1.35. Type Generator

Type Generator  
type generators

## 1.36. Object Generator

object generators  
std::pair  
Fabrikmethoden

## 1.37. Parameterized Base Class

Parameterized Base Class  
Mixin-Based Programming in C++

## 1.38. Barton-Nackman Trick (Restricted Template Expansion)

More C++ Idioms: Barton-Nackman trick  
Wiki: Barton-Nackman trick

## 1.39. Die Boost Concept Check Library (BCCL)

Using Concept Checks:

```
#include <boost/concept_check.hpp>

template <class T>
void generic_library_function(T x)
{
    BOOST_CONCEPT_ASSERT(( EqualityComparable<T>));
    // ...
};

template <class It>
class generic_library_class
{
    BOOST_CONCEPT_ASSERT(( RandomAccessIterator<It >));
    // ...
};
//... BOOST_CONCEPT_ASSERT()
//    only for things seeable in the
//    template function definition
oder

#include <boost/concept/requires.hpp>
#include <boost/concept_check.hpp>

template<typename RanIter>
BOOST_CONCEPT_REQUIRES(
    (( Mutable_RandomAccessIterator<RanIter>))
    (( LessThanComparable<typename Mutable_RandomAccessIterator<RanIter >::
        value_type >)),
    (void)) // return type
    stable_sort (RanIter, RanIter);
//... otherwise
```

## 1.39.1. Difference of Draft C++ Cocepts and BCCL

What's the difference between C++0x concepts and The Boost Concept Check Library:

- Compiler brauchen Templates bis zur entgültigen Instanziierung nicht zu übersetzen, also auch nicht syntaktisch zu analysieren. Das Auftreten möglicher Fehlermeldungen ist deshalb bis zur Instanziierung aufgeschoben. Um eine vollständige Testabdeckung zu erreichen, benötigt man ein „Urmuster“ des Gebrauchs aller nach Konzept vorgeschriebenen Operationen, die testcompiliert vorhandene fehlende Operationsdefinitionen aufdecken würde: einen **Archetyp** des Konzepts.

Die ursprünglich in den C++0x geplanten Konzepte hätten Archtypen automatisch erzeugt und benutzt, somit die Templatedefinition automatisch vollständig typgechecked. Das bisherige C++-template-Handling läßt mögliche in der Dokumentation einer Template-Bibliothek unerwähnte Requirements eines generischen Objekts lange unentdeckt und frustriert zu unvorhergesehenen Zeiten dessen Benutzer mit einer bis dahin nie aufgetretenen Fehlermeldung(skaskade): **Motivierendes BCCL-Beispiel.**

Bei Benutzung der BCCL hat man eigene Konzepte selbst mit Archtypen auszustatten und diese testzucompilieren (siehe Abschnitt 1.21.3). Die in der BCCL vordefinierten für die STL nötigen Konzepte sind in der Datei `boost/concept_archetype.hpp` mit Archtypen ausgestattet.

Zum Beispiel das Konzept `InputIterator` mit den geforderten Operationen `++i, (void)i++, *i++,*i; Defaultkonstruktor, operator=, operator-> (TrivialIterator); Kopierkonstruktor, swap() (Assignable); operator==, operator!= (EqualityComparable); Defaultkonstruktor (DefaultConstructible)` (vgl. [STL InputIterator](#)) und dem folgenden Archetyp dafür:

```
//  
  


---

  
// Iterator Archetype Classes  
  
template <class T, int I = 0>  
class input_iterator_archetype  
{  
private:  
    typedef input_iterator_archetype self;  
public:  
    typedef std::input_iterator_tag iterator_category;  
    typedef T value_type;  
    struct reference {  
        operator const value_type&() const { return static_object<T>::  
            get(); }  
    };  
    typedef const T* pointer;  
    typedef std::ptrdiff_t difference_type;  
    self& operator=(const self&) { return *this; }  
    bool operator==(const self&) const { return true; }  
    bool operator!=(const self&) const { return true; }  
    reference operator*() const { return reference(); }  
    self& operator++() { return *this; }  
    self operator++(int) { return *this; }  
};
```

- BCCL unterstützt die Überprüfung von semantischen Requirements wie z.B. Benutzbarkeit in Multipass-Algorithmen ... **nicht!**



- BCCL unterstützt das **Syntaxremapping** (temporäres renaming, Context\_maps) **nicht**.
- BCCL unterstützt Kontext-basiertes Überladen **nicht**. (Ausweg: `enable_if` mit Concept Traits)

## 1.39.2. In der Boost-Bibliothek vordefinierte (STL-)Konzepte

Zur Referenz: [boost/concept\\_check.hpp](#)

### Basic Concept Checking Classes

Zum Unterschied von Assignable und SGIAssignable:

```
template <class TT>
struct AssignableConcept
{
    void constraints() {
#ifdef _ITERATOR_ // back-insert-it broken for VC++ STL
        a = a; // require assignment operator
#endif
        const_constraints(a);
    }
    void const_constraints(const TT& b) {
#ifdef _ITERATOR_ // back-insert-it broken for VC++ STL
        a = b; // const required for arg to assignment
#endif
    }
    TT a;
};
```

im Vergleich zu:

```
// The SGI STL version of Assignable requires copy constructor and
// operator=
template <class TT>
struct SGIAssignableConcept
{
    void constraints() {
        TT b(a);
#ifdef _ITERATOR_ // back-insert-it broken for VC++ STL
        a = a; // require assignment operator
#endif
        const_constraints(a);
        ignore_unused_variable_warning(b);
    }
    void const_constraints(const TT& b) {
        TT c(b);
#ifdef _ITERATOR_ // back-insert-it broken for VC++ STL
        a = b; // const required for arg to assignment
#endif
        ignore_unused_variable_warning(c);
    }
    TT a;
};
```

### Iterator Concept Checking Classes

```
template <class TT, class ValueT>
struct OutputIteratorConcept
{
    void constraints() {
        function_requires< AssignableConcept<TT> >();
        ++i;           // require preincrement operator
        i++;           // require postincrement operator
        *i++ = t;      // require postincrement and assignment
    }
    TT i, j;
    ValueT t;
};
```

### Function Object Concept Checking Classes

```
template <class Func, class Return>
struct GeneratorConcept
{
    void constraints() {
        const Return& r = f(); // require operator() member fct
        ignore_unused_variable_warning(r);
    }
    Func f;
};
```

### Container Concept Checking Classes

```
template <class Container>
struct ContainerConcept
{
    typedef typename Container::value_type value_type;
    typedef typename Container::difference_type difference_type;
    typedef typename Container::size_type size_type;
    typedef typename Container::const_reference const_reference;
    typedef typename Container::const_pointer const_pointer;
    typedef typename Container::const_iterator const_iterator;

    void constraints() {
        function_requires< InputIteratorConcept<const_iterator> >();
        function_requires< AssignableConcept<Container> >();
        const_constraints(c);
    }
    void const_constraints(const Container& c) {
        i = c.begin();
        i = c.end();
        n = c.size();
        n = c.max_size();
        b = c.empty();
    }
    Container c;
};
```

```
bool b;  
const_iterator i;  
size_type n;  
};
```

## 1.40. Ein Blick zurück (2003..2008) — und vorwärts 2017? Usage-Pattern oder Pseudosignatur

(Am Anfang) „Usage Pattern“-FallstudienAnsatz:

Stroustrup: N1510 (2003)

```
template<class Value_type> struct Forward_iterator {
    static void constraints(Forward_iterator p)
    {
        Forward_iterator q = p; p = q; // can be copied
        p++; ++p; // can be incremented
        Value_type v = *p; // points to Value_types
    }
    // ...
}
// ...
concept Element {
    constraints(Element e1, Element e2) {
        bool b = e1 < e2; // Elements can be compared using <
        // and the result can be used as a
        // bool
        swap(e1, e2); // Elements can be swapped
    }
};
// ...
concept Add { // We can copy and add Adds
    constraints(Add x) {
        Add y = x; x = y; x = x+y; Add xx = x+y;
    }
};
// ...
```

Der Pseudosignatur-Ansatz führt zunächst zu Problemen bei ähnlichen unterschiedlichen Signaturen.

Var<>-Platzhalter, where, &&(2006):

### Stroustrup: Specifying C++ Concepts

```
concept Mutable_fwd<typename Iter, typename T> {
    Var<Iter> p;           // a variable of type Iter.
    Var<const T> v;       // a variable of type const T.

    Iter q = p;           // an Iter must be copy-able

    bool b = (p != q);    // must support != operation,
                          // and the resulting expression
                          // must be convertible to bool
    ++p;                  // must support pre-increment, no
                          // requirements on the result type

    *p = v;               // must be able to dereference p,
                          // and assign a const T to the
                          // result of that dereference; no
                          // requirements on the result type
};
//...
concept Small<typename T, int N>
    where sizeof (T) <= N
{ };
// ...
concept C<typename X>
    where C1<X> && C2<X>
{ };
// ...
concept Mutable_fwd<typename Iter, typename T> {
    Var<Iter> p;           // placeholder for variable of type Iter.
    Var<const T> v;
    Var<T> v2;
    // ... as before ...
    *p = v;               // we can write to *p
    v2 = *p;              // we can read from *p
};
// ...
```

```

concept Forward_iterator<typename Iter> {
    Var<Iter> p;           // a variable of type Iter.
    typename Iter::value_type // must have a named member
                          // associated type value_type.

    Iter q = p;           // an Iter must be copy-able
    bool b = (p != q);    // must support == and !=
    b = (p == q);         // operations, and the resulting
                          // expressions must be convertible
                          // to bool.

    ++p;                  // must support pre- and
    p++;                  // post-increment operations, no
                          // assumption on the result type
};
// ...
concept Copy_constructible<typename T> {
    Var<T> a;
    T b = a;              // copy construction
    T c(a);               // direct copy construction
};

concept Assignable<typename T, typename U = T> {
    Var<T> a;
    Var<const U> b;
    a = b;                // copy (non-destructive read)
};

concept Movable<typename T, typename U = T> {
    Var<T> a;
    Var<U> b;
    a = b;                // potentially-destructive read
};

concept Equality_comparable<typename T, typename U = T> {
    Var<const T> a;
    Var<const U> b;
    bool eq = (a == b);
    bool neq = (a != b);
};
// ...
concept Container<typename X> {
    X::X(int n);
    X::~~X();
    bool X::empty() const;
}

```

## 1.41. Creating Own Boost Concept Checking Classes

### 1.41.1. Ein Beispiel

Example

```
// #include <type_traits>
template <class X>
struct InputIterator
    : Assignable<X>, EqualityComparable<X>
{
private:
    typedef std::iterator_traits<X> t;
public:
    typedef typename t::value_type value_type;
    typedef typename t::difference_type difference_type;
    typedef typename t::reference reference;
    typedef typename t::pointer pointer;
    typedef typename t::iterator_category iterator_category;

    BOOST_CONCEPT_ASSERT(( SignedInteger<difference_type >));
    BOOST_CONCEPT_ASSERT(( Convertible<iterator_category ,
        std::input_iterator_tag >));

    BOOST_CONCEPT_USAGE(InputIterator)
    {
        X j(i);           // require copy construction
        same_type(*i++,v); // require postincrement-
                        // dereference returning value_type
// or in C++11:
//     static_assert(is_same(*i++,v)::value);
        X& x = ++j;      // require preincrement returning X&
    }

private:
    X i;
    value_type v;

    // Type deduction will fail unless the arguments have the
    // same type.
    template <typename T>
    void same_type(T const&, T const&);
};
```

Kochrezept:

- Concept Checking „Template Class“ mit Namen des Konzepts erstellen
- Notwendige Vater-Konzepte mittels Vererbung spezifizieren (hier: `Assignable` und `EqualityComparable`)
- Nötige assoziierte Typen mittels `typedef` als Membertype definieren
- Durch `BOOST_CONCEPT_ASSERT()` nötige Einschränkungen an die assoziierten Typen formulieren
- Mittels `BOOST_CONCEPT_USAGE()` die geforderten Eigenschaften (Operationen) der das Konzept erfüllenden Klassen (als Usage-Pattern) spezifizieren
- Für `BOOST_CONCEPT_USAGE()` nötige Objektdeklarationen als Datamember der „Concept Checking Class“ und eventuell nötige Funktionsdeklarationen (wie hier `same_type(.,.)`) vornehmen

## 1.41.2. Erstellung eines zugehörigen Archetypes

Concept Covering and Archetypes

archetypes

see: 5 Concept Covering

Ein Beispiel:

```
template <class T>
class input_iterator_archetype
{
private:
    typedef input_iterator_archetype self;
public:
    typedef std::input_iterator_tag iterator_category;
    typedef T value_type;
    struct reference {
        operator const value_type&() const { return static_object<T>::get(); }
    };
    typedef const T* pointer;
    typedef std::ptrdiff_t difference_type;
    self& operator=(const self&) { return *this; }
    bool operator==(const self&) const { return true; }
    bool operator!=(const self&) const { return true; }
    reference operator*() const { return reference(); }
    self& operator++() { return *this; }
    self& operator++(int) { return *this; }
};
```

Die Testabdeckungsüberprüfung kann durch einen Instanzierungsversuch der Archetypen der Requirements aller generischen Algorithmen erreicht werden, zum Beispiel:

```
typedef less_than_comparable_archetype<
    sgi_assignable_archetype<> ValueType;
random_access_iterator_archetype<ValueType> ri;
std::stable_sort(ri, ri);
```



## 1.42. Programmieren mit Konzepten

Minimiere die **Requirements** an die Input-Parameter generischer Komponenten, um deren Wiederverwendbarkeit zu steigern. Damit erreicht man die größtmögliche Allgemeinheit!

Konzepte (Kombinationen von Requirements) sind ein Sprachmechanismus, der es gestattet, dem Compiler (oder zur Zeit wenigstens dem Benutzer einer generischen Bibliothek in Kommentarform) verbindlich mitzuteilen, welche Typeigenschaften ein generischer Algorithmus (ein Template-Konstrukt) benutzen darf beziehungsweise welche Eigenschaften ein Typ besitzen muß, damit er als aktueller generischer Parameter für eine Template-Instanziierung benutzt werden darf.

**Table 1.** Concepts, constraints, and axioms

<i>Concepts</i>		<i>Constraints</i>	
<i>Regularity</i>	<i>Iterators</i>	<i>Operators</i>	<i>Language</i>
<b>Comparable</b>	<b>Iterator</b>	<b>Equal</b>	<b>Same</b>
<b>Ordered</b>	<b>Forward_iterator</b>	<b>Less</b>	<b>Common</b>
<b>Copyable</b>	<b>Bidirectional_iterator</b>	<b>Logical_and</b>	<b>Derived</b>
<b>Movable</b>	<b>Random_access_iterator</b>	<b>Logical_or</b>	<b>Convertible</b>
<b>Regular</b>		<b>Logical_not</b>	<b>Signed_int</b>
		<b>Callable</b>	
<i>Functional</i>	<i>Types</i>	<i>Initialization</i>	<i>Other</i>
<b>Function</b>	<b>Boolean</b>	<b>Destructible</b>	<b>Procedure</b>
<b>Operation</b>		<b>Constructible</b>	<b>Input_iterator</b>
<b>Predicate</b>		<b>Assignable</b>	<b>Output_iterator</b>
<b>Relation</b>			
<i>Axioms</i>			
<b>Equivalence_relation</b>			
<b>Strict_weak_order</b>			
<b>Strict_total_order</b>			
<b>Boolean_algebra</b>			

```

concept Ordered<Regular T> {
  requires constraint Less<T>;
  requires axiom Strict_total_order<less<T>, T>;
  requires axiom Greater<T>;
  requires axiom Less_equal<T>;
  requires axiom Greater_equal<T>;
}

```

We factor out the axioms just to show that we can, and because they are examples of axioms that might find multiple uses:

```

template<typename T>
axiom Greater(T x, T y) {
  (x>y) == (y<x);
}
template<typename T>
axiom Less_equal(T x, T y) {
  (x<=y) == !(y<x);
}
template<typename T>
axiom Greater_equal(T x, T y) {
  (x>=y) == !(x<y);
}

```

## 2. Metaprogrammierung

Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime. In some cases, this allows programmers to minimize the number of lines of code to express a solution (hence reducing development time), or it gives programs greater flexibility to efficiently handle new situations without recompilation. ...

aus: [Metaprogramming](#)

```
promote_trait <.,.>::T_promote  
  
numeric_limits <.>::max()  
numeric_limits <.>::is_integer  
  
iterator_traits <.>::value_type
```

### 2.1. Metafunktionen

Metafunktionen (Metaprogrammierung) sind uns schon an diversen Stellen begegnet:

```
promote_trait<T1,T2>::T_promote in  
template <typename T1, typename T2>  
    typename promote_trait<T1,T2>::T_promote my_function(T1 x, T2 y)  
//  
{  
    \\...  
}
```

stellt den mit T1, T2 assoziierten Typ T\_promote bereit (vergleiche Abschnitt 1.16.4).

```
cout << "Minimum value for int: " << numeric_limits<int>::min() << endl;  
//  
cout << "Maximum value for int: " << numeric_limits<int>::max() << endl;  
//  
cout << "int is signed: " << numeric_limits<int>::is_signed << endl;  
//  
cout << "Non-sign bits in int: " << numeric_limits<int>::digits << endl;  
//  
cout << "int has infinity: " << numeric_limits<int>::has_infinity << endl;  
//
```

druckt Eigenschaften der numerischen Typen mittels Typfunktionen aus. Siehe

```
static constexpr T min() noexcept;  
static constexpr T max() noexcept;  
static constexpr bool is_signed;  
static constexpr int digits;  
static constexpr bool has_infinity;
```

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

in [http://publib.boulder.ibm.com/infocenter/comphelp/v9v111/index.jsp?topic=/com.ibm.xlcpp9.aix.doc/stdlib/header\\_limits.htm](http://publib.boulder.ibm.com/infocenter/comphelp/v9v111/index.jsp?topic=/com.ibm.xlcpp9.aix.doc/stdlib/header_limits.htm).

```
template <class InputIterator>  
iterator_traits<InputIterator>::value_type  
last_value(InputIterator first, InputIterator last) {
```

```
//...  
}
```

greift auf den `InputIterator` assoziierten `value_type` zu, siehe [http://www.sgi.com/tech/stl/iterator\\_traits.html](http://www.sgi.com/tech/stl/iterator_traits.html).

„A metafunction is a construct to map some types or constants to other entities like types, constants, functions, or objects at **compile time**. The name metafunction comes from fact that they can be regarded as part of a meta-programming language that is evaluated during compilation.“ (<http://trac.seqan.de/wiki/Tutorial/Metafunctions>)

Run time, compile time vs. runtime, mit dem GCC zur ausführbaren Datei (Binary)

Implementiert wurden sie im geschichtlichen Verlauf durch **enum**, **static const**, `integral_constant` <> beziehungsweise `constexpr`-Konstruktionen:

```
                                                                    // vor C++03  
#include <iostream>  
  
template<int i>  
class fact {  
    public:  
        enum { result = i * fact<i-1>::result };  
};  
  
template<>class fact<1> {  
    public:  
        enum { result = 1 };  
};  
  
int main(){  
    std::cout << fact<5>::result << std::endl;  
}  
  
////////////////////////////////////  
  
#include <iostream>                // C++03  
  
template<int i>  
class fact {  
    public:  
        static const long long result = i * fact<i-1>::result;  
};  
  
template<>class fact<1> {  
    public:  
        static const long long result = 1;  
};  
  
int main(){  
    std::cout << fact<5>::result << std::endl;  
};
```

```

////////////////////////////////////
#include <iostream> // ab C++11
#include <type_traits>

template <unsigned n>
struct fact : std::integral_constant<int, n * fact<n-1>::value> {};

template <>
struct fact<0> : std::integral_constant<int,1> {};

int main() {
    std::cout << fact<5>::value; // constexpr (no calculations on runtime)
    return 0;
}

////////////////////////////////////
#include <iostream> // oder

constexpr long long fact(int value) {
    return (value==0) ? 1 : value * fact(value-1);
};

int main(){
    std::cout << fact(5) << std::endl;
    return 0;
}

```

## 2.2. Metafunktionen in `/usr/include/c++/4.7/type_traits` und Feldlängen

Metafunktion `integral_constant` in `type_traits`:

```

/// integral_constant
template<typename _Tp, _Tp __v>
struct integral_constant
{
    static constexpr _Tp value = __v;
    typedef _Tp value_type;
    typedef integral_constant<_Tp, __v> type;
    constexpr operator value_type() { return value; }
};

```

```

/// The type used as a compile-time boolean with true value.
typedef integral_constant<bool, true>    true_type;

/// The type used as a compile-time boolean with false value.
typedef integral_constant<bool, false>    false_type;

template<typename _Tp, _Tp --v>
    constexpr _Tp integral_constant<_Tp, --v>::value;
(http://publib.boulder.ibm.com/infocenter/comphelp/v9v111/index.jsp?topic=/com.ibm.xlcpp9.aix.doc/stdlib/header.type\_traits.htm)

/// is_array
template<typename>
    struct is_array
    : public false_type { };

template<typename _Tp, std::size_t _Size>
    struct is_array<_Tp[_Size]>
    : public true_type { };

template<typename _Tp>
    struct is_array<_Tp[]>
    : public true_type { };

////////////////////////////////////

/// is_unsigned
template<typename _Tp>
    struct is_unsigned
    : public __and_<is_arithmetic<_Tp>, __not_<is_signed<_Tp>>>::type
    { };

////////////////////////////////////

/// Define a member typedef @c type only if a boolean constant is true.
template<bool, typename _Tp = void>
    struct enable_if
    { };

// Partial specialization for true.
template<typename _Tp>
    struct enable_if<true, _Tp>
    { typedef _Tp type; };

////////////////////////////////////

```

Benutzung von Nontype-Templateparametern für die Länge von Containern:

```
#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 i = 0; i < 10; i++)
        x[i] = 1;

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

    return(0);
}
```

Nontype template parameter

What does template <unsigned int N> mean?

## 2.3. Factorial, Combinations, IF, id, add und die Rekursion statt der Schleife

Template-Metaprogrammierung:

```
template <int N>
struct Factorial
{
    enum { value = N * Factorial<N - 1>::value };
};

template <>
struct Factorial<0>
{
    enum { value = 1 };
};

// Factorial<4>::value == 24
// Factorial<0>::value == 1
void foo()
{
    int x = Factorial<4>::value; // == 24
    int y = Factorial<0>::value; // == 1
}

//
//  $C(k, n) = \frac{n!}{k! (n-k)!}$ 
//

template <int k, int n>
struct Combinations
{
    enum { RET = Factorial<n>::RET / (Factorial<k>::RET * Factorial<n-k>::RET) };
};

cout << Combinations<2,4>::RET << endl;
```

Templates mit zwei non-type int-Parametern statt eines nicht erlaubten non-type float-Parameters:  
[C++ Template Metaprogramming](#).



Bedingte Typauswahl:

```
// IF
template <bool condition, class Then, class Else>
struct IF
{
    typedef Then RET;
};

template <class Then, class Else>
struct IF<false, Then, Else>
{
    typedef Else RET;
};

// if sizeof(int) < sizeof(long) then use long else use int
IF< sizeof(int)<sizeof(long), long, int >::RET i;
```

C++-Metaprogrammierung:

```
template <unsigned int x>
struct id
{
    enum { value = x };
};

template <unsigned int x, unsigned int y>
struct add
{
    enum { value = x + y };
};

//...

template <int x>
struct id_static
{
    static const int value = x;
};

template <int x>
struct id_static_constexpr
{
    static constexpr int value = x;
};
```

## 2.4. Rechnende Compiler:

Erwin Unruh: Die Entdeckung des rechnenden Compilers:

```
// Erwin Unruh, untitled program,
// ANSI X3J16-94-0075/ISO WG21-462, 1994.

template <int i>
struct D
{
    D(void *);
    operator int ();
};

template <int p, int i>
struct is_prime
{
    enum { prim = (p%i) && is_prime<(i>2?p:0), i>::prim };
};

template <int i>
struct Prime_print
{
    Prime_print<i-1> a;
    enum { prim = is_prime<i, i-1>::prim };
    void f() { D<i> d = prim; }
};

struct is_prime<0,0> { enum { prim = 1 }; };
struct is_prime<0,1> { enum { prim = 1 }; };
struct Prime_print<2>
{
    enum { prim = 1 };
    void f() { D<2> d = prim; }
};

void foo ()
{
    Prime_print<10> a;
}

// output:
// unruh.cpp 30: conversion from enum to D<2> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<3> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<5> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<7> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<11> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<13> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<17> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<19> requested in Prime_print

constexpr is_prime_recursive
```

## 2.5. Typfunktionen: längerer Datentyp, IfThenElse-Werte

Typfunktionen liefern anstelle eines Datenwertes (einer Konstanten) einen oder mehrere Datentypen *oder* sind von Datentypen abhängig:

```
template <int bits>
struct number_type
{
    typedef int type;
};
template <>
struct number_type <16>
{
    typedef short type;
};
template <>
struct number_type <8>
{
    typedef char type;
};
template <typename arg>
struct bitsize
{
    enum { value = sizeof(arg) * 8 };
};
template <typename arg>
struct bigger_type
{
    typedef typename number_type<bitsize<arg>::value * 2 >::type type;
};
```

Bedingte Werte:

```
template <bool cond , int true_part , int false_part >
struct IfThenElse ;
template <int true_part , int false_part >
struct IfThenElse <true , true_part , false_part >
{
    enum { value = true_part };
};
template <int true_part , int false_part >
struct IfThenElse <false , true_part , false_part >
{
    enum { value = false_part };
};
```

## 2.6. Template Nontype Parameter

Abschnitt 14.3.2: Template non-type arguments

Abschnitt 5.19: Constant expressions

Non-type template parameters

What are the requirements for C++ template parameters?

C++: Why can't I use float value as a template parameter?

template+floating point non type

How can I use a floating-point value as a non-type template parameter?

Why are floating point types invalid template parameter types for template functions?

c++ template parameter is ambiguous

Workaround:

Floating point arithmetic in C++ templates.

Beachte dabei auch:

ldexp() und frexp().

ieee754.h und der Zugriff auf Vorzeichen, Exponent und Mantisse:

```
#include <iostream>
#include <cmath>
#include <ieee754.h>

int main()
{
    float num = 4.0 * atan(1.0); // PI

    ieee754_float ft;
    ft.f = num;

    std::cout << ft.ieee.negative << std::endl;
    std::cout << ft.ieee.exponent << std::endl;
    std::cout << ft.ieee.mantissa << std::endl;

    // prints:
    // 0
    // 128
    // 4788187

    ieee754_float ft2;
    ft2.ieee.negative = ft.ieee.negative;
    ft2.ieee.exponent = ft.ieee.exponent;
    ft2.ieee.mantissa = ft.ieee.mantissa;

    std::cout << ft2.f << std::endl;

    // prints:
    // 3.14159

    return 0;
}
```

## 2.7. Compilezeit-Fehlermeldungen in constexpr-Metafunktionen

Aus [Andrzej's C++ blog](#):

```
constexpr int factorial( int i )
{
    return ( i > 1 ) ? i * factorial(i - 1)
                   : 1;
}
```

oder besser:

```
constexpr int safe_factorial( int i )
{
    return ( i < 0 ) ?          // error condition
               throw exception() // error reporting
               : factorial(i);  // real computation
}
```

Mittels einer mit selbstsprechendem Bezeichnernamen gewählten Hilfsmetafunktion:

```
constexpr int requires_nonnegative( int i )
{
    return ( i < 0 ) ? throw exception()
                   : i;
}
```

wird die Compiler-Fehlermeldung selbstdentifizierend:

```
constexpr int safe_factorial( int i )
{
    return requires_nonnegative(i),
           factorial(i);
}
```

Eine Klasse mit Compiletime Kontruktor:

```
class Date
{
    unsigned d;
    Month m;
    unsigned y;

public:
    constexpr Date( unsigned d, Month m, unsigned y );
    // other stuff ...
};

constexpr Date::Date( unsigned d, Month m, unsigned y )
: d( requires_goodDay(d, m, y) )
, m(m)
, y( requires_positive(y) )
{} // empty body
```

```
constexpr unsigned requires_goodDay( unsigned d, Month m, unsigned y )
{
    return (d == 0 || d > 31) ?      throw BadDayName(d)           :
           (is30day(m) && d == 31) ? throw BadDayOfMonth(d, m)    :
           (m == Feb && d >= 30) ?   throw BadDayOfMonth(d, m)    :
           (!isLeap(y) && m == Feb && d == 29) ? throw Bad29Feb(y) :
           d; // real return value
}
```

Vergleiche:

[BOOST::optional](#)

[constexpr-unions](#)

[utility class to represent optional objects \(Revision 2\)](#)

[Parsing strings at compile-time — Part I](#)

[Parsing strings at compile-time — Part II](#)

[User-defined literals — Part I](#)

[User-defined literals — Part II](#)

[User-defined literals — Part III](#)

## 2.8. Vor- und Nachteile der Metaprogrammierung

Vor- und Nachteile der Template-Metaprogrammierung:

- **Längere Übersetzungszeit und kürzere Ausführungszeit:** Da der gesamte Template-Quelltext während der Übersetzung verarbeitet, ausgewertet und eingesetzt wird, dauert die Übersetzung insgesamt länger, während der ausführbare Code dadurch an Effizienz gewinnen kann. Obwohl dieser Zusatzaufwand im Allgemeinen sehr gering ausfällt, kann er auf große Projekte oder Projekte, in denen intensiv Templates eingesetzt werden, großen Einfluss auf die Dauer der Übersetzung besitzen.
- **Kürzerer Quelltext:** Templatemetaprogrammierung erlaubt es dem Programmierer, sich auf die Architektur des Programms zu konzentrieren und dem Compiler die Erzeugung von jeglichen Implementierungen, die vom aufrufenden Quelltext benötigt werden, zu überlassen. Daher kann Templatemetaprogrammierung zu kürzerem Quelltext und erhöhter Wartbarkeit führen.
- **Schlechtere Lesbarkeit:** Verglichen mit konventioneller C++-Programmierung wirken Syntax und Schreibweisen der Templatemetaprogrammierung ungewohnt. Fortgeschrittene oder sogar die meiste nicht-triviale Templatemetaprogrammierung kann daher schwer zu verstehen sein. Dadurch können Metaprogramme von Programmierern, die in Templatemetaprogrammierung unerfahren sind, schwer zu pflegen sein. Letzteres hängt allerdings auch davon ab, wie die Templatemetaprogrammierung im speziellen Fall umgesetzt wurde.
- **Geringere Portierbarkeit:** Die Portierbarkeit von Quelltext, der von Template-Metaprogrammierung starken Gebrauch macht, kann auf Grund von Unterschieden zwischen den verschiedenen Compilern eingeschränkt sein.
- **Ungewohnter Programmierstil:** Durch die rein-funktionale Struktur der Templates wären zwar theoretisch Optimierungen wie etwa in Haskell (Glasgow Haskell Compiler) möglich, praktisch werden solche Vorteile jedoch von keinem Compiler ausgenutzt und statt dessen verursacht diese Struktur in erster Linie (insbesondere für Programmierer, die strukturierte Programmierung aus C++ gewohnt sind) schwer verständlichen Code.
- **Schlechte Fehlermeldungen und schlechte Debuggbarkeit**

## 2.9. C++11 Metaprogramming Examples

```
// Here are a few tricks I've used with the trunk versions of clang and libc++
// with C++11 compilation turned on. Some might be obvious, some not, but at
// least they are some kind of improvement over their C++03 counterparts.
//
// Public domain.
//
// =====
// 1) Using variadic class templates recursively, like in the definitions for
// "add<T...>" here:
```

```
#include <type_traits>
```

```
// A few convenience aliases first
template <typename T, T N> using ic = std::integral_constant<T, N>;
template <int N> using int_ = std::integral_constant<int, N>;
```

```
// Sum any number of integral constants:
template <typename... Args> struct add;
template <>
struct add<>
    : ic<int, 0> {};
```

```
template <typename A>
struct add<A>
    : ic<decltype(+A::value), A::value> {};
```

```
template <typename A, typename B, typename... More>
struct add<A, B, More...>
    : add<ic<decltype(A::value + B::value),
          A::value + B::value>, More...> {};
```

```
// — example —
```

```
add<>::value; // 0
add<int_<1>>::value; // 1
add<int_<1>, int_<2>>::value; // 3
add<int_<1>, int_<2>, int_<3>>::value; // 6
add<int_<1>, int_<2>, int_<3>, int_<4>>::value; // 10
// etc.
```

```
// =====
// 2a) With decltype, the "sizeof(yes-type)" trick is no longer needed for
// implementing traits. This one tests whether there is a type T::type
// defined:
```

```
using std::true_type;
using std::false_type;
```

```
namespace detail {
    template <typename T, typename Type=typename T::type>
    struct has_type_helper;

    template <typename T> true_type has_type_test(has_type_helper<T> *);
    template <typename T> false_type has_type_test(...);
}
```

```
template <typename T>
    struct has_type : decltype(detail::has_type_test<T>(nullptr)) {};
```

```
// — example —
```

```
has_type<int>::value; // false
has_type<std::is_integral<int>>::value; // true, said type is "bool"
has_type<std::integral_constant<int,1>>::value; // true, said type is "int"
```



```

// -----
// 2b) This trait tests whether T is an integral constant:
namespace detail {
    template <typename T, decltype(T::value)>
        struct integral_constant_helper;

    template <typename T> true_type integral_constant_test(
        integral_constant_helper<T,T::value> *);

    template <typename T> false_type integral_constant_test(...);
}

template <typename T>
    struct is_integral_constant
        : decltype(detail::integral_constant_test<T>(nullptr)) {};

// ----- example -----
is_integral_constant<int>::value; // false
is_integral_constant<std::is_integral<int>>::value; // true
is_integral_constant<std::integral_constant<int,1>>::value; // true
// =====

// 3) Selection of the first matching type from a list of cases (or "pattern
// matching", if you will):

template <typename... When> struct match;
template <> struct match<> { static constexpr bool value = false; };
template <typename When, typename... More> struct match<When, More...>
    : std::conditional<When::value, When, match<More...>>::type {};

// 'match' is meant to be used together with 'when', 'otherwise' and
// friends:

template <bool Cond, typename Then=void> struct when_c;
template <typename Then> struct when_c<true, Then> {
    typedef Then type;
    static constexpr bool value = true;
};

template <typename Then> struct when_c<false, Then> {
};

template <bool Cond, typename Then=void>
    struct when_not_c : when_c<!Cond, Then> {};

template <typename Cond, typename Then=void>
    struct when : when_c<Cond::value, Then> {};

template <typename Cond, typename Then=void>
    struct when_not : when_not_c<Cond::value, Then> {};

template <typename Then> struct otherwise {
    typedef Then type;
    static constexpr bool value = true;
};

// ----- example -----
struct fizz {};
struct buzz {};
struct fizzbuzz {};

```

```

template <int N>
    struct game : match<
        when_c<N % 3 == 0 && N & 5 == 0, fizzbuzz>,
        when_c<N % 3 == 0, fizz>,
        when_c<N % 5 == 0, buzz>,
        otherwise< int_c<N>>
    > {};

game<1>::type; // int_<1>
game<2>::type; // int_<2>
game<3>::type; // fizz
game<4>::type; // int_<4>
game<5>::type; // buzz
game<6>::type; // fizz
game<7>::type; // int_<7>
game<8>::type; // int_<8>
game<9>::type; // fizz
game<10>::type; // buzz
game<11>::type; // int_<11>
game<12>::type; // fizz
game<13>::type; // int_<13>
game<14>::type; // int_<14>
game<15>::type; // fizzbuzz
game<16>::type; // int_<16>

// =====
// 4a) Variadic template template parameters. For instance,
// boost::mpl::quoteN<...> can be reimplemented with just:

template <template <typename...> class F>
    struct quote {
        template <typename... Args> struct apply : F<Args...> {};
    };

// =====
// 4b) Here's another use for variadic template template parameters. Of course,
// the standard library offers std::tuple_size<T> for getting the number of
// elements in a tuple. But that metafunction cannot be used for any other
// tuple-like class. Suppose we defined boost::mpl::vector like:

template <typename... T>
    struct vector {};

// By using a variadic template template, we can define a metafunction which
// works equally for both std::tuple<T...> as well as vector<T...>:

template <typename T>
    struct size {}; // (no size defined by default)

template <template <typename...> class C, typename... T>
    struct size<C<T...>> : ic<std::size_t, sizeof...(T)> {};

template <typename T> struct size<T &&> : size<T> {};
template <typename T> struct size<T &&&> : size<T> {};
template <typename T> struct size<T const> : size<T> {};
template <typename T> struct size<T volatile> : size<T> {};
template <typename T> struct size<T const volatile> : size<T> {};

// — example —
size<tuple<int, int> &&>::value; // 2
size<vector<int, int, int>>::value; // 3
size<vector<> const &&>::value; // 0

// =====
// 5) Using nested variadic templates to get many template parameter packs to

```

```

// play with:
namespace detail {
    template <typename A> struct con;
    template <typename... T> struct con<vector<T...>> {
        template <typename B> struct cat;
        template <typename... U> struct cat<vector<U...>> {
            typedef vector<T..., U...> type;
        };
    };
}

template <typename A, typename B>
    struct concat : detail::con<A>::template cat<B> {};

// — example —————

struct a; struct b; struct c; struct d; struct e;
concat<vector<a, b>, vector<c, d, e>>::type; // vector<a, b, c, d, e>

// =====
// 6) Defining function result and result type at once.
#define RETURNS(...) decltype((--VA_ARGS--)) { return (--VA_ARGS--); }

// — example —————

template <typename A, typename B>
auto plus(A const & a, B const & b) -> RETURNS(a + b)

// It can't be used with recursive definitions like here, though:

struct mul_ {
    int operator()() const { return 1; }

    template <typename A>
    A operator()(A const & a) const { return a; }

    // template <typename A, typename B, typename... C>
    // auto operator()(A const & a, B const & b, C const &... c) const ->
    // RETURNS(mul_()(a * b, c...))

    // —> Error: invalid use of incomplete type mul_

    // std::declval helps, but duplicates the multiplication part:
    template <typename A, typename B, typename... C>
    auto operator()(A const & a, B const & b, C const &... c) const ->
    decltype(std::declval<mul_>()(a * b, c...)) {
        return mul_()(a * b, c...);
    }
};

constexpr mul_ mul = {}; // global function object

// — example —————

mul(); // 1
mul(10); // 10
mul(10, -20, 30.0); // -6000.0

// =====
// 7) Counted template recursion. The function "apply_tuple(f, t)" calls the
// function (function object) "f" with the elements of the tuple "t" as
// arguments. (To simplify things a bit, I omitted the perfect forwarding
// support in this example.)
//

```

```

// The count is tracked with a total number of iterations N, and the running
// index I. R is the precalculated result type.

namespace detail {
    template <typename R, std::size_t N, std::size_t I=0>
        struct apply_tuple {
            template <typename F, typename T, typename... Args>
            R operator()(F f, T const & t, Args const &... args) const {
                typedef apply_tuple<R, N, I + 1> next;
                return next()(f, t, args..., std::get<I>(t));
            }
        };

    template <typename R, std::size_t N> struct apply_tuple<R, N, N> {
        template <typename F, typename T, typename... Args>
        R operator()(F f, T const &, Args const &... args) const {
            return f(args...);
        }
    };
}

template <typename F, typename... T>
decltype(std::declval<F>()(std::declval<T const &>()...))
apply_tuple(F f, std::tuple<T...> const & t) {
    typedef decltype(std::declval<F>()(std::declval<T const &>()...))
        result;
    return detail::apply_tuple<result, sizeof...(T)>(f, t);
}

// — example —————
int f(int a, int b) { return a + b; }
apply_tuple(f, std::make_tuple(10, 20)); // 30

auto t = std::make_tuple(10, -20, 30.0);
apply_tuple(mul, t); // -6000.0

(aus C++11 metaprogramming)

```

## 2.10. Fortgeschrittene Metaprogrammierung

### 2.10.1. Domain specific language extensions: C++11 Compile-time rational arithmetic

Bruch-Arithmetik mit Zähler/Nenner aus `intmax_t`.

```
namespace std {
// 20.10.3, class template ratio
template <intmax_t N, intmax_t D = 1> class ratio;
// 20.10.4, ratio arithmetic
template <class R1, class R2> using ratio_add = see below;
template <class R1, class R2> using ratio_subtract = see below;
template <class R1, class R2> using ratio_multiply = see below;
template <class R1, class R2> using ratio_divide = see below;
// 20.10.5, ratio comparison
template <class R1, class R2> struct ratio_equal;
template <class R1, class R2> struct ratio_not_equal;
template <class R1, class R2> struct ratio_less;
template <class R1, class R2> struct ratio_less_equal;
template <class R1, class R2> struct ratio_greater;
template <class R1, class R2> struct ratio_greater_equal;
// 20.10.6, convenience SI typedefs
typedef ratio<1, 1000000000000000000000000000> yocto; // see below
typedef ratio<1, 1000000000000000000000000> zepto; // see below
typedef ratio<1, 100000000000000000000000> atto;
typedef ratio<1, 10000000000000000000000> femto;
typedef ratio<1, 1000000000000000000000> pico;
typedef ratio<1, 100000000000000000000> nano;
typedef ratio<1, 10000000> micro;
typedef ratio<1, 1000> milli;
typedef ratio<1, 100> centi;
typedef ratio<1, 10> deci;
typedef ratio<10, 1> deca;
typedef ratio<100, 1> hecto;
typedef ratio<1000, 1> kilo;
typedef ratio<1000000, 1> mega;
typedef ratio<1000000000, 1> giga;
typedef ratio<1000000000000, 1> tera;
typedef ratio<1000000000000000, 1> peta;
typedef ratio<1000000000000000000, 1> exa;
typedef ratio<1000000000000000000000, 1> zetta; // see below
typedef ratio<1000000000000000000000000, 1> yotta; // see below
}
```

```

namespace std {
template <intmax_t N, intmax_t D = 1>
class ratio {
public:
typedef ratio<num, den> type;
static constexpr intmax_t num;
static constexpr intmax_t den;
};
}

static_assert(ratio_add<ratio<1,3>, ratio<1,6>>::num == 1, "1/3+1/6
== 1/2");
static_assert(ratio_add<ratio<1,3>, ratio<1,6>>::den == 2, "1/3+1/6
== 1/2");
static_assert(ratio_multiply<ratio<1,3>, ratio<3,2>>::num == 1, "
1/3*3/2 == 1/2");
static_assert(ratio_multiply<ratio<1,3>, ratio<3,2>>::den == 2, "
1/3*3/2 == 1/2");

```

## 2.10.2. Unrolled Loops: Durch Rekursion wegoptimierte Schleifen

Schleifen ohne Verwaltungsoverhead:

Tailrekursion statt Iteration

Ausgangspunkt (laufzeit-iterativ):

```
template <typename T>
inline T dot_product (T* a, T* b, int dim)
{
    T result = T();
    for (int i = 0; i < dim; i++)
    {
        result += a[i] * b[i];
    }
    return result;
}
```

Optimiert (Compilezeit-tailrekursive Metafunktion)

```
template <int N, typename T>
struct dotproduct_s
{
    static T result (T* a, T* b)
    {
        return (*a)*(*b) + dotproduct_s<N-1,T>::result(a+1, b+1);
    };
};
template <typename T>
struct dotproduct_s<1, T>
{
    static T result (T* a, T* b)
    {
        return (*a)*(*b);
    };
};
template <int N, typename T>
inline T dotproduct(T* a, T* b)
{
    return dotproduct_s<N, T>::result(a, b);
}
int main ()
{
    int a [3] = {1, 2, 3};
    int b [3] = {4, 5, 6};
    std :: cout << dot_product(a, b, 3) << std :: endl ;
    std :: cout << dotproduct<3>(a, b) << std :: endl ;
    return 0;
}
```

Codedisassembly:

Listing: Mit Schleife (dot\_product)

```
push ebp
mov ebp , esp
push edi
push esi
```

```

push ebx
mov edi , DWORD PTR [ ebp +8]
mov esi , DWORD PTR [ ebp +12]
mov ebx , DWORD PTR [ ebp +16]
mov ecx , 0
mov edx , 0
cmp ecx , ebx
jge L32
L30 :
mov eax , DWORD PTR [ edi + edx *4]
imul eax , DWORD PTR [esi +edx *4]
add ecx , eax
inc edx
cmp edx , ebx
jl L30
L32 :
mov eax , ecx
pop ebx
pop esi
pop edi
pop ebp
ret

```

Listing: Ohne Schleife (dotproduct\_s)

```

push ebp
mov ebp , esp
push ebx
mov edx , DWORD PTR [ ebp +8]
mov ebx , DWORD PTR [ ebp +12]
mov eax , DWORD PTR [ edx ]
imul eax , DWORD PTR [ebx ]
mov ecx , DWORD PTR [ edx +4]
imul ecx , DWORD PTR [ebx +4]
mov edx , DWORD PTR [ edx +8]
imul edx , DWORD PTR [ebx +8]
add ecx , edx
add eax , ecx
pop ebx
pop ebp
ret

```

Acht Anweisungen statt 18 Anweisungen.  
(Metaprogrammierung Seite 45ff.)

Endrekursion  
Tailrekursion



A generic loop unroller based on template meta-programming

C++11 anonyme Funktion

## 2.10.3. Expression templates

### C++ Expression templates:

Expression templates are a category of C++ template meta programming which delays evaluation of subexpressions until the full expression is known, so that optimizations (especially the elimination of temporaries) can be applied. (lazy evaluation)

Zum Beispiel:

statt `x = a + b + c` der Aufruf von `Expression<Expression<Array,plus,Array>,plus,Array>` mit

```
struct plus {
    static int apply(int a, int b) {
        return (a + b); }
};

template < typename L, typename OpTag, typename R >
struct Expression {
    Expression(L const& l, R const& r) : l(l), r(r) {}
    int operator [] (unsigned index) const {
        return OpTag::apply(l[index], r[index]);
    }
    L const& l;
    R const& r;
};

template< typename L, typename R >
Expression<L, plus, R> operator+(L const& l, R const& r) {
    return Expression<L, plus, R>(l, r);
}
...
// verzogerte Ausdrucksauswertung (lazy evaluation)
// — sofort wird nur der Parsbaum aufgebaut —
// bis zur Aktivierung von operator=
template<typename Expr> {
Array& Array::operator=(Expr const& x) {
    for(unsigned i = 0; i < this->size(); ++i) {
        (*this)[i] = x[i];
    }
    return (*this);
}
}
(Metaprogrammierung)
```

Matrixdimension	Zeit (nicht opt.)[s]	Zeit (opt.)[s]
100x100	1.776	1.059
200x200	7.204	4.237
300x300	17.091	9.796
400x400	30.305	18.087
500x500	46.842	28.154
600x600	85.316	54.125
700x700	114.167	73.604

Dr.Dobbs: Expression Templates  
A. Langer: Expression Templates  
Blitz++

Lazy evaluation  
Lazy Evaluation in C++11  
Expression templates and C++11

## 2.11. Nachteile der Metaprogrammierung (Forts.)

Metaprogrammierung-Nachteile Seite 52

Metaprogrammierung-Nachteile Seite 17

## 2.12. Die BOOST Metaprogramming Library MPL

Will man selbst Metafunktionsbibliotheken schreiben sollte man die MPL nutzen:

[MPL](#)

## 2.13. Metaprogramme für die Manipulation von Typen in C++

[Typelists](#)

Metafunktionen für Container (Sequenzen, Listen, ...) von Typen:

```
template<class List1 , class List2>
struct TypeListAppend
{
    typedef TypeList<typename List1::Head, typename TypeListAppend<
        typename List1::Tail , List2 >::Result> Result;
};
template<class List2>
struct TypeListAppend<NullType , List2>
{
    typedef List2 Result;
};
// Auf die Implementierung von TypeListBeforePivot und TypeListAfterPivot
// soll hier verzichtet werden
template<class List , template<typename A, typename B> class Comparator>
struct TypeListSort
{
    typedef typename TypeListAppend<
        typename TypeListSort<
            typename TypeListBeforePivot<
                typename List::Tail ,
                typename List::Head,
                Comparator >::Result ,
            Comparator >::Result ,
        TypeList<
            typename List::Head,
            typename TypeListSort<
                typename TypeListAfterPivot<
                    typename List::Tail ,
                    typename List::Head,
                    Comparator >::Result ,
                Comparator >::Result
            >
        >::Result Result;
};
```

Funktionale Programmierung  
Lisp  
Scheme  
Scala

## 2.14. Spracherweiterung Maßeinheiten

### 2.14.1. Eine Softwarekatastrophe und ihr Einfluß auf neue Programmiersprachen

Anlaß: 1999 verpasste die **NASA-Sonde Mars Climate Orbiter** den Landeanflug auf den Mars, weil die Programmierer das falsche Maßsystem verwendeten - Pfund x Sekunde statt Newton x Sekunde. Die NASA verlor dadurch die Sonde.

Einheiten können durch geeignete Klassen (**Euro**, **Franken**, **Pfund**, ... statt **double**) mit (automatisch durchgeführten) Typkonversionen ähnlich wie in

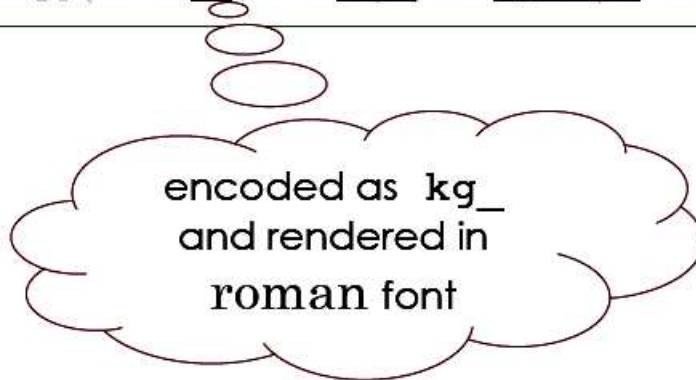
```
class Euro
{
private:
    double Wert;
public:
    Euro() : Wert(0.0) {};
    Euro(double w) : Wert(w) {};
    Euro(const Euro &e) : Wert(e.Wert) {};
    Euro(DM dw);
    double ZeigeWert() const { return Wert; };
    friend Euro operator+(Euro a, Euro b);
    friend Euro operator-(Euro a, Euro b);
    friend Euro operator*(Euro a, double d);
    friend Euro operator/(Euro a, double d);
    friend bool operator<(Euro a, Euro b);
    friend ostream& operator<<(ostream& os, const Euro& e);
};
Euro::Euro(DM dw)
{
    Wert = dw.ZeigeWert() / 1.95583;
}
```

realisiert werden.

Flexibler ist es jedoch, wenn die Programmiersprache Maßeinheiten unterstützt:

- Units und Dimensions in Fortress:

*kineticEnergy*(*m* :  $\mathbb{R}$  kg, *v* :  $\mathbb{R}$  m/s) :  $\mathbb{R}$  kg m<sup>2</sup>/s<sup>2</sup> = (*m v*<sup>2</sup>)/2



<code>m_</code>	is rendered as	<code>m</code>	<code>s_</code>	is rendered as	<code>s</code>
<code>km_</code>	is rendered as	<code>km</code>	<code>kg_</code>	is rendered as	<code>kg</code>
<code>v_</code>	is rendered as	<code>V</code>	<code>kW_</code>	is rendered as	<code>kW</code>
<code>_v</code>	is rendered as	<code>v</code>	<code>_foo13</code>	is rendered as	<code>foo<sub>13</sub></code>

*v* :  $\mathbb{R}$  m/s = (3 meters + 4 meters)/5 seconds

Correct

*v* :  $\mathbb{R}$  m/s = (3 meters + 4 seconds)/5 seconds

static error

*v* :  $\mathbb{R}$  m/s = (3 meters + 4 meters)/5

static error

*kineticEnergy*(3.14 kg, 32 f/s in m/s)



- Units und Dimensions in der Programmiersprache F#

```
let gravityOnEarth = 9.81<m/s^2> // Beschleunigung
let heightOfDrop   = 3.5<m>      // Laenge
let speedOfImpact  = sqrt(2.0 * gravityOnEart * heightOfDrop)
```

C++11 bleibt leider bei den SI-Skalierfaktoren

```
// 20.10.6, convenience SI typedefs
typedef ratio<1, 1000000000000000000000000> yocto; // see below
typedef ratio<1, 1000000000000000000000000> zepto; // see below
typedef ratio<1, 1000000000000000000000000> atto;
typedef ratio<1, 1000000000000000000000000> femto;
typedef ratio<1, 1000000000000000000000000> pico;
typedef ratio<1, 1000000000000000000000000> nano;
typedef ratio<1, 1000000000000000000000000> micro;
typedef ratio<1, 1000000000000000000000000> milli;
typedef ratio<1, 1000000000000000000000000> centi;
typedef ratio<1, 1000000000000000000000000> deci;
typedef ratio<10, 1> deca;
typedef ratio<100, 1> hecto;
typedef ratio<1000, 1> kilo;
typedef ratio<1000000, 1> mega;
typedef ratio<1000000000, 1> giga;
typedef ratio<1000000000000, 1> tera;
typedef ratio<1000000000000000, 1> peta;
typedef ratio<1000000000000000000, 1> exa;
typedef ratio<1000000000000000000000000, 1> zetta; // see below
typedef ratio<1000000000000000000000000000, 1> yotta; // see below
```

stehen, was die Unterstützung von Maßeinheiten angeht.

Kann uns da Metaprogrammierung helfen?



## 2.14.2. DSLs

In software development and domain engineering, a **domain-specific language (DSL)** is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique. The concept isn't new—special-purpose programming languages and all kinds of modeling/specification languages have always existed, but the term has become more popular due to the rise of domain-specific modeling.

Scala DSLs

DSLs

C++ template metaprogramming for DSLs

## 2.14.3. Ausflug in die Domain des technische-wissenschaftlichen Rechnens: Units and Measure in F#

Introducing Units:

```
[<Measure>] type kg
[<Measure>] type m
[<Measure>] type s
```

```
let gravityOnEarth = 9.81<m/s^2>
let heightOfMyOfficeWindow = 3.5<m>
```

```
let speedOfImpact = sqrt(2.0 * gravityOnEarth * heightOfMyOfficeWindow)
...
```

speedOfImpact hat die Einheit <m/s>.

Der Fehler

```
let speedOfImpact = sqrt(2.0 * gravityOnEarth + heightOfMyOfficeWindow)
...
```

führt zur Compiler-Fehlermeldung

The unit measure 'm' does not match the unit measure 'm/s^2'.

Unit Conversions:

```
...
let heightOfMyOfficeWindow = 11.5<ft>
let FeetPerMetre = 3.28084<ft/m>
...
let heightOfMyOfficeWindowInMetres = heightOfMyOfficeWindow /
    FeetPerMetre
...
type float = float<1>
...
```

Generic Units  
Parameterized Types

## 2.14.4. SI-Einheitssystem

Internationales Einheitensystem

Dezimale Vielfache und Teile der SI-Einheiten: Seite 23

Sieben Dimensionen von Meßgrößen: Seite 9

Basiseinheiten, abgeleitete Einheiten, ...: Seite 18, 19, 20, 21, 24, 27, 29ff.

## 2.14.5. Boost.Units

Automatische Einheiten-Dimensionsrechnung in C++

```
#include <complex>
#include <iostream>

#include <boost/typeof/std/complex.hpp>

#include <boost/units/systems/si/energy.hpp>
#include <boost/units/systems/si/force.hpp>
#include <boost/units/systems/si/length.hpp>
#include <boost/units/systems/si/electric_potential.hpp>
#include <boost/units/systems/si/current.hpp>
#include <boost/units/systems/si/resistance.hpp>
#include <boost/units/systems/si/io.hpp>

using namespace boost::units;
using namespace boost::units::si;

quantity<energy>
work(const quantity<force>& F, const quantity<length>& dx)
{
    return F * dx; // Defines the relation: work = force * distance.
}

int main()
{
    /// Test calculation of work.
    quantity<force>      F(2.0 * newton); // Define a quantity of force.
    quantity<length>     dx(2.0 * meter); // and a distance,
    quantity<energy>     E(work(F,dx));  // and calculate the work done.

    std::cout << "F = " << F << std::endl
               << "dx = " << dx << std::endl
               << "E = " << E << std::endl
               << std::endl;

    /// Test and check complex quantities.
```

```

typedef std::complex<double> complex_type; // double real and
    imaginary parts.

// Define some complex electrical quantities.
quantity<electric_potential, complex_type> v = complex_type(12.5,
    0.0) * volts;
quantity<current, complex_type> i = complex_type(3.0,
    4.0) * amperes;
quantity<resistance, complex_type> z = complex_type(1.5,
    -2.0) * ohms;

std::cout << "V = " << v << std::endl
    << "I = " << i << std::endl
    << "Z = " << z << std::endl
    // Calculate from Ohm's law voltage = current *
    resistance.
    << "I * Z = " << i * z << std::endl
    // Check defined V is equal to calculated.
    << "I * Z == V? " << std::boolalpha << (i * z == v) <<
    std::endl
    << std::endl;
return 0;
}

```

produziert folgende Ausgabe:

```

F = 2 N
dx = 2 m
E = 4 J

V = (12.5,0) V
I = (3,4) A
Z = (1.5,-2) Ohm
I*Z = (12.5,0) V
I*Z == V? true

```

Boost.Units benutzt **Metafunktionen**, um quantity<.>-Werte mit automatischer Dimensionsanalyse zu ermöglichen:

```

quantity<length> L = 2.0*meters; // quantity of
    length
quantity<time> E = 14.5*seconds; // quantity of
    time
// mit:
// template<class Unit, class Y = double> class quantity;
//

```

**Conversions**

Pool von vordefinierten Konstanten  
alphabetische Liste der Grundeinheiten

Meßgenauigkeiten und Fehlerfortpflanzung:

```
quantity<length,measurement<double> >  
    u(measurement<double>(1.0,0.0)*meters),  
    w(measurement<double>(4.52,0.02)*meters),  
    x(measurement<double>(2.0,0.2)*meters),  
    y(measurement<double>(3.0,0.6)*meters);
```

mit den Ergebniswerten (Fehlerbalken):

```
x+y-w      = 0.48(+/-0.632772) m  
w*x        = 9.04(+/-0.904885) m^2  
x/y        = 0.666667(+/-0.149071) dimensionless
```

...

```
w*y^2/(u*x)^2 = 10.17(+/-3.52328) m^-1  
w/(u*x)^(1/2) = 3.19612(+/-0.160431) dimensionless
```

Dabei wurde die folgende Benutzererweiterung verwendet:

```
// Boost.Units - A C++ library for zero-overhead dimensional  
// analysis and  
// unit/quantity manipulation and conversion  
//  
// Copyright (C) 2003-2008 Matthias Christian Schabel  
// Copyright (C) 2008 Steven Watanabe  
//  
// Distributed under the Boost Software License, Version 1.0. (  
// See  
// accompanying file LICENSE_1_0.txt or copy at  
// http://www.boost.org/LICENSE_1_0.txt)
```

```
#ifndef BOOST_UNITS_MEASUREMENT_HPP  
#define BOOST_UNITS_MEASUREMENT_HPP  
  
#include <cmath>  
#include <cstdlib>  
#include <iomanip>  
#include <iostream>  
  
#include <boost/io/ios_state.hpp>  
#include <boost/units/static_rational.hpp>  
  
namespace boost {
```

```

namespace units {

namespace sqr_namespace /**/ {

template<class Y>
Y sqr(Y val)
{ return val*val; }

} // namespace

using sqr_namespace::sqr;

template<class Y>
class measurement
{
public:
    typedef measurement<Y>          this_type;
    typedef Y                       value_type;

    measurement(const value_type& val = value_type(),
                const value_type& err = value_type()) :
        value_(val),
        uncertainty_(std::abs(err))
    { }

    measurement(const this_type& source) :
        value_(source.value_),
        uncertainty_(source.uncertainty_)
    { }

    //~measurement() { }

    this_type& operator=(const this_type& source)
    {
        if (this == &source) return *this;

        value_ = source.value_;
        uncertainty_ = source.uncertainty_;

        return *this;
    }

    operator value_type() const    { return value_; }

    value_type value() const       { return value_; }
    value_type uncertainty() const { return uncertainty_; }
}

```

```

value_type lower_bound() const { return value_ -
    uncertainty_; }
value_type upper_bound() const { return value_ +
    uncertainty_; }

this_type& operator+=(const value_type& val)
{
    value_ += val;
    return *this;
}

this_type& operator-=(const value_type& val)
{
    value_ -= val;
    return *this;
}

this_type& operator*=(const value_type& val)
{
    value_ *= val;
    uncertainty_ *= val;
    return *this;
}

this_type& operator/=(const value_type& val)
{
    value_ /= val;
    uncertainty_ /= val;
    return *this;
}

this_type& operator+=(const this_type& /*source*/);
this_type& operator-=(const this_type& /*source*/);
this_type& operator*=(const this_type& /*source*/);
this_type& operator/=(const this_type& /*source*/);

private:
    value_type          value_,
                      uncertainty_;
};

}

}

#if BOOST_UNITS_HAS_BOOST_TYPEOF

```

```

BOOST_TYPEOF_REGISTER_TEMPLATE(boost::units::measurement, 1)

#endif

namespace boost {

namespace units {

template<class Y>
inline
measurement<Y>&
measurement<Y>::operator+=(const this_type& source)
{
    uncertainty_ = std::sqrt(sqr(uncertainty_)+sqr(source.
        uncertainty_));
    value_ += source.value_;

    return *this;
}

template<class Y>
inline
measurement<Y>&
measurement<Y>::operator-=(const this_type& source)
{
    uncertainty_ = std::sqrt(sqr(uncertainty_)+sqr(source.
        uncertainty_));
    value_ -= source.value_;

    return *this;
}

template<class Y>
inline
measurement<Y>&
measurement<Y>::operator*=(const this_type& source)
{
    uncertainty_ = (value_*source.value_)*
        std::sqrt(sqr(uncertainty_/value_)+
            sqr(source.uncertainty_/source.value_));
    value_ *= source.value_;

    return *this;
}

```

```

template<class Y>
inline
measurement<Y>&
measurement<Y>::operator/=(const this_type& source)
{
    uncertainty_ = (value_/source.value_)*
        std::sqrt(sqr(uncertainty_/value_)+
            sqr(source.uncertainty_/source.value_));
    value_ /= source.value_;

    return *this;
}

// value_type op measurement
template<class Y>
inline
measurement<Y>
operator+(Y lhs, const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs, Y(0))+=rhs);
}

template<class Y>
inline
measurement<Y>
operator-(Y lhs, const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs, Y(0))-=rhs);
}

template<class Y>
inline
measurement<Y>
operator*(Y lhs, const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs, Y(0))*=rhs);
}

template<class Y>
inline
measurement<Y>
operator/(Y lhs, const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs, Y(0))/=rhs);
}

```



```

// measurement op value_type
template<class Y>
inline
measurement<Y>
operator+(const measurement<Y>& lhs,Y rhs)
{
    return (measurement<Y>(lhs)+=measurement<Y>(rhs,Y(0)));
}

template<class Y>
inline
measurement<Y>
operator-(const measurement<Y>& lhs,Y rhs)
{
    return (measurement<Y>(lhs)-=measurement<Y>(rhs,Y(0)));
}

template<class Y>
inline
measurement<Y>
operator*(const measurement<Y>& lhs,Y rhs)
{
    return (measurement<Y>(lhs)*=measurement<Y>(rhs,Y(0)));
}

template<class Y>
inline
measurement<Y>
operator/(const measurement<Y>& lhs,Y rhs)
{
    return (measurement<Y>(lhs)/=measurement<Y>(rhs,Y(0)));
}

// measurement op measurement
template<class Y>
inline
measurement<Y>
operator+(const measurement<Y>& lhs,const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs)+=rhs);
}

template<class Y>
inline
measurement<Y>
operator-(const measurement<Y>& lhs,const measurement<Y>& rhs)

```

```

{
    return (measurement<Y>(lhs)--rhs);
}

template<class Y>
inline
measurement<Y>
operator*(const measurement<Y>& lhs,const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs)*=rhs);
}

template<class Y>
inline
measurement<Y>
operator/(const measurement<Y>& lhs,const measurement<Y>& rhs)
{
    return (measurement<Y>(lhs)/=rhs);
}

/// specialize power typeof helper
template<class Y,long N,long D>
struct power_typeof_helper<measurement<Y>,static_rational<N,D> >
{
    typedef measurement<
        typename power_typeof_helper<Y,static_rational<N,D> >::
        type
    > type;

    static type value(const measurement<Y>& x)
    {
        const static_rational<N,D> rat;

        const Y m = Y(rat.numerator())/Y(rat.denominator()),
            newval = std::pow(x.value(),m),
            err = newval*std::sqrt(std::pow(m*x.uncertainty
                ()/x.value(),2));

        return type(newval,err);
    }
};

/// specialize root typeof helper
template<class Y,long N,long D>
struct root_typeof_helper<measurement<Y>,static_rational<N,D> >
{

```

```

typedef measurement<
    typename root_typeof_helper<Y,static_rational<N,D> >::
        type
> type;

static type value(const measurement<Y>& x)
{
    const static_rational<N,D> rat;

    const Y m = Y(rat.denominator())/Y(rat.numerator()),
        newval = std::pow(x.value(),m),
        err = newval*std::sqrt(std::pow(m*x.uncertainty
            ()/x.value(),2));

    return type(newval,err);
}
};

// stream output
template<class Y>
inline
std::ostream& operator<<(std::ostream& os,const measurement<Y>&
    val)
{
    boost::io::ios_precision_saver precision_saver(os);
    boost::io::ios_flags_saver flags_saver(os);

    os << val.value() << "(+/-" << val.uncertainty() << ")";

    return os;
}

} // namespace units

} // namespace boost

#endif // BOOST_UNITS_MEASUREMENT_HPP

```

## 2.14.6. Erweiterung des C++-Typsystems um Units

... mit Hilfe der Boost MPL-Library (Implementierungs-idee der Boost.Units-Bibliothek):

**Dimensions:**

```
// die sieben Grundeinheiten:
typedef int dimension[7]; // m l t ...
dimension const mass      = {1, 0, 0, 0, 0, 0, 0};
dimension const length    = {0, 1, 0, 0, 0, 0, 0};
dimension const time      = {0, 0, 1, 0, 0, 0, 0};
...
// und die abgeleiteten Einheiten:
dimension const force     = {1, 1, -2, 0, 0, 0, 0};
...
```

Diese Dimensionen sind jedoch alle vom gleichen C++-Typ, führen also nicht zu den gewünschten Fehlermeldungen bei Dimensionsrechnungsabweichungen. Mit Hilfe des Datentyps `vector_c` der MPL ändert sich das:

```
#include <boost/mpl/vector_c.hpp>

typedef mpl::vector_c<int,1,0,0,0,0,0,0> mass;
typedef mpl::vector_c<int,0,1,0,0,0,0,0> length; // or position
typedef mpl::vector_c<int,0,0,1,0,0,0,0> time;
typedef mpl::vector_c<int,0,0,0,1,0,0,0> charge;
typedef mpl::vector_c<int,0,0,0,0,1,0,0> temperature;
typedef mpl::vector_c<int,0,0,0,0,0,1,0> intensity; // in cd
typedef mpl::vector_c<int,0,0,0,0,0,0,1> angle; // oder mol
// abgeleitete Einheiten:
typedef mpl::vector_c<int,0,1,-1,0,0,0,0> velocity; // l/t
typedef mpl::vector_c<int,0,1,-2,0,0,0,0> acceleration; // l/(t2)
typedef mpl::vector_c<int,1,1,-1,0,0,0,0> momentum; // ml/t
typedef mpl::vector_c<int,1,1,-2,0,0,0,0> force; // ml/(t2)
...
typedef mpl::vector_c<int,0,0,0,0,0,0,0> scalar;
...
```

### Quantities:

```
template <class T, class Dimensions>
struct quantity
{
    explicit quantity(T x)
        : m_value(x)
    {}

    T value() const { return m_value; }
private:
    T m_value;
};
```

...

```
quantity<float, length> l( 1.0f );
```

```
quantity<float, mass> m( 2.0f );
```

...

```
m = l; // compile-time type error
```

Quantity-Arithmetik (value und dimension):

**add/subtract:**

```
template <class T, class D>
quantity<T,D>
operator+(quantity<T,D> x, quantity<T,D> y)
{
    return quantity<T,D>(x.value() + y.value());
}
```

```
template <class T, class D>
quantity<T,D>
operator-(quantity<T,D> x, quantity<T,D> y)
{
    return quantity<T,D>(x.value() - y.value());
}
```

```
// ...
```

```
quantity<float, length> len1( 1.0f );
```

```
quantity<float, length> len2( 2.0f );
```

```
len1 = len1 + len2; // OK
```

```
len1 = len2 + quantity<float, mass>( 3.7f ); // error
```

**multiply:**

```
template <class T, class D1, class D2>
quantity<
    T
    , typename mpl::transform<D1,D2,plus_f>::type // new dimensions
>
operator*(quantity<T,D1> x, quantity<T,D2> y)
{
    typedef typename mpl::transform<D1,D2,plus_f>::type dim;
    return quantity<T,dim>( x.value() * y.value() );
}
// mit MPL-Hilfe:
template <class OtherDimensions>
quantity(quantity<T,OtherDimensions> const& rhs)
: m_value(rhs.value())
{
    BOOST_STATIC_ASSERT((
        mpl::equal<Dimensions,OtherDimensions>::type::value
    ));
}
}
```

**divide:**

```
template <class T, class D1, class D2>
quantity<
    T
    , typename mpl::transform<D1,D2,mpl::minus<-1,-2>>::type
>
operator/(quantity<T,D1> x, quantity<T,D2> y)
{
    typedef typename
        mpl::transform<D1,D2,mpl::minus<-1,-2>>::type dim;

    return quantity<T,dim>( x.value() / y.value() );
}
}
```

**MPL-Metafunktionsklasse:**

The most basic way to formulate a compile-time function so that it can be treated as polymorphic metadata; that is, as a type. A metafunction class is a class with a nested metafunction called apply.

Gewöhnungsbedürftige Syntax der Metaprogrammierung: [MPL-ManualSeite 99](#)

```
typedef vector<int , char , long , short , char , short , double , long> types;
typedef count<types , short >::type n;
BOOST_MPL_ASSERT_RELATION( n::value , ==, 2 );
```

(Werte des Metaprogramms werden durch typedefs an „Typnamen-Aliases“ gebunden, ...)

## 2.14.7. Nachteile von DSLs

- meist Nischensprachen, häufig fehlende Sprachstandards, fehlende freie Implementierungen, ...
- hoher Aufwand für das Erlernen der nur in wenigen Fällen benutzbaren DSL
- Risiko, dass der Anwender zusätzlich viel Entwicklung in der Hostsprache (hier C++) statt der DSL selbst erledigen muß.
- Risiko des Bindens an den Anbieter einer Nischensprache
- Risiko des zukünftigen Vermeidens der Entwicklung von Problemlösungen in etablierten allgemeinen Hochsprachen
- hoher Aufwand der Spezifikation, Entwicklung und Wartung der DSL
- Schwierigkeit, die langfristig benötigten Eigenschaften der DSL richtig abzuschätzen
- Risiko der schleichenden Entwicklung der DSL zu einer allgemeinen Programmiersprache
- Schwierigkeit der Findung des der DSL angemessenen Abstraktionsniveaus
- Hoher Anspruch an die Kompetenz der Entwickler des DSLs

(Nachteil DSLs)

## 2.15. Literaturhinweise zum Metaprogrammieren

C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond  
Generative Programming — Methods, Tools and Applications, Kapitel 10 Static Metaprogramming in C++





# 3. Template template-Parameter, Policy-basiertes Klassendesign

## 3.1. Templates als Template-Parameter

Die Template-Klassen `vector`, `deque`, `list`, ... als `Container`.

`deque`  
`vector`

Template Template-Parameter:

```
template <template <typename, typename> class Container, typename Type>
class Example
{
    Container<Type, std::allocator <Type> > baz;
};
```

```
// Beispiel der Verwendung:
//
// statt xxxxx<std::deque<int>, int> ...
```

```
Example <std::deque, int> example;
```

## 3.2. Policies

Policies bei der Template-Metaprogrammierung:

„Policies sind Klassen-Templates, die dazu dienen, Verhalten auszulagern.“

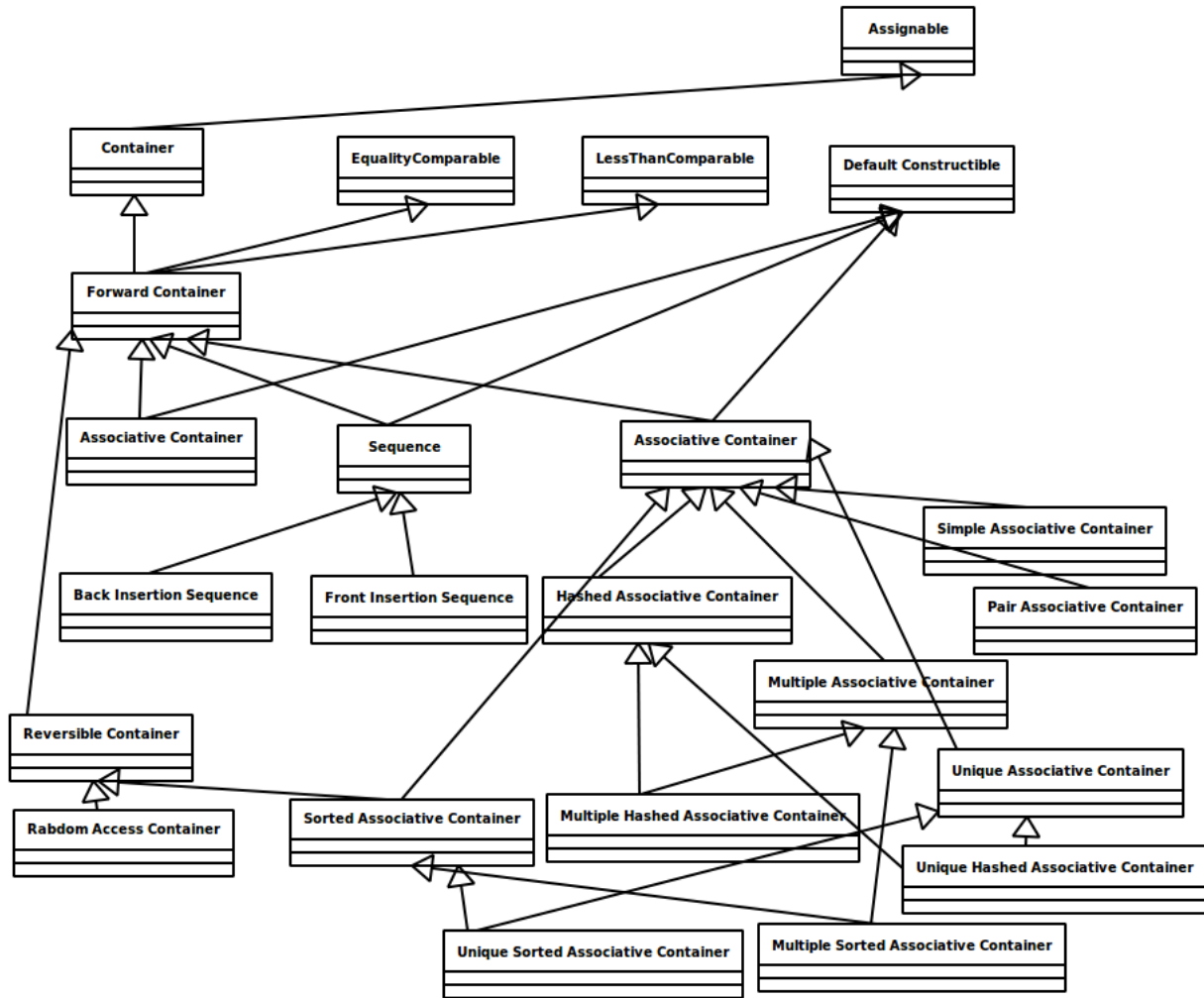
Ein Beispiel:

```
struct MultiThreadingPolicy {
    typedef /* ... */ Mutex;
    struct Lock {
        Lock(Mutex& mtx) : mtx_(mtx) {lock(mtx_);}
        ~Mutex() {unlock(mtx_);}
        Mutex& mtx_;
    };
};

struct SingleThreadingPolicy {
    class Mutex {};
    struct Lock {
        Lock(Mutex&) {}
        ~Mutex() {}
    };
};

//
// Ein Algorithmus koennte jetzt so aussehen:
//
template< class ThreadingPolicy >
void f(typename ThreadingPolicy::Mutex& mtx)
{
    // ...
    if( needToTouchThreadSensibleData() ) {
        typename ThreadingPolicy::Lock lock(mtx); // lock mutex
        // thread-safe section
    }
    // ...
}
```

STL-Container:



Policies (Implementierungsvarianten/-verhaltensweisen) der STL-Container:

- assoziativ/nichtassoziativ
- sortiert/unsortiert
- hashed/ohne hash
- unique/multiple

Andere Policies:

- threadsave
- errorhandling (exception, errno, ...)
- allocator (threadsave, singleclient, malloc-based, ...)
- ...

STL Allocators

## Policy based Design:

Implementiert wird durch eine zur Compilezeit durchgeführte Template-Instantiierung der gewünschten Policy-Ausprägungen der Template-Kindklasse aller Policies:

```
template < typename output_policy , typename language_policy >
class HelloWorld : public output_policy , public language_policy
{
    using output_policy :: Print ;
    using language_policy :: Message ;

public :
    //behaviour method
    void Run()
    {
        //two policy methods
        Print( Message() );
    }
};

#include <iostream>

class HelloWorld_OutputPolicy_WriteToCout
{
    protected :
        template< typename message_type >
        void Print( message_type message )
        {
            std::cout << message << std::endl ;
        }
};

#include <string>

class HelloWorld_LanguagePolicy_English
{
    protected :
        std::string Message()
        {
            return "Hello , World!" ;
        }
};

class HelloWorld_LanguagePolicy_German
{
    protected :
        std::string Message()
        {
            return "Hallo Welt!" ;
        }
};

int main()
```

```

{
    /* example 1 */
    typedef HelloWorld<HelloWorld_OutputPolicy_WriteToCout ,
        HelloWorld_LanguagePolicy_English> my_hello_world_type;

    my_hello_world_type hello_world;
    hello_world.Run(); // Prints "Hello, World!"

    /* example 2
    * does the same but uses another policy, the language has changed
    */
    typedef HelloWorld< HelloWorld_OutputPolicy_WriteToCout ,
        HelloWorld_LanguagePolicy_German > my_other_hello_world_type;

    my_other_hello_world_type hello_world2;
    hello_world2.Run(); // Prints "Hallo Welt!"
}

```

Seite 8: ThreadingPolicy

CreationPolicy

Generic Pool Design: CreationPolicy, ExpirationPolicy

Boost numeric conversions: OverflowHandler, Float2IntRounder, RawConverter, UserRangeChecker

Policies and the STL: AllocationPolicy, CharTPolicy

### 3.3. Entwurfsmuster Strategie

Policies als Compile-Time-Variante des Strategie-Designmusters

C++ Design Pattern: What is a Design Pattern?

Einführung in Design Patterns: 4.4 Das Strategy Pattern

Strategy pattern

The Strategy design motif

Implementing the Strategy Pattern

Design patterns

Entwurfsmuster Iterator

Entwurfsmuster Observer

Wikibook Entwurfsmuster

### 3.4. Policies als Template Template-Parameter

(aus: <http://ess.cs.tu-dortmund.de/Teaching/SS2013/SuS/Downloads/06.2-PL-mit-Templates.pdf#page=15>;) Konfigurierbare `CreationPolicy` mit den Ausprägungen

```
template <typename T>
class NewCreator {
public:
    static T *create() {
        return new T;
    }
};
// ...
Widget *w = NewCreator<Widget>::create();
```

und

```
template <typename T>
class MallocCreator {
public:
    static T *create() {
        void *buf = malloc(sizeof(T));
        if (!buf) return 0; // oder Exception werfen
        return new(buf) T; // placement new
    }
};
...
Widget *w = MallocCreator<Widget>::create();
```

Diese (und eventuell andere ausfaktorisierte Policies werden benutzt zum Beispiel in der Host-Klasse:

```
template <typename CreationPolicy>
class WidgetManager {
public:
    void newWidget() {
        Widget *w = CreationPolicy::create();
        // ...
    }
    // ...
};
// ...
typedef WidgetManager<NewCreator<Widget> > WM1;
// ...
typedef WidgetManager<MallocCreator<Widget> > WM2;
// ...
```

Ärgerlich, aber durch Nutzung von Template Template-Parameter vermeidbar, ist die Notwendigkeit, als aktuellen template-Parameter immer `Widget` angeben zu müssen.

Hier also die entgültige Lösung:

```
template <template <typename> class CreationPolicy>
class WidgetManager {
public:
    void newWidget () {
        Widget *w = CreationPolicy<Widget>::create ();
        // ...
    }
    // ...
};
// jetzt kompakter und einfacher:
typedef WidgetManager<NewCreator> WM1;
// ...
typedef WidgetManager<MallocCreator> WM2;
// ...
```

Die Benutzung von Template Template-Parametern macht den Code einfacher zu lesen, weniger fehleranfällig und flexibler.

## 3.5. Orthogonale Policy-Dimensionen

[A Case for Orthogonality in Design](#)  
[Orthogonality](#)

Policies sollten minimale orthogonale Implementierungsvarianten sein.

## 3.6. Policies (Fortsetzung)

(aus WordIQ.com:)

**Policy-based design** is a programming technique, which one could call the compile-time equivalent of the Strategy pattern. ...

The technique is used to create a flexible set of types, providing the same interface, but employing different implementation behind. Therefore this technique has a lot of similarity to the Strategy pattern. However while Strategy allows the type to “change its ways“ runtime, policy-based design fixes the implementation during compilation. In fact, it creates a new type for each different implementation. While their interface (functions present, their names etc.) will be the same, they will be different type – as opposed to Strategy, where the same type behaves differently.

The main idea is to use commonality-variability analysis to divide the type into fixed implementation and interface (the policy based class) and the different policies. The main class, the policy based class takes template arguments (types, templates of types etc.) and delegates parts of the work to the policies.

...

[Policy-based design in C++](#)

## 3.7. Loki

[Loki \(C++\): example designs of common design patterns and idioms](#)  
[Loki Pattern visitor](#)

## 3.8. A Policy-Based `flex_string` Implementation

A Policy-Based `basic_string` Implementation



# 4. Aspektorientiertes Programmieren in komplexen Unternehmensanwendungen

Policy-basiertes Design = Template-gesteuerte statische Wahl einer Implementierungsvariante durch den entwickelnden Bibliotheksbenutzer.

**Aspect-Oriented Programming (AOP)** = Ergänzung einer Programmiersprache um neue Sprachmittel (aspekt, advice) zur lokalisierten übersichtlichen, wartbaren und insbesondere erweiterbaren Codierung von Querschnittsanforderungen (= **Cross-Cutting Concerns (CCC)**)

## AOP:

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects.

## AOP als Ergänzung des OOP

Code Scattering der Wirkungsstellen einzelner Belange/Anforderungen

Seite 21: Aspectual Decomposition, Aspectual Recomposition

I want my AOP!:

- Aspectual decomposition: Decompose the requirements to identify crosscutting and common concerns. You separate module-level concerns from crosscutting system-level concerns. For example, in the aforementioned credit card module example, you would identify three concerns: core credit card processing, logging, and authentication.
- Concern implementation: Implement each concern separately. For the credit card processing example, you'd implement the core credit card processing unit, logging unit, and authentication unit.
- Aspectual recomposition: In this step, an aspect integrator specifies recomposition rules by creating modularization units – aspects. The recomposition process, also known as weaving or integrating, uses this information to compose the final system. For the credit card processing example, you'd specify, in a language provided by the AOP implementation, that each operation's start and completion be logged. You would also specify that each operation must clear authentication before it proceeds with the business logic.

Weitere Beispiele von Aspekten:

- logging
- object counting
- locking
- errorhandling
- creational OO design patterns

- Singleton
- ...
- structural design patterns
  - Proxy
  - ...
- behavioral design patterns
  - Strategy
  - ...
- Class/Component Scale Aspects
- Application Scale Aspect
- Enterprise Scale Aspect
- Director Design Pattern
- ...

(aus: [AspectJ Cookbook: Aspect Oriented Solutions to Real-World Problems](#) )

### Scattering und Tangling Code

Needed an AOP language with:

- implementation of concerns: Mapping an individual requirement into code so that a compiler can translate it into executable code. Since implementation of concerns takes the form of specifying procedures, you can to use traditional languages like C, C++, or Java with AOP.

```
.NET Framework languages (C# / VB.NET) [10]
ActionScript [11]
Ada [12]
AutoHotkey [13]
C / C++ [14]
COBOL [15]
The Cocoa Objective-C frameworks [16]
ColdFusion [17]
Common Lisp [18]
Delphi [19] [20] [21]
Delphi Prism [22]
e (IEEE 1647)
Emacs Lisp [23]
Groovy
Haskell [24]
Java [25]
    AspectJ
JavaScript [26]
Logtalk [27]
Lua [28]
Matlab [29]
make [30]
ML [31]
PHP [32]
Racket [33]
```

Perl [34]  
Prolog [35]  
Python [36]  
Ruby [37] [38] [39]  
Squeak Smalltalk [40] [41]  
UML 2.0 [42]  
XML [43]

(aus: [AOP Implementations](#))

- Weaving rules specification: How to compose independently implemented concerns to form the final system. For this purpose, an implementation needs to use or create a language for specifying rules for composing different implementation pieces to form the final system. The language for specifying weaving rules could be an extension of the implementation language, or something entirely different.
- Recompile the whole enterprise application.

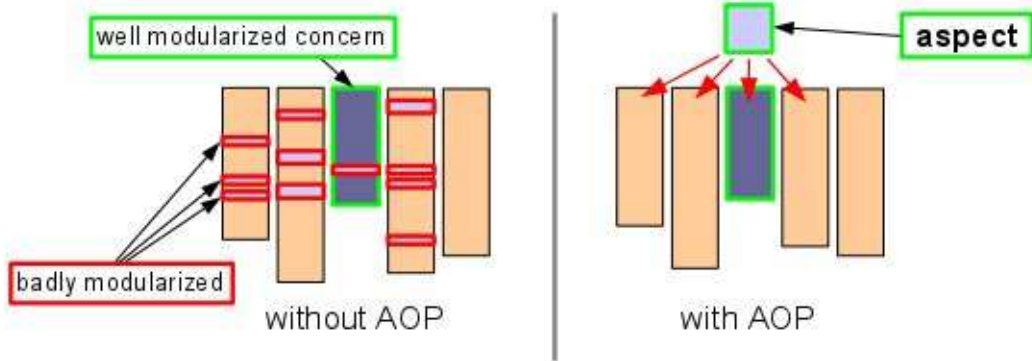
AOP vocabulary: aspect, join point, pointcut, advice

[AspectC++ Quick Reference](#)

[Aspect-Oriented Programming with C++ and AspectC++:](#)

# Aspect-Oriented Programming

➤ AOP is about modularizing crosscutting concerns



without AOP

with AOP

➤ Examples: tracing, synchronization, security, buffering, error handling, constraint checks, ...

Introduction © 2007 Daniel Lohmann and Olaf Spinczyk 1/5

## The Simple Queue Class Revisited

```

namespace util {
class Item {
    friend class Queue;
    Item* next;
public:
    Item() : next(0){}
};

class Queue {
    Item* first;
    Item* last;
public:
    Queue() : first(0), last(0) {}

    void enqueue( Item* item ) {
        printf( " > Queue::enqueue()\n" );
        if( last ) {
            last->next = item;
            last = item;
        } else
            last = first = item;
        printf( " < Queue::enqueue()\n" );
    }

    Item* dequeue() {
        printf( " > Queue::dequeue()\n" );
        Item* res = first;
        if( first == last )
            first = last = 0;
        else
            first = first->next;
        printf( " < Queue::dequeue()\n" );
        return res;
    }
}; // class Queue
} // namespace util

```

Erweiterung „gemeinsamer Elementzähler“

## ElementCounter1

```

aspect ElementCounter {

    int counter;
    ElementCounter() {
        counter = 0;
    }

    advice execution("% util::Queue::enqueue(...)") : after() {
        ++counter;
        printf( " Aspect ElementCounter: # of elements = %d\n", counter );
    }
    advice execution("% util::Queue::dequeue(...)") : after() {
        if( counter > 0 ) --counter;
        printf( " Aspect ElementCounter: # of elements = %d\n", counter );
    }
};

```

ElementCounter1.ah

modifizierte Erweiterung „Elementzähler für jede Warteschlange getrennt“

## ElementCounter2



```
aspect ElementCounter {
    advice "util::Queue" : slice class {
        int counter;
    public:
        int count() const { return counter; }
    };
    advice execution("% util::Queue::enqueue(...)")
        && that(queue) : after( util::Queue& queue ) {
        ++queue.counter;
        printf( " Aspect ElementCounter: # of elements = %d\n", queue.count() );
    }
    advice execution("% util::Queue::dequeue(...)")
        && that(queue) : after( util::Queue& queue ) {
        if( queue.count() > 0 ) --queue.counter;
        printf( " Aspect ElementCounter: # of elements = %d\n", queue.count() );
    }
    advice construction("util::Queue")
        && that(queue) : before( util::Queue& queue ) {
        queue.counter = 0;
    }
};
```

Erweiterung Errorhandling

## ErrorException



```
namespace util {
    struct QueueInvalidItemError {};
    struct QueueEmptyError {};
}

aspect ErrorException {

    advice execution("% util::Queue::enqueue(...)") && args(item)
        : before(util::Item* item) {
        if( item == 0 )
            throw util::QueueInvalidItemError();
    }
    advice execution("% util::Queue::dequeue(...)") && result(item)
        : after(util::Item* item) {
        if( item == 0 )
            throw util::QueueEmptyError();
    }
};
```

ErrorException.ah

## LockingMutex



```
aspect LockingMutex {
  advice "util::Queue" : slice class { os::Mutex lock; };

  pointcut sync_methods() = "% util::Queue::%queue(...)";

  advice execution(sync_methods()) && that(queue)
  : around( util::Queue& queue ) {
    queue.lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      queue.lock.leave();
      throw;
    }
    queue.lock.leave();
  }
};
```

LockingMutex.ah

## LockingIRQ2



```
aspect LockingIRQ {

  pointcut sync_methods() = "% util::Queue::%queue(...)";
  pointcut kernel_code() = "% kernel::%(...)";

  advice execution(sync_methods())
  && !cflow(execution(kernel_code())) : around() {
    os::disable_int();
    try {
      tjp->proceed();
    }
    catch(...) {
      os::enable_int();
      throw;
    }
    os::enable_int();
  }
};
```

### Solution

Using the **cflow** pointcut function

LockingIRQ2.ah

Spezifikation der Cutpoints (Name Matching):

<http://www.aspectc.org/fileadmin/documentation/ac-languageref.pdf#page=17>

„Named Types“-Matching sollte mittels `type_traits` (oder später besser noch Concept-Erfüllung) möglich sein.

.Net Introduction to AOP:

## Advice types

Now that we have defined *what* is advised and *where* we can add the minor detail of *when* it is applied, because we have different kinds of advices. The example above uses a so called *around* advice. That is, the advice is wrapped around the joinpoint. It controls the moment when the joinpoint is executed explicitly by calling `Proceed()`. Other examples of advice types are:

*before* and *after* advices, which are executed before or after the execution of the joinpoint respectively. Those advices can't explicitly define when the joinpoint is executed.

*throwing* advices, which are only executed if the execution of the joinpoint raised an exception

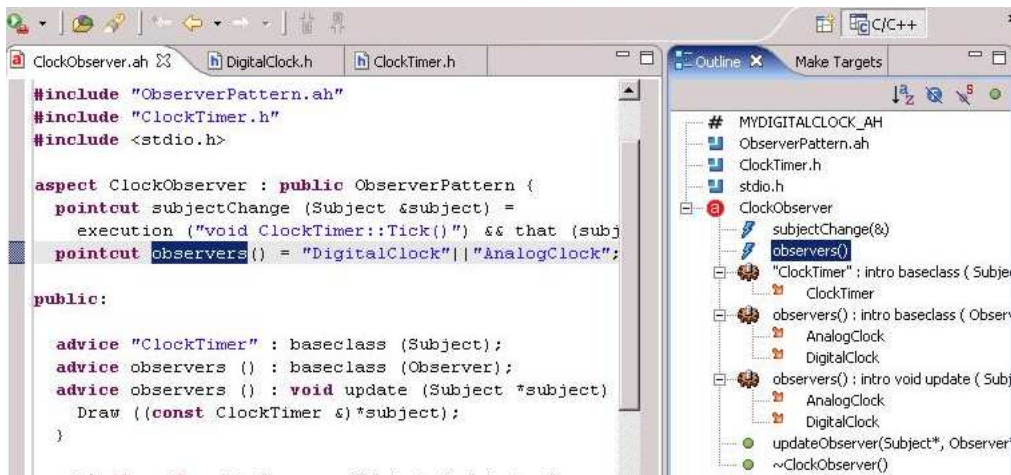
*returning* advices, which are only executed if the execution of the joinpoint did *not* raise an exception

Finally, here's a fine-tuned version of our pseudo code example:

```
1: pointcut ServiceMethods : call( Service.* )
2:
3: around(context) : ServiceMethods {
4:     try {
5:         context.Proceed();
6:     } catch(Exception e) {
7:         if(ExceptionHandler.ShouldRethrow(e)) {
8:             throw;
9:         }
10:    }
11: }
```

By the way, I didn't come up with this notation all by myself. It resembles the syntax used with [AspectJ](#), which is more or less the most prominent (Java) AOP framework available. It has grown quite mature and if you're interested in AOP I highly recommend to have a look at it. Yes - even it uses Java :-)

## Eclipse AspectC/C++ Development Tools:



(aus: Screenshots)

### Nachteile:

Schlechte Unterstützung beim Debuggen, Profilen, ...

(mögliche) Codeexplosion beim Aspekt-Einweben

Setzt AOP-Begriffe und -Ideologien als Bekannt voraus (dann allerdings leicht erlernbar))

Erfordert Pattern-Matching-Erfahrungen (Filterdefinition)

Erfordert Recompileation der in der Regel riesigen Unternehmensapplikationen

Probleme der Abhängigkeit von der Reihenfolge des Einwebens(?)

evtl. schlecht lesbarer neu entstehender Code

Sind wirklich alle relevanten Codestellen mit Advices geändert worden? (fehlende direkte Sprachkonstrukte von C++, z.B. Annotationen mit Aspekt-Bezug, ...)

### Vorteile:

Schnell und einfach aufzusetzen

selektiv einsetzbar

keine Modifikation der Originalquellen nötig

leicht entfernenbar

gute Performance

### AOP-Implementierung

[http://en.wikipedia.org/wiki/Aspect-oriented\\_programming#Implementation](http://en.wikipedia.org/wiki/Aspect-oriented_programming#Implementation)

Code Injection

Bytecode Instrumentation

Dynamic Bytecode Instrumentation

The Design and Implementation of AspectC++

AspectC++ Quick Reference (page 8f.)

Advances in AOP with AspectC++



**Entwicklungsstand:**

(siehe [Entwicklungsstand der aspektorientierten Programmierung](#))

- noch in den Kinderschuhen (erfaßt unter anderem nur selbst Compiliertes, nicht jedoch lediglich Benutztes, ...)
- Realisierung des AOP noch unausgereift (Patchlisten, ...)
- Bis zu akzeptabler Reife dürfte es noch einige Jahre und einige Programmiersprachengenerationen dauern
- hohe Abstraktion und völlig andere ungewohnte Herangehensweise stellt hohe Anforderungen an den (zukünftigen) Entwickler



# 5. Unternehmensanwendungen und Frameworks

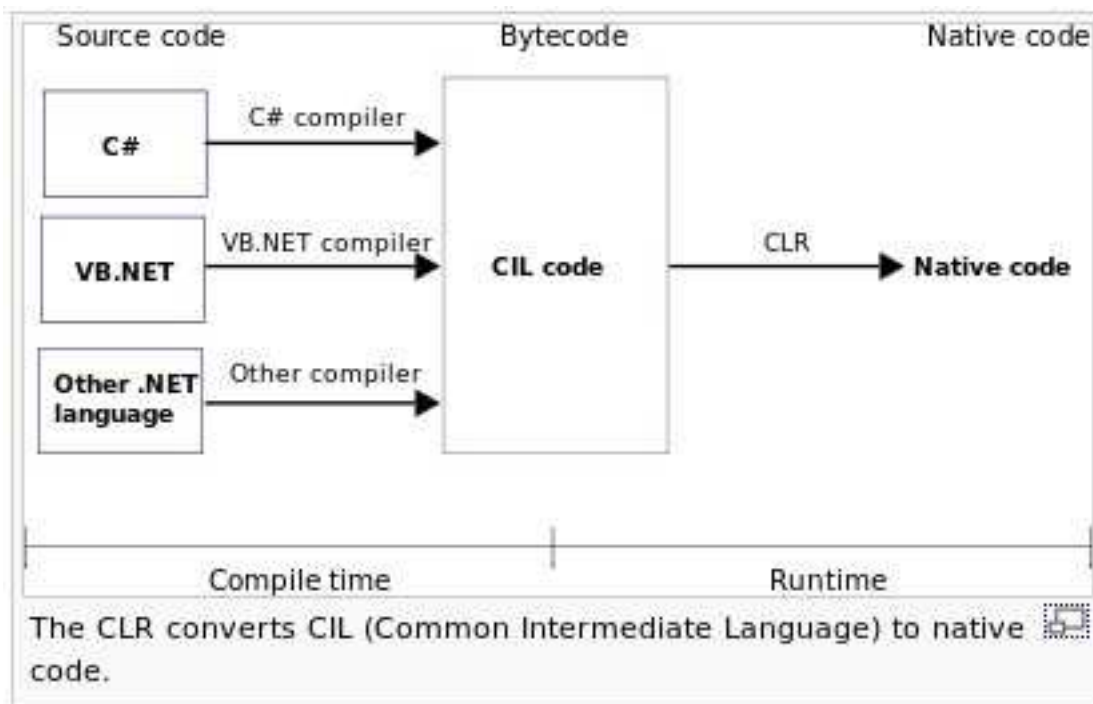
## 5.1. Frameworks

framework = Rahmenwerk, Grundstruktur, ...

<http://www.itwissen.info/definition/lexikon/Framework-framework.html>

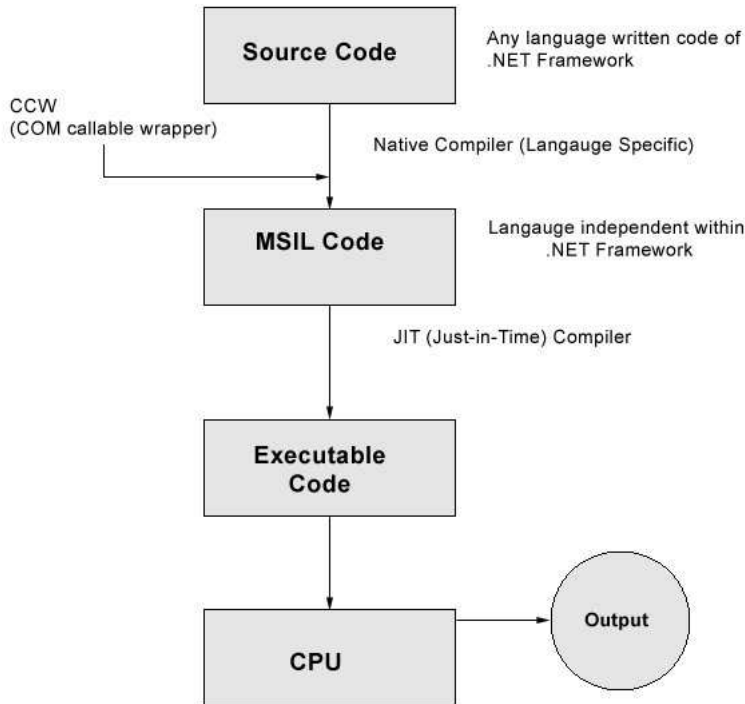
Beispiele:

1. eclipse Rich Client Platform mit Equinox-OSGi-Kern
  2. Spring Framework, inklusive AOP AspectJ, Spring – Hello World, Chakraborty: Pro Spring 2.5
  3. .NET Framework-Plattform, .NET.Framework, Managed Code
  4. SAP R/3 Business Framework
  5. OA-Netzwerk-Projekte
  6. web\_start, IcedTea
- früher: dezentrale Systeme/verteiltes Rechnen, DCE, ...

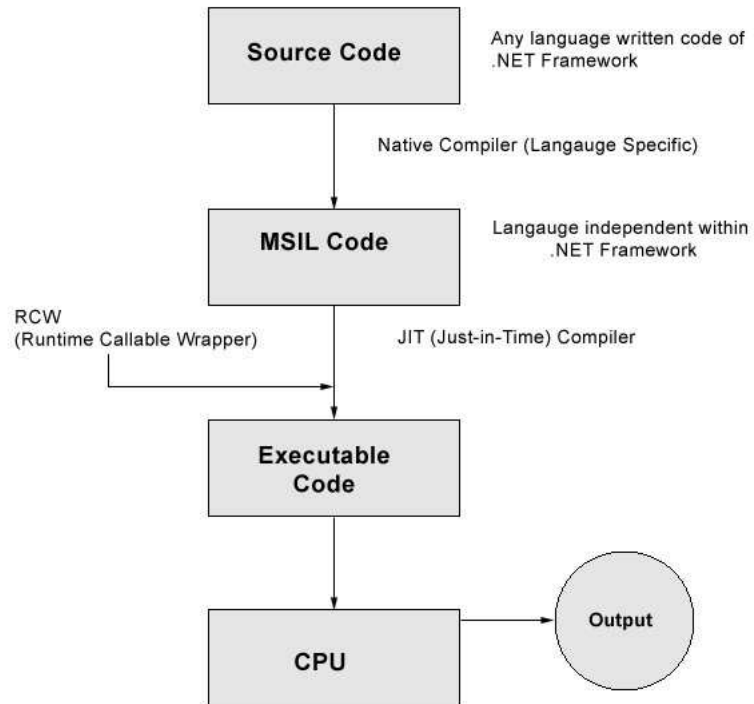


(vgl. Managed code and unmanaged code, Code Execution Process)

### Unmanaged Code Execution Process



### Unmanaged Code Execution Process



...  
Framework:

#### Framework-Typen:

- Applikationenframeworks
- Anwendungsbereichsframeworks
- Klassenframeworks
- Komponentenframeworks
- Koordinationsframeworks
- Testframeworks
- Webframeworks

unter anderem:

- JavaFX
- Eclipse IDE
- Java EE
- HTML5
- Spring
- Netbeans IDE
- JQuery
- JUnit Testframework
- JavaServer Faces
- MFC
- .NET
- Cocoa-API (Foundation-, Application Kit-, Core Data-Framework), für Objective-C
- Zend für PHP 5

Horde (Groupware-Webanwendungen in PHP 5)

Ruby on Rails (Webanwendungen in Ruby)

TYPO 3 CMS in PHP

...

## 5.2. Testframeworks

### 5.2.1. Testumgebungen in dyn. Programmiersprachen: pydev und „Stand der Technik“-Testumgebungen

dynamischen Programmiersprachen

Python-Testumgebungen

Python-Testenvironments: integrierte Dokumentation und Testumgebung: doctest, Codeverträge

### 5.2.2. cxxtest (mit minimalen Systemanforderungen, Ubuntu)

C++-Test Frameworks

CxxTest User Guide:

- Benutzungsanleitung: Test Suites

- Test Assertions

- Test Runner

- Seite 9f.: qt Testrunner

- CxxTest test fixtures: Setup and Teardown

- TS\_ASSERT\_EQUALS() benötigt operator==

- Mock-Objekte (Attrappen)

- Makefiles

### 5.2.3. Gleitkomma-Maschinengenauigkeit

Machine epsilon

Values for standard hardware floating point arithmetics

```
#include <iostream>
#include <limits>

int main()
{
    //using a built-in function to display the machine-epsilon ...
    std::cout << "The machine precision for double is : "
              << std::numeric_limits<double>::epsilon()
              << std::endl;
    std::cout << "The machine precision for long double is : "
              << std::numeric_limits<long double>::epsilon()
              << std::endl;
    return 0;
}

vgl.:
namespace std {
    template<T> class numeric_limits<float> {
    public:
        static constexpr bool is_specialized = true;
        inline static constexpr float min() noexcept { return 1.17549435E-38F; }
        inline static constexpr float max() noexcept { return 3.40282347E+38F; }
        inline static constexpr float lowest() noexcept { return -3.40282347E+38F; }
        // ...
        inline static constexpr float epsilon() noexcept { return 1.19209290E-07F; }
        //...
    }
}
(C++11)
```

## 5.2.4. Tests auf annähernde Gleichheit: Boost.Test, ...

[Boost Floating-point comparison algorithms: close\\_at\\_tolerance\(\)](#)

üblicher:

```
static bool approximatelyEqualTo(double left, double right, double factor){
    return fabs(left - right) <= std::numeric_limits<double>::epsilon() *
        factor *
        std::max(fabs(left), fabs(right));
}
```

[Floating-point tolerances revisited](#)

## 5.2.5. UnitTest++

[UnitTest++](#)

[Kurzdokumentation](#)

[Money, a step by step example for Visual Studio .NET 2005](#)

## 5.2.6. CppUnit (OpenSUSE)

[CppUnit](#)

[CppUnit Wiki](#)

[Unit-Tests unter C++ mit dem Framework CppUnit](#)

[CppUnit Documentation](#)

[CppUnit FAQ](#)

[CppUnit Cookbook](#)

```
//...
public:
    static CppUnit::Test *suite()
    {
        CppUnit::TestSuite *suiteOfTests =
            new CppUnit::TestSuite( "ComplexNumberTest" );
        suiteOfTests->addTest( new CppUnit::TestCaller<ComplexNumberTest>(
            "testEquality",
            &ComplexNumberTest::testEquality ) );
        suiteOfTests->addTest( new CppUnit::TestCaller<ComplexNumberTest>(
            "testAddition",
            &ComplexNumberTest::testAddition ) );

        return suiteOfTests;
    }
// ...
int main( int argc, char **argv)
{
    CppUnit::TextUi::TestRunner runner;
    runner.addTest( ComplexNumberTest::suite() );
    runner.run();
    return 0;
}
```

QtTestRunner  
CPPUNIT\_ASSERT\_...  
CppUnitWithEclipse

Zukunft: CppUnit 2 ...  
... und Open Test

## 5.2.7. Vergleich UnitTest++ mit CppUnit

unittest++ vs cppunit



# 6. C++-Contract Programming with AOP

## C++11-Contracting Spracherweiterungsvorschlag (Proposal n1866)

*Spezifikation wie sie einmal in C++ aussehen könnte:*

```
double sqrt( double r )
{
    precondition
    {
        r >= 0.;
    }
    postcondition( result )
    {
        equal_within_precision( result * result , r );
    }
    {
        ...
    }
}

int factorial( int n )
precondition
{
    0 <= n && n <= 12;
}
postcondition( result )
{
    result >= 1;
}
{
    if ( n < 2 )
        return 1;
    else
        return n * factorial( n - 1 );
}

template< class T >
class vector
{
    invariant
    {
        ( size() == 0 ) == empty();
        size() == std::distance( begin(), end() );
        size() == std::distance( rbegin(), rend() );
        size() <= capacity();
        capacity() <= max_size();
    }

    void resize( size_type newsz )
    postcondition
    {
        size() == newsz;
        if( newsz > oldof size() )
            all_equals( begin() + oldof size(), end(), T() );
    }
};
```

```
    }  
    void clear();  
        postcondition { empty(); }  
    void swap( vector& right )  
        postcondition  
        {  
            oldof *this == right;  
            oldof right == *this;  
        }  
    // ...  
}; // class 'vector'
```

(Proposal to add Contract Programming to C++ (revision 4),

siehe insbesondere: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1962.html#vector-example-hpp>)

State of C++ Evolution (Mid-term 2008 Mailing): vorerst nicht im Standard

## 6.1. Contract enforcement with AOP

Contract enforcement with AOP

Designing by Contract with JML for Aspect-Oriented Programs

# 7. Aspects in AspectJ Cookbook

Russ Miles: AspectJ Cookbook:

...

## Chapter 17 Implementing Creational Object-Oriented Design Patterns

- Introduction
- Implementing the Singleton Pattern
- Implementing the Prototype Pattern
- Implementing the Abstract Factory Pattern
- Implementing the Factory Method Pattern
- Implementing the Builder Pattern

## Chapter 18 Implementing Structural Object-Oriented Design Patterns

- Introduction
- Implementing the Composite Pattern
- Implementing the Flyweight Pattern
- Implementing the Adapter Pattern
- Implementing the Bridge Pattern
- Implementing the Decorator Pattern
- Implementing the Proxy Pattern

## Chapter 19 Implementing Behavioral Object-Oriented Design Patterns

- Introduction
- Implementing the Observer Pattern
- Implementing the Command Pattern
- Implementing the Iterator Pattern
- Implementing the Mediator Pattern
- Implementing the Chain of Responsibility Pattern
- Implementing the Memento Pattern
- Implementing the Strategy Pattern
- Implementing the Visitor Pattern
- Implementing the Template Method Pattern
- Implementing the State Pattern
- Implementing the Interpreter Pattern

## Chapter 20 Applying Class and Component Scale Aspects

- Introduction
- Validating Parameters Passed to a Method
- Overriding the Class Instantiated on a Call to a Constructor
- Adding Persistence to a Class
- Applying Mock Components to Support Unit Testing

## Chapter 21 Applying Application Scale Aspects

- Introduction
- Applying Aspect-Oriented Tracing
- Applying Aspect-Oriented Logging
- Applying Lazy Loading
- Managing Application Properties

## Chapter 22 Applying Enterprise Scale Aspects

- Introduction
- Applying Development Guidelines and Rules
- Applying Transactions
- Applying Resource Pooling
- Remoting a Class Transparently Using RMI
- Applying a Security Policy

## Chapter 23 Applying Aspect-Oriented Design Patterns

- Introduction
- Applying the Cuckoo's Egg Design Pattern
- Applying the Director Design Pattern
- Applying the Border Control Design Pattern
- Applying the Policy Design Pattern

...

# A. Ausblick

Concepts-Lite — experimental branch of the GCC C++ compiler

Concepts Lite — Constraining Template Arguments with Predicates, Presentation

Concepts Lite — Constraining Template Arguments with Predicates, Paper

A Concept Design for the STL

Alexander Stepanov, Paul McJones: *Elements of Programming*, mathematisch (algebraische) Fundierung von Sortieren, Iteratoren, ...