Materialsammlung - generische Programmierung

Prof. Dr. Hans-Jürgen Buhl

Winersemester 2011/2012

Fachgruppe Mathematik und Informatik
FB C
Bergische Universität Wuppertal

Praktische Informatik
PIBUW - WS11/12
Oktober 2011
2. Auflage, 2011

Inhaltsverzeichnis

Vorbemerkungen:

Literatur

- B. Stroustrup: Einführung in die Programmierung mit C++, Pearson 2010, München, Kapitel 19.3 ff.
- D. Vandervoorde, N. M. Josuttis: C++ Templates — The Complede Guide, Pearson 2003, Boston
- Scott Meyers: Effective STL, Addison-Wesley 2001, Indianapolis
- Björn Karlsson: Beyond the C++ Standard Library — An Introduction to Boost, Pearson 2006, Boston
- A. Alexandrescu: Modern C++ Design — Generic Programming and Design Patterns Applied, Pearson 2001, Indianapolis

Einordnung in Programmierparadigmen
imperativ, objektbasiert, objektorientiert, funktional, generisch, ...

Die Entwicklung der Aussagekraft der formalen generischen Parameter

- von einfallslosen Parameternamen wie `class T1, class T2, ...`
  vergleiche http://www.cplusplus.com/doc/tutorial/templates/
- über semantisch inhaltsvolle Parameternamen wie `typename InputIterator1, typename InputIterator2, typename NumericT, ...`
  vergleiche http://www.iue.tuwien.ac.at/phd/heinzl/node32.html#SECTION010223000000000000000
- hin zur Nennung der Requirements an die zur Instantiierung benutzbaren aktuellen Parameter wie `T shall meet the requirements of CopyConstructible and CopyAssignable types`
  (Seite 970 von http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3126.pdf)
  mit der Erläuterung:

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

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

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

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

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

Expression	Post-condition
`T u(v);`	the value of `v` is unchanged and is equivalent to `u`
`T(v)`	the value of `v` is unchanged and is equivalent to `T(v)`

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

Expression	Return type	Return value	Post-condition
`t = rv`	`T&`	`t`	`t` is equivalent to the value of `rv` before the assigment
[*Note:* `rv` remains a valid object. Its state is unspecified. — *end note*]			

(Seite 483 des Drafts)

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

C++11 ohne „Concepts“
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf
beziehungsweise
http://en.wikipedia.org/wiki/C%2B%2B11

TR1 - ein „Zwischenstandard“ für die C++-Bibliothek
TR1
http://en.wikipedia.org/wiki/Technical_Report_2#Mathematical_special_functions

Was hätten Concepts gebracht?
http://en.wikipedia.org/wiki/Concepts_(C%2B%2B)

Ziele des Draft-Desings
http://www.artima.com/cppsource/cpp0x.html

TR2 call for proposals
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1810.html

Typsicherheit ... bei generischen Konstrukten
http://de.wikipedia.org/wiki/Typsicherheit

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
Generics in Java

C++11 Standard Library
C++ standard library changes

C++ Reference

1 Generische Programmierung

1.1 Was ist generische Programmierung?

http://pdfcast.org/pdf/the-java-generic-programming-system
http://foldoc.org/generic+programming
http://www.boost.org/community/generic_programming.html
http://en.wikipedia.org/wiki/Generic_programming

1.2 Beispiel einer generische 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 „gerneral 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
`T u = rv;`	`u` is equivalent to the value of `rv` before the construction
`T(rv)`	`T(rv)` is equivalent to the value of `rv` before the construction
`rv`'s state is unspecified [*Note:* `rv` must still meet the requirements of the library component that is using it. The operations listed in those requirements must work as specified whether `rv` has been moved from or not. — *end note*]	

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

Expression	Post-condition
`T u = v;`	the value of `v` is unchanged and is equivalent to `u`
`T(v)`	the value of `v` is unchanged and is equivalent to `T(v)`

Table 22 — `MoveAssignable` requirements **[moveassignable]**

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

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

Expression	Return type	Return value	Post-condition
`t = v`	`T&`	`t`	`t` is equivalent to `v`, the value of `v` is unchanged

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

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)?

http://en.wikipedia.org/wiki/Executable_and_Linkable_Format
http://www.ibm.com/developerworks/aix/library/au-unixtools/index.html
http://cpp.comsci.us/process/build.html

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++  -g -I. -I/home/username/include   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 Programmlauf 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\0\3\0>\0\1\0\0\0\260\305\5\0\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\0\3\0>\0\1\0\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\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\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", 41 5
```

```
)                   = 4
write(1, "5 1\n", 45 1
)                   = 4
write(1, "3.1415 15.1055\n", 153.1415 15.1055
)       = 15
write(1, "15.1055 3.1415\n", 1515.1055 3.1415
)       = 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.
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<c
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
```

```
> c++filt -n _Z4swapIiEvRT_S1_
void swap<int>(int&, int&)
```

Vergleiche:
C++ name mangling
names in object files
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 0x00000000004007e0

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000400040 paddr 0x0000000000400040 align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off    0x0000000000000238 vaddr 0x0000000000400238 paddr 0x0000000000400238 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                 0x0000000000400720
  FINI                 0x0000000000400ba8
...
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 l    d  .fini  0000000000000000              .fini
0000000000400bb8 l    d  .rodata        0000000000000000              .rodata
0000000000400bc0 l    d  .eh_frame_hdr  0000000000000000              .eh_frame_hdr
0000000000400c08 l    d  .eh_frame      0000000000000000              .eh_frame
0000000000600de0 l    d  .ctors 0000000000000000              .ctors
0000000000600df8 l    d  .dtors 0000000000000000              .dtors
0000000000600e08 l    d  .jcr   0000000000000000              .jcr
0000000000600e10 l    d  .dynamic       0000000000000000              .dynamic
0000000000600fe0 l    d  .got   0000000000000000              .got
0000000000600fe8 l    d  .got.plt       0000000000000000              .got.plt
0000000000601048 l    d  .data  0000000000000000              .data
```

```
0000000000601060 l    d  .bss   0000000000000000              .bss
0000000000000000 l    d  .comment.SUSE.OPTs      0000000000000000              .comment.SUSE.OPTs
0000000000000000 l    d  .comment       0000000000000000              .comment
0000000000000000 l    d  .debug_aranges 0000000000000000              .debug_aranges
0000000000000000 l    d  .debug_pubnames         0000000000000000              .debug_pubnames
0000000000000000 l    d  .debug_info    0000000000000000              .debug_info
0000000000000000 l    d  .debug_abbrev  0000000000000000              .debug_abbrev
0000000000000000 l    d  .debug_line    0000000000000000              .debug_line
0000000000000000 l    d  .debug_str     0000000000000000              .debug_str
0000000000000000 l    d  .debug_loc     0000000000000000              .debug_loc
0000000000000000 l    d  .debug_pubtypes         0000000000000000              .debug_pubtypes
0000000000000000 l    d  .debug_ranges  0000000000000000              .debug_ranges
0000000000000000 l    df *ABS*  0000000000000000              init.c
0000000000000000 l    df *ABS*  0000000000000000              initfini.c
000000000040080c l     F .text  0000000000000000              call_gmon_start
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
0000000000600de0 l     O .ctors 0000000000000000              __CTOR_LIST__
0000000000600df8 l     O .dtors 0000000000000000              __DTOR_LIST__
0000000000600e08 l     O .jcr   0000000000000000              __JCR_LIST__
0000000000400830 l     F .text  0000000000000000              __do_global_dtors_aux
0000000000601170 l     O .bss   0000000000000001              completed.5939
0000000000601178 l     O .bss   0000000000000008              dtor_idx.5941
00000000004008a0 l     F .text  0000000000000000              frame_dummy
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
0000000000600df0 l     O .ctors 0000000000000000              __CTOR_END__
0000000000400d08 l     O .eh_frame       0000000000000000              __FRAME_END__
0000000000600e08 l     O .jcr   0000000000000000              __JCR_END__
0000000000400b70 l     F .text  0000000000000000              __do_global_ctors_aux
0000000000000000 l    df *ABS*  0000000000000000              initfini.c
0000000000000000 l    df *ABS*  0000000000000000              swap1.cpp
0000000000601180 l     O .bss   0000000000000001              std::__ioinit
0000000000400a15 l     F .text  0000000000000040              __static_initialization_and_destruction_0(int, int)
0000000000400a55 l     F .text  0000000000000015              global constructors keyed to main
0000000000000000 l    df *ABS*  0000000000000000              elf-init.c
0000000000600fe8 l     O .got.plt        0000000000000000              .hidden _GLOBAL_OFFSET_TABLE_
0000000000600ddc l       .ctors 0000000000000000              .hidden __init_array_end
0000000000600ddc l       .ctors 0000000000000000              .hidden __init_array_start
0000000000600e10 l     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   0000000000000110              std::cout@@GLIBCXX_3.4
0000000000601050 g     O .data  0000000000000000              .hidden __dso_handle
0000000000600e00 g     O .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> >::oper
00000000004007c8       F *UND*  0000000000000000              std::basic_ostream<char, std::char_traits<char> >& std:
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 _ZStlsISt11char_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_traits<char> >&
    10: 00000000004007c8     0 FUNC    GLOBAL DEFAULT  UND _ZSt4endlIcSt11char_trait@GLIBCXX_3.4 (2)
    11: 0000000000400798     0 FUNC    GLOBAL DEFAULT  UND std::ios_base::Init::~Init()@GLIBCXX_3.4 (2)
    12: 0000000000601060   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: 0000000000400238     0 SECTION LOCAL  DEFAULT    1
     2: 0000000000400254     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000400274     0 SECTION LOCAL  DEFAULT    3
     4: 000000000040028c     0 SECTION LOCAL  DEFAULT    4
     5: 00000000004002b0     0 SECTION LOCAL  DEFAULT    5
     6: 00000000004002f8     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000400328     0 SECTION LOCAL  DEFAULT    7
     8: 0000000000400460     0 SECTION LOCAL  DEFAULT    8
     9: 00000000004005be     0 SECTION LOCAL  DEFAULT    9
    10: 00000000004005d8     0 SECTION LOCAL  DEFAULT   10
    11: 0000000000400618     0 SECTION LOCAL  DEFAULT   11
    12: 0000000000400648     0 SECTION LOCAL  DEFAULT   12
    13: 0000000000400720     0 SECTION LOCAL  DEFAULT   13
    14: 0000000000400738     0 SECTION LOCAL  DEFAULT   14
    15: 00000000004007e0     0 SECTION LOCAL  DEFAULT   15
    16: 0000000000400ba8     0 SECTION LOCAL  DEFAULT   16
    17: 0000000000400bb8     0 SECTION LOCAL  DEFAULT   17
    18: 0000000000400bc0     0 SECTION LOCAL  DEFAULT   18
    19: 0000000000400c08     0 SECTION LOCAL  DEFAULT   19
    20: 0000000000600de0     0 SECTION LOCAL  DEFAULT   20
    21: 0000000000600df8     0 SECTION LOCAL  DEFAULT   21
    22: 0000000000600e08     0 SECTION LOCAL  DEFAULT   22
    23: 0000000000600e10     0 SECTION LOCAL  DEFAULT   23
    24: 0000000000600fe0     0 SECTION LOCAL  DEFAULT   24
    25: 0000000000600fe8     0 SECTION LOCAL  DEFAULT   25
    26: 0000000000601048     0 SECTION LOCAL  DEFAULT   26
    27: 0000000000601060     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: 000000000040080c     0 FUNC    LOCAL  DEFAULT   15 call_gmon_start
    42: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    43: 0000000000600de0     0 OBJECT  LOCAL  DEFAULT   20 __CTOR_LIST__
    44: 0000000000600df8     0 OBJECT  LOCAL  DEFAULT   21 __DTOR_LIST__
    45: 0000000000600e08     0 OBJECT  LOCAL  DEFAULT   22 __JCR_LIST__
    46: 0000000000400830     0 FUNC    LOCAL  DEFAULT   15 __do_global_dtors_aux
    47: 0000000000601170     1 OBJECT  LOCAL  DEFAULT   27 completed.5939
    48: 0000000000601178     8 OBJECT  LOCAL  DEFAULT   27 dtor_idx.5941
    49: 00000000004008a0     0 FUNC    LOCAL  DEFAULT   15 frame_dummy
    50: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    51: 0000000000600df0     0 OBJECT  LOCAL  DEFAULT   20 __CTOR_END__
    52: 0000000000400d08     0 OBJECT  LOCAL  DEFAULT   19 __FRAME_END__
    53: 0000000000600e08     0 OBJECT  LOCAL  DEFAULT   22 __JCR_END__
    54: 0000000000400b70     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: 0000000000601180     1 OBJECT  LOCAL  DEFAULT   27 std::__ioinit
    58: 0000000000400a15    64 FUNC    LOCAL  DEFAULT   15 _Z41__static_initializati
    59: 0000000000400a55    21 FUNC    LOCAL  DEFAULT   15 global constructors keyed to main
    60: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS elf-init.c
    61: 0000000000600fe8     0 OBJECT  LOCAL  HIDDEN   25 _GLOBAL_OFFSET_TABLE_
    62: 0000000000600ddc     0 NOTYPE  LOCAL  HIDDEN   20 __init_array_end
    63: 0000000000600ddc     0 NOTYPE  LOCAL  HIDDEN   20 __init_array_start
    64: 0000000000600e10     0 OBJECT  LOCAL  HIDDEN   23 _DYNAMIC
    65: 0000000000601048     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: 0000000000400b60     2 FUNC    GLOBAL DEFAULT   15 __libc_csu_fini
    69: 00000000004007e0     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_t
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:
test, data und bss

Hinweis zu verfügbaren Softwareentwicklungssystemen:
GNU g++ für Linux
cygwin für Windows
Visual Studio 2010 für Windows

1.5.2 Erstellen und Benutzen von statischen Bibliotheken

*.a-Bibiotheken als Sammlungen von Objektdateien

Erzeugen statischer Bibliotheken

ar Manualpage

Static Libraries

Wo waren einmal statisch gelinkte Binaries positioniert?

```
> 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<<(
                 U std::basic_ostream<char, std::char_traits<char> >::operator<<(
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 =>  (0x00007fffdfbff000)
        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 =>  (0x00007fffc85fa000)
        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 =>  (0x00007fff6ebff000)
        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)

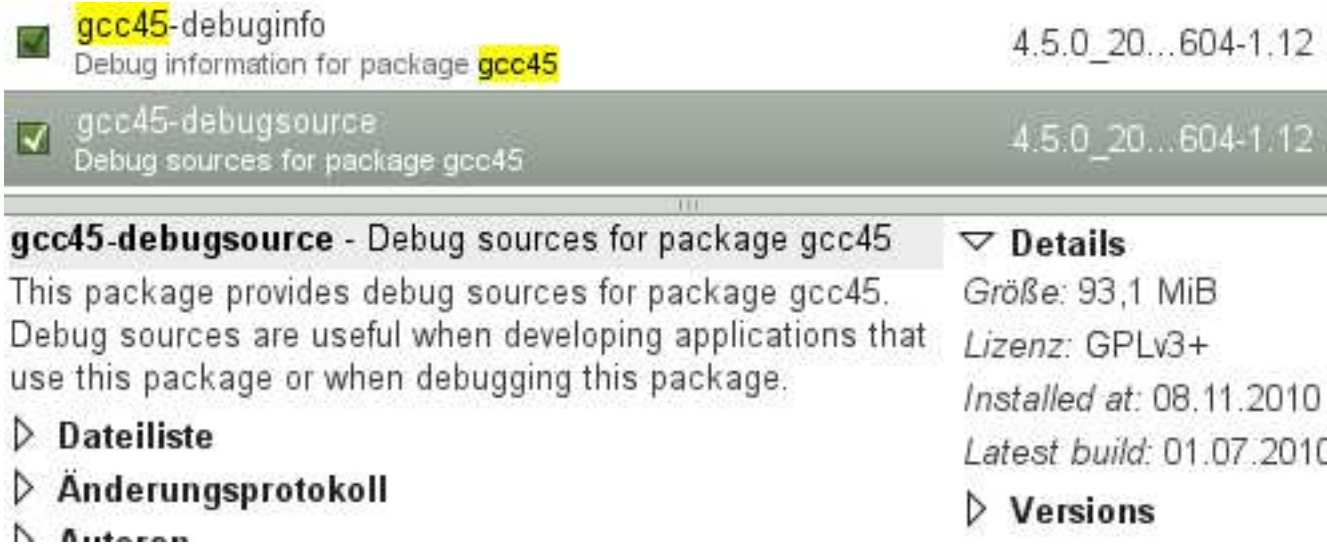
Creating shared object libraries
Anatomy of Linux dynamic libraries

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)

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



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:9:1: Fehler: Ungültige Anwendung von ≫sizeof≪ auf unvollständigen Typen ≫boost::STATIC_ASSERTION_FAILURE<false>≪
geomMittel2.cpp: In Funktion ≫double geomMittel2(const ArithmeticLike1&, const ArithmeticLike2&) [with ArithmeticLike1 = double, ArithmeticLike2 = char [2]]≪:
geomMittel2.cpp:28:33:   instantiated from here
geomMittel2.cpp:15:1: Fehler: Ungültige Anwendung von ≫sizeof≪ auf unvollständigen Typen ≫boost::STATIC_ASSERTION_FAILURE<false>≪
geomMittel2.cpp:17:25: Fehler: ungültige Operanden der Typen ≫const double≪ und ≫const char [2]≪ für binäres ≫operator*≪
```

Vergleiche is_arithmetic und type_traits der Boost

oder

1.7.2 mit Hilfe des c++0x-Modus des g++

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

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(::boost::is_arithmetic<ArithmeticLike1>::value, "
   ArithmeticLike1 is not arithmetic");
static_assert(::boost::is_arithmetic<ArithmeticLike2>::value, "
   ArithmeticLike2 is not arithmetic");

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

// uebersetze mit -std=c++0x
// oder make CXXFLAGS="-std=c++0x" ...

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 Funktion »double geomMittel2(const ArithmeticLike1&, const ArithmeticLike2&) [with ArithmeticLike1 = double, ArithmeticLike2 = char
geomMittel-sa2.cpp:29:33:   instantiated from here
geomMittel-sa2.cpp:14:1: Fehler: statische Behauptung gescheitert: "ArithmeticLike2 is not arithmetic"
geomMittel-sa2.cpp:16:25: Fehler: ungültige Operanden der Typen »const double« und »const char [2]« für binäres »operator*«
make: *** [geomMittel-sa2] Fehler 1
```

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 kompliziertere 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 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_arithmetic;
...
template <class T> struct is_const;
template <class T> struct is_trivially_copyable;
...
template <class T> struct is_abstract;

template <class T> struct is_same;
template <class T> struct is_base_of;
template <class T> struct is_convertible;
```

1.8.2 BOOST type_traits

Boost: Type Traits

```
template <class T> struct is_array;
template <class T> struct is_complex;
template <class T> struct is_void;
template <class T> struct is_integral;
...
template <class T> struct is_arithmetic;
...
template <class T> struct is_const;
template <class T> struct is_trivially_copyable;
...
template <class T> struct is_abstract;

template <class T> struct is_same;
template <class T> struct is_base_of;
template <class T> struct is_convertible;
template <class T> struct has_new_operator;
template <class T> struct has_nothrow_assign;
template <class T> struct has_nothrow_constructor;
...
template <class T> struct is_empty;
template <class T> struct is_polymorphic;
template <class T> struct has virtual_destructor;
```

1.8.3 numeric_limits als Typ-Abbildung

numeric_limits als generische Klasse

mit traits-ähnlichem Charakter für die Benutung zum Beispiel für Requirements von Tempate-Parametern:
UnsignedInt Template-Parameter

```
#include <limits>
#include <boost/static_assert.hpp>

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;
}
```

1.9 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.

1.10 Die Boost-Bibliothek

Boost.StaticAssert mit RandomAccessIterator, UnsignedInt, ...
The Boost Concept Check Library (BCCL)
enable_if für property based template overloading
(Boost.Foreach)
Math Special Functions

1.11 Wo ist die Template-Instanz?

Abschnitt 7.5: Where's the template?

Abschnitt 7.9: GNU-Compilerunterstützung für Type Traits

g++-Compileroptionen mit Template-Relevanz:

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

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

1.12 C++11 extern template

C++0x Draft
N1448

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 Fehlermeldungen bei uneingeschränkter Generizität

```
#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>::operat
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::comp
/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 == last - first) comparisons; if enough extra memory is available, it is $N \log(N)$.

4 *Remarks:* Stable.

SGI:

stable_sort

Algorithms
Category: algorithms

Function
Component type: function

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 `*j` is not less than `*i`. `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 `operator<`, 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

C++ Standard Concept Traits (abandoned Boost project):

```
::boost::is_less_than_comparable<T1,T2>::value
```

1.15 Verbesserte Fehlermeldungen bei Nutzung von StaticAssert

1.15.1 RandomAccessIterator

Zunächst die uneingeschränkt generische Variante:

```
#include <iostream>
#include <vector>
#include <set>
#include <complex>
#include <algorithm>
#include <iterator>
#include <boost/static_assert.hpp>
#include <boost/type_traits.hpp>

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

   //
   // detail goes here...
   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++0x" ra2b
```

```
g++ -std=c++0x -g -I. -I/home/user/include   ra2b.cpp   -o ra2b
```

und funktioniert einwandfrei, für

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

erscheint jedoch keine vernünftige Fehlermeldung

```
> make CXXFLAGS="-std=c++0x" ra2b
g++ -std=c++0x -g -I. -I/home/buhl/include   ra2b.cpp   -o ra2b
In file included from /usr/include/c++/4.5/bits/char_traits.h:41:0,
                 from /usr/include/c++/4.5/ios:41,
                 from /usr/include/c++/4.5/ostream:40,
                 from /usr/include/c++/4.5/iostream:40,
                 from ra2b.cpp:1:
/usr/include/c++/4.5/bits/stl_algobase.h: In static member function »static void
/usr/include/c++/4.5/bits/stl_algobase.h:138:7:   instantiated from »void std::it
/usr/include/c++/4.5/bits/stl_algo.h:1395:6:   instantiated from »void std::__rev
/usr/include/c++/4.5/bits/stl_algo.h:1441:7:   instantiated from »void std::rever
ra2b.cpp:19:4:   instantiated from »RandomAccessIterator foo(RandomAccessIterator
ra2b.cpp:30:28:   instantiated from here
/usr/include/c++/4.5/bits/stl_algobase.h:89:11: Fehler: Zuweisung der schreibgesc
/usr/include/c++/4.5/bits/stl_algobase.h:138:7:   instantiated from »void std::it
/usr/include/c++/4.5/bits/stl_algo.h:1395:6:   instantiated from »void std::__rev
/usr/include/c++/4.5/bits/stl_algo.h:1441:7:   instantiated from »void std::rever
ra2b.cpp:19:4:   instantiated from »RandomAccessIterator foo(RandomAccessIterator
ra2b.cpp:30:28:   instantiated from here
/usr/include/c++/4.5/bits/stl_algobase.h:90:11: Fehler: Zuweisung der schreibgesc
make: *** [ra2b] Fehler 1
```

RandomAccessIterator

Durch ein geeignetes statisches Assert der Art

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <set>
#include <complex>
#include <boost/static_assert.hpp>
#include <boost/type_traits.hpp>

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(
      (::boost::is_convertible<
         cat,
         const std::random_access_iterator_tag&>::value),
       "no random access iterator");
   //
   // detail goes here...
   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++0x" ra2
```

```
g++ -std=c++0x -g -I. -I/home/buhl/include   ra2.cpp   -o ra2
ra2.cpp: In Funktion »RandomAccessIterator foo(RandomAccessIterator, RandomAccess
ra2.cpp:36:28:   instantiated from here
ra2.cpp:18:4: Fehler: statische Behauptung gescheitert: "no random access iterato
...
```

1.15.2 Nicht instantiierbare Klassen

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

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_47_0/doc/html/boost_staticassert.html#boost_staticassert.templates

1.15.3 Erzwingung gleicher Typen

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

template <class UnsignedInt>
class myclass
{
private:
  static_assert(::boost::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;
}
```

1.15.4 Funktionen mit (int/float/...) type promotion

```
#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:

```
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 vergleiche auch: automatisch bestimmter return-Typ

1.15.5 Auf Unterklassen eingeschränkte Generizität

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

1.15.6 Überladene Templatefunktionen

1.15.6.1 enable_if-Funktionen

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

using namespace std;

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

// uebersetze mit -std=c++0x
// oder g++ CXXFLAGS="-std=c++0x" ...

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;
}
```

1.15.6.2 enable_if-Funktionsüberladen

3.2 Overlapping enabler conditions

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

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

1.15.6.3 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<::boost::is_integral<T> >::type> { ... };

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

1.16 Rückblick: typsichere Funktionsbenutzung

Der Prototypen (Signaturen) von Funtionen:

```
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

Im C++11-Standard (zumindest schon dokumentarisch, d.h. nicht maschinell überprüft) zusätzlich:
„Descriptions of function semantics contain the following elements (as appropriate):

- Requires: the preconditions for calling the function
- Effects: the actions performed by the function
- Synchronization: the synchronization operations (1.10) applicable to the function
- Postconditions: the observable results established by the function
- Returns: a description of the value(s) returned by the function
- Throws: any exceptions thrown by the function, and the conditions that would cause the exception
- Complexity: the time and/or space complexity of the function
- Remarks: additional semantic constraints on the function
- Error conditions: the error conditions for error codes reported by the function.
- Notes: non-normative comments about the function“

Ein Beispiel:

```
logic_error(const string& what_arg);
    Effects: Constructs an object of class logic_error.
    Postcondition: strcmp(what(), what_arg.c_str()) == 0.
```

Bemerkung:
Software durch Verträge/ DbC in C++(?)
Not ready for C++0x, but open to resubmit in future

1.17 Ausblick: C++2x eventuell mit eingeschränkter Generizität: Concepts (Prototypen von Klassen)

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 Konzept-Eigenschaften 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

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))

Link zu conceptg++:
http://www.generic-programming.org/software/ConceptGCC/download.php

Zitat: „ConceptC++ makes programming with C++ templates easier, because the 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++11 without Concepts
C++ Concepts: a Postmortem

1.18 Generic Programming

www.generic-programming.org mit:

- (viele/meherere/...) 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
  ...

- Concepts:
  Associative Container
  Assignable
  ...
  Default Constructible

1.19 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 Concepsts sein (Vererbung))

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

Requirement-Klauseln ermöglichen vollständige zielgerichtete Fehlermeldungen bei Konzeptverletzung sowohl bei Instantiierungsversuch eines Templates mit einem falschen generischen Parameter als auch bei der Nutung 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

1.20 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> >'
```

(vergleiche Concepts in Generic Programming, Concept Cpp)

1.21 Concepts, concept_maps, axioms

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
}
```

1.22 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 Instantiierungsversuch

```
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 Vervolllständigung 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;
  }
```

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 Instantiierung mittels

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

führt zur Fehlermeldung, dass Zeiger keine Forwarditeratoren sin.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.

Eine Sammlung von Konzepten findet man unter ConceptC++ Specification.

1.23 Generic Programming Techniques of the BOOST Librarie

Anatomy of a Concept
Traits
C++ type_traits
Tag dispatching
Adaptors
Type generators
Metaprogramming
Object generators
Policy classes
Policy-based design

1.24 SFINAE

SFINAE als Voraussetzung für `enable_if`:

```
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 (even though there
       is no int::foo) thanks to SFINAE.
}
```

1.25 Assoziierte Typen

Associated Types

Ein Beispiel:

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

1.26 POD-Typen

POD Types

1.27 Checking Concepts without Concepts in C++

static_assert, enable_if-pod-Overloading for copy():

```
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:

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++;
    }
}
```

2 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
```

2.1 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 Instantiierung nicht zu übersetzen, also auch nicht syntaktisch zu analysieren. Das Auftreten möglicher Fehlermeldungen ist deshalb bis zur Instantiierung 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 unerwähne 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 Archetypen auszustatten und diese testzucompilieren (siehe Abschmitt 1.21.3). Die in der BCCL vordefinierten für die STL nötigen Konzepte sind in der Datei `boost/concept_archetype.hpp` mit Archetypen ausgestattet.

  Zum Beispiel das Konzept `InputIterator` mit den geforderten Operationen `++i, (void)i++, *i++,*i; Defaultkonstruktor, operator=, operator->` `(TrivialIterator); Kopierkonstruktor, swap() (Assignable);` `operator==, operator!= (EqualtityComparable); 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 Syntaxremappping (temporäres renaming, Context_maps) **nicht**.
- BCCL unterstützt Kontext-basiertes Überladen **nicht**. (Ausweg: `enable_if` mit Concept Traits)

2.2 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() {
#if !defined(_ITERATOR_) // back_insert_it broken for VC++ STL
      a = a;                 // require assignment operator
#endif
      const_constraints(a);
    }
    void const_constraints(const TT& b) {
#if !defined(_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);
#if !defined(_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);
#if !defined(_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;
};
```

2.3 Ein Blick zurück (2003..2008) — und vorwärts (202x)? 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;
}
```

Pseudo-Signaturen (2008..2009):

C++11-FAQ:

„Since it is likely that concepts in some form or other will be part of a later version of C++, I have not deleted the concept sections from this document.“ (Stroustrup)

ConceptClang (2011):

```
concept Hashset<typename X> : HasEmpty<X> {
  typename element;
  requires (Hashable<element>);
  int size(X);
}
// modeling
template<typename K>
requires (Hashable<K>)
concept_map Hashset<list<K>> {
  typedef K element;
  int size(list<K> m) { ... }
}
// ...
```

Concepts: Linguistic Support for Generic Programming in C++
ConceptC++ Publications
The C++0x “Concepts” Effort

2.4 Creating Own Boost Concept Checking Classes

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 `EqualitiyComparable`)
- 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

2.5 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 Instantiierungsversuch 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);
```

2.6 Programmieren mit Konzepten

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

Konzepte 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-Instantiierung benutzt werden darf.

3 Trait-Klassen als Mittel der Absicherung vorhandener Eigenschaften aktueller generischer Parameter

3.1 Type Traits in D (Template Constraints)

Eingeschränkte Generizität in D:

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

und deren Anwendung für ADTs:

```
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

3.2 gcc Type-Traits

Type-Traits.html

3.3 C++11 type_traits

C++11 Kapitel 20.9

```
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;
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;
// 20.9.4.2, composite type categories:
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;
// 20.9.4.3, type properties:
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, class... Args> 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... Args> struct
   is_trivially_constructible;
template <class T> struct is_trivially_default_constructible;
template <class T> struct is_trivially_copy_constructible;
template <class T> struct is_trivially_move_constructible;
template <class T, class U> struct is_trivially_assignable;
template <class T> struct is_trivially_copy_assignable;
template <class T> struct is_trivially_move_assignable;
template <class T> struct is_trivially_destructible;
template <class T, class... Args> struct
   is_nothrow_constructible;
template <class T> struct is_nothrow_default_constructible;
template <class T> struct is_nothrow_copy_constructible;
template <class T> struct is_nothrow_move_constructible;
template <class T, class U> struct is_nothrow_assignable;
template <class T> struct is_nothrow_copy_assignable;
template <class T> struct is_nothrow_move_assignable;
template <class T> struct is_nothrow_destructible;
template <class T> struct has_virtual_destructor;
// 20.9.5, type property queries:
template <class T> struct alignment_of;
template <class T> struct rank;
template <class T, unsigned I = 0> struct extent;
// 20.9.6, type relations:
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;
// 20.9.7.1, const-volatile modifications:
template <class T> struct remove_const;
template <class T> struct remove_volatile;
template <class T> struct remove_cv;
template <class T> struct add_const;
```

```
template <class T> struct add_volatile;
template <class T> struct add_cv;
// 20.9.7.2, reference modifications:
template <class T> struct remove_reference;
template <class T> struct add_lvalue_reference;
template <class T> struct add_rvalue_reference;

// 20.9.7.3, sign modifications:
template <class T> struct make_signed;
template <class T> struct make_unsigned;

// 20.9.7.4, array modifications:
template <class T> struct remove_extent;
template <class T> struct remove_all_extents;
// 20.9.7.5, pointer modifications:
template <class T> struct remove_pointer;
template <class T> struct add_pointer;
// 20.9.7.6, other transformations:
template <std::size_t Len, std::size_t Align> struct
   aligned_storage;
template <std::size_t Len, class... Types> struct aligned_union
   ;
template <class T> struct decay;
template <bool, class T = void> struct enable_if;
template <bool, class T, class F> struct conditional;
template <class... T> struct common_type;
template <class T> struct underlying_type;
template <class> class result_of; // undefined
template <class F, class... ArgTypes> class result_of<F(
   ArgTypes...)>;
} // namespace std
```

3.4 Boost Type-Traits

Boost.TypeTraits

3.5 Boost Concept Traits Library

Concept Traits Library, Boost.UnderConstruction (abandoned)

3.6 C++0x mit ersten Concepts: Numerics Library

Draft N2736, ...

```
namespace std {
auto concept HasAbs<typename T> see below ;
auto concept HasAcos<typename T> see below ;
auto concept HasAsin<typename T> see below ;
auto concept HasAtan<typename T> see below ;
auto concept HasAtan2<typename T> see below ;
auto concept HasCos<typename T> see below ;
auto concept HasCosh<typename T> see below ;
auto concept HasExp<typename T> see below ;
auto concept HasLog<typename T> see below ;
auto concept HasLog10<typename T> see below ;
auto concept HasPow<typename T> see below ;
auto concept HasSin<typename T> see below ;
auto concept HasSinh<typename T> see below ;
auto concept HasSqrt<typename T> see below ;
auto concept HasTan<typename T> see below ;
auto concept HasTanh<typename T> see below ;
}

namespace std {
template<ArithmeticLike T> class complex;
template<> class complex<float>;
template<> class complex<double>;
template<> class complex<long double>;
}

template<FloatingPointType T> T abs(const complex<T>&);
template<FloatingPointType T> T arg(const complex<T>&);
template<ArithmeticLike T> T norm(const complex<T>&);
template<ArithmeticLike T> complex<T> conj(const complex<T>&);
template<ArithmeticLike T> complex<T> fabs(const complex<T>&);
template<FloatingPointType T> complex<T> polar(const T&, const
   T& = 0);
// 26.3.8 transcendentals:
```

```
template<FloatingPointType T> complex<T> acos(const complex<T
    >&);
template<FloatingPointType T> complex<T> asin(const complex<T
    >&);
template<FloatingPointType T> complex<T> atan(const complex<T
    >&);
template<FloatingPointType T> complex<T> acosh(const complex<T
    >&);
template<FloatingPointType T> complex<T> asinh(const complex<T
    >&);
template<FloatingPointType T> complex<T> atanh(const complex<T
    >&);
template<FloatingPointType T> complex<T> cos (const complex<T
    >&);
template<FloatingPointType T> complex<T> cosh (const complex<T
    >&);
template<FloatingPointType T> complex<T> exp (const complex<T
    >&);
template<FloatingPointType T> complex<T> log (const complex<T
    >&);
template<FloatingPointType T> complex<T> log10(const complex<T
    >&);
template<class T> complex<T> pow(const complex<T>&, int);
template<FloatingPointType T> complex<T> pow(const complex<T>&,
     const T&);
template<FloatingPointType T> complex<T> pow(const complex<T>&,
     const complex<T>&);
template<FloatingPointType T> complex<T> pow(const T&, const
    complex<T>&);
template<FloatingPointType T> complex<T> sin (const complex<T
    >&);
template<FloatingPointType T> complex<T> sinh (const complex<T
    >&);
template<FloatingPointType T> complex<T> sqrt (const complex<T
    >&);
template<FloatingPointType T> complex<T> tan (const complex<T
    >&);
template<FloatingPointType T> complex<T> tanh (const complex<T
    >&);
```

3.7 Das ConceptWeb in ConceptC++

ConceptWeb

3.8 Die (dokumentarisch genutzten) Konzepte der SGI STL — Übersicht

Vergleiche http://www.sgi.com/tech/stl/stl_index.html

- Functors

  ```
  Adaptable Binary Function
  Adaptable Binary Predicate
  Adaptable Generator
  Adaptable Predicate
  Adaptable Unary Function
  Binary Function
  Binary Predicate
  Generator
  Hash Function
  MonoidOperation
  Predicate
  Random Number Generator
  Strict Weak Ordering
  Unary Function
  ```

- Utilities

  ```
  Assignable
  Character Traits
  Default Constructible
  Equality Comparable
  LessThan Comparable
  ```

- Containers

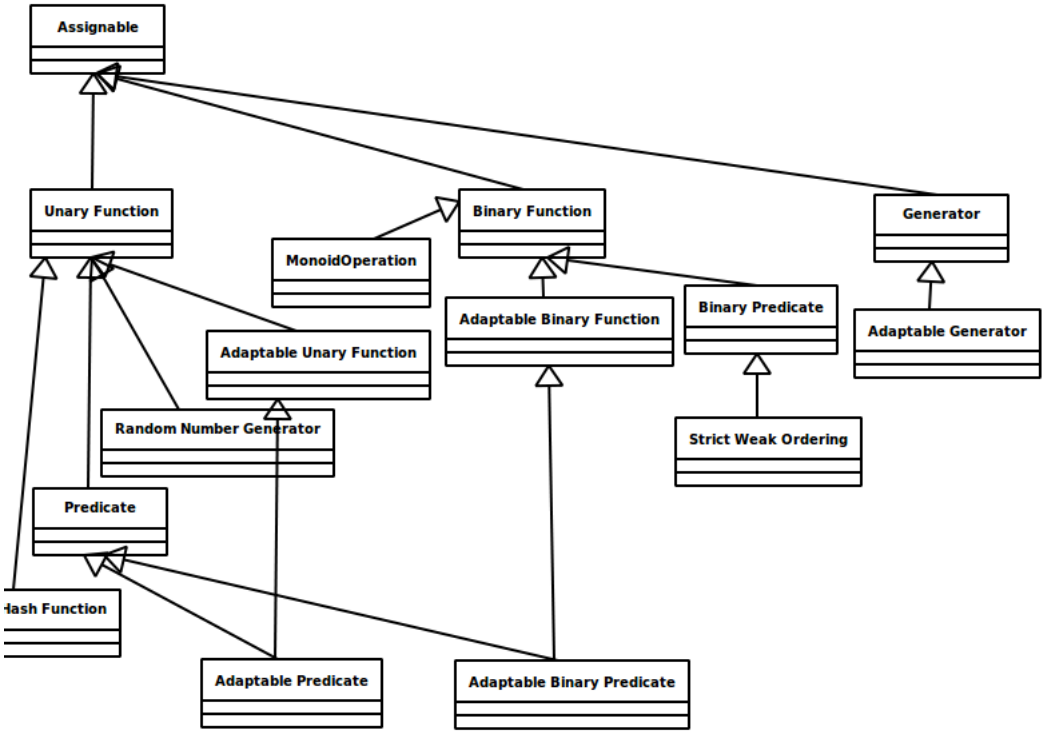
  ```
  Associative Container
  Back Insertion Sequence
  Container
  Forward Container
  Front Insertion Sequence
  Hashed Associative Container
  ```

Multiple Associative Container
Multiple Hashed Associative Container
Multiple Sorted Associative Container
Pair Associative Container
Random Access Container
Reversible Container
Sequence
Simple Associative Container
Sorted Associative Container
Unique Associative Container
Unique Hashed Associative Container
Unique Sorted Associative Container

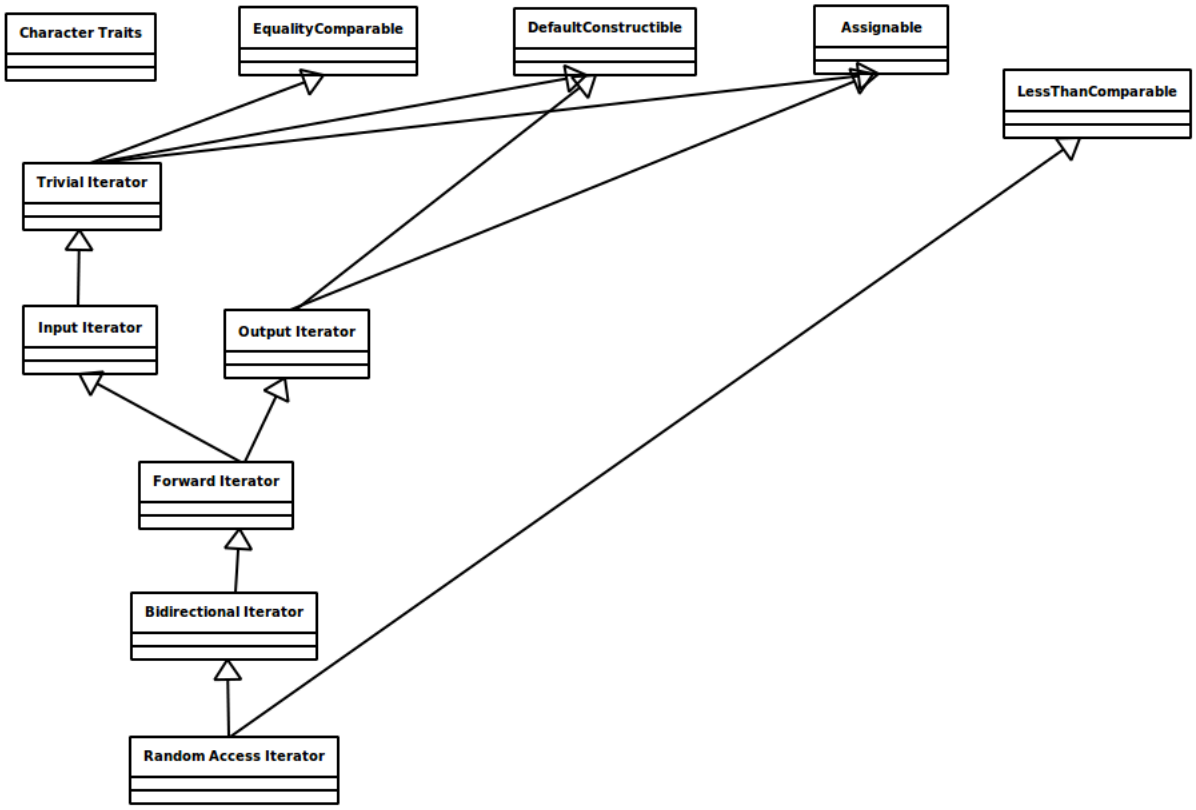
- Iterators

Bidirectional Iterator
Forward Iterator
Input Iterator
Output Iterator
Random Access Iterator
Trivial Iterator

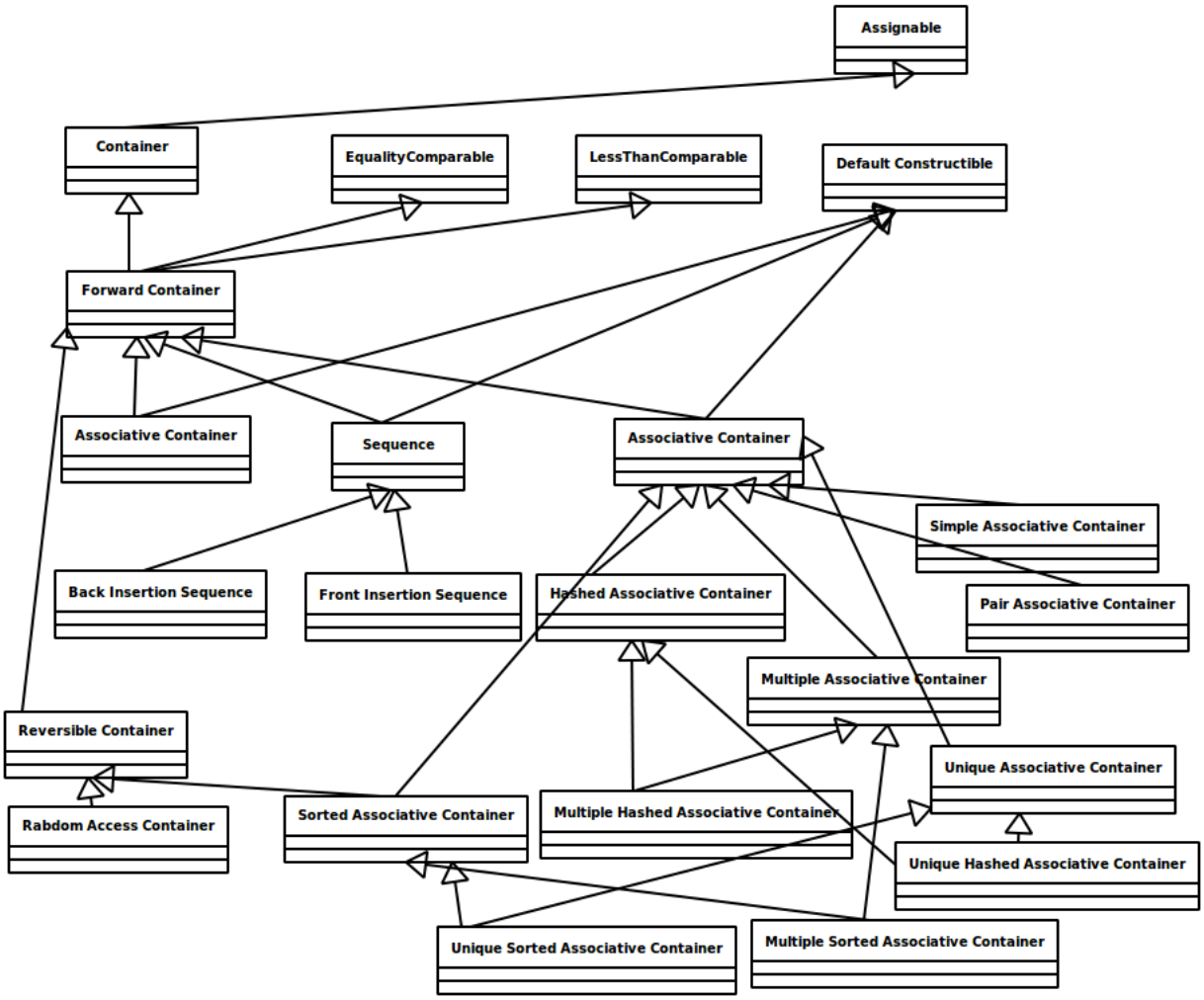
Verfeinerungshierarchie der Functor-Konzepte:



Verfeinerungshierarchie der Iterator-Konzepte:



Verfeinerungshierarchie der Container-Konzepte:



3.9 Glossar der generischen Programmierung

Konzept A concept contains a set of requirements that describe a family of abstractions, typically data types. Examples of concepts include InputIterator, Graph, and EqualityComparable.

Requirement/Anforderung A requirement is part of a concept that describes the behavior of an abstraction. Requirements tend to be syntactic (e.g., all InputIterators have a dereference operation), semantic (e.g., one can traverse the sequence of values returned from a ForwardIterator muliple times), or performance-related (e.g., incrementing an InputIterator occurs in constant amortized time).

Modell A model is a type or set of types that meets the requirements of a concept. An integer pointer is a model of the InputIterator concept. „Model“ can also be used as a verb to describe the relationship between a type or set of types and a concept, e.g., an adjacency list models the Graph concept.

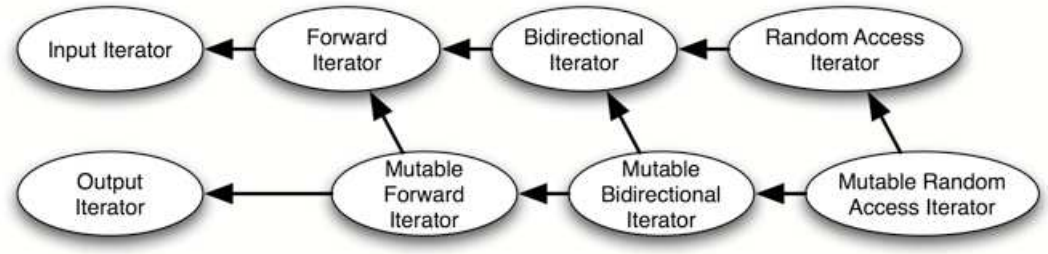
Archetyp An archetype class is an exact implementation of the interface associated with a particular concept. The run-time behavior of the archetype class is not important, the functions can be left empty. A simple test program can then be compiled with the archetype classes as the inputs to the component. If the program compiles then one can be sure that the concepts cover the component.

assoziierter Typ 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 InputIterator 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.

konzeptbasiertes Überladen Concept-based overloading selects the most specific algorithm from a set of specializations of a given algorithm.

3.10 Verallgemeinerung von Algorithmen: Lifting

- Iterative finding communality among similar implementations
  1. Constructing template version, using default constructor (documenting requirements)
  2. Lifting Containers (instead of arrays)
  3. Lifting iterators (instead of indexing), avoiding modifying types
- Concepts = organized lists of requirements
- nested requirements
- assoziierte Typen
- hirarchisch verschachtelte Anforderungen
- Beispiel: Concept taxonomy Iterator



3.11 Modelle für Konzepte

http://www.generic-programming.org/about/intro/models.php

3.12 Retroaktive Modellierung (rückwirkend gültig, mit Methoden-Namen-„Umbenennung“ gültig)

http://www.generic-programming.org/about/glossary.php?term=retroactive modeling
http://www.generic-programming.org/faq/?category=paradigms#object-oriented-programming

Temporäres `rename` von Methoden zum Beispiel in Eiffel (hier zur Auflösung von Namenskonflikten bei der Mehrfachvererbung oder zur verbesserten Verbalisierung):

```
class Multiindex inherit
   ARRAY[CARDINAL]rename
                        count as Dimension,
                        clear_all as Null
                     redefine
                        abs,
                        put,
                        make
                     undefine
                        has
                     end;
feature
   abs: CARDINAL;
   put(v:like item; i:INTEGER)
       - - replace i-th entry, if in index interval, by v
       ⋮
end - - class Multiindex
```

Temporäres Umbenennen (map) von Methoden eines Datentyps zur Modellierung von Konzepten in ConceptC++:
http://www.generic-programming.org/languages/conceptcpp/tutorial/#adapting
„Concept maps: show *how* a set of types meets the requirements of a concept.“
(aus: Taming C++ Templates with Concepts)

```
concept DreieckLike<typename D>{
    typename length_type;
    requires std::FloatingPointLike<length_type>;

    length_type getSide(const D&);
    length_type getHeight(const D&);
    //...
}

concept ParallelogrammLike<typename P>{
    typename length_type;
```

```
    requires std::FloatingPointLike<length_type>;

    length_type getSide(const P&);
    length_type getHeight(const P&);
    //...
}

class Dreieck{
    public:
     float a; float b; float c;
     float h_a; float h_b; float h_c;
};

class Parallelogramm{
    public:
      double a; double b;
      double h_a; double h_b;
};

concept_map DreieckLike<Dreieck>{
    typedef float length_type;
    float getSide(const Dreieck& d){ return d.a; };
    float getHeight(const Dreieck& d){ return d.h_a; };
};

concept_map ParallelogrammLike<Parallelogramm>{
    typedef double length_type;
    double getSide(const Parallelogramm& p){ return p.b; };
    double getHeight(const Parallelogramm& p){ return p.h_b; };
};

template <DreieckLike D>
D::length_type Flaeche(const D& d){
    return 0.5 * getSide(d) * getHeight(d);
};

template <ParallelogrammLike P>
P::length_type Flaeche(const P& p){
     return getSide(p) * getHeight(p);
};
```

Language Comparison:
A Comparative Study of Language Support for Generic Programming

3.13 Checking Concepts without Concepts: Eigene Concept Traits

(http://drdobbs.com/article/print?articleId=227500449)

Kombinierte Concept-Nutzung in ConceptC++ (Sets of Requirements):

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

Nach 2009 („Aus“ für Concepts) zurück zu:

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

mit textuellen Requirement-Beschreibungen in Form des SGI-STL-Manuals:

```
    * Iter is a model of Input Iterator.
    * T is a model of Assignable.
    * If x is an object of type T and y is an object of InputIterator's
      value type, then x + y is defined.
    * The return type of x + y is convertible to T.
```

Diese textuellen Requirements werden leider nicht maschinell beim Instantiieruingsversuch überprüft!

`static_asserts` mit geeigneten `type_traits`:

```
template<typename Iter, typename T>
T sum(Iter first, Iter last, T result)
{
  static_assert(NNN::is_input_iterator<Iter>::value, "Iter kein InputIterator!");
```

Vergleiche StandardConceptTraits oder jaredhoberock-thrust-concept oder http://fcppt.net/fcppt/type_traits/is_input_iterator.html oder ...

```
...
  static_assert(std::is_assignable<T,T>::value, "T not assignable!");
```

Vergleiche C++11, Seite 583.

```
...
  static_assert(::boost::has_plus_assign<
                                          T,
                                          iterator_traits<iter>::value_type,
                                          T
                                         >::value, "no plusassign!");
  for (; first != last; ++first)
    result += *first;
  return result;
}
```

Vergleiche boost::has_plus_assign.

(Alternativ `BOOST_CONCEPT_ASSERT(())`, `BOOST_CONCEPT_REQUIRE(())`, isbesondere wenn vordefinierte CONCEPTs existieren. Muß man eigene CONCEPTs definieren ist der technische Aufwand (Archetypes) eventuell abschreckend.)

Ohne BCCL bietet sich die Erstellung eigener Trait-Klassen als Gruppierungsmittel von Sets von Requirements zu „Concepts“ an, die man analog zu Seite 577 des C++11-Standards

```
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;
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;
```

```
// 20.9.4.2, composite type categories:
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;
// 20.9.4.3, type properties:
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, class U> struct is_assignable;
template <class T> struct is_copy_assignable;
template <class T> struct is_move_assignable;
...
```

etwa folgendermaßen erstellen kann:

```
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
  >
{};
```

Dieses selbserstellte Konzept

```
is_summable<.,.>
```

ist dann in einem einzigen `static_assert()` oder sogar zum „Concept“-basierten Überladen gemäß Abschnitt 1.15.6.2f nutzbar.

Aufgabe: Erstellen Sie ein Konzept
`is_input_iterator<.>`
nach diesem Muster gemäß Abschnitt 1.15.1.

3.14 Design of Concept Libraries for C++ (2011)

Design of Concept Libraries for C++ (2011), Sutton, Sroustrup

Concepts = Constraints + Axioms

Table 1. Concepts, constraints, and axioms

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

```
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);
}
```

4 Metaprogrammierung

4.1 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 1.15.4).

```
  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://www2.math.uni-wuppertal.de/images/pdf.gif beziehungsweise in
http://publib.boulder.ibm.com/infocenter/comphelp/v9v111/index.jsp?topic=/com.ibm.xlcpp9.aix.doc/standlib/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.

„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)

4.2 integral_constant

Metafunktion `integral_constant` in `type_traits`:

```
template <class _T, _T _V>
 struct integral_constant
 {
    static  const _T                    value = _V;
    typedef _T                          value_type;
    typedef integral_constant<_T,_V> type;
 };
 typedef integral_constant<bool, true> true_type;
 typedef integral_constant<bool, false> false_type;
```

(http://publib.boulder.ibm.com/infocenter/comphelp/v9v111/index.jsp?topic=/com.ibm.xlcpp9.aix.doc/standlib/header_type_traits.htm)

```
// C++0x type_traits -*- C++ -*-
// @file /usr/include/c++/4.5/type_traits
//...
/// is_rvalue_reference
  template<typename>
    struct is_rvalue_reference
    : public false_type { };

  template<typename _Tp>
    struct is_rvalue_reference<_Tp&&>
    : public true_type { };
// ...
/// is_unsigned
  template<typename _Tp>
    struct is_unsigned
    : public integral_constant<bool, (is_arithmetic<_Tp>::value
                                      && !is_signed<_Tp>::value
                                      )>
    { };
 // ...
 // Define a nested type if some predicate holds.
 // Primary template.
 /// enable_if
  template<bool, typename _Tp = void>
    struct enable_if
    { };
 // Partial specialization for true.
  template<typename _Tp>
    struct enable_if<true, _Tp>
    { typedef _Tp type; };
// ...
```

4.3 enum vs. static const vs. static constexpr

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
}
```

$$
\begin{aligned}
&// \\
&// \; C(k,n) = \frac{n!}{k!\;(n-k)!} \\
&//
\end{aligned}
$$

```
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. Siehe auch ldexp() und frexp().

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;
};
```

4.4 Typfunktionen

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 };
};
```

4.5 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.

4.6 Fortgeschrittene Metaprogrammierung

4.6.1 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, 1000000000000000000000000> yocto; // see below
typedef ratio<1, 1000000000000000000000> zepto; // see below
typedef ratio<1, 1000000000000000000> atto;
typedef ratio<1, 1000000000000000> femto;
typedef ratio<1, 1000000000000> pico;
typedef ratio<1, 1000000000> nano;
typedef ratio<1, 1000000> 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");
```

4.6.2 Unrolled Loops

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;
}
```

Codedissassembly:

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)

```
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
```

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

Endrekursion
Tailrekursion

4.6.3 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.
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);
}
...
// verzoegerte 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++

4.7 Nachteile der Metaprogrammierung (Forts.)

Metaprogrammierung-Nachteile Seite 52
Metaprogrammierung-Nachteile Seite 17

4.8 Die BOOST Metaprogramming Library MPL

Will man selbst Metafunktionsbibliotheken schreiben sollte man die MPL nutzen:
MPL

4.9 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

4.10 Spracherweiterung Maßeinheiten

4.10.1 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.

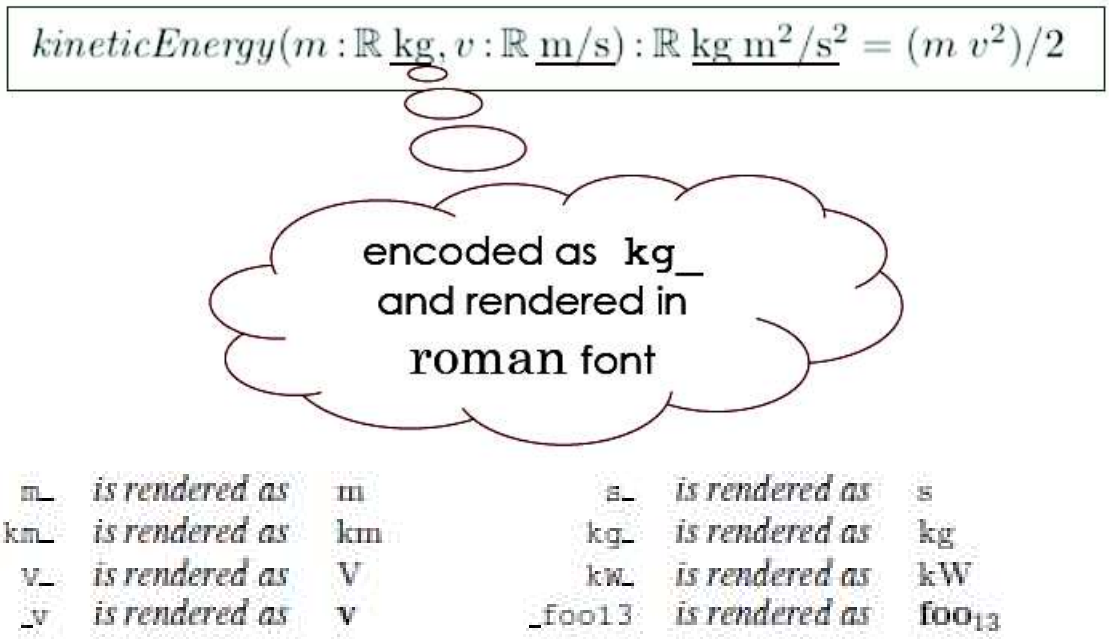
Einheiten können durch geeignete Klassen (`Euro, Franken, Pfund,` ... statt `double`) mit (automatisch durchgefürten) Typkonversionen ähnlich wie in

```
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;
}
```

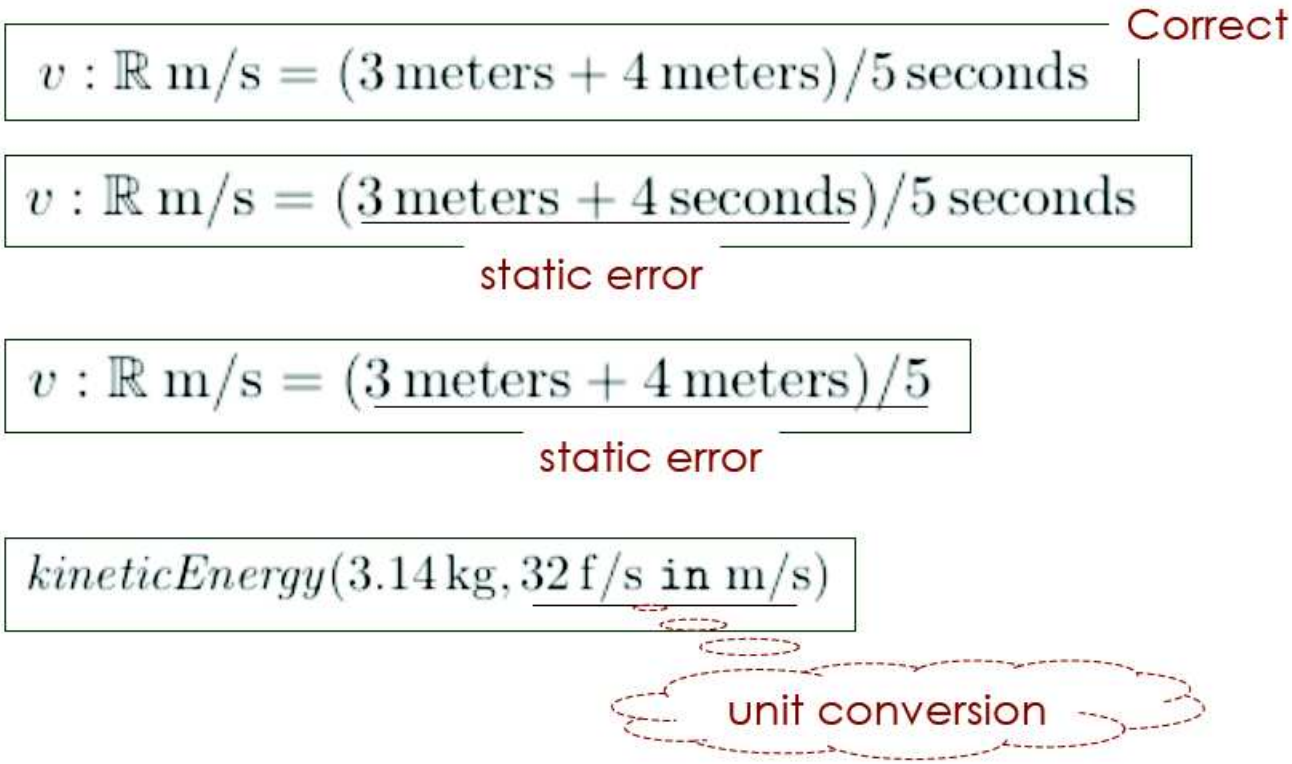
realisiert werden.

Flexibler ist es jedoch, wenn die Programmiersprache Maßeinheiten unterstützt:

- Units und Dimensions in Fortress:



Vgl. http://www.ipl.t.u-tokyo.ac.jp/~takeichi/attachments/Fortress.pdf



- 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, 1000000000000000000000> zepto; // see below
typedef ratio<1, 1000000000000000000> atto;
typedef ratio<1, 1000000000000000> femto;
typedef ratio<1, 1000000000000> pico;
typedef ratio<1, 1000000000> nano;
typedef ratio<1, 1000000> 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
```

stehen, was die Unterstützung von Maßeinheiten angeht.

Kann uns da Metaprogrammierung helfen?

4.10.2 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

4.10.3 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 FeedPerMetre = 3.28084<ft/m>
...
let heightOfMyOfficeWindowInMetres = heightOfMyOfficeWindow /
   FeedPerMetre
...
type float = float<1>
...
```

Generic Units
Parameterized Types

4.10.4 SI-Einheitssystem

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.

4.10.5 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

Meßungenauigkeiten 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
```

4.10.6 Erweiterung des C++-Typsystems durch Units

... mit Hilfe der Boost MPL-Library (Implementierungsidee 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
```

multiplicate:

```
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 `typedef`s an „Typnamen-Aliases" gebunden, ...)

4.10.7 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)

4.11 Literaturhinweise zum Metaprogrammieren

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

5 Template template-Parameter, Policy-basiertes Klassendesign

5.1 Templates als Template-Parameter

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;
```

5.2 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
  }
  // ...
}
```

Policies (Implementierungsvarianten/-verhaltensweisen) der STL-Container: (Vergleiche Seite 92)

```
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:

```
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: OverflowhHandler, Float2IntRounder, RawConverter, UserRangeChecker
Policies and the STL: AllocationPolicy, CharTPolicy
Policy-basiertes Klassendesign, CreationPolicy, Seite 16: CheckingPolicy, ThreadingPolicy

5.3 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
Wikibook Entwurfsmuster

5.4 Orthogonale Policy-Dimensionen

A Case for Orthogonality in Design
Orthogonality

Policies sollten minimale orthogonale Implementierungsvarianten sein.

5.5 Policies (Fortsetzung)

Policy-based Design — Definition

5.6 Aspektorientiertes Programmiern in komplexen Unternehmensanwendungen

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.

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.
- 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: join point, pointcut, advice
AspectC++ Quick Reference
AspectC++ Home

Aspect-Oriented Programming with C++ and AspectC++
An Introduction to AOP
Eclipse AspectC/C++ Development Tools

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 Recompilation 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 Sprachkonstruckte von C++, z.B. Annotationen mit Aspekt-Bezug, ...)

Vorteile:
Schnell und einfach aufzusetzen
selektiv einsetzbar
keine Modifikation der Originalquellen nötig
leicht entfernbar
gute Performance

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