Modernes C++

Mit C++98, C++11 und C++14

FJW

20. 6. 2014

C++

1983 - Bjarne Stroustrup bolts everything he's ever heard of onto C to create C++. The resulting language is so complex that programs must be sent to the future to be compiled by the Skynet artificial intelligence. Build times suffer. Skynet's motives for performing the service remain unclear but spokespeople from the future say "there is nothing to be concerned about, baby," in an Austrian accented monotones. There is some speculation that Skynet is nothing more than a pretentious buffer overrun.

— James Iry: A Brief, Incomplete, and Mostly Wrong History of Programming Languages

Grundprinzipien

RAII

- Implizites Ressourcen-Managment

```
void safe_fun(std::string path) {
    path += "file.txt"; //can throw
    std::ifstream file{path};
    util::enforce_opened(file, path); //can throw
    // string-reallocation can throw:
    for (std::string line; getline(file, line);) {
        std::lock_guard<std::mutex> guard{global_mutex};
        global_vec.emplace_back(line); //can throw
    }
}
```

RAII

- Implizites Ressourcen-Managment

```
void safe_fun(std::string path) {
    path += "file.txt"; //can throw
    std::ifstream file{path};
    util::enforce_opened(file, path); //can throw
    // string-reallocation can throw:
    for (std::string line; getline(file, line);) {
        std::lock_guard<std::mutex> guard{global_mutex};
        global_vec.emplace_back(line); //can throw
    }
}
```

- kein explizites Ressourcen-Handling,
- trotzdem keine Leaks (Speicher, Filehandles, Mutex),

Ownership

- Problem: Shared Mutability
  - Lösung in funktionalen Sprachen: Mutability verbieten
  - Lösung in C++: Unterscheidung zwischen Besitzer und Benutzern

Ownership

- Problem: Shared Mutability
  - Lösung in funktionalen Sprachen: Mutability verbieten
  - Lösung in C++: Unterscheidung zwischen Besitzer und Benutzern

- Besitzer darf Dinge verändern, zerstören und Zugriff gewähren
- Sofern sinnvoll möglich: Genau **ein** Besitzer

Ownership

- Problem: Shared Mutability
  - Lösung in funktionalen Sprachen: Mutability verbieten
  - Lösung in C++: Unterscheidung zwischen Besitzer und Benutzern

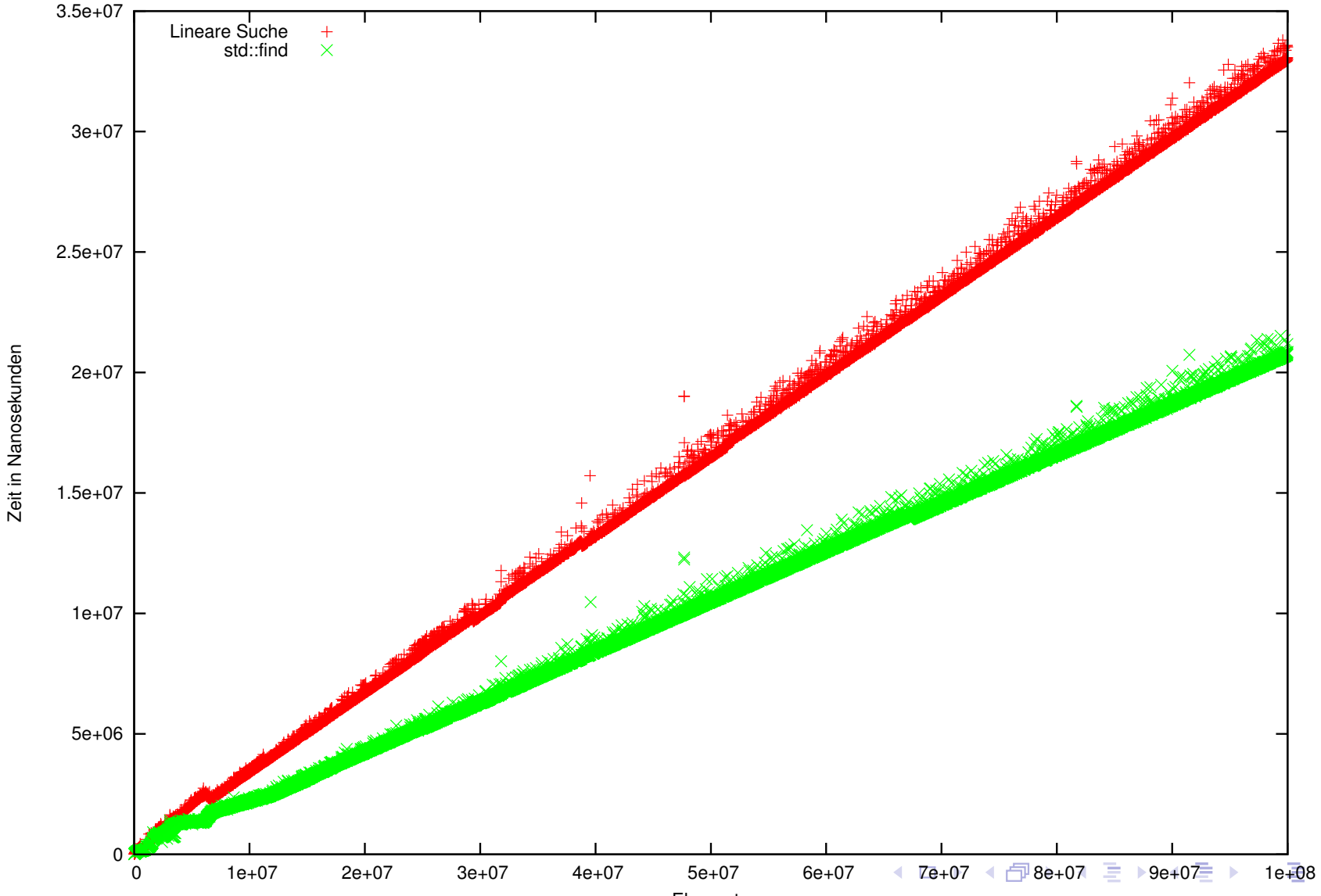
- Besitzer darf Dinge verändern, zerstören und Zugriff gewähren
- Sofern sinnvoll möglich: Genau **ein** Besitzer

```
int main() {
    std::string str; // owner = main
    str += "foobar"; // changes by owner are fine
    fun(str); // main() stays owner, should only
              // pass str by const reference
    // cleanup by main(), because it is the owner
}
```

Low- vs. Highlevel

- „Lightweight Abstraction Language“ (Stroustrup)
  - Die Abstraktionen sind leichtgewichtig, nicht die Sprache
- Inline-C für die Implementierung dieser Abstraktionen
- Lowlevel ist selten schneller, aber oft schwer verständlich

Highlevel = schnell



Bibliotheken

- Bibliothekslösungen statt nativer Sprachunterstützung:
  - `std::string` statt eingebautem Strings
  - `std::vector` statt verwendbaren eingebauten Arrays
  - `std::tuple` statt eingebauter Tupel
  - ...
- Kein Kompromis bezüglich Performance

Pass-by-Value vs. Pass-by-Reference

```
class my_class {
public:
    my_class(std::string str): str{std::move(str)} {}
    // my_class(const std::string& str): str{str} {}

private:
    std::string str;
};

int main() {
    my_class var1{"some pretty long string"};
    std::string str = "long string with many chars";
    my_class var2{str};
    my_class var3{std::move(str)};
}
```

Metaprogrammierung

- „$O(0)$“-Berechnungen
- Bedingte Vererbung
- Policies

C++14

Übersicht

- Viele kleine Verbesserungen
- Große Änderungen erst wieder in C++17
- Keine Science-Fiction!

Literale

- Binärliterale: `0b0011 == 3`
- Integer-Trennzeichen: `1'000'000'000`

Lambdas

- polymorphe Lambdas

```
[](auto a, auto b){return a+b;}
```

- Allgemeineres Lambda-Capture

```
[ptr = std::move(ptr)] {return ptr->foo();}
```

Funktionen

- Typinferenz für normale Funktionen

```
auto fun() {return 42;}
```

- Liberalere constexpr-Funktionen

```
constexpr unsigned exp(unsigned b, unsigned e) {
    unsigned returnval = 1;
    for(unsigned i = 0; i < e; ++i) {
        returnval *= b;
    }
    return returnval;
}
```

Templates

- Variable-Templates

```
template<typename T> constexpr T pi = 3.14159;
```

Sonstige Kernsprachenfeatures

- deprecated-Attribut

```
[[deprecated("use new_good_function instead")]]
int* ancient_bad_function();
```

- Verbesserte Aggregat-Initialisierung:

```
std::array<int, 3> arr = {1, 2, 3};
```

std::make_unique

```
// efficient thanks to perfect forwarding:
auto ptr = std::make_unique<std::string>("foo");
```

- Teil von „Never write new again“
- Exception-safe

```
//spot the bug:
void f(std::unique_ptr<int>, std::unique_ptr<int>);

f(std::unique_ptr<int>(new int(3)),
  std::unique_ptr<int>(new int(4)));

f(std::make_unique<int>(3),
  std::make_unique<int>(4));
```

Benutzerdefinierte Literale für die Standard-Bibliothek

- Namensraum `std::literals::foo_literals`
  - `literals` und `foo_literals` sind inline-Namensräume
  - `using namespace std::literals`
  - `using namespace std::string_literals`

```
using namespace std::literals;

int main() {
    auto s = "foo"s; // std::string
    auto seconds = 3s; //std::chrono::seconds
}
```

Robustere Sequenz-Algorithmen

- Nicht-modifizierende Sequenz-Algorithmen erhalten Ende-Iterator für die zweite Sequenz
- Betrifft `std::equal`, `std::mismatch`, und `std::is_permutation`

```
std::vector<int> vec1 = {1,2,3}, vec2 = {1,2,3,4};

// oops: returns true
std::equal(begin(vec1), end(vec1), begin(vec2));

// oops: UB
std::equal(begin(vec2), end(vec2), begin(vec1));

// correct: returns false
std::equal(begin(vec1), end(vec1),
           begin(vec2), end(vec2));
```

„Quoted“ Strings

- Problem:

```
std::stringstream ss;
std::string str;
ss << "foo bar";
ss >> str; // "foo"
```

- Lösung:

```
ss << std::quoted("foo bar");
ss >> std::quoted(str); // "foo bar"
```

- `std::quoted("x\"\\$ y") // -> "x\"\\$ y"`
- `std::quoted("x\"\\$ y", '$', '%') // -> $x"\%$ y$`

Tuple-Indexierung via Typ

```
using std::string;
std::tuple<string, string, int> t("foo", "bar", 7);

int i = get<int>(t); // i == 7
int j = get<2>(t); // Equivalent to the above: j == 7
string s = get<string>(t); // Compile-time error
```

std::exchange

- Weist einer Variablen einen neuen Wert zu und gibt den alten zurück

```
template<typename T, typename U=T>
T exchange(T& obj, U&& new_val) {
    T old_val = std::move(obj);
    obj = std::forward<U>(new_val);
    return old_val;
}
```

The benefit isn't huge, but neither is the specification cost.

Shared Locking

- std::shared_timed_mutex
- std::shared_lock

```
using namespace std::literals;
std::shared_timed_mutex mut;
using guard = std::shared_lock<std::shared_timed_mutex>;

void reader() {
    if(guard g{mut, 100us}){
        read();
    } else {
        panic();
    }
}
```

Concepts TS

Motivierendes Beispiel

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

int main() {
    std::list<int> lst = {4, 3, 2, 1};
    sort(begin(lst), end(lst));
}
```

Clang und GCC sind empört!

```
In file included from /usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/algorithm:62:
/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_algo.h:1868:22: error: invalid operands to binary expression ('std::_List_iterator<int>' and 'std::_List_iterator<int>')
      std::__lg(__last - __first) * 2,
                        ~~~~~~~ - ~~~~~~
/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_algo.h:4876:12: note: in instantiation of function template specialization 'std::__sort<std::_List_iterator<int>, __gnu_cxx::__ops::_Iter_less_iter>' requested here
    std::__sort(__first, __last, __gnu_cxx::__ops::__iter_less_iter());

main.cpp:15:7: note: in instantiation of function template specialization 'std::sort<std::_List_iterator<int> >' requested here
    std::sort(begin(lst), end(lst));

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_bvector.h:208:3: note: candidate function not viable: no known conversion from 'std::_List_iterator<int>' to 'const std::_Bit_iterator_base' for 1st argument
  operator-(const _Bit_iterator_base& __x, const _Bit_iterator_base& __y)

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:328:5: note: candidate template ignored: could not match 'reverse_iterator' against '_List_iterator'
    operator-(const reverse_iterator<_Iterator>& __x,

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:1380:5: note: candidate template ignored: could not match 'reverse_iterator' against '_List_iterator'
    operator-(const reverse_iterator<_Iterator>& __x,

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:1121:5: note: candidate template ignored: could not match 'move_iterator' against '_List_iterator'
    operator-(const move_iterator<_IteratorL>& __x,

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:1128:5: note: candidate template ignored: could not match 'move_iterator' against '_List_iterator'
    operator-(const move_iterator<_IteratorL>& __x,

In file included from main.cpp:11:
In file included from /usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/algorithm:62:
/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_algo.h:1886:18: error: invalid operands to binary expression ('std::_List_iterator<int>' and 'std::_List_iterator<int>')
    if (__last - __first > __int(_S_threshold))

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_algo.h:1970:9: note: in instantiation of function template specialization 'std::__final_insertion_sort<std::_List_iterator<int>, __gnu_cxx::__ops::_Iter_less_iter>' requested here
        std::__final_insertion_sort(__first, __last, __comp);

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_algo.h:4876:12: note: in instantiation of function template specialization 'std::__sort<std::_List_iterator<int>, __gnu_cxx::__ops::_Iter_less_iter>' requested here
    std::__sort(__first, __last, __gnu_cxx::__ops::__iter_less_iter());

main.cpp:15:7: note: in instantiation of function template specialization 'std::sort<std::_List_iterator<int> >' requested here
    std::sort(begin(lst), end(lst));

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_bvector.h:208:3: note: candidate function not viable: no known conversion from 'std::_List_iterator<int>' to 'const std::_Bit_iterator_base' for 1st argument
  operator-(const _Bit_iterator_base& __x, const _Bit_iterator_base& __y)

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:328:5: note: candidate template ignored: could not match 'reverse_iterator' against '_List_iterator'
    operator-(const reverse_iterator<_Iterator>& __x,

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:1380:5: note: candidate template ignored: could not match 'reverse_iterator' against '_List_iterator'
    operator-(const reverse_iterator<_Iterator>& __x,

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:1121:5: note: candidate template ignored: could not match 'move_iterator' against '_List_iterator'
    operator-(const move_iterator<_IteratorL>& __x,

/usr/bin/../lib64/gcc/x86_64-unknown-linux-gnu/4.9.0/../../../../include/c++/4.9.0/bits/stl_iterator.h:1128:5: note: candidate template ignored: could not match 'move_iterator' against '_List_iterator'
    operator-(const move_iterator<_IteratorL>& __x,

2 errors generated.
makefile:25: recipe for target 'main.o' failed
makefile: *** [main.o] Error 1
florian@kerra /tmp/tmp.WFQWXIEDM5 % yfc -cg++ -c
gcc -Wall -Wextra -pedantic -std=c++11g  -c -o main.o main.cpp
In file included from /usr/include/c++/4.9.0/algorithm:62:0,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_algo.h: In Instanziierung von »void std::__sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = std::_List_iterator<int>; _Compare = __gnu_cxx::__ops::_Iter_less_iter]«:
/usr/include/c++/4.9.0/bits/stl_algo.h:4876:7:  erfordert durch »void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator<int>]«
main.cpp:15:13:  von hier erfordert
/usr/include/c++/4.9.0/bits/stl_algo.h:1968:22: Fehler: no match for »operator-« (operand types are »std::_List_iterator<int>« and »std::_List_iterator<int>«)
      std::__lg(__last - __first) * 2,
                         ~~~~~~~ - ~~~~~~
/usr/include/c++/4.9.0/bits/stl_algo.h:1968:22: Anmerkung: Kandidaten sind:
In file included from /usr/include/c++/4.9.0/bits/stl_algobase.h:67:0,
                 from /usr/include/c++/4.9.0/algorithm:61,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_iterator.h:328:5: Anmerkung: template<class _Iterator> typename std::reverse_iterator<_Iterator>::difference_type std::operator-(const std::reverse_iterator<_Iterator>&, const std::reverse_iterator<_Iterator>&)
    operator-(const reverse_iterator<_Iterator>& __x,

/usr/include/c++/4.9.0/bits/stl_iterator.h:328:5: Anmerkung:   Herleitung/Ersetzung von Templateargument gescheitert:
In file included from /usr/include/c++/4.9.0/algorithm:62:0,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_algo.h:1968:22: Anmerkung:  »std::_List_iterator<int>« ist nicht vom Typ »const std::reverse_iterator<_Iterator>« abgeleitet
      std::__lg(__last - __first) * 2,
                         ~~~~~~~ - ~~~~~~
/usr/include/c++/4.9.0/bits/stl_algobase.h:67:0,
                 from /usr/include/c++/4.9.0/algorithm:61,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_iterator.h:1380:5: Anmerkung: template<class _IteratorL, class _IteratorR> decltype ((__y.base() - __x.base())) std::operator-(const std::reverse_iterator<_Iterator>&, const std::reverse_iterator<_IteratorR>&)
    operator-(const reverse_iterator<_IteratorL>& __x,

/usr/include/c++/4.9.0/bits/stl_iterator.h:1380:5: Anmerkung:   Herleitung/Ersetzung von Templateargument gescheitert:
In file included from /usr/include/c++/4.9.0/algorithm:62:0,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_algo.h:1968:22: Anmerkung:  »std::_List_iterator<int>« ist nicht vom Typ »const std::reverse_iterator<_Iterator>« abgeleitet
      std::__lg(__last - __first) * 2,
                         ~~~~~~~ - ~~~~~~
/usr/include/c++/4.9.0/bits/stl_algobase.h:67:0,
                 from /usr/include/c++/4.9.0/algorithm:61,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_iterator.h:1121:5: Anmerkung: template<class _IteratorL, class _IteratorR> decltype ((__x.base() - __y.base())) std::operator-(const std::move_iterator<_Iterator>&, const std::move_iterator<_IteratorR>&)
    operator-(const move_iterator<_IteratorL>& __x,

/usr/include/c++/4.9.0/bits/stl_iterator.h:1121:5: Anmerkung:   Herleitung/Ersetzung von Templateargument gescheitert:
In file included from /usr/include/c++/4.9.0/algorithm:62:0,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_algo.h:1968:22: Anmerkung:  »std::_List_iterator<int>« ist nicht vom Typ »const std::move_iterator<_Iterator>« abgeleitet
      std::__lg(__last - __first) * 2,
                         ~~~~~~~ - ~~~~~~
/usr/include/c++/4.9.0/bits/stl_algobase.h:67:0,
                 from /usr/include/c++/4.9.0/algorithm:61,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_iterator.h:1128:5: Anmerkung: template<class _Iterator> decltype ((__x.base() - __y.base())) std::operator-(const std::move_iterator<_Iterator>&, const std::move_iterator<_Iterator>&)
    operator-(const move_iterator<_IteratorL>& __x,

/usr/include/c++/4.9.0/bits/stl_iterator.h:1128:5: Anmerkung:   Herleitung/Ersetzung von Templateargument gescheitert:
In file included from /usr/include/c++/4.9.0/algorithm:62:0,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_algo.h:1968:22: Anmerkung:  »std::_List_iterator<int>« ist nicht vom Typ »const std::move_iterator<_Iterator>« abgeleitet
      std::__lg(__last - __first) * 2,
                         ~~~~~~~ - ~~~~~~
In file included from /usr/include/c++/4.9.0/vector:65:0,
                 from /usr/include/c++/4.9.0/bits/random.h:34,
                 from /usr/include/c++/4.9.0/random:49,
                 from /usr/include/c++/4.9.0/bits/stl_algo.h:66,
                 from /usr/include/c++/4.9.0/algorithm:62,
                 from main.cpp:11:
/usr/include/c++/4.9.0/bits/stl_bvector.h:208:3: Anmerkung: std::ptrdiff_t std::operator-(const std::_Bit_iterator_base&, const std::_Bit_iterator_base&)
  operator-(const _Bit_iterator_base& __x, const _Bit_iterator_base& __y)

/usr/include/c++/4.9.0/bits/stl_bvector.h:208:3: Anmerkung:   keine bekannte Umwandlung für Argument 1 von »std::_List_iterator<int>« nach »const std::_Bit_iterator_base&«
makefile:25: recipe for target 'main.o' failed
makefile: *** [main.o] Error 1
florian@kerra /tmp/tmp.WFQWXIEDM5 %
```

Concepts

- Im Debugmodus erheblich schlimmer.
- Ähnliches Szenario mit Concept-GCC:

```
error: no matching call to ‘sort(forward_list<int>&)’
note: candidate is ‘sort(C& container)’
note:   where C = forward_list<int>
note: template constraints not satisfied
note:   ‘C’ is not a/an ‘Sortable’ type since
note:      ‘c[n]’ is not valid syntax
```

C++17

Übersicht

- Module
- Dateisystem
- Netzwerk
- Nebenläufigkeit
- Ranges
- ...

Idiomatische Verwendung

Rückgabewerte

- Ausgabe-Parameter sind oft hässlich und unintuitiv:

```
void fun(int arg, int& ret1, int& ret2);
```

Rückgabewerte

- Ausgabe-Parameter sind oft hässlich und unintuitiv:

```
void fun(int arg, int& ret1, int& ret2);
```

- Seit C++11: Tupel:

```
std::tuple<int, int> fun(int arg);
```

Rückgabewerte

- Ausgabe-Parameter sind oft hässlich und unintuitiv:

```
void fun(int arg, int& ret1, int& ret2);
```

- Seit C++11: Tupel:

```
std::tuple<int, int> fun(int arg);
```

- Entgegennahme der Argumente mit `std::tie`:

```
long val; // sic
std::tie(val, std::ignore) = fun(42);
```

Lexikographische Vergleiche

- Für Container: `std::lexicographical_compare` und `std::equal`
- Sonst: `std::tie`:

```
struct col {
    unsigned char r, g, b;
};

bool operator<(const col& l, const col& r) {
    return std::tie(l.r, l.g, l.b)
         < std::tie(r.r, r.g, r.b);
}
```

Typinferenz

```
void fun1(const std::map<std::string, int>& map) {
    for(const std::pair<std::string, int>& pair: map) {
        std::cout << pair.first << ": "
                  << pair.second << '\n';
    }
}


void fun2(const std::map<std::string, int>& map) {
    for(auto&& pair: map) {
        std::cout << pair.first << ": "
                  << pair.second << '\n';
    }
}
```

Typinferenz

```
void fun1(const std::map<std::string, int>& map) {
    for(const std::pair<std::string, int>& pair: map) {
        std::cout << pair.first << ": "
                  << pair.second << '\n';
    }
}


void fun2(const std::map<std::string, int>& map) {
    for(auto&& pair: map) {
        std::cout << pair.first << ": "
                  << pair.second << '\n';
    }
}
```

- Bug: unnötige Kopie in erster Version

Algorithmen

If you want to improve the code quality in your organization, replace all your coding guidelines with one goal:
No raw loops!
– Sean Parent

Algorithmen

If you want to improve the code quality in your organization, replace all your coding guidelines with one goal:
No raw loops!
– Sean Parent

- Aufgabe: Verschiebe einen Teilbereich eines Containers an den Anfang

Algorithmen

If you want to improve the code quality in your organization, replace all your coding guidelines with one goal:
No raw loops!
– Sean Parent

- Aufgabe: Verschiebe einen Teilbereich eines Containers an den Anfang

```
std::rotate(begin(container), sub_range_begin,
            sub_range_end);
```

Algorithmen

- Aufgabe: Sammle alle Elemente eines Containers die ein Prädikat erfüllen an gegebener Stelle, ohne die relative Reihenfolge zu ändern

Algorithmen

- Aufgabe: Sammle alle Elemente eines Containers die ein Prädikat erfüllen an gegebener Stelle, ohne die relative Reihenfolge zu ändern

```
std::stable_partition(begin, target, std::not1(pred));
std::stable_partition(target, end, pred);
```

Algorithmen

- Aufgabe: Sammle alle Elemente eines Containers die ein Prädikat erfüllen an gegebener Stelle, ohne die relative Reihenfolge zu ändern

```
std::stable_partition(begin, target, std::not1(pred));
std::stable_partition(target, end, pred);
```

- Aufgabe: erzeuge einen std::vector mit den Zahlen von 3 bis n

Algorithmen

- Aufgabe: Sammle alle Elemente eines Containers die ein Prädikat erfüllen an gegebener Stelle, ohne die relative Reihenfolge zu ändern

```
std::stable_partition(begin, target, std::not1(pred));
std::stable_partition(target, end, pred);
```

- Aufgabe: erzeuge einen std::vector mit den Zahlen von 3 bis n

```
std::vector vec(n - 3);
std::iota(begin(vec), end(vec), 3);
```

Argument-Dependent-Lookup

```
int main() {
    std::vector<int> vec;
    foo::bar x; // foo::begin(foo::bar) exists
    bla::blub y; // bla::begin(bla::blub) doesn't exist

    using std::begin;

    auto it1 = begin(vec); // -> std::begin
    auto it2 = begin(x);   // -> foo::begin
    auto it3 = begin(y);   // -> std::begin
}
```

Einsatz von Exceptions

- Für Bugs: `std::terminate` oder `std::abort` vielleicht besser
- Für ungültige Usereingaben: „das kommt darauf an“
  - gegen DoS: ungeeignet
  - sonst: sicherer als Fehlercodes
- Potentiell schneller als Fehlercodes
  - solange nicht geworfen
- Keine Zeitgarantien (gilt auch für Speicherallokation, `dynamic_cast`, ...)

Umgang mit Exceptions

- schwache Exception-safety (keine Leaks) is Pflicht
  - Nur sinnvoll mit RAII möglich
- starke Exception-safety (transaktionelles Verhalten) is schön
- Ungefangene Exception → `std::terminate()`
  - `main()` sollte mindestens `std::runtime_error` fangen:

```
int main() try {
    do_something();
} catch(std::runtime_error& e) {
    std::cerr << "Runtime-error: " << e.what() << '\n';
}
```

Type-Erasure

```
struct base {
    virtual std::unique_ptr<base> clone() const = 0;
    virtual ~base = default
};

template<typename T> class wrapper: base {
    T value;
    virtual std::unique_ptr<base> clone() const override {
        return std::make_unique<wrapper<T>>(value);
    }
};
```

- Funktioniert auch für int
- Container mit tiefen Kopien trivial

Sonstiges

Probleme

- Keine guten und freien Tutorials (oft veraltet und ausnahmslos zu low-level)
  - Ja das gilt auch für `$YOUR_FAVOURITE_TUTORIAL`
  - `https://github.com/FlorianJW/cpp_tut`
    - Pull-requests sind gerne gesehen ;-)

Probleme

- Keine guten und freien Tutorials (oft veraltet und ausnahmslos zu low-level)
  - Ja das gilt auch für `$YOUR_FAVOURITE_TUTORIAL`
  - `https://github.com/FlorianJW/cpp_tut`
    - Pull-requests sind gerne gesehen ;-)

- Abwärtskompatibilität
- Schlechter Ruf
- Biliotheken aus der Steinzeit (Qt, I'm looking at you!)
- Keine einheitlichen Code-Konventionen
  - `foo.h` bitte für C reservieren (`foo.H` nur zum trollen)
  - Die Standardbibliothek nutzt `snake_case`, nicht `PascalCase`

Stärken

- Implizites Ressourcenmanagment
- Mächtige Sprachmittel
- Starkes, statisches Typsystem
- High-Level
- Performance

Antipatterns

std::endl

- `std::endl` leert den Puffer und ist sonst identisch zu `'\n'`

```
std::cout << "bad" << std::endl;
std::cout << "good\n";

std::cerr << "NEVER what you want!" << std::endl;
std::cerr << "since std::cerr is unbuffered.\n";
```

std::endl

- `std::endl` leert den Puffer und ist sonst identisch zu `'\n'`

```
std::cout << "bad" << std::endl;
std::cout << "good\n";

std::cerr << "NEVER what you want!" << std::endl;
std::cerr << "since std::cerr is unbuffered.\n";
```

- (normalerweise: `'\n'` → plattformspezifischer Zeilenumbruch)

Spezielle Methoden

```
//foo.hpp:
class foo {
    foo();
    virtual ~foo();
    // stuff
};

//foo.cpp
foo::foo() {}
foo::~foo() {}
```

Spezielle Methoden

```
//foo.hpp:
class foo {
    foo();
    virtual ~foo();
    // stuff
};

//foo.cpp
foo::foo() {}
foo::~foo() {}
```

- Manchmal völlig unnötige TU hierdurch (Compilezeit!)
- Indirekter Funktionsaufruf: Inlining RIP
- Eclipse CDT: I am looking at you

Spezielle Methoden

```
// better: Rule of 5
class foo {
    foo() = default;
    virtual ~foo() = default;
    foo(const foo&) = default;
    foo(foo&&) = default;
    foo& operator(const foo&) = default;
    foo& operator(foo&&) = default;
    // stuff
};
```

Spezielle Methoden

```
// better: Rule of 5
class foo {
    foo() = default;
    virtual ~foo() = default;
    foo(const foo&) = default;
    foo(foo&&) = default;
    foo& operator(const foo&) = default;
    foo& operator(foo&&) = default;
    // stuff
};

// usually best: Rule of zero
class foo{
    // stuff
};
```

Klassenmissbrauch (1)

```
class math {
    static double sin(double);
    static double cos(double);
    static double tan(double);
};
```

Klassenmissbrauch (1)

```
class math {
    static double sin(double);
    static double cos(double);
    static double tan(double);
};

int main() {
    math m; // err...
    auto m2 = m; // You can copy a math?
    assert(&m != nullptr); // math has an adress?
}
```

Klassenmissbrauch (1)

```
class math {
    static double sin(double);
    static double cos(double);
    static double tan(double);
};

int main() {
    math m; // err...
    auto m2 = m; // You can copy a math?
    assert(&m != nullptr); // math has an adress?
}
```

- Missbrauch von Klassen als Namensraum
- C++ hat *native* Sprachunterstützung für Namensräume

Klassenmissbrauch (2)

```
class searcher {
    std::size_t search(const std::string& haystack,
                       char needle) const {
        /* ... */
    }
};
```

Klassenmissbrauch (2)

```
class searcher {
    std::size_t search(const std::string& haystack,
                       char needle) const {
        /* ... */
    }
};

std::size_t search(const std::string& haystack,
                   char needle) { /* ... */ }
```

- C++ ist nicht Java: Funktionen werden nativ unterstützt
- „Aber Kapselung“: Siehe vorherige Folie (Namensräume!)