C++ PROGRAMMING

Lecture 10

Secure Software Engineering Group
Philipp Dominik Schubert

CONTENTS

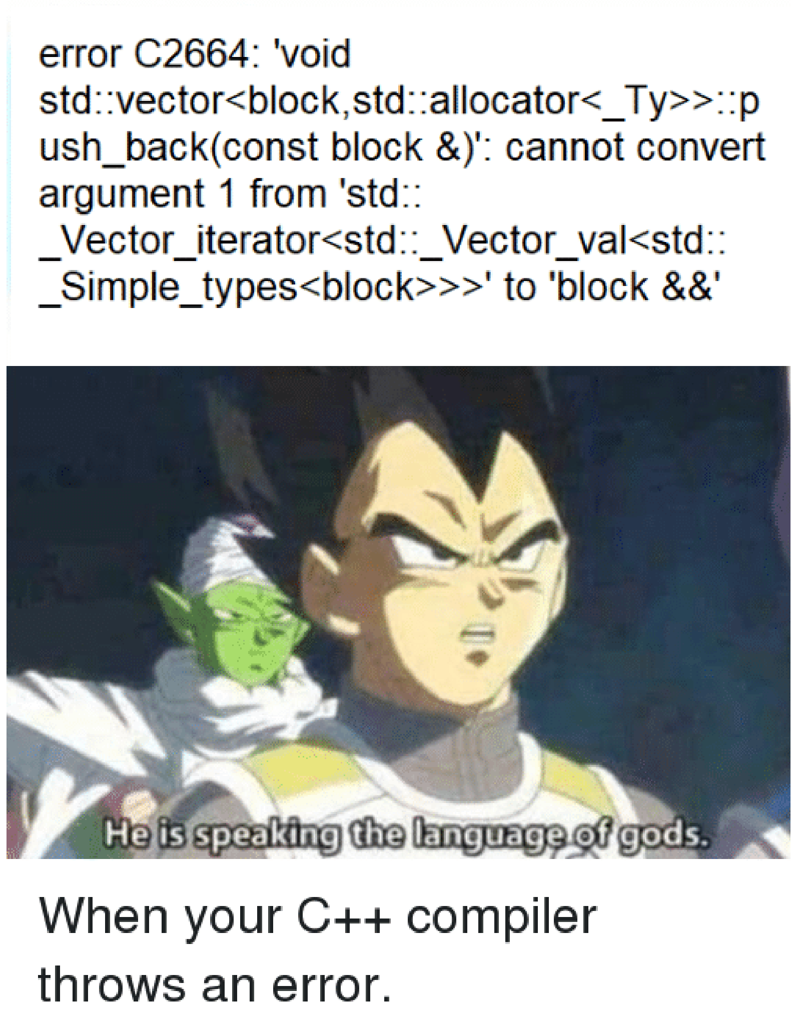
Libraries

- Why solve a task that has already been solved?
  - Waste of time and energy
  - You cannot solve all tasks
    - There are too many
    - They are too hard
- Much effort and clever design is put into libraries
- If possible prefer STL over other libraries
- Be sure to use high quality libraries
- Prefer libraries over ad-hoc / hand-crafted solutions

[Figure taken from https://upload.wikimedia.org/wikipedia/commons/thumb/2/2b/Melk_-_Abbey_-_Library.jpg/1200px-Melk_-_Abbey_-_Library.jpg]

STL – Standard template library

- Contains a huge amount of useful things
- Specified by C++ standardization committee
- Different compiler vendors provide different implementations
  - GCC – libstdc++, Clang – libc++, …
- Is automatically linked with your application
- Prefer STL over other third party libraries
- Optimized for performance
  - Hard to read for humans
- But not all platforms offer an STL implementation
- Do not think STL is perfect for every task



error C2664: 'void std::vector<block,std::allocator<_Ty>>::push_back(const block &)': cannot convert argument 1 from 'std::_Vector_iterator<std::_Vector_val<std::_Simple_types<block>>>' to 'block &&'

When your C++ compiler throws an error.

[Figure taken from https://me.me/i/error-c2664-void-std-vector%3Cblock-std-allocator%3C-tv%3E%3Ep-ush-back-const-block-a144ab0a485f49628babcf0933861e76]

BOOST

- Was founded by C++ standardization committee members
- A collection of portable sub-libraries
- Sub-libraries are distinguished by the task they solve
- Most parts of these libraries are implemented in header files (.hpp)
  - Why? → Templates
- Highly compatible with STL
- Heavily used by C++ programmers of all domains
- High quality
  - New sub-libraries have to undergo an extensive review-process
- Not restricted to a specific domain
- Goal is to increase productivity
- Boost has suitable licenses for commercial and non-commercial use

[Figure taken from http://images.google.de/imgres?imgurl=https%3A%2F%2Fsvn.boost.org%2Ftrac%2Fboost%2Fraw-attachment%2Fwiki%2FUnifiedLookAndFeelProject%2Funified_look_and_feel.

Using BOOST and other libraries in general

- Boost is a huge collection of libraries
- You cannot look things up manually
- Google for the task to be solved
  - *C++ <what you want> <whatever library you want to use>*
    - “*c++ serialize objects boost*”
  - Focus on what looks promising
  - Do not use code blindly! (never just copy and paste)
    - Dunning-Kruger effect
  - Try to understand the code
  - Write small test programs
  - You have to learn to distinguish good code from rubbish code
    - Sadly there is much rubbish out there

[See http://stackoverflow.com/questions/7264402/when-to-use-template-vs-inheritance/]

boost/filesystem

```cpp
#include <boost/filesystem.hpp>
#include <iostream>
namespace bfs = boost::filesystem;
int main() {
  bfs::path p("files/");
  bfs::path q("data.txt");
  bfs::path r = p / q;
  std::cout << r.string() << '\n';
  if (bfs::exists(r) && !bfs::is_directory(r)) {
    std::cout << r.stem().string() << '\n';
    std::cout << r.extension().string() << '\n';
  }
  if (bfs::exists(p) && bfs::is_directory(p)) {
    bfs::directory_iterator dit(p);
    while (dit != bfs::directory_iterator{}) std::cout << *dit++ << '\n';
  }
  return 0;
}
```

Or use `std::filesystem` (C++17)

```cpp
#include <filesystem>
```

boost/program_options

```
#include <boost/program_options.hpp>
#include <iostream>
#include <stdexcept>
namespace bpo = boost::program_options;

int main(int argc, char **argv) {
  bpo::variables_map VarMap;
  bpo::options_description Options("My awesome program");
  // clang-format off
  Options.add_options()
    ("num,N", bpo::value<int>(), "A number")
    ("msg,M", bpo::value<std::string>()->multitoken()->zero_tokens()->composing(), "A message");
  // clang-format on
  try {
    bpo::store(bpo::command_line_parser(argc, argv).options(Options).allow_unregistered().run(), VarMap);
    bpo::notify(VarMap);
  } catch (const bpo::error &e) {
    std::cerr << "error: could not parse options, message: " << e.what() << ", abort\n";
    return 1;
  }
  if (argc == 1) { std::cout << Options; return 0; }
  if (VarMap.count("num")) { std::cout << VarMap["num"].as<int>() << '\n'; }
  if (VarMap.count("msg")) { std::cout << VarMap["msg"].as<std::string>() << '\n'; }
  return 0;
}
```

boost/logger

```
enum severity_level { INFO, DEBUG, WARNING, ERROR, CRITICAL };
BOOST_LOG_INLINE_GLOBAL_LOGGER_DEFAULT(lg, bl::sources::severity_logger<severity_level>)
void LogFormatter(const bl::record_view &view, bl::formatting_ostream &os) {
  os << "[" << view.attribute_values()["Severity"].extract<severity_level>()
     << "] " << view.attribute_values()["Message"].extract<std::string>();
}
void initializeLogger() {
  bl::core::get()->set_logging_enabled(true);
  typedef bl::sinks::synchronous_sink<bl::sinks::text_ostream_backend> text_sink;
  boost::shared_ptr<text_sink> sink = boost::make_shared<text_sink>();
  boost::shared_ptr<std::ostream> stream(&std::clog, boost::empty_deleter{});
  sink->locked_backend()->add_stream(stream);
  sink->set_formatter(&LogFormatter);
  bl::core::get()->add_sink(sink);
}
int main() {
  initializeLogger();
  BOOST_LOG_SEV(lg::get(), DEBUG)
    << "I am debugging!";
  return 0;
}
```

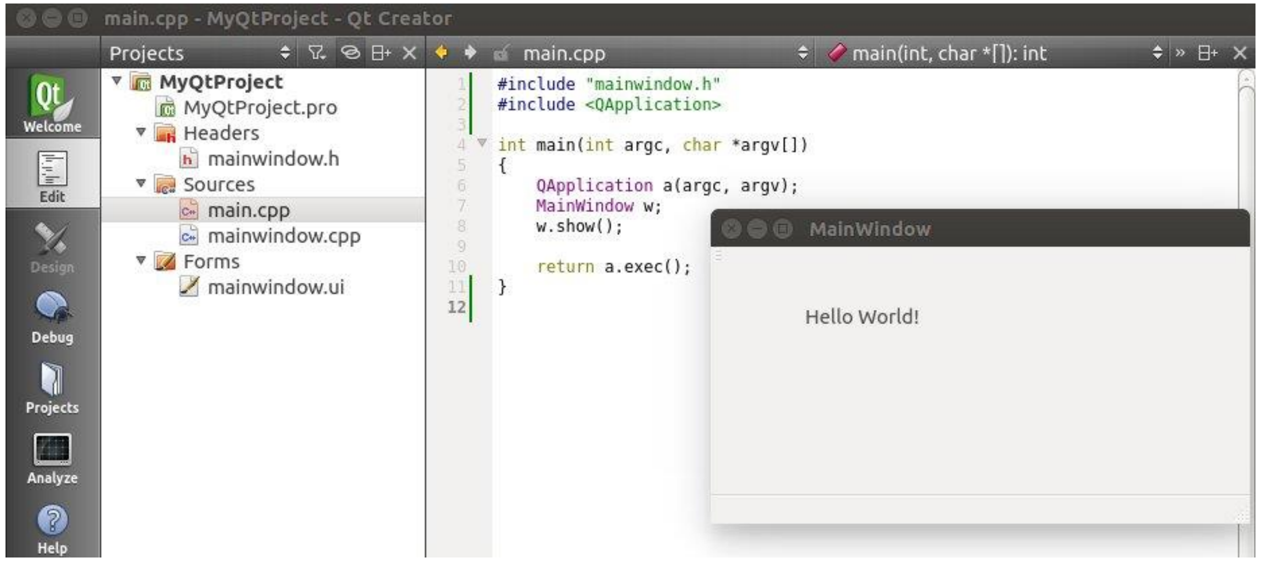
```
#include <boost/log/common.hpp>
#include <boost/log/sinks.hpp>
#include <boost/log/sources/global_logger_storage.hpp>
#include <boost/log/sources/logger.hpp>
#include <boost/log/sources/severity_logger.hpp>
#include <boost/utility/empty_deleter.hpp>
#include <iostream>
namespace bl = boost::log;
```

boost/…

- There are many more useful libraries
- Check if boost can solve your problem
- Boost documentation
  - http://www.boost.org/doc/libs/
- A hands-on tutorial guide (with code examples)
  - The Boost C++ Libraries
    - https://theboostcpplibraries.com/

Qt

- Qt (cute)
- Platform independent C++ class library
- Primary goal: graphical user interfaces
  - X11, OS X, Windows, iOS, Android
- Covers other domains as well
- Qt comes with MOC preprocessor (meta object compiler) allowing signals, slots, and reflection
- Provides interfaces for other languages
  - Python, Ruby, C#, Java, …
- High-quality IDE Qt Creator
  - Includes GUI designer
  - You want to use Qt Creator when developing graphical Qt applications



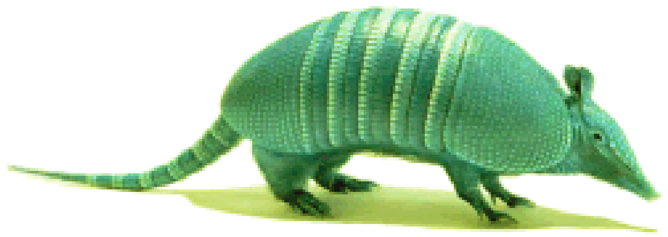
```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

[Figure taken from http://i1-linux.softpedia-static.com/screenshots/Qt_1.jpg]

Armadillo



- High-quality linear algebra library
- Good balance between performance and ease of use
- High-level syntax
- Used for
  - Machine learning
  - Pattern recognition
  - Computer vision
  - Signal processing
  - Bioinformatics
  - …
- Algorithms can be easily implemented using Armadillo
- Really good documentation (including examples)

```
#include <iostream>
#include <armadillo>
int main() {
  // arma::mat is arma::Mat<double> → type alias
  arma::mat A = arma::randu<arma::mat>(3,3);
  arma::mat B = arma::randu<arma::mat>(3,3);
  arma::mat C = A * B;
  std::cout << C;
  std::cout << "------------\n";
  arma::mat D =  {{-1, 8, 2, 8, 7},
                  {5, 6, -5, 7, 2},
                  {-9, 0, 1, 2, -3}};
  arma::mat moore_penrose_inverse =
    arma::pinv(D);
  std::cout << moore_penrose_inverse;
  return 0;
}
```

[Figure taken from http://arma.sourceforge.net/img/armadillo_logo.png]

OpenCV

- High-quality C++ library for computer vision
- For academic and industrial use
- As efficient as possible: allows for real-time applications
- Optimized C/C++ code
- Multi-core and GPU support
- Contains lots of useful stuff
  - Fourier transformation
  - Support vector machine (SVM)
  - Edge detection
  - GUI elements

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
int main(int argc, char **argv) {
  cv::Mat lena =
    cv::imread("pictures/lena.png",
                     CV_LOAD_IMAGE_COLOR);
  cv::imshow("opencvtest", lena);
  cv::waitKey();
  return 0;
}
```

- DFT would be only a few lines of code

[Figure taken from https://upload.wikimedia.org/wikipedia/commons/thumb/5/53/OpenCV_Logo_with_text.png/180px-OpenCV_Logo_with_text.png]

OpenGL and Vulkan

- API for 2D and 3D computer graphics applications
- API implemented in a library
- Platform independent
- Real-time rendering of complex 3D scenes on graphics cards
- Modern computer games can be programmed in OpenGL and Vulkan
- OpenGL → Vulkan
- Animated movies are usually done by ray tracing techniques (currently too slow for real time graphics)

```
#include <GL/freeglut.h>
static void dispfun() {
  glClear(GL_COLOR_BUFFER_BIT);
  glutSwapBuffers();
}
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("A nice green window");
  glutDisplayFunc(dispfun);
  glClearColor(0.0f, 1.0f, 0.0f, 0.0f);
  glutMainLoop();
  return 0;
}
```

[Figure taken from https://upload.wikimedia.org/wikipedia/commons/thumb/e/e9/Opengl-logo.svg/2000px-Opengl-logo.svg.png and https://developer.nvidia.com/open-source-vulkan-c-api]

OpenCL and CUDA

- Use the NVCC compiler for CUDA
- Nowadays: copying to and from GPU RAM happens implicitly

```
 #include <cstdio>
const short N = 10;
// CUDA Kernel for Vector Addition
__global__ void Vector_Addition (
  const int *dev_a,
  const int *dev_b,
  int *dev_c) {
  unsigned short tid = threadIdx.x ;
  if ( tid < N ) {
    dev_c [tid] = dev_a[tid] + dev_b[tid];
  }
}

int main () {
  int Host_a[N], Host_b[N], Host_c[N];
  int *dev_a , *dev_b, *dev_c;
  cudaMalloc((void **) &dev_a , N*sizeof(int));
  cudaMalloc((void **) &dev_b , N*sizeof(int));
  cudaMalloc((void **) &dev_c , N*sizeof(int));
  for ( int i = 0; i <N ; i++ ) {
    Host_a[i] = -i ;
    Host_b[i] = i*i ;
  }
```

```
  cudaMemcpy(dev_a , Host_a , N*sizeof(int),
             cudaMemcpyHostToDevice);

  cudaMemcpy(dev_b , Host_b , N*sizeof(int),
             cudaMemcpyHostToDevice);

  Vector_Addition<<<1, N>>>(dev_a,
                            dev_b,
                            dev_c);

  cudaMemcpy(Host_c, dev_c, N*sizeof(int),
             cudaMemcpyDeviceToHost);

  for ( int i = 0; i<N; i++ ) {
    printf ("%d + %d = %d\n",
      Host_a[i],
      Host_b[i],
      Host_c[i]);
  }
  cudaFree(dev_a);
  cudaFree(dev_b);
  cudaFree(dev_c);
  return 0;
}
```

[Figure taken from http://opencl.org/OpenCL_Logo.png, https://s3.amazonaws.com/nvlabs-qwiklab-website-prod/badges/images/8/original/badge-nvidia-cuda-cpp.png?1433877963]

OpenCL and CUDA

- OpenCL
  - Open standard for all graphics cards / (accelerated) multi-core architectures
- CUDA
  - Nvidia’s programming environment
- Programming-Technique
- Certain program parts can be computed on GPU
- Data-parallelism problems
- Linear algebra
  - Graphics computations
  - Numeric computations
  - Computer simulations
  - N-dimensional vector problems

- General idea
  - Copy data from CPU RAM to GPU RAM
  - Call graphics kernel on that data
  - Copy results back from GPU to CPU RAM
- Kernel functions
  - “All happens at the same time”
  - Branching must be avoided
  - Different model of thinking
    - Quite hard at the beginning
- More general applicable than OpenGL, which is “graphics only”-computations

[Figure taken from http://opencl.org/OpenCL_Logo.png, https://s3.amazonaws.com/nvlabs-qwiklab-website-prod/badges/images/8/original/badge-nvidia-cuda-cpp.png?1433877963]

OpenMP

- API for shared-memory programming in C/C++
- Developed by hardware- / compiler vendors
- A collection of functions but mostly preprocessor directives for parallelization
- Mostly parallelization of loops
  - Distribute computation using different threads
- Synchronization via `#pragma omp cirtical`
- Programs work correctly even if compiler does not support OpenMP
- Every C/C++ programmer should know the basics
- OMP introduces embarrassing parallel problems

```
#include <iostream>
#include <vector>
#include <omp.h>
int main() {
  std::vector<int> vi(100000000, 2);
  size_t i;
  #pragma omp parallel for private(i) shared(vi)\
    schedule(static)
  for (i = 0; i < vi.size(); ++i) {
    vi[i] *= 2;
  }
  for (size_t i = 0; i < 10; ++i) {
    std::cout << vi[i] << '\n';
  }
  return 0;
}
```

[Figure taken from http://www.openmp.org/wp-content/uploads/openmp-menu-logo.jpg]

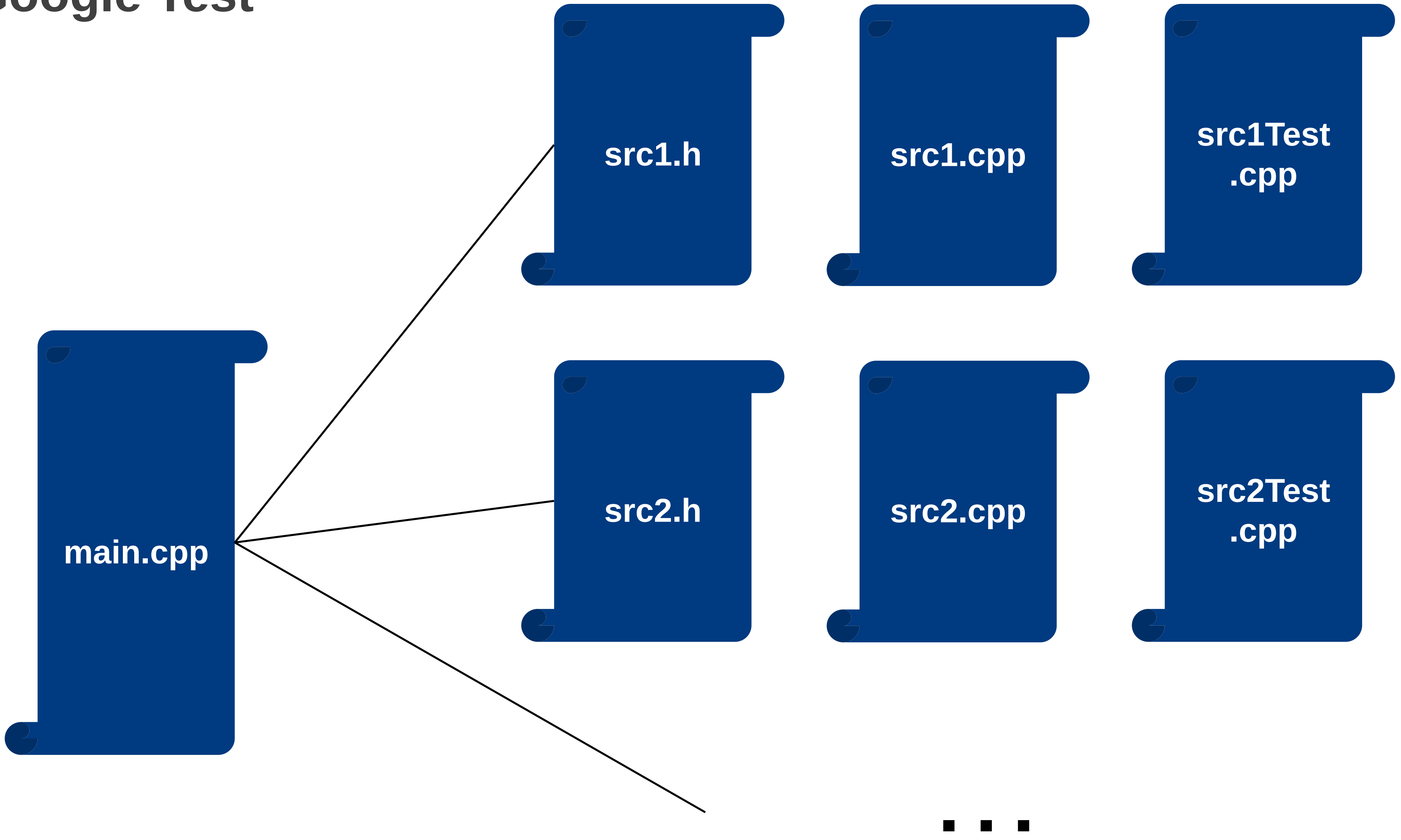
Google Test

googletest
Google C++ Testing Framework

- Testing framework for C and C++
- Provides basic infrastructure for automated testing
- Allows one to write standardized test cases
- Can be compiled and executed fully automatically
- Detailed information are provided when tests fail
  - Test name, line number, assertion that failed, …
- De facto standard testing framework
- Is used by many modern C++ projects
- First stable release: August 2016 (Wiki)

[Figure taken from http://www.qatestingtools.com/code.google/googletest]

Google Test



[Figure taken from http://www.qatestingtools.com/code.google/googletest]

Google Test

```
#ifndef SRC_H_
#define SRC_H_

unsigned f(unsigned n);

#endif
```

```
#include "src.h"

unsigned f(unsigned n) {
 return (n <= 1) ?
        1 :
        n * f(n - 1);
}
```

```
#include <iostream>
#include "src.h"

int main() {
  unsigned k = 8;
  unsigned result = f(k);
  std::cout << result << '\n';
  return 0;
}
```

```
#include <gtest/gtest.h>
#include "src.h"
// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
  EXPECT_EQ(1, f(0));
}
// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(1, f(1));
  EXPECT_EQ(2, f(2));
  EXPECT_EQ(6, f(3));
  EXPECT_EQ(40320, f(8));
}

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

Google Abseil

- Designed to augment C++’s STL
- Collection of Google’s C++ code base
- Provides libraries components to solve various tasks
  - base
  - algorithm
  - container
  - debugging
- Can be build via Bazel and Cmake
- C++ has one problem: don’t break compatibility
  - ABI (application binary interface) and linking
    - Disallows modifications and improvements on STL containers
  - Abseil’s container implementations may be more efficient for your problem

```cpp
#include <iostream>
#include <string>
#include <vector>

#include "absl/strings/str_join.h"

int main() {
  std::vector<std::string> v
        = {"foo", "bar", "baz"};
  std::string s = asbl::StrJoin(v, "-");
  std::cout << "Joined string: " << s << "\n";
  return 0;
}
```

Google Protobuf

- Protocol buffers
- Language-neutral, platform-neutral, extensible mechanism for serializing structured data
- Supports virtually all modern languages
- XML, but smaller, faster and simpler
- Binary format
- Define how data is structured
  - Special source code will be generated automatically by protoc (compiler)
    - Generated code allows for reading and writing your data
- Use protocol buffers to communicate data between server(s) and clients

```
syntax = "proto2";
package tutorial;

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2; }
  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2
      [default = HOME];
   }
  repeated PhoneNumber phones = 4;
}
message AddressBook {
  repeated Person people = 1;
}
```

Libraries and the linker

```cpp
#include <iostream>
#include <boost/filesystem.hpp>
namespace bfs = boost::filesystem;
int main(int argc, char **argv) {
  bfs::path p(argv[1]);
  if (bfs::exists(p)) {
    std::cout << "path exists\n";
  }
  return 0;
}
```

- The compiler will automatically link against C++'s STL
- Taming the linker: important compiler switches (for the linker)
  - If a library is installed system-wide, use
    - `-lLIBRARY`
      - Search the library named `LIBRARY` when linking
    - `clang++ -std=c++17 -Wall test.cpp -o test -lboost_system -lboost_filesystem`
  - If a library is not installed system-wide
    - `-IDIRECTORY`
      - Add the directory `DIRECTORY` to the list of directories to search for header files
    - `-LDIRECTORY`
      - Add the directory `DIRECTORY` to the list of directories to be searched for the `-l` flag
    - Assuming install directory is `/home/my_user/my_library/`

```
$ clang++ -std=c++17 -Wall -I/home/my_user/my_library/include/
-L/home/my_user/my_library/lib/ test.cpp -o test -lmy_library
```

Libraries and the linker

- Suffix for static libraries: `.a` (archive)
- Suffix for dynamic libraries: `.so` (shared object)
- How to write your own basic C/C++ library?
  - Compile to static library
  - `$ g++ -std=c++17 -Wall -I../include/ -c mycode.cpp`
  - `ar rcs libmylibrary.a mycode.o`
  - Compile to dynamic library
  - `$ g++ -std=c++17 -Wall -I../include/ -shared -fPIC mycode.cpp -o libtest.so`

- Library structure
- ❖ `mylibrary/`
  - ❖ `include/`
    - ❖ `mylibrary/`
      - ❖ `mycode.h`
  - ❖ `lib/`
    - ❖ `mycode.cpp`
- Contents
- ❖ `mylibrary.h`

```
#ifndef MYCODE_H_
#define MYCODE_H_
int addIntegers(int a, int b);
#endif
```

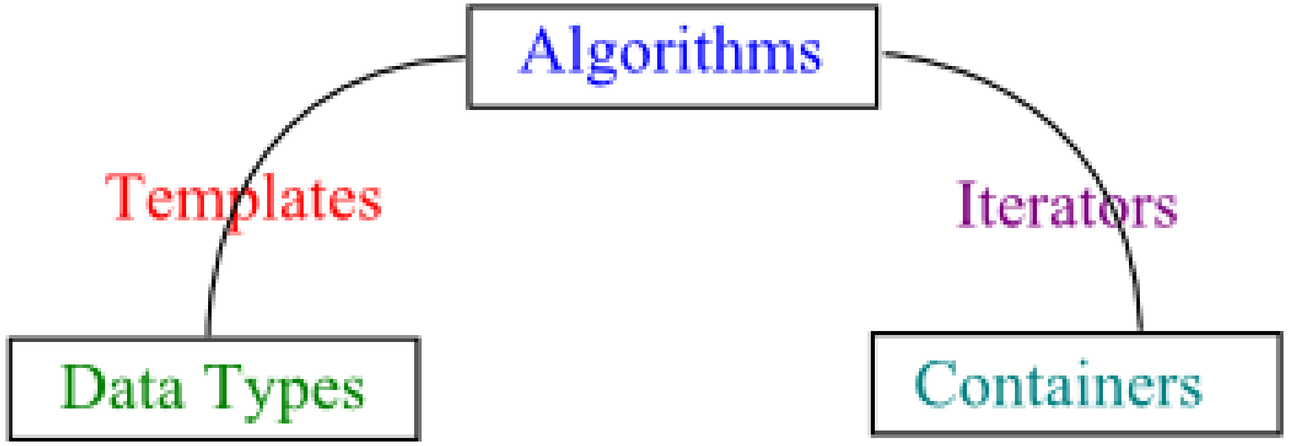
- ❖ `mylibrary.cpp`

```
#include "mylibrary/mycode.h"
int addIntegers(int a, int b) {
  return a + b;
}
```

Iterators

- Data is often stored in containers
- Containers must be inspected / iterated
- Iteration of data is used all the time
- A data type (usually) needs to provide some functionalities for iteration
- Idea:
  - Provide some ad-hoc functionalities
  - Problem:
    - Every container type looks different / must be used in a different manner
  - Solution:
    - Specify a common concept ‘Iterator’ that can / must be implemented

The benefits of iterators



1. Templates

make algorithms independent of the data types

2. Iterators

make algorithms independent of the containters

[Figure taken from http://www.bogotobogo.com/cplusplus/stl3_iterators.php]

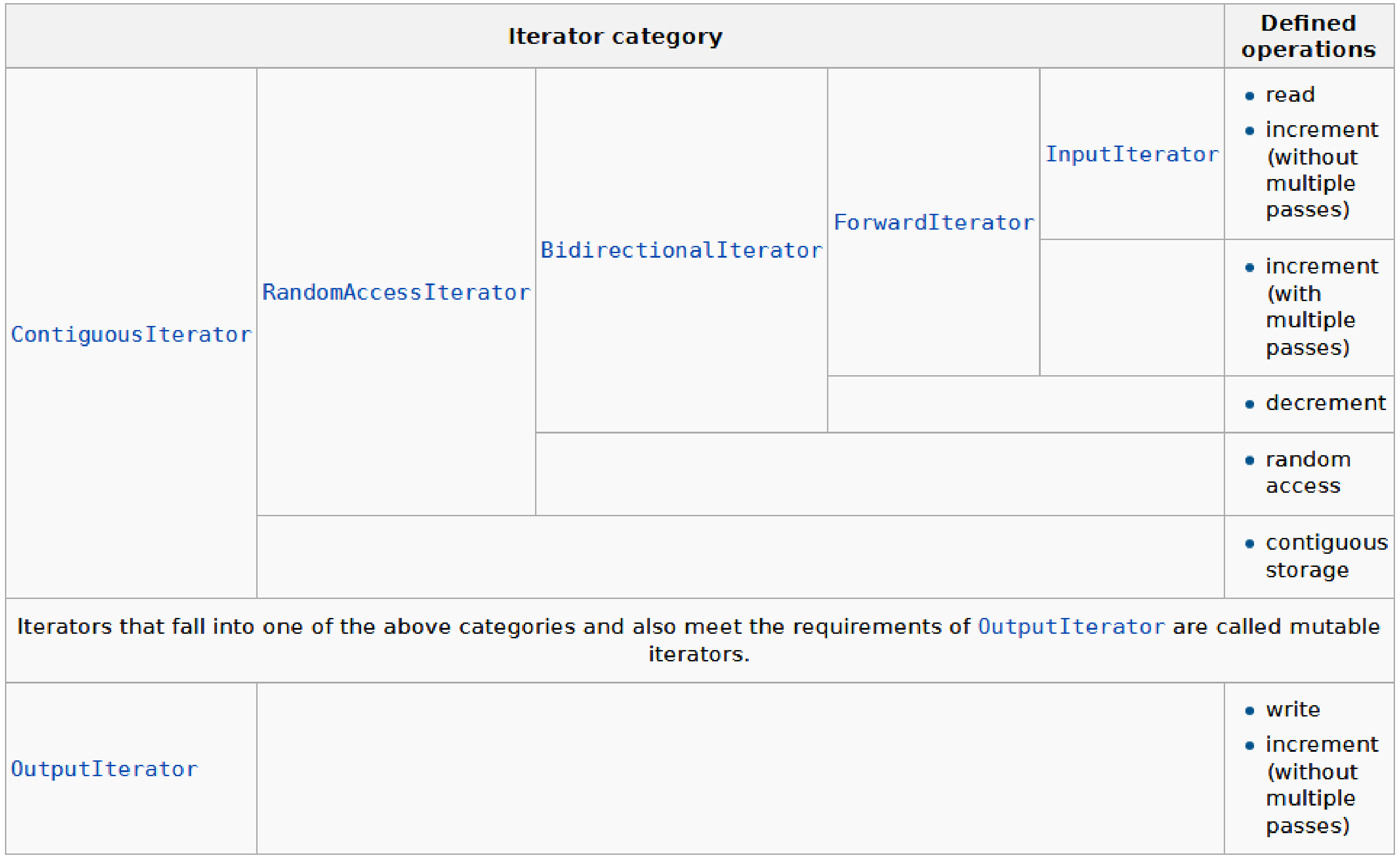
The benefits of iterators

- Achieve higher abstraction and flexibility
- Functions and algorithm can now be implemented using iterators
  - They do not care about the specific container

- You get very much for free: example `#include <algorithm>`
  - ”The algorithms library defines functions for a variety of purposes (e.g. searching, sorting, counting, manipulating) that operate on ranges of elements. Note that a range is defined as [first, last) where the last refers to the element past the last element to inspect or modify.” en.cppreference.com/w/cpp/algorithm
  - Use `algorithm` rather than some hand-crafted solutions
  - Since C++17 you can choose an execution policy
    - `sequenced_policy`
    - `parallel_policy`
    - `parallel_unsequenced_policy`

Serialization

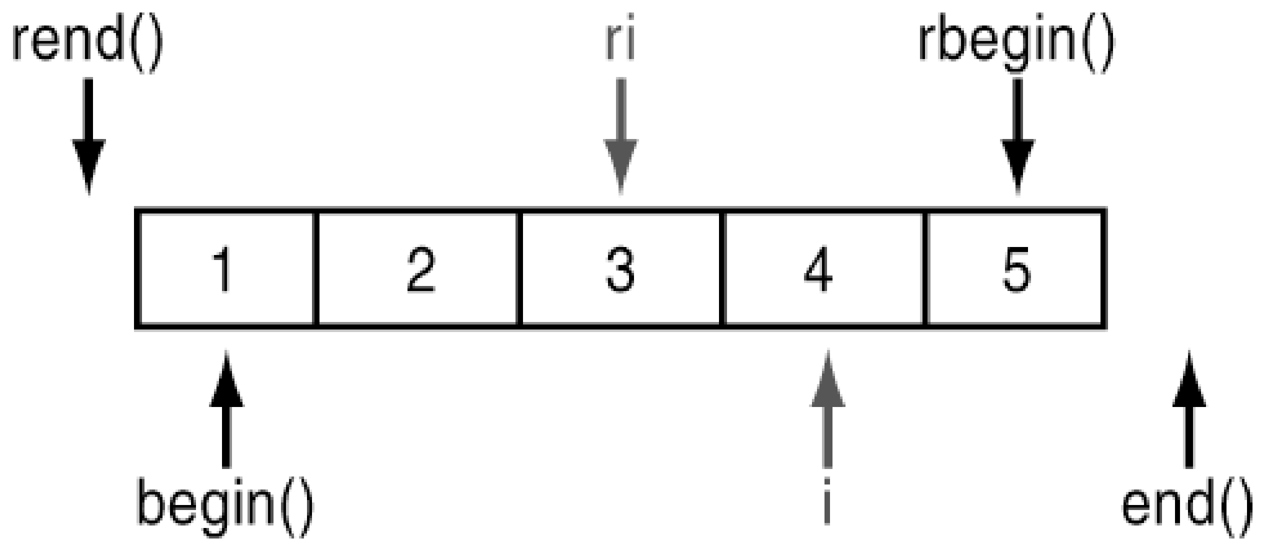
- Six categories of iterators exist
  - InputIterator
  - OutputIterator
  - ForwardIterator
  - BidirectionalIterator
  - RandomAccessIteratior
  - ContiuguousIterator (C++ 17)



Iterator category					Defined operations
ContiguousIterator	RandomAccessIterator	BidirectionalIterator	ForwardIterator	InputIterator	• read • increment (without multiple passes)
					• increment (with multiple passes)
					• decrement
					• random access
					• contiguous storage
Iterators that fall into one of the above categories and also meet the requirements of OutputIterator are called mutable iterators.					
OutputIterator					• write • increment (without multiple passes)

[Figure taken from http://en.cppreference.com/w/cpp/iterator]

Iterators



[Figure taken from http://www.drdobbs.com/cpp/three-guidelines-for-effective-iterator/184401406?pgno=3]

Examples using `#include <algorithm>`

```
int main() {
  std::vector<int> vi = {1, 2, 3, 4, 5, 6, 6, 0};
  std::cout << std::all_of(vi.begin(), vi.end(), [](int i) { return i > 0; }) << '\n';
  std::cout << std::any_of(vi.begin(), vi.end(), [](int i) { return i < 2; }) << '\n';
  std::cout << std::count(vi.begin(), vi.end(), 6) << '\n';
  std::multiset< std::string> ms = {"Hello", "World", "!", "!", "!"};
  std::cout << (std::find(ms.begin(), ms.end(), "World") != ss.end()) << '\n';
  std::list<int> li = {4, 5, 6, 1, 2, 19, 32};
  std::vector<int> vli(li.size());
  std::copy(li.begin(), li.end(), vli.begin());
  std::sort(vli.begin(), vli.end());
  std::copy(vli.begin(), vli.end(), std::ostream_iterator<int>(std::cout, ", "));
  std::array<int, 3> ai = {100, 200, 300};
  std::set<int> si;
  std::set_union(li.begin(), li.end(), ai.begin(), ai.end(),
  std::inserter(si, si.begin()));
  std::copy(si.begin(), si.end(), std::ostream_iterator<int>(std::cout, ", "));
  return 0;
}
```

How could an implementation of `std::find` look like?

```cpp
#include <iostream>
#include <vector>
template<typename InputIt, typename T>
InputIt find(InputIt first, InputIt last, const T &value) {
  for (; first != last; ++first) {
    if (*first == value) {
      return first;
    }
  }
  return last;
}
int main() {
  std::vector<int> vi = {1, 2, 13, 6, 0};
  std::cout << (find(vi.begin(), vi.end(), 13) != vi.end()) << '\n';
  return 0;
}
```

Too good to be true?

- A few caveats
  - Iterators are only pointers
  - Pointers are not very smart
    - They only point to memory
  - Iterators can be invalid (you better know when they become invalid)
    - Leads to unnecessary and time-consuming debugging sessions
    - **Check if a member function invalidates your iterator(s)**
      - Do not use member functions blindly
  - Implementing your own iterators can be really hard
    - Depending on the structure of your container

C++ iterator bug I

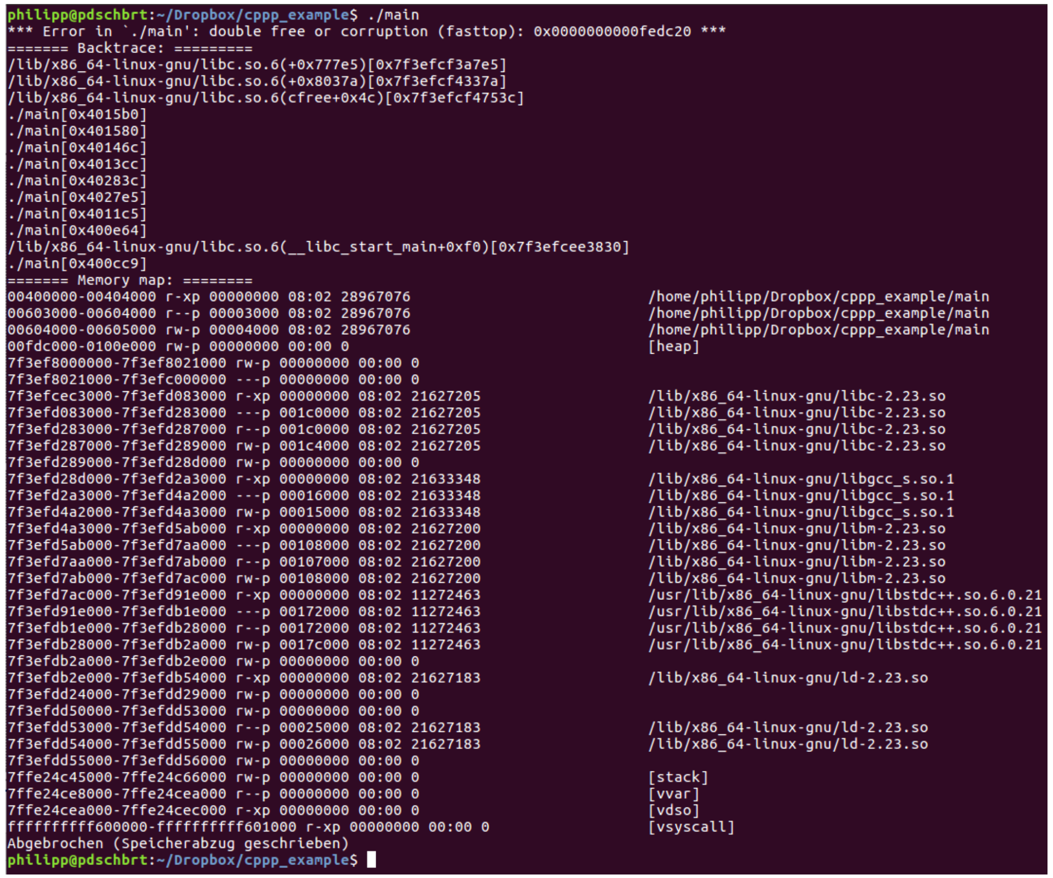
- **Lookup member functions when dealing with iterators**

```cpp
#include <iostream>
#include <set>

int main() {
  std::set<int> c = {1, 2, 3, 4, 5, 6, 7, 8, 9};
  // erase all odd numbers from c
  for (auto it = c.begin(); it != c.end();)
    if (*it % 2 == 1)
       c.erase(it);
    else
      ++it;
  for (int n : c) std::cout << n << ' ';
  return 0;
}
```

Must have been:

```cpp
it = c.erase(it);
```

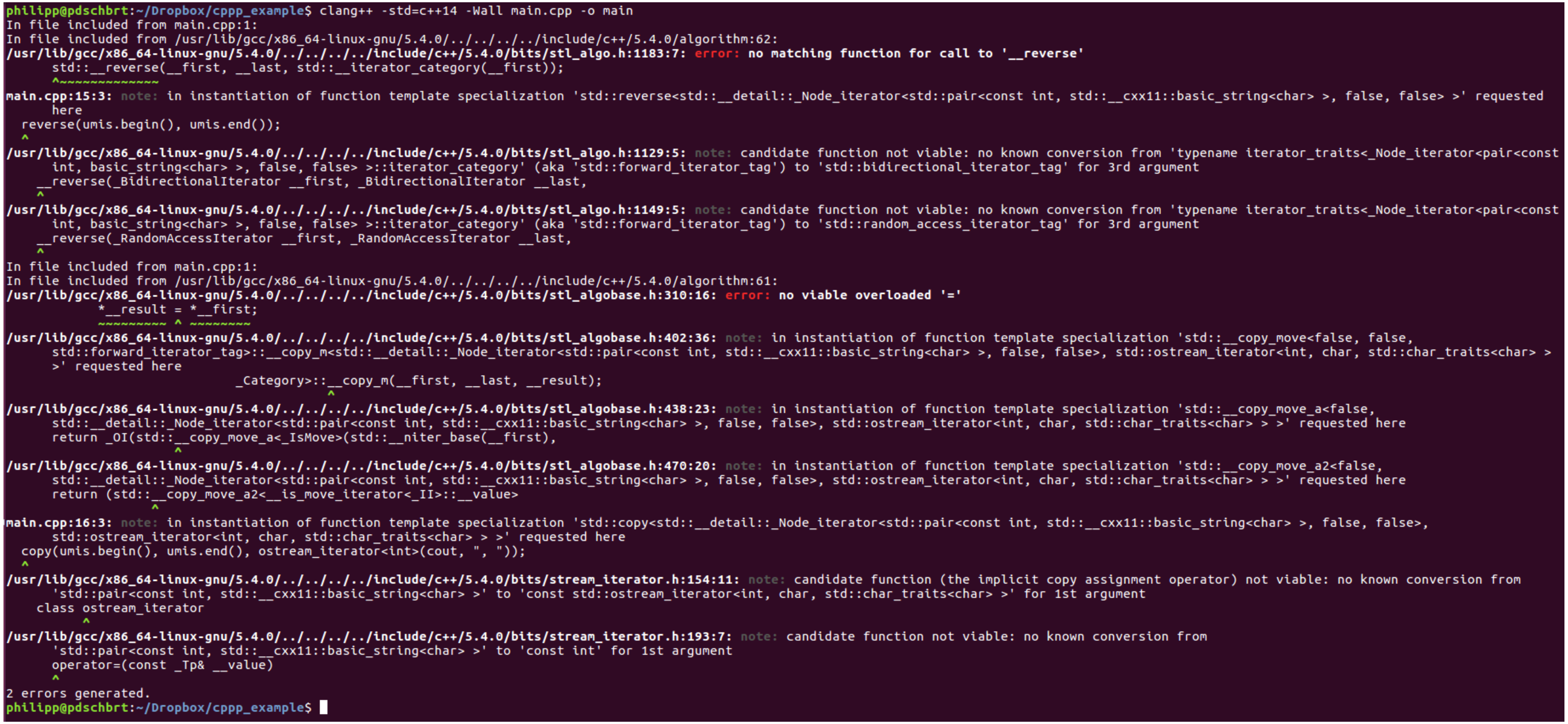


```
philipp@pdschbrt:~/Dropbox/cppp_example$ ./main
*** Error in `./main': double free or corruption (fasttop): 0x0000000000fedc20 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f3efcf3a7e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x8037a)[0x7f3efcf4337a]
/lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7f3efcf4753c]
./main[0x4015b0]
./main[0x401580]
./main[0x40146c]
./main[0x4013cc]
./main[0x40283c]
./main[0x4027e5]
./main[0x4011c5]
./main[0x400e64]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7f3efcee3830]
./main[0x400cc9]
======= Memory map: ========
00400000-00404000 r-xp 00000000 08:02 28967076                           /home/philipp/Dropbox/cppp_example/main
00603000-00604000 r--p 00003000 08:02 28967076                           /home/philipp/Dropbox/cppp_example/main
00604000-00605000 rw-p 00004000 08:02 28967076                           /home/philipp/Dropbox/cppp_example/main
00fdc000-0100e000 rw-p 00000000 00:00 0                                  [heap]
7f3ef8000000-7f3ef8021000 rw-p 00000000 00:00 0
7f3ef8021000-7f3efc000000 ---p 00000000 00:00 0
7f3efcec3000-7f3efd083000 r-xp 00000000 08:02 21627205                   /lib/x86_64-linux-gnu/libc-2.23.so
7f3efd083000-7f3efd283000 ---p 001c0000 08:02 21627205                   /lib/x86_64-linux-gnu/libc-2.23.so
7f3efd283000-7f3efd287000 r--p 001c0000 08:02 21627205                   /lib/x86_64-linux-gnu/libc-2.23.so
7f3efd287000-7f3efd289000 rw-p 001c4000 08:02 21627205                   /lib/x86_64-linux-gnu/libc-2.23.so
7f3efd289000-7f3efd28d000 rw-p 00000000 00:00 0
7f3efd28d000-7f3efd2a3000 r-xp 00000000 08:02 21633348                   /lib/x86_64-linux-gnu/libgcc_s.so.1
7f3efd2a3000-7f3efd4a2000 ---p 00016000 08:02 21633348                   /lib/x86_64-linux-gnu/libgcc_s.so.1
7f3efd4a2000-7f3efd4a3000 rw-p 00015000 08:02 21633348                   /lib/x86_64-linux-gnu/libgcc_s.so.1
7f3efd4a3000-7f3efd5ab000 r-xp 00000000 08:02 21627200                   /lib/x86_64-linux-gnu/libm-2.23.so
7f3efd5ab000-7f3efd7aa000 ---p 00108000 08:02 21627200                   /lib/x86_64-linux-gnu/libm-2.23.so
7f3efd7aa000-7f3efd7ab000 r--p 00107000 08:02 21627200                   /lib/x86_64-linux-gnu/libm-2.23.so
7f3efd7ab000-7f3efd7ac000 rw-p 00108000 08:02 21627200                   /lib/x86_64-linux-gnu/libm-2.23.so
7f3efd7ac000-7f3efd91e000 r-xp 00000000 08:02 11272463                   /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
7f3efd91e000-7f3efdb1e000 ---p 00172000 08:02 11272463                   /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
7f3efdb1e000-7f3efdb28000 r--p 00172000 08:02 11272463                   /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
7f3efdb28000-7f3efdb2a000 rw-p 0017c000 08:02 11272463                   /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
7f3efdb2a000-7f3efdb2e000 rw-p 00000000 00:00 0
7f3efdb2e000-7f3efdb54000 r-xp 00000000 08:02 21627183                   /lib/x86_64-linux-gnu/ld-2.23.so
7f3efdd24000-7f3efdd29000 rw-p 00000000 00:00 0
7f3efdd50000-7f3efdd53000 rw-p 00000000 00:00 0
7f3efdd53000-7f3efdd54000 r--p 00025000 08:02 21627183                   /lib/x86_64-linux-gnu/ld-2.23.so
7f3efdd54000-7f3efdd55000 rw-p 00026000 08:02 21627183                   /lib/x86_64-linux-gnu/ld-2.23.so
7f3efdd55000-7f3efdd56000 rw-p 00000000 00:00 0
7ffe24c45000-7ffe24c66000 rw-p 00000000 00:00 0                          [stack]
7ffe24ce8000-7ffe24cea000 r--p 00000000 00:00 0                          [vvar]
7ffe24cea000-7ffe24cec000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
Abgebrochen (Speicherabzug geschrieben)
philipp@pdschbrt:~/Dropbox/cppp_example$
```

C++ iterator bug II

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <list>
#include <unordered_map>
#include <string>
#include <vector>
int main() {
  std::vector<int> vi = {3, 2, 1};
  std::reverse(vi.begin(), vi.end());
  std::unordered_map<int, string> umis = {{3, "C"}, {2, "B"}, {1, "A"}};
  std::reverse(umis.begin(), umis.end());
  return 0;
}
```

C++ iterator bug II



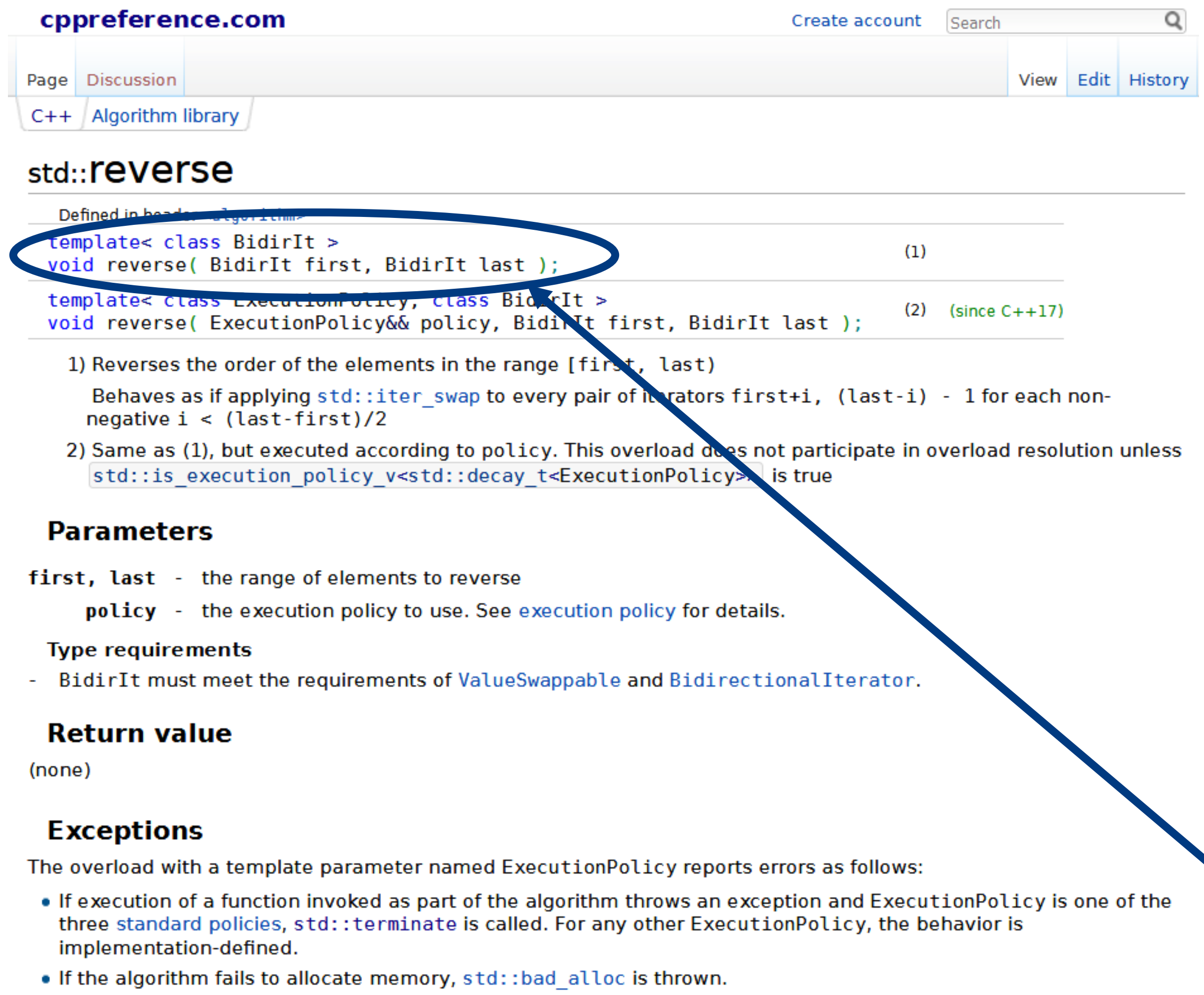
```
philipp@pdschbrt:~/Dropbox/cppp_example$ clang++ -std=c++14 -Wall main.cpp -o main
In file included from main.cpp:1:
In file included from /usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/algorithm:62:
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algo.h:1183:7: error: no matching function for call to '__reverse'
      std::__reverse(__first, __last, std::__iterator_category(__first));
      ^~~~~~~~~~~~~~
main.cpp:15:3: note: in instantiation of function template specialization 'std::reverse<std::__detail::_Node_iterator<std::pair<const int, std::__cxx11::basic_string<char> >, false, false> >' requested
      here
  reverse(umis.begin(), umis.end());
  ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algo.h:1129:5: note: candidate function not viable: no known conversion from 'typename iterator_traits<_Node_iterator<pair<const
      int, basic_string<char> >, false, false> >::iterator_category' (aka 'std::forward_iterator_tag') to 'std::bidirectional_iterator_tag' for 3rd argument
    __reverse(_BidirectionalIterator __first, _BidirectionalIterator __last,
    ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algo.h:1149:5: note: candidate function not viable: no known conversion from 'typename iterator_traits<_Node_iterator<pair<const
      int, basic_string<char> >, false, false> >::iterator_category' (aka 'std::forward_iterator_tag') to 'std::random_access_iterator_tag' for 3rd argument
    __reverse(_RandomAccessIterator __first, _RandomAccessIterator __last,
    ^
In file included from main.cpp:1:
In file included from /usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/algorithm:61:
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algobase.h:310:16: error: no viable overloaded '='
            *__result = *__first;
            ~~~~~~~~~ ^ ~~~~~~~~
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algobase.h:402:36: note: in instantiation of function template specialization 'std::__copy_move<false, false,
      std::forward_iterator_tag>::__copy_m<std::__detail::_Node_iterator<std::pair<const int, std::__cxx11::basic_string<char> >, false, false>, std::ostream_iterator<int, char, std::char_traits<char> >
      >' requested here
                              _Category>::__copy_m(__first, __last, __result);
                                          ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algobase.h:438:23: note: in instantiation of function template specialization 'std::__copy_move_a<false,
      std::__detail::_Node_iterator<std::pair<const int, std::__cxx11::basic_string<char> >, false, false>, std::ostream_iterator<int, char, std::char_traits<char> > >' requested here
      return _OI(std::__copy_move_a<_IsMove>(std::__niter_base(__first),
                      ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algobase.h:470:20: note: in instantiation of function template specialization 'std::__copy_move_a2<false,
      std::__detail::_Node_iterator<std::pair<const int, std::__cxx11::basic_string<char> >, false, false>, std::ostream_iterator<int, char, std::char_traits<char> > >' requested here
      return (std::__copy_move_a2<__is_move_iterator<_II>::__value>
                   ^
main.cpp:16:3: note: in instantiation of function template specialization 'std::copy<std::__detail::_Node_iterator<std::pair<const int, std::__cxx11::basic_string<char> >, false, false>,
      std::ostream_iterator<int, char, std::char_traits<char> > >' requested here
  copy(umis.begin(), umis.end(), ostream_iterator<int>(cout, ", "));
  ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stream_iterator.h:154:11: note: candidate function (the implicit copy assignment operator) not viable: no known conversion from
      'std::pair<const int, std::__cxx11::basic_string<char> >' to 'const std::ostream_iterator<int, char, std::char_traits<char> >' for 1st argument
    class ostream_iterator
          ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stream_iterator.h:193:7: note: candidate function not viable: no known conversion from
      'std::pair<const int, std::__cxx11::basic_string<char> >' to 'const int' for 1st argument
      operator=(const _Tp& __value)
      ^
2 errors generated.
philipp@pdschbrt:~/Dropbox/cppp_example$ 
```

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer
IEM

C++ iterator bug II

- Templates do not carry type information ☹
- **Check the stuff you are using**



std::reverse

Defined in header `<algorithm>`

```
template< class BidirIt >
void reverse( BidirIt first, BidirIt last );                                        (1)

template< class ExecutionPolicy, class BidirIt >
void reverse( ExecutionPolicy&& policy, BidirIt first, BidirIt last );              (2) (since C++17)
```

1) Reverses the order of the elements in the range `[first, last)`

Behaves as if applying `std::iter_swap` to every pair of iterators `first+i, (last-i) - 1` for each non-negative `i < (last-first)/2`

2) Same as (1), but executed according to `policy`. This overload does not participate in overload resolution unless `std::is_execution_policy_v<std::decay_t<ExecutionPolicy>>` is true

Parameters

first, last	-	the range of elements to reverse
policy	-	the execution policy to use. See execution policy for details.

Type requirements

- `BidirIt` must meet the requirements of `ValueSwappable` and `BidirectionalIterator`.

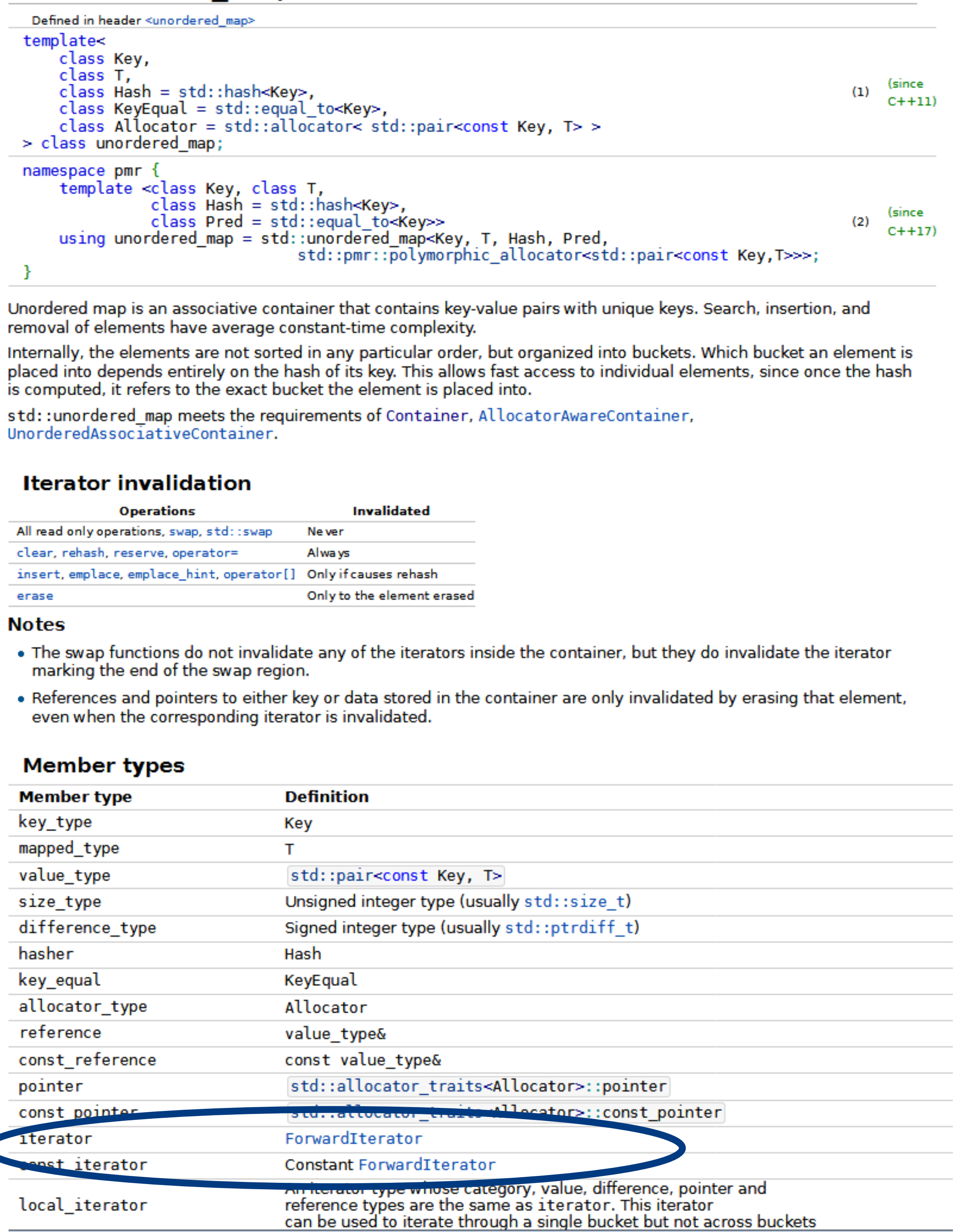
Return value

(none)

Exceptions

The overload with a template parameter named `ExecutionPolicy` reports errors as follows:

- If execution of a function invoked as part of the algorithm throws an exception and `ExecutionPolicy` is one of the three standard policies, `std::terminate` is called. For any other `ExecutionPolicy`, the behavior is implementation-defined.
- If the algorithm fails to allocate memory, `std::bad_alloc` is thrown.



Defined in header `<unordered_map>`

```
template<
    class Key,
    class T,
    class Hash = std::hash<Key>,                                                   (1) (since
    class KeyEqual = std::equal_to<Key>,                                               C++11)
    class Allocator = std::allocator< std::pair<const Key, T> >
> class unordered_map;

namespace pmr {
    template <class Key, class T,
              class Hash = std::hash<Key>,                                         (2) (since
              class Pred = std::equal_to<Key>>                                         C++17)
    using unordered_map = std::unordered_map<Key, T, Hash, Pred,
                              std::pmr::polymorphic_allocator<std::pair<const Key,T>>>;
}
```

Unordered map is an associative container that contains key-value pairs with unique keys. Search, insertion, and removal of elements have average constant-time complexity.

Internally, the elements are not sorted in any particular order, but organized into buckets. Which bucket an element is placed into depends entirely on the hash of its key. This allows fast access to individual elements, since once the hash is computed, it refers to the exact bucket the element is placed into.

`std::unordered_map` meets the requirements of `Container`, `AllocatorAwareContainer`, `UnorderedAssociativeContainer`.

Iterator invalidation

Operations	Invalidated
All read only operations, `swap`, `std::swap`	Never
`clear`, `rehash`, `reserve`, `operator=`	Always
`insert`, `emplace`, `emplace_hint`, `operator[]`	Only if causes rehash
`erase`	Only to the element erased

Notes

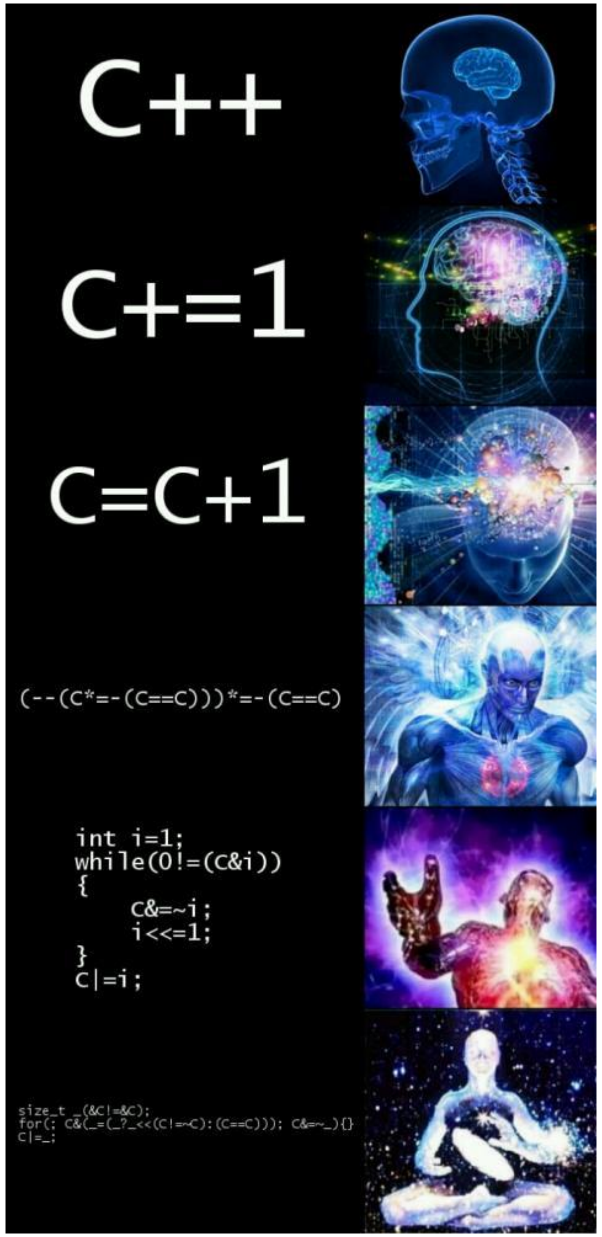
- The swap functions do not invalidate any of the iterators inside the container, but they do invalidate the iterator marking the end of the swap region.
- References and pointers to either key or data stored in the container are only invalidated by erasing that element, even when the corresponding iterator is invalidated.

Member types

Member type	Definition
`key_type`	`Key`
`mapped_type`	`T`
`value_type`	`std::pair<const Key, T>`
`size_type`	Unsigned integer type (usually `std::size_t`)
`difference_type`	Signed integer type (usually `std::ptrdiff_t`)
`hasher`	`Hash`
`key_equal`	`KeyEqual`
`allocator_type`	`Allocator`
`reference`	`value_type&`
`const_reference`	`const value_type&`
`pointer`	`std::allocator_traits<Allocator>::pointer`
`const_pointer`	`std::allocator_traits<Allocator>::const_pointer`
`iterator`	`ForwardIterator`
`const_iterator`	Constant `ForwardIterator`
`local_iterator`	An iterator type whose category, value, difference, pointer and reference types are the same as `iterator`. This iterator can be used to iterate through a single bucket but not across buckets

Always be critical and suspicious

- A nice talk by Felix von Leitner
- “A Case Against C++, why C++ is bad for the environment, causes global warming and kills puppies”
  - https://media.ccc.de/v/cccamp07-en-1951-A_Case_Against_C++



```
C++
```

```
C+=1
```

```
C=C+1
```

```
(--(C*=-(C==C)))*=-(C==C)
```

```
int i=1;
while(0!=(C&i))
{
    C&=~i;
    i<<=1;
}
C|=i;
```

```
size_t _(&C!=&C);
for(; C&(_=(_?_<<(C!=~C):(C==C))); C&=~_){}
C|=_;
```

Recap

- Libraries
  - STL
  - BOOST
  - Qt
  - Armadillo
  - OpenCV
  - OpenGL / Vulkan
  - OpenCL / CUDA
  - OpenMP
  - Google Test
  - Google Abseil
  - Google Protobuf
- Iterators

Thank you for your attention
Questions?