Contents

1. Libraries
2. Iterators
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
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 a STL implementation
- Do not think STL is perfect for every task

When your C++ compiler throws an error.
BOOST

- Was founded by C++ standardization committee members
- A collection of **portable** sub-libraries
- Sub-libraries are distinguished by the task to 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

Using BOOST and other libraries in general

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

```cpp
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;
```
There are many more useful libraries
Check if boost can solve your problem

Boost documentation

A hands-on tutorial guide (with code examples)
  * The Boost C++ Libraries
    * [https://theboostcpplibraries.com/](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

[Figure taken from http://i1-linux.softpedia-static.com/screenshots/Qt_1.jpg]
#include <iostream>
#include <armadillo>
using namespace std;
using namespace arma;

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

- 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)
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
- Lots of useful stuff
  - Fourier transformation
  - Support vector machine (SVM)
  - Edge detection
  - GUI elements
- DFT would be just a few lines of code

```cpp
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace std;
using namespace cv;
int main(int argc, char** argv) {
    Mat lena = imread("pictures/lenna.png", CV_LOAD_IMAGE_COLOR);
    imshow("opencvtest", lena);
    waitKey();
    return 0;
}
```
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 is usually done by ray tracing techniques (currently too slow for real time graphics)

```c
#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;
}
```
#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;
  }
  ...
OpenCL / 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
    - 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
#include <iostream>
#include <vector>
#include <omp.h> // note .h
using namespace std;

int main() {
    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)
        cout << vi[i] << '\n';
    return 0;
}
Google Test

- 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

main.cpp

src1.h
src1.cpp
src1Test.cpp

src2.h
src2.cpp
src2Test.cpp

...

[Figure taken from http://www.qatestingtools.com/code.google/googletest]
#include <iostream>
#include "src.h"
using namespace std;

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

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 <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
  - Example: 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 = absl::StrJoin(v, "-");
  std::cout << "Joined string: " << s << "\n";
  return 0;
}
```
Libraries and the linker

- 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
  - `-IDLIRECTORY`
  - 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 `/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`

```cpp
#include <boost/filesystem.hpp>
#include <iostream>

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;
    }
}
```
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 datatype (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 containers
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 defined 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](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`
Iterators

- Six categories of iterators exist
  - InputIterator
  - OutputIterator
  - ForwardIterator
  - BidirectionalIterator
  - RandomAccessIterator
  - ContiguousIterator (C++ 17)

[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>`

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

```cpp
#include <iostream>
#include <vector>
using namespace std;

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() {
    vector<int> vi = {1, 2, 13, 6, 0};
    cout << (find(vi.begin(), vi.end(), 13) != vi.end()) << '
';
    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
  - 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>
using namespace std;

int main() {
    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) cout << n << ' ';
}
```

Must have been:

```
    it = c.erase(it);
```
C++ iterator bug II

```cpp
#include <algorithm>
#include <iostream>
#include <iterator>
#include <list>
#include <unordered_map>
#include <string>
#include <vector>

using namespace std;

int main() {
    vector<int> vi = {3, 2, 1};
    reverse(vi.begin(), vi.end());

    unordered_map<int, string> umis = {{3, "C"}, {2, "B"}, {1, "A"}};
    reverse(umis.begin(), umis.end());

    return 0;
}
```
C++ iterator bug II

```
philippdpschrbt@Dropbox/cpp_example$ clang++ -std=c++14 -Wall main.cpp -o main
In file included from main.cpp:
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/.../.../include/c++/5.4.0/algorithm:
/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:1513:12: note: in instantiation of function template specialization 'std::reverse<__detail::node_iterator<std::pair<const int, std::__cxx11::basic_string<char>>>>, false, false>' requested here
   reverse(unis.begin(), unis.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>>>>::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:1149:15: note: candidate function not viable: no known conversion from 'typename iterator_traits<node_iterator<pair<const int, basic_string<char>>>>::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:
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/.../.../include/c++/5.4.0/algorithm:
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.../.../include/c++/5.4.0/bits/stl_algsbase.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_algsbase.h:402:36: note: in instantiation of function template specialization 'std::__copy_move<false, false, std::forward_iterator_tag::copy_move, 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
   _result = _first;
               ^
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.../.../include/c++/5.4.0/bits/stl_algsbase.h:438:23: note: candidate function not viable: no known conversion from 'typename iterator_traits<node_iterator<pair<const int, basic_string<char>>>>::iterator_category' (aka 'std::forward_iterator_tag') to 'std::iterator_base' for 1st argument
    return _01(stl::copy_move_in_ismove(stl::iterator_base(_first)),
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.../.../include/c++/5.4.0/bits/stl_algsbase.h:470:20: note: candidate function not viable: no known conversion from 'typename iterator_traits<node_iterator<pair<const int, basic_string<char>>>>::iterator_category' (aka 'std::forward_iterator_tag') to 'std::ostream_iterator<int, char, std::char_traits<char>>' for 1st argument
    return stl::copy_move_in_ismove(stl::iterator_base(_first),
main.cpp:1619:34: note: in instantiation of function template specialization 'std::__copy_move<false, false, std::forward_iterator_tag::copy_move, 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(unis.begin(), unis.end()), ostrom.iterator(int,cout, "s");
            ^
/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 std::ostream_iterator<int, char, std::char_traits<char>>' for 1st argument
  operator<<(const T& __value)
       ^
```

2 errors generated.

philippdpschrbt@Dropbox/cpp_example$
C++ iterator bug II

- Templates do not carry type information 😐

- Check the stuff you are using

```cpp
#include <algorithm>

template<
    class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = std::allocator<std::pair<const Key, T>>,
>
class unordered_map;

namespace std {
    template<class Key, class T, class Hash = std::hash<Key>,
    class Pred = std::equal_to<Key>,
>
    class unordered_map;

    using unordered_map = std::unordered_map<Key, T, Hash, Pred,
    std::pair<policy::policy<Allocator, std::pair<const Key, T>>>::type>;
}
```

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, and UnorderedAssociativeContainer.

### Iterator Invalidation

#### Operations

- `swap`, `equal_range`, `swap_ranges`:
  - `swap` does not invalidate any of the iterators inside the container, but they do invalidate the iterator marking the end of the swap region.
  - `equal_range` and `swap_ranges` invalidate the iterator marking the end of the range.

#### Notes

- 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

<table>
<thead>
<tr>
<th>Member type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_type</td>
<td>Key</td>
</tr>
<tr>
<td>mapped_type</td>
<td>T</td>
</tr>
<tr>
<td>value_type</td>
<td>std::pair&lt;const Key, T&gt;</td>
</tr>
<tr>
<td>size_type</td>
<td>Unsigned integer type (usually std::size_t)</td>
</tr>
<tr>
<td>difference_type</td>
<td>Signed integer type (usually std::ptrdiff_t)</td>
</tr>
<tr>
<td>hasher</td>
<td>Hash</td>
</tr>
<tr>
<td>key_equal</td>
<td>KeyEqual</td>
</tr>
<tr>
<td>allocator_type</td>
<td>Allocator</td>
</tr>
<tr>
<td>reference</td>
<td>value_type</td>
</tr>
<tr>
<td>const_reference</td>
<td>const value_type</td>
</tr>
<tr>
<td>pointer</td>
<td>std::allocator_traits&lt;Allocator&gt;::pointer</td>
</tr>
<tr>
<td>const_pointer</td>
<td>std::allocator_traits&lt;Allocator&gt;::const_pointer</td>
</tr>
<tr>
<td>iterator</td>
<td>ForwardIterator</td>
</tr>
<tr>
<td>const_iterator</td>
<td>Constant ForwardIterator</td>
</tr>
<tr>
<td>local_iterator</td>
<td>An iterator of 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.</td>
</tr>
</tbody>
</table>

### 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.
Always be highly 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++
Recap

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

- Iterators
Thank you for your attention

Questions?