Contents

1. Libraries
2. Iterators
Libraries

- Why solve a task that has been solved already?
  - 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
- (Do not forget to link against the library to be used)
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++, …
- 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
  - But for most of your purposes it probably is
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 (header only/.hpp)
  - Why? Because of template code
- Contains almost all useful C++ code you can think of
- 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 in programming C++
- Boost has usable 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
- Use google for task to solve/ library to use
  - *C++ <what you want> <what ever library you want to use>*
    - e.g.: *c++ serialize objects boost*
- Open first few links of google results
- Focus on what looks promising
- Do not use code blindly! (Never just copy and paste)
- Try to understand the code
- Write small minimal 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() << '
';
  if (bfs::exists(r) && !bfs::is_directory(r)) {
    std::cout << r.stem().string() << '
';
    std::cout << r.extension().string() << '
';
  }
  if (bfs::exists(p) && bfs::is_directory(p)) {
    bfs::directory_iterator dit(p);
    while (dit != bfs::directory_iterator{}) std::cout << *dit++ << '
';
  }
  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; }
    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();
    auto &lg = lg::get();
    BOOST_LOG_SEV(lg, DEBUG) << "I am debugging!";
    return 0;
}
```

The `boost/logger` module provides a set of logging utilities for the Boost library. The `severity_level` enum defines a set of severity levels for log messages. The `LogFormatter` function formats log messages by extracting severity and message attributes from a `record_view`. The `initializeLogger` function sets up the logging infrastructure by registering a `synchronous_sink` that outputs to the standard log stream. The `main` function demonstrates the use of the logger by logging a severe message.
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
- The Boost C++ Libraries
  - https://theboostcpplibraries.com/
Qt

- Qt (cutie)
- 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 & reflection
- Interfaces for other languages provided
  - Python, Ruby, C#, Java, …
- High quality IDE Qt Creator
  - Includes GUI designer
  - You want to use Qt Creator when developing Qt applications

[Figure 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> - just a typedef
    mat A = randu<mat>(3,3);
    mat B = randu<mat>(3,3);
    mat C = A * B;
    cout << C;
    cout << "------------
";
    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;
}
OpenCV

- High quality C++ library for computer vision
- For academic and industrial use
- As efficient as possible 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
  - All you can think of corresponding to digital image processing
- DFT would be just a few lines of code more

```c++
#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/lena.png", CV_LOAD_IMAGE_COLOR);
    imshow("opencvtest", lena);
    waitKey();
    return 0;
}
```
OpenGL

- 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
- Today: Vulkan

- Animated movies is 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;
}
```
OpenCL/ CUDA

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

```c
#include <stdio.h>
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;
    cudaMemcpy((void **) &dev_a, Host_a ,N*sizeof(int),
               cudaMemcpyHostToDevice);
    cudaMemcpy((void **) &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 ;
}
```
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
- Perform computations on data
- 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”
  - Different model of thinking
    - Quite hard at the beginning
- More general applicable than OpenGL, which is “graphics only”-computations
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
  - Computation in different threads
- Synchronization via \#pragma omp critical
- Programs work correct even if compiler does not support OpenMP
- Every C/C++ programmer should know about the basics
- OMP introduces embarrassing parallel problems

```cpp
#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;
}
```

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

- Testing framework for C and C++
- Provides basic infrastructure for automatic code testing
- Allows to write standardized test cases
- Can be compiled and executed fully automatically
- Detail information are provided when tests fail
  - Test name, line number, assertion that was made, …
- De facto standard testing framework
- Is used by many meaningful and 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

...
Consider the example from the website

```cpp
#include <iostream>
#include "src.h"
using namespace std;

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

```cpp
#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));
}
```

```cpp
int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```
What are 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 iterating functionality

Idea:
- Just provide some functionality

Problem:
- Every container type looks different

Solution:
- Specify a common concept `Iterator` that can/must be implemented

[Figure taken from http://en.cppreference.com/w/cpp/iterator]
The benefits of iterators

1. Templates make algorithms independent of the data types
2. Iterators make algorithms independent of the containers

[Figure taken from http://www.bogotobogo.com/cplusplus/stl3_iterators.php]
The benefits of iterators

- Achieve more 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. Not 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](https://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)
Iterators

- How can one obtain an iterator?

[Figure taken from http://www.drdobbs.com/cpp/three-guidelines-for-effective-iterator/184401406?pgno=3]
Example 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> ss = {"Hello", "World", "!", "!", "!"};
    cout << (find(ss.begin(), ss.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()) << 'n';
    return 0;
}
```
Too good to be true?

A few caveats

- Iterators are just pointers under the hood
- Pointers are not very smart
  - They only point to memory
- Iterators can be invalid
  - Leads to unnecessary and time-consuming bugs
- Check if a member function invalidates your iterator(s)
  - Do not use member functions blindly
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

```
philipp@gdschrbt@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/.......
  /usr/include/c++/5.4.0/algorithm:62:
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/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:153: 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(unis.begin(), unis.end());

/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/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/.......
  /usr/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>>> >::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:
In file included from /usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/include/c++/5.4.0/algorithm:81:
/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/algorithm:310:16: error: no viable overloaded '='
 _result = _first;

/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/algorithm: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/.......
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/algorithm:438:23: note: in instantiation of function template specialization 'std::__copy_move<false, 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 __01(stcl::copy_move_or_ismove(stcl::move_base(_first),

/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/algorithm:470:20: note: in instantiation of function template specialization 'std::__copy_move_or_ismove<false, 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 __01(stcl::copy_move_or_ismove(stcl::move_base(_first),

main.cpp:161:3: note: in instantiation of function template specialization 'std::copy<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
 copy(unis.begin(), unis.end()), ostream_iterator<int>(cout, "");

/usr/lib/gcc/x86_64-linux-gnu/5.4.0/.......
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/algorithm: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/.......
  /usr/lib/gcc/x86_64-linux-gnu/5.4.0/algorithm: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(constTp& value)

2 errors generated.
philipp@gdschrbt@Dropbox/cpp_example$`
```
C++ iterator bug II

- Templates do not carry type information

Check the stuff you are using

cppreference.com

```cpp
std::reverse

Defined in header `<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 Allocator = std::allocator<std::pair<const Key, T>>,
    >
    class unordered_map;
```

```}
```

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 HashableContainer.
```

**Iterator Invalidation**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Invalidated</th>
</tr>
</thead>
<tbody>
<tr>
<td>All read-only operations, swap, std::swap</td>
<td>Never</td>
</tr>
<tr>
<td>clear, reallocate, operator[]</td>
<td>Only if caused by rehash</td>
</tr>
<tr>
<td>insert, emplace, emplace_hint, operator()</td>
<td>Only to the element erased</td>
</tr>
</tbody>
</table>

**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**
- **value_type**
- **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

- A nice talk from 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++](https://media.ccc.de/v/cccamp07-en-1951-A_Case_Against_C++)
Recap

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

- Iterators
Thank you for your attention

Questions?