C++ Programming

Lecture 10
Software Engineering Group

Philipp D. Schubert
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

Library at Melk Abbey in Austria
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? → 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
- 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
- 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
```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;
}
```
```cpp
#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>().c_str() << '\n'; }
  return 0;
}
```
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;
}
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 (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 Qt applications

[Figure taken from http://i1-linux.softpedia-static.com/screenshots/Qt_1.jpg]
```cpp
#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 << "------------\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;
}
```

**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
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
  - Everything that corresponds to digital image processing

DFT would be just a few lines of code more

```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/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
- Nowadays: 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;
}
```
OpenCL / CUDA

- Use the NVCC compiler for CUDA
- Nowadays: 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 (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]);
  }
}
```

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 split into 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;
}
```
Google Test

- Testing framework for C and C++
- Provides basic infrastructure for automated testing
- Allows 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 meaningful, modern C++ projects
- First stable release: August 2016 (Wiki)
Google Test

[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) {
    return (n <= 1) ?
            1 :
            n * f(n - 1);
}
#endif

#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();
}
**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++14 -Wall test.cpp -o test -lboost_system -lboost_filesystem
  - If a library is not installed system-wide
    - `-DIDIRECTORY`
    - **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 `-l`**
  - Assuming install directory `/home/my_user/my_library/`
    - clang++ -std=c++14 -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;
  }
  
  bfs::path p(argv[1]);
  if (bfs::exists(p)) {
    std::cout << "path exists\n";
    return 0;
  }
```

```cpp
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++14 -Wall -I../include/ -c mycode.cpp
    ar rcs libmylibrary.a mycode.o
    ```
  - Compile to dynamic library
    ```
    g++ -std=c++14 -Wall -I../include/ -shared -fPIC mycode.cpp -o libtest.so
    ```

Library structure

- mylibrary/
  - include/
    - 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;
  }
  ```
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 functionality for iteration

Idea:

- Just provide some functionality

Problem:

- Every container type looks different

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

[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 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
- 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?

![Diagram of iterators and range functions](http://www.drdobbs.com/cpp/three-guidelines-for-effective-iterator/184401406?pgno=3)
Examples using `#include <algorithm>`

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

```cpp
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

```cpp
#include <algorithm>
#include <iostream>

int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};
    std::for_each(v.begin(), v.end(), [] (int i) { std::cout << i << std::endl; });
    return 0;
}
```
C++ iterator bug II

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

cppreference.com

```
std::reverse

Defined in header <algorithm>

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

```template<class ExecutionPolicy, class BidIt first, class BidIt last>  
void reverse( ExecutionPolicy& policy, BidIt first, BidIt last );

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
   - 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

- BidIt 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++](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
- Iterators
Thank you for your attention

Questions?