Research Computing with C++

Jim Dobson Matt Clarkson Jonathan Cooper
Roma Klapaukh James Hetherington

Contents

Chapter 1

Lecture 1: C++ for Research

Course Overview

Part 1

- Using C++ in research
- Better C++
  - Reliable
  - Reproducible
  - Good science
  - Libraries

Part 2

- HPC concepts
- Shared memory parallelism - OpenMP
- Distributed memory parallelism - MPI

Course Aims

- Teach how to do research with C++
- Optimise your research output
- A taster for various technologies
- Not just C++ syntax, Google/Compiler could tell you that!

Pre-requisites

- Use of command line (Unix) shell
- You are already doing some C++
- You are familiar with your compiler
- (Maybe) You are happy with the concept of classes
- (Maybe) You know C++ up to templates?
- You are familiar with development eg. version control
  - Git: https://git-scm.com/

Course Notes

- Revise some software Engineering: MPHY0021
- Register with Moodle: PHAS0100
  - contact lecturer for key to self-register
  - guest key to look around is "996135"
- Online notes: PHAS0100

Course Assessment

- 2 pieces coursework - 40 hours each
  - See assessment section for details

Course Community

- UCL Research Programming Hub: http://research-programming.ucl.ac.uk
- Slack: https://ucl-programming-hub.slack.com

Lecture 1: C++ In Research

Problems In Research

- Poor quality software
- Excuses
  - I'm not a software engineer
  - I don't have time
  - It's just a prototype
  - I'm unsure of my code (scared to share)

C++ Disadvantages

Some people say:

- Compiled language
    - (compiler versions, libraries, platform specific etc)
- Perceived as difficult, error prone, wordy, unfriendly syntax
- Result: It's more trouble than its worth?

C++ Advantages

- Fast, code compiled to machine code
- Stable, evolving standard, powerful notation, improving
- Lots of libraries, Boost, Qt, VTK, ITK etc.
- Nice integration with CUDA, OpenACC, OpenCL, OpenMP, OpenMPI
- Result: Good reasons to use it, or you may *have* to use it

Research Programming

- Software is always expensive
    - Famous Book: Mythical Man Month
- Research programming is different to product development:
    - What is the end product?

Development Methodology?

- Will software engineering methods help?
    - Waterfall
    - Agile
- At the 'concept discovery' stage, probably too early to talk about product development

Approach

- What am I trying to achieve?
- How do I maximise my research output?
- What is the best pathway to success?
- How do I de-risk (get results, meet deadlines) my research?
- Software is an important part of scientific reproducibility, authorship, credibility.

1. Types of Code

- What are you trying to achieve?
- Divide code:
  - Your algorithm
  - Testing code
  - Data analysis
  - User Interface
  - Glue code
  - Deployment code
  - Scientific output (a paper)

Question: What type of code is C++ good for? Question: Should all code be in C++?

2. Maximise Your Value

- Developer time is expensive
- Your brain is your asset
- Write as little code as possible
- Focus tightly on your hypothesis
- Write the minimum code that produces a paper

Don't fall into the trap "Hey, I'll write a framework for that"

3. Ask Advice

- Before contemplating a new piece of software
  - Ask advice - Slack Channel
  - Review libraries and use them.
  - Check libraries are suitable, and sustainable.
  - Read Libraries section from Software Engineering course
  - Ask about best practices

Debunking The Excuses

- I'm not a software engineer
  - Learn effective, minimal tools
- I don't have time

  - Unit testing to save time
  - Choose your battles/languages wisely
- I'm unsure of my code
  - Share, collaborate

What Isn't This Course?

We are NOT suggesting that:

- C++ is the solution to all problems.
- You should write all parts of your code in C++.

What Is This Course?

We aim to:

- Improve your C++ (and associated technologies).
- Introduction to High Performance Computing (HPC).

So that:

- Apply it to research in a pragmatic fashion.
- You use the right tool for the job.

Git

Git Introduction

- This is a practical course
- We will use git for version control
- Submit git repository for coursework
- Here we provide a very minimal introduction

Git Resources

- Complete beginner - Try Git
- Git book by Scott Chacon
- Git section of MPHYG001
- MPHYG001 repo

Git Walk Through - 1

(demo on command line)

- Creating your own repo:
  - git init
  - git add
  - git commit
  - git log
  - git status
  - git remote add
  - git push

Git Walk Through - 2

(demo on command line)

- Working on existing repo:
  - git clone
  - git add
  - git commit
  - git log
  - git status

  - git push
  - git pull

Git Walk Through - 3

- Cloning or forking:
  - If you have write permission to a repo: clone it, and make modifications
  - If you don't: fork it, to make your own version, then clone that and make modifications.

Homework - 1

- Register Github
- Register for private repos - free for academia.
- Find project of interest - try cloning it, make edits, can you push?
- Find project of interest - try forking it, make edits, can you push?

CMake

Ever worked in Industry?

- (0-3yrs) Junior Developer - given environment, team support
- (4-6yrs) Senior Developer - given environment, leading team
- (7+ years) Architect - chose tools, environment, design code
- Only cross-platform if product/business demands it
- All developers told to use the given platform no choice

Ever worked in Research?

- All prototyping, no scope
- Start from scratch, little support
- No end product, no nice examples
- Cutting edge use of maths/science/technology
- Share with others on other platforms
- Develop on Windows, run on cluster (Linux)

Research Software Engineering Dilemma

- Comparing Research with Industry, in Research you have:
  - Least experienced developers
  - with the least support
  - developing cross-platform
  - No clear specification or scope
- Struggle of C++ is often not the language its the environment

Build Environment

- Windows: Visual Studio solution files
- Linux: Makefiles
- Mac: XCode projects / Makefiles

Question: How was your last project built?

CMake Introduction

- This is a practical course

- We will use CMake as a build tool
- CMake produces
  - Windows: Visual Studio project files
  - Linux: Make files
  - Mac: XCode projects, Make files
- So you write 1 build language (CMake) and run on multi-platform.
- This course will provide most CMake code and boiler plate code for you, so we can focus more on C++. But you are expected to google CMake issues and work with CMake.

CMake Usage Linux/Mac

Demo an "out-of-source" build

```
cd ~/build
git clone https://github.com/MattClarkson/CMakeHelloWorld
mkdir CMakeHelloWorld-build
cd CMakeHelloWorld-build
ccmake ../CMakeHelloWorld
make
```

Homework - 2

- Build https://github.com/MattClarkson/CMakeHelloWorld.git
- Ensure you do "out-of-source" builds
- Use CMake to configure separate Debug and Release versions
- Add code to hello.cpp:
  - On Linux/Mac re-compile just using make

Homework - 3

- Build https://github.com/MattClarkson/CMakeHelloWorld.git
- Exit all code editors
- Rename hello.cpp
- Change CMakeLists.txt accordingly
- Notice: The executable name and .cpp file name can be different
- In your build folder, just try rebuilding.
- You should see that CMake is re-triggered, so you get a cmake/compile cycle.

CMake Basics

Compiling Basics

Question: How does a compiler work?

How does a Compiler Work?

Question: How does a compiler work?

- (Don't quote any of this in a compiler theory course!)
- Preprocessing .cpp/.cxx into pure source files
- Source files compiled, one by one into .o/.obj
- executable compiled to .o/.obj
- executable linked against all .o, and all libraries

That's what you are trying to describe in CMake.

CMake - Directory Structure

- CMake starts with top-level CMakeLists.txt
- CMakeLists.txt is read top-to-bottom
- All CMake code goes in CMakeLists.txt or files included from a CMakeLists.txt
- You can sub-divide your code into separate folders.
- If you `add_subdirectory`, CMake will go into that directory and start to process the CMakeLists.txt therein. Once finished it will exit, go back to directory above and continue where it left off.
- e.g. top level CMakeLists.txt

```
project(MYPROJECT VERSION 0.0.0)
add_subdirectory(Code)
if(BUILD_TESTING)
  set()
  include_directories()
  add_subdirectory(Testing)
endif()
```

CMake - Define Targets

- Describe a target, e.g. Library, Application, Plugin

```
add_executable(hello hello.cpp)
```

- Note: You don't write compile commands
- You tell CMake what things need compiling to build a given target. CMake works out the compile commands!

CMake - Order Dependent

- You can't say "build Y and link to X" if X not defined
- So, imagine in a larger project

```
add_library(libA a.cpp b.cpp c.cpp)
add_library(libZ x.cpp y.cpp z.cpp)
target_link_libraries(libZ libA)
add_executable(myAlgorithm algo.cpp) # contains main()
target_link_libraries(myAlgorithm libA libZ ${THIRD_PARTY_LIBS})
```

- So, logically, its a big, ordered set of build commands.

Homework - 4

- Build https://github.com/MattClarkson/CMakeLibraryAndApp.git
- Look through .cpp/.h code. Ask questions if you don't understand it.
- What is an "include guard"?
- What is a namespace?
- Look at .travis.yml and appveyor.yml - cross platform testing, free for open-source
- Look at myApp.cpp, does it make sense?
- Look at CMakeLists.txt, does it make sense?
- Look for examples on the web, e.g. VTK

Intermediate CMake

What's next?

- Most people learn CMake by pasting snippets from around the web
- As project gets larger, its more complex
- Researchers tend to just "stick with what they have."
- i.e. just keep piling more code into the same file.
- Want to show you a reasonable template project.

Homework - 5

- Build https://github.com/MattClarkson/CMakeCatch2.git
- If open-source, use travis and appveyor from day 1.
- We will go through top-level CMakeLists.txt in class.
- See separate `Code` and `Testing` folders
- Separate `Lib` and `CommandLineApps` and `3rdParty`
- You should focus on
  - Write a good library
  - Unit test it
  - Then it can be called from command line, wrapped in Python, used via GUI.

Homework - 6

- Try renaming stuff to create a library of your choice.
- Create a github account, github repo, Appveyor and Travis account
- Try to get your code running on 3 platforms
- If you can, consider using this repo for your research
- Discuss
  - Debug / Release builds
  - Static versus Dynamic
  - declspec import/export
  - Issues with running command line app? Windows/Linux/Mac

Looking forward

In the remainder of this course we cover

- Some compiler options
- Using libraries
- Including libraries in CMake
- Unit testing
- i.e. How to put together a C++ project
- in addition to actual C++ and HPC

Unit Testing

What is Unit Testing?

At a high level

- Way of testing code.
- Unit
    - Smallest ‘atomic’ chunk of code
    - i.e. Function, could be a Class
- See also:
    - Integration/System Testing
    - Regression Testing
    - User Acceptance Testing

Benefits of Unit Testing?

- Certainty of correctness
- (Scientific Rigour)
- Influences and improves design
- Confidence to refactor, improve

Drawbacks for Unit Testing?

- Don’t know how
    - This course will help
- Takes too much time
    - Really?
    - IT SAVES TIME in the long run

Unit Testing Frameworks

Generally, all very similar

- JUnit (Java), NUnit (.net?), CppUnit, phpUnit,
- Basically
    - Macros (C++), methods (Java) to test conditions
    - Macros (C++), reflection (Java) to run/discover tests
    - Ways of looking at results.
        - Java/Eclipse: Integrated with IDE
        - Log file or standard output

Unit Testing Example

How To Start

We discuss

- Basic Example
- Some tips

Then its down to the developer/artist.

C++ Frameworks

To Consider:

- Catch
- GoogleTest
- QTestLib
- BoostTest
- CppTest
- CppUnit

Worked Example

- Borrowed from
  - Catch Tutorial
  - and Googletest Primer
- We use Catch, so notes are compilable
- But the concepts are the same

Code

To keep it simple for now we do this in one file:

```
#define CATCH_CONFIG_MAIN  // This tells Catch to provide a main() - only do this in one cpp
#include "../catch/catch.hpp"

unsigned int Factorial( unsigned int number ) {
    return number <= 1 ? number : Factorial(number-1)*number;
}
```

```
TEST_CASE( "Factorials are computed", "[factorial]" ) {
    REQUIRE( Factorial(1) == 1 );
    REQUIRE( Factorial(2) == 2 );
    REQUIRE( Factorial(3) == 6 );
    REQUIRE( Factorial(10) == 3628800 );
}
```

Produces this output when run:

```
===============================================================================
All tests passed (4 assertions in 1 test case)
```

Principles

So, typically we have

- Some `#include` to get test framework
- Our code that we want to test
- Then make some assertions

Catch / GoogleTest

For example, in Catch:

```
    // TEST_CASE(<unique test name>, <test case name>)
    TEST_CASE( "Factorials are computed", "[factorial]" ) {
        REQUIRE( Factorial(2) == 2 );
        REQUIRE( Factorial(3) == 6 );
    }
```

In GoogleTest:

```
    // TEST(<test case name>, <unique test name>)
    TEST(FactorialTest, HandlesPositiveInput) {
      EXPECT_EQ(2, Factorial(2));
      EXPECT_EQ(6, Factorial(3));
    }
```

all done via C++ macros.

Tests That Fail

What about Factorial of zero? Adding

```
    REQUIRE( Factorial(0) == 1 );
```

Produces something like:

```
    factorial2.cc:9: FAILED:
    REQUIRE( Factorial(0) == 1 )
    with expansion:
    0 == 1
```

Fix the Failing Test

Leading to:

```
#define CATCH_CONFIG_MAIN  // This tells Catch to provide a main() - only do this in one cpp
#include "../catch/catch.hpp"

unsigned int Factorial( unsigned int number ) {
    //return number <= 1 ? number : Factorial(number-1)*number;
    return number > 1 ? Factorial(number-1)*number : 1;
}

TEST_CASE( "Factorials are computed", "[factorial]" ) {
    REQUIRE( Factorial(0) == 1 );
    REQUIRE( Factorial(1) == 1 );
    REQUIRE( Factorial(2) == 2 );
    REQUIRE( Factorial(3) == 6 );
    REQUIRE( Factorial(10) == 3628800 );
}
```

which passes:

```
===============================================================================
All tests passed (5 assertions in 1 test case)
```

Test Macros

Each framework has a variety of macros to test for failure. Catch has:

```
REQUIRE(expression); // stop if fail
CHECK(expression);   // doesn't stop if fails
```

If an exception is thrown, it's caught, reported and counts as a failure.

Examples:

```
CHECK( str == "string value" );
CHECK( thisReturnsTrue() );
REQUIRE( i == 42 );
```

Others:

```
REQUIRE_FALSE( expression )
CHECK_FALSE( expression )
REQUIRE_THROWS( expression ) # Must throw an exception
CHECK_THROWS( expression ) # Must throw an exception, and continue testing
REQUIRE_THROWS_AS( expression, exception type )
CHECK_THROWS_AS( expression, exception type )
REQUIRE_NOTHROW( expression )
CHECK_NOTHROW( expression )
```

Testing for Failure

To re-iterate:

- You should test failure cases
  - Force a failure
  - Check that exception is thrown
  - If exception is thrown, test passes
  - (Some people get confused, expecting test to fail)
- Examples
  - Saving to invalid file name
  - Negative numbers passed into double arguments
  - Invalid Physical quantities (e.g. -300 Kelvin)

Setup/Tear Down

- Some tests require objects to exist in memory
- These should be set up
  - for each test
  - for a group of tests
- Frameworks do differ in this regards

Setup/Tear Down in Catch

Referring to the Catch Tutorial:

```
TEST_CASE( "vectors can be sized and resized", "[vector]" ) {

    std::vector<int> v( 5 );

    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }
    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );
    }
    SECTION( "reserving bigger changes capacity but not size" ) {
        v.reserve( 10 );

        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 10 );
    }
    SECTION( "reserving smaller does not change size or capacity" ) {
        v.reserve( 0 );

        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );
    }
}
```

So, Setup/Tear down is done before/after each section.

Unit Testing Tips

C++ design

- Stuff from above applies to Classes / Functions

- Think about arguments:
    - Code should be hard to use incorrectly.
    - Use `const`, `unsigned` etc.
    - Testing forces you to sort these out.

Test Driven Development (TDD)

- Methodology
    1. Write a test
    2. Run test, should fail
    3. Implement/Debug functionality
    4. Run test
        1. if succeed goto 5
        2. else goto 3
    5. Refactor to tidy up

TDD in practice

- Aim to get good coverage
- Some people quote 70% or more
- What are the downsides?
- Don't write 'brittle' tests

Behaviour Driven Development (BDD)

- Behaviour Driven Development (BDD)
    - Refers to a whole area of software engineering
    - With associated tools and practices
    - Think about end-user perspective
    - Think about the desired behaviour not the implementation
    - See Jani Hartikainen article.

TDD Vs BDD

- TDD
    - Test/Design based on methods available
    - Often ties in implementation details
- BDD

    - Test/Design based on behaviour
    - Code to interfaces (later in course)
- Subtly different
- Aim for BDD

Anti-Pattern 1: Setters/Getters

Testing every Setter/Getter.

Consider:

```
class Atom {

  public:
    void SetAtomicNumber(const int& number) { m_AtomicNumber = number; }
    int GetAtomicNumber() const { return m_AtomicNumber; }
    void SetName(const std::string& name) { m_Name = name; }
    std::string GetName() const { return m_Name; }
  private:
    int m_AtomicNumber;
    std::string m_Name;
};
```

and tests like:

```
TEST_CASE( "Testing Setters/Getters", "[Atom]" ) {

    Atom a;

    a.SetAtomicNumber(1);
    REQUIRE( a.GetAtomicNumber() == 1);
    a.SetName("Hydrogen");
    REQUIRE( a.GetName() == "Hydrogen");
```

- It feels tedious
- But you want good coverage
- This often puts people off testing
- It also produces "brittle", where 1 change breaks many things

Anti-Pattern 1: Suggestion.

- Focus on behaviour.

  - What would end-user expect to be doing?
  - How would end-user be using this class?
  - Write tests that follow the use-case
  - Gives a more logical grouping
  - One test can cover > 1 function
  - i.e. move away from slavishly testing each function

- Minimise interface.

  - Provide the bare number of methods
  - Don't provide setters if you don't want them
  - Don't provide getters unless the user needs something
  - Less to test. Use documentation to describe why.

Anti-Pattern 2: Constructing Dependent Classes

- Sometimes, by necessity we test groups of classes
- Or one class genuinely Has-A contained class
- But the contained class is expensive, or could be changed in future

Anti-Pattern 2: Suggestion

- Read up on Dependency Injection
- Enables you to create and inject dummy test classes
- So, testing again used to break down design, and increase flexibility

Summary BDD Vs TDD

Aim to write:

- Most concise description of requirements as unit tests
- Smallest amount of code to pass tests
- … i.e. based on behaviour

Any Questions?

Homework - Overview

- Example git repo, CMake, Catch template project:

  - CMakeCatch2 - Simple

  - CMakeCatchTemplate - Complex

- You should

  - Clone, Build.
  - Add unit test in Testing
  - Run via ctest
  - Find log file

Homework - 7

- Imagine a simple function, e.g. to add two numbers.

- Play with unit tests until you understand the difference between:

  ```
  int AddTwoNumbers(int a, int b);
  int AddTwoNumbers(const int& a, const int&b);
  void AddTwoNumbers(int* a, int*b, int* output);
  void AddTwoNumbers(const int* const a, const int* const b);
  ```

  Now imagine, instead of integers, the variables all contained a large Image. Which type of function declaration would you use?

Homework - 8

- Write a Fraction class
- Write a print function to print nicely formatted fractions
- Does the print function live inside or outside of the class?
- Write a method `simplify()` which will simplify the fraction.
- Unit test until you have at least got the hang of unit testing
- Review your function arguments and return types