Report from Manchester C++ Training Event

SWIFT-HEP #6 - Bristol

Tobias Fitschen on behalf of the mentors, lecturers, organisers:
M. Amerl, C. Doglioni, P. Jawahar, D. Lange, S. Ponce, N. Skidmore

21 November 2023

University of Manchester

About Me

Tobias Fitschen

- PostDoc at Manchester in Caterina Doglioni's group
- Was one of the demonstrators in the course

Interests

- Sustainable computing
    - Co-investigator of grant on sustainable ML
- DM search with alternative analysis strategies
    - Trigger level analysis (TLA)
    - Analysis contact for Full Run 2 dijet TLA
    - Involved in Run3 TLA
- Jets and ML
    - Jet/EtMiss - ML-Forum Liaison
    - Generally active in Jet/EtMiss

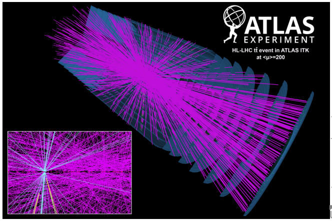
HEP Software Foundation (HSF): A Reminder

MANCHESTER
1824
The University of Manchester

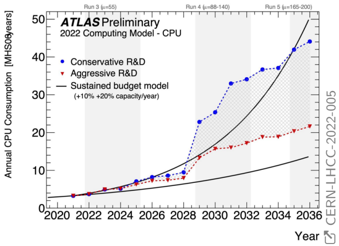
HSF Goals

HSF
HEP Software Foundation

- Established 2015 to facilitate common efforts in software and computing in HEP
- HEP software needs to be maintained over long timescales
  - E.g.: ROOT was initiated in 1994
- In environments they were not originally designed for
  - E.g.: ATLAS main software athena rewritten for multi-threading (athenaMT) for release 22



Future detector environments require performant algorithms



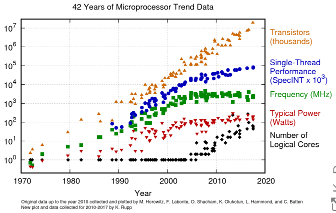
Constant improvement of software to stay within computing budget

Motivation: Teaching Software at the PhD Level

MANCHESTER 1824 The University of Manchester

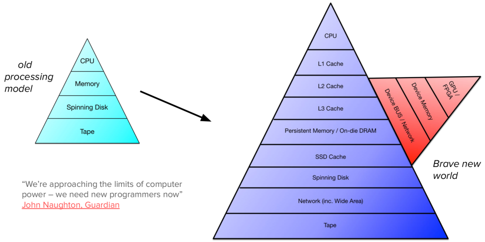
Software is the most prevalent of all instruments in modern science

- Nearly every HEP scientist is actively developing code
- Most have learned it mostly through self-study
- Code has to be performant:
    - Grid utilises 1M CPU cores processing data on the exabite scale
- And well written:
    - Each year many new students interact with vast code base



K. Rupp

Moore's law upheld only by more cores + other modern hardware optimisations



"We're approaching the limits of computer power – we need new programmers now" John Naughton, Guardian

Graeme A Stewart, Lepton-Photon 2021

Making use of complex modern hardware requires solid understanding

The Course

- 8th HEP C++ Course and Hands-on Training - The Essentials
  - In-person Aug 29 - Sep 1 in Manchester (indico)
- Advanced course (indico) later: Oct 16 - 20 at CERN
- Part of series of HSF-training courses
  - Sign up here for upcoming HSF-trainings
  - Other HSF modules (git, ROOT,...) are available here
- Funded by SWIFT-HEP, material by SIDIS, and HSF

Workshop Organisation

MANCHESTER 1824 The University of Manchester

Costs: Very low!

- Teachers/demonstrators expenses covered by SWIFT-HEP
- Participants expenses covered by themselves/their groups
- No registration fee
- 1,200 GBP for two teacher's hotels
- 850 GBP for two teacher's flights (relatively high, due to dates)
- 150 GBP for social dinner
- 450 GBP for refreshments (coffee breaks)

Effort: Very manageable!

- Course content and material from well-tested HSF course
- Plus invited lecture from E. Chadwick (Software Sustainability Institute and Software Carpentry)
- Local organiser: Closing remarks and help during hands-on sessions but also first lecture due to air traffic control strike (easy to teach!)
  - University-lab style: After lectures, participants work on their own ore in small groups, ask demonstrators for help

Target Audience

Who should attend?

- Starts with the absolute basics of C++
- But not intended for complete beginners to programming
- Possible (but challenging) for participants who are not at all familiar with C++ but with other languages
- Still interesting for experienced C++ users
    - For me: Especially the tools (compiler chain, debugging) and modern C++ (smart pointers) sessions
- And in general as a very well-structured refresher

Lecture Structure and Content

Structure

- 2h interactive lecture session in the morning by Sebastien Ponce
    - HSF course slides (Sebastien: main author)
    - Monday to Thursday
- Afternoons focused on hands-on excercises based on course
- Wednesday: 1h invited talk: software sustainability institute
- Friday: Hands-on excercises and close-out at lunch-time
- Sessions are recorded (albeit with technical outages)
    - Earlier course's recordings available by HSF

Content

- **Basics**: Syntax, pointers, references, compound types, operators
- **Tools**: Code management, compiler chain, debugging
- **Object Orientation**: Classes, inheritence
- **Modern C++**: Constness, exceptions, templates, STL, lambdas

Setup

MANCHESTER 1824
The University of Manchester

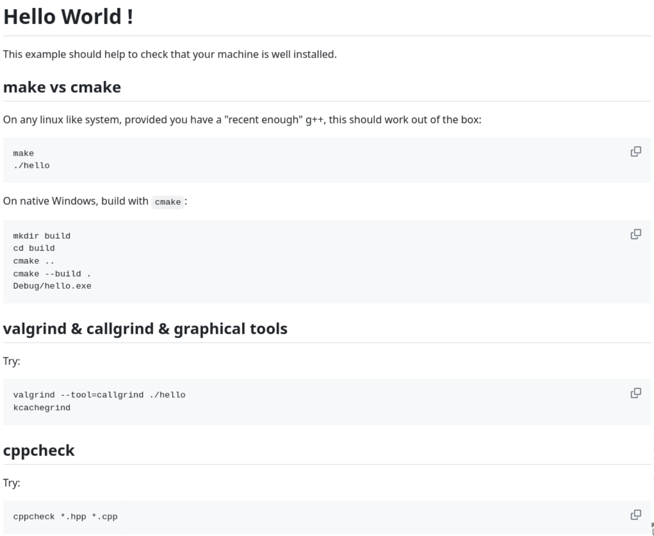
Lesson learned from previous instances of course:

- Very important to provide solid setup instructions
- Be prepared for MAC, Linux, Windows users
- Spend first hands-on session to get all participants fully setup
- Send out instructions well in advance before course
- Work together with participants to improve them
- Instructions

xkcd.com/979

Setup

1st exercise checks for correct C++ version and tools' installation



Hello World !

This example should help to check that your machine is well installed.

make vs cmake

On any linux like system, provided you have a "recent enough" g++, this should work out of the box:

```
make
./hello
```

On native Windows, build with `cmake`:

```
mkdir build
cd build
cmake ..
cmake --build .
Debug/hello.exe
```

valgrind & callgrind & graphical tools

Try:

```
valgrind --tool=callgrind ./hello
kcachegrind
```

cppcheck

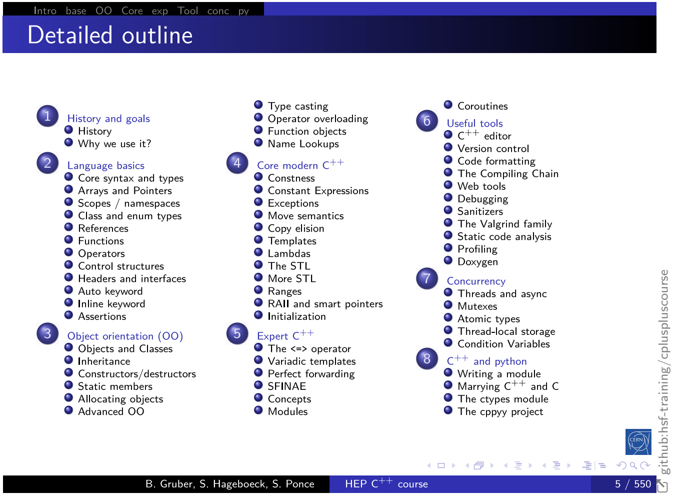
Try:

```
cppcheck *.hpp *.cpp
```

course's github

Material

- HSF C++ course slides: 550 pages of well-structured content



Intro base OO Core exp Tool conc py

Detailed outline

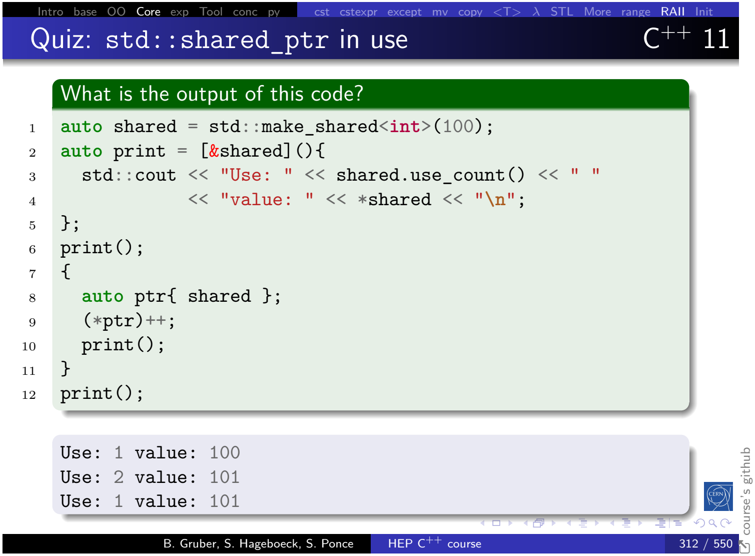
github:hsf-training/cpluspluscourse

B. Gruber, S. Hageboeck, S. Ponce | HEP C^{++} course | 5 / 550

- Great reference also for after the course

Example: Lecture

Lecture slides continously improved, tracked on github



Quiz: std::shared_ptr in use C++ 11

What is the output of this code?

```cpp
auto shared = std::make_shared<int>(100);
auto print = [&shared](){
  std::cout << "Use: " << shared.use_count() << " "
            << "value: " << *shared << "\n";
};
print();
{
  auto ptr{ shared };
  (*ptr)++;
  print();
}
print();
```

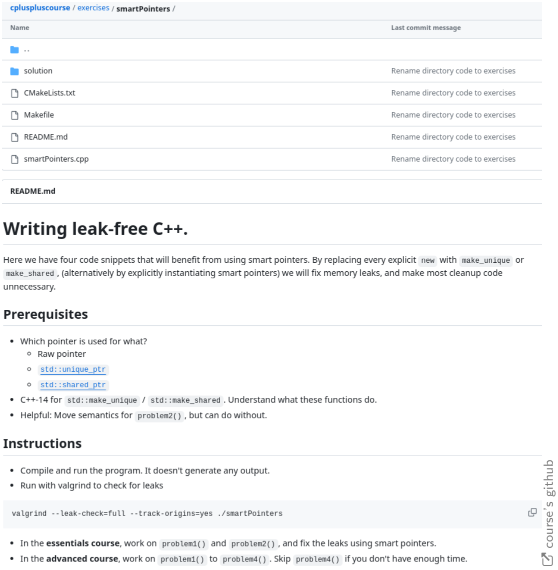
```
Use: 1 value: 100
Use: 2 value: 101
Use: 1 value: 101
```

course's github

Example: Exercise

MANCHESTER
1824
The University of Manchester

All exercises with solutions available on github



cpluspluscourse / exercises / smartPointers /

Name	Last commit message
..	
solution	Rename directory code to exercises
CMakeLists.txt	Rename directory code to exercises
Makefile	Rename directory code to exercises
README.md	Rename directory code to exercises
smartPointers.cpp	Rename directory code to exercises

README.md

Writing leak-free C++.

Here we have four code snippets that will benefit from using smart pointers. By replacing every explicit `new` with `make_unique` or `make_shared`, (alternatively by explicitly instantiating smart pointers) we will fix memory leaks, and make most cleanup code unnecessary.

Prerequisites

- Which pointer is used for what?
  - Raw pointer
  - `std::unique_ptr`
  - `std::shared_ptr`
- C++-14 for `std::make_unique` / `std::make_shared`. Understand what these functions do.
- Helpful: Move semantics for `problem2()`, but can do without.

Instructions

- Compile and run the program. It doesn't generate any output.
- Run with valgrind to check for leaks

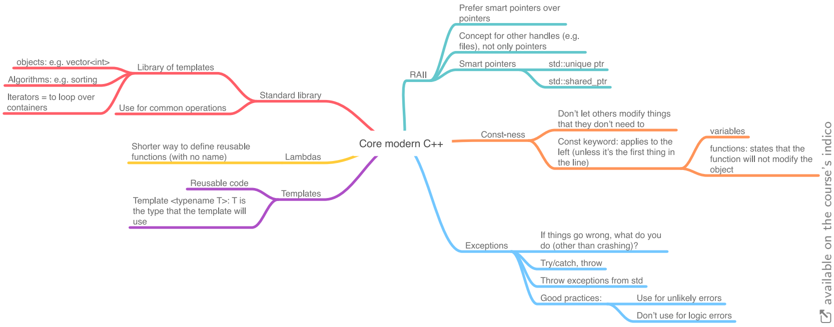
```
valgrind --leak-check=full --track-origins=yes ./smartPointers
```

- In the **essentials course**, work on `problem1()` and `problem2()`, and fix the leaks using smart pointers.
- In the **advanced course**, work on `problem1()` to `problem4()`. Skip `problem4()` if you don't have enough time.

course's github

Closeout: Summary of Concepts

MANCHESTER 1824
The University of Manchester

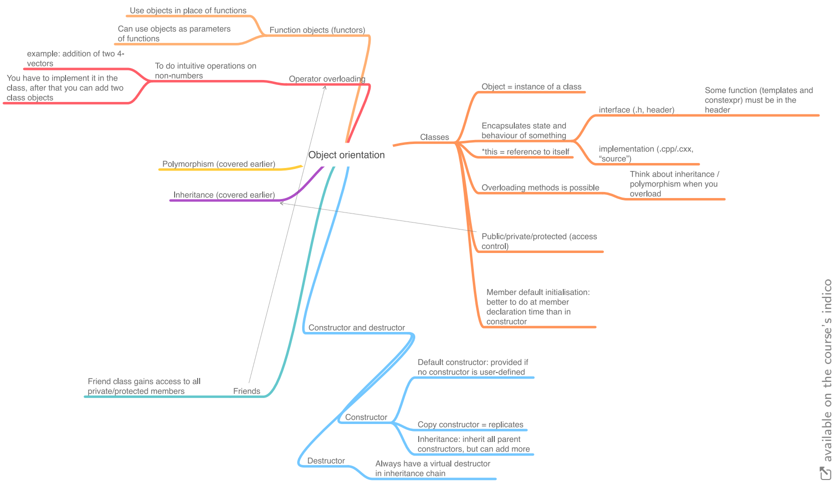


Core modern C++
Standard library
Library of templates
objects: e.g. vector<int>
Algorithms: e.g. sorting
Iterators = to loop over containers
Use for common operations
Lambdas
Shorter way to define reusable functions (with no name)
Templates
Reusable code
Template <typename T>: T is the type that the template will use
RAII
Prefer smart pointers over pointers
Concept for other handles (e.g. files), not only pointers
Smart pointers
std::unique ptr
std::shared_ptr
Const-ness
Don't let others modify things that they don't need to
Const keyword: applies to the left (unless it's the first thing in the line)
variables
functions: states that the function will not modify the object
Exceptions
If things go wrong, what do you do (other than crashing)?
Try/catch, throw
Throw exceptions from std
Good practices:
Use for unlikely errors
Don't use for logic errors

↗ available on the course's indico

- Made mind-map with participants of concepts encountered
- During close-out session of Friday
- To structure and better maintain memories of event

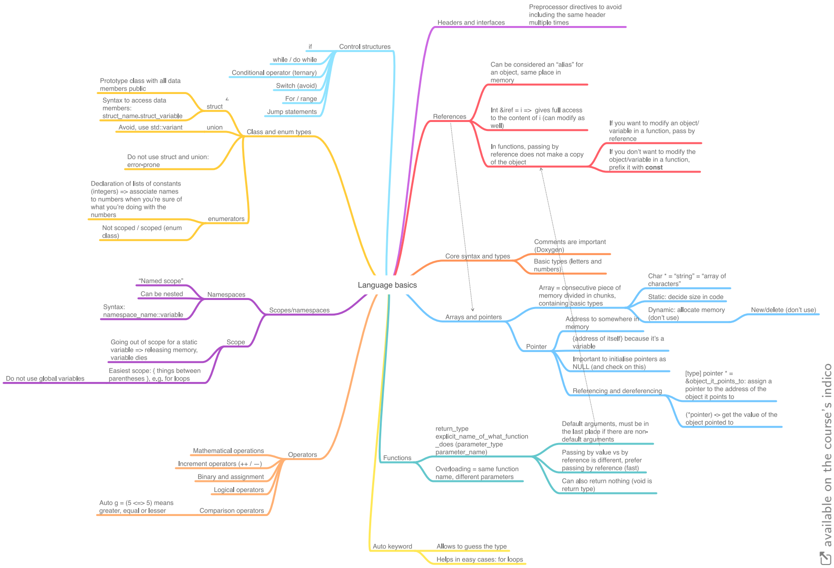
Closeout: Summary of Concepts



Object orientation
Function objects (functors)
Use objects in place of functions
Can use objects as parameters of functions
Operator overloading
To do intuitive operations on non-numbers
example: addition of two 4-vectors
You have to implement it in the class, after that you can add two class objects
Polymorphism (covered earlier)
Inheritance (covered earlier)
Friends
Friend class gains access to all private/protected members
Classes
Object = instance of a class
Encapsulates state and behaviour of something
interface (.h, header)
Some function (templates and constexpr) must be in the header
implementation (.cpp/.cxx, "source")
*this = reference to itself
Overloading methods is possible
Think about inheritance / polymorphism when you overload
Public/private/protected (access control)
Member default initialisation: better to do at member declaration time than in constructor
Constructor and destructor
Constructor
Default constructor: provided if no constructor is user-defined
Copy constructor = replicates
Inheritance: inherit all parent constructors, but can add more
Destructor
Always have a virtual destructor in inheritance chain

available on the course's indico

Closeout: Summary of Concepts


Language basics
Control structures
if
while / do while
Conditional operator (ternary)
Switch (avoid)
For / range
Jump statements
Class and enum types
struct
Prototype class with all data members public
Syntax to access data members: struct_name.struct_variable
union
Avoid, use std::variant
Do not use struct and union: error-prone
enumerators
Declaration of lists of constants (integers) => associate names to numbers when you're sure of what you're doing with the numbers
Not scoped / scoped (enum class)
Scopes/namespaces
Namespaces
"Named scope"
Can be nested
Syntax: namespace_name::variable
Scope
Going out of scope for a static variable => releasing memory, variable dies
Easiest scope: { things between parentheses }, e.g. for loops
Do not use global variables
Operators
Mathematical operations
Increment operators (++ / —)
Binary and assignment
Logical operators
Comparison operators
Auto g = (5 <=> 5) means greater, equal or lesser
Auto keyword
Allows to guess the type
Helps in easy cases: for loops
Headers and interfaces
Preprocessor directives to avoid including the same header multiple times
References
Can be considered an "alias" for an object, same place in memory
Int &iref = i => gives full access to the content of i (can modify as well)
In functions, passing by reference does not make a copy of the object
If you want to modify an object/ variable in a function, pass by reference
If you don't want to modify the object/variable in a function, prefix it with const
Core syntax and types
Comments are important (Doxygen)
Basic types (letters and numbers)
Arrays and pointers
Array = consecutive piece of memory divided in chunks, containing basic types
Char * = "string" = "array of characters"
Static: decide size in code
Dynamic: allocate memory (don't use)
New/delete (don't use)
Pointer
Address to somewhere in memory
{address of itself} because it's a variable
Important to initialise pointers as NULL (and check on this)
Referencing and dereferencing
[type] pointer * = &object_it_points_to: assign a pointer to the address of the object it points to
(*pointer) -> get the value of the object pointed to
Functions
return_type explicit_name_of_what_function_does (parameter_type parameter_name)
Overloading = same function name, different parameters
Default arguments, must be in the last place if there are non-default arguments
Passing by value vs by reference is different, prefer passing by reference (fast)
Can also return nothing (void is return type)

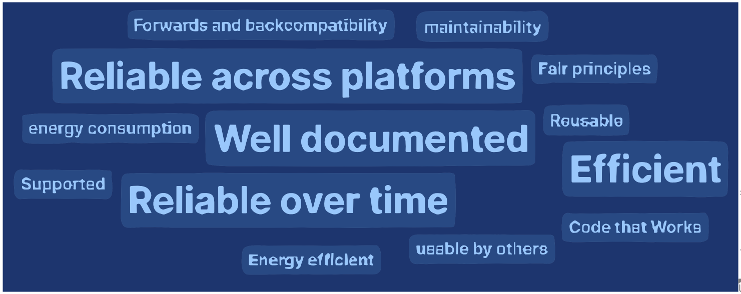
↗ available on the course's indico

Invited Talk: Making Software Sustainable

by **Eli Chadwick** - The Carpentries, Software sustainability Institute

→ recording available

Live Poll: What comes to mind when you hear "sustainable software"?



session recording

Invited Talk: Making Software Sustainable

by **Eli Chadwick** - The Carpentries, Software sustainability Institute

How to improve sustainability of my software?

1 Admit you have a problem

2 Use version control (little and often)

3 Build and run your code on another machine

4 Formalise your tests

5 Modularise and share

6 Give and receive code review

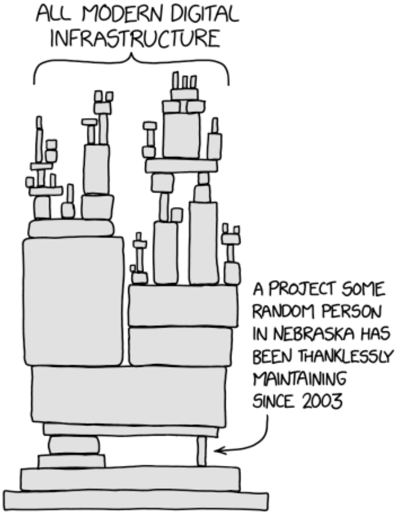
7 Automate as much as you can

8 Join community of practive

Personal Experience with ATLAS coding

MANCHESTER 1824 The University of Manchester

- Often core software is maintained by single contributors
- Many users rely on the software to be maintained
- Recently commited to be xAODAnaHelpers maintainer
- Core software many exotics analyses rely on
- Would have been discontinued at end of year
- Would lead to big delay in many analyses


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

xkcd.com/2347

EVERSE

EVERSE

- **E**uropean **V**irtual **I**nstitute for **R**esearch **S**oftware **E**xcellence
- Website with more information: (EVERSE)
- Start: late 2023, duration: 3 years
- Funded as part of Horizon Europe Initiative
  - EU's key (100 € billion) funding programme until 2027
  - Strong focus on environmental sustainability
- Manchester HEP: WP4 leader for Science Clusters pilot cases (includes particle physics)

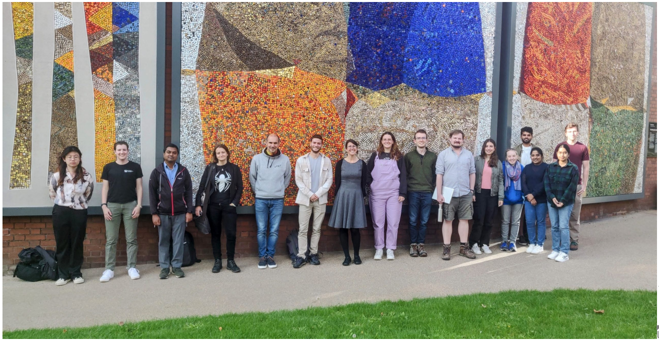
Goals

- Build community-led structure for evaluating and improving code
- Establish sustainable and reliable ecosystem of stakeholders
- Create framework to ensure appropriate recognition for software careers

Summary

Hosting the HSF C++ course is cheap
and does not require much organisation

Well-tested, excellent lectures + exercises provided entirely by HSF



Workshop photo

Contact me if you're interested in collaborating further on software training in the UK (or if you want to know more on EVERSE)