Programming in Modern C++

Module M60: Closing Comments

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Module Recap

- Understood synchronization issues in multi-thread programming in C++
- Studied various synchronization mechanisms through example
- Provided detail for self-study of synchronization mechanisms:
  - *Mutex*
  - *Lock*
  - *Atomics*
  - *Condition Variable*
  - *Future and Promises*
  - *Async*
- Explored use of the synchronization mechanisms to alleviate race condition and data race and left practice examples

Module Objectives

- Review C++ Course
- Key take-backs
- What next?

Module Outline

Course Summary

What we learnt in 59 Modules

- **Programming in C++ is Fun**: C programs in C++, Equivalent programs [**Module 01-05**]
- **C++ as Better C**: Procedural Extensions of C [**Module 06-10**]
- **OOP in C++**: Object-Oriented Programming with Classes [**Module 11-20**]
- **Inheritance**: Generalization / Specialization of Object Modeling in C++ [**Module 21-25**]
- **Polymorphism**: Static & Dynamic Binding and Multiple Inheritance [**Module 26-30**]
- **Type Casting**: Cast Operators [**Module 31-35**]
- **Exceptions**: Error Handling in C & C++ [**Module 36-37**]
- **Templates**: Generic Programming in C++ [**Module 38-40**]
- **Streams**: File Handling in C & Stream in C++ for IO [**Module 41-42**]
- **STL**: Generic Programming Library in C++ Containers [**Module 43-45**]
- **Modern C++**[1]: *General Features*: `auto`, `decltype`, initializer list, `constexpr`, rvalue & move semantics, λ & closure Objects, etc. [**Module 46-53**]
- **Modern C++**: *Class Features*: `=default`, `=delete`, delegating & inherited constructors, in-class initialization, etc. [**Module 54-54**]
- **Modern C++**: *Non-class Types* & *Template Features* [**Module 55-55**]
- **Modern C++**: *Resource Mgmt.*: `unique_ptr`, `shared_ptr`, `weak_ptr` [**Module 56-57**]
- **Modern C++**: *Concurrency Support*: Threads and synchronization [**Module 58-59**]

[1] Mostly **C++11**, references to **C++14** at time, occasionally to **C++17** & **C++20**

What we learnt in 12 Tutorials

- **How to build a C/C++ program?**: C Preprocessor (CPP), Build Pipeline, `make` Utility, Static and Dynamic Library [**Tutorial 01-04**]
- **Mixing C and C++ Code**: Issues and Resolutions, Project Example [**Tutorial 05-06**]
- **How to design a UDT like built-in types?**: Fraction UDT, Int & Poly UDT, Updates and Mixes of UDTs [**Tutorial 07-09**]
- **How to optimize C++11 programs using Rvalue and Move Semantics?** [**Tutorial 10-10**]
- **Compatibility of C and C++**: Significant Features, Summary [**Tutorial 11-12**]

Note that in some module and tutorial presentations some slides have been added after recording. These deal with more clarifications, examples, and minor added features. These are marked with "Post-Recording" in the slide title. So please refer to the presentation file along with the video

Upcoming Tutorials

- **Embedded C++**
- **CMAKE**
- **Resource Management**: Managoing memory and other resources by smart pointers, RAII & RDDI
- **C++ Coding Styles: How to write good code?**
- **C++ Processes and Tools**: Design, Development, Build (**Tutorial 01-04**), Debugging, Test, and Source Code & Version Management, and Bug Tracking
- **Design Patterns**: Singleton, Iterator, Command, Factory, Abstract Factory, Visitor, etc.
- **We may record more tutorials based on request from students. Feel free to suggest**

Sources:

- C++11, isocpp.org
- C++14, isocpp.org
- C++17, isocpp.org
- C++20, isocpp.org
- Modern cpp Features: C++20/17/14/11, github
- Modern C++ Tutorial: C++11/14/17/20 On the Fly, Changkun Ou, O'Reilly, 2022

Modern C++ Features

C++ Standards

C++98	C++11	C++14	C++17	C++20
1998	2011	2014	2017	2020
Templates	Move Semantics	Reader-Writer Locks	Fold Expressions	Coroutines
STL with Containers and Algorithms	Unified Initialization	Generic Lambda Functions	constexpr if	Modules
Strings	auto and decltype		Structured Binding	Concepts
I/O Streams	Lambda Functions		`std::string_view`	Ranges Library
	constexpr		Parallel Algortihms of the STL	
	Multi-threading and Memory Model		File System Library	
	Regular Expressions		`std::any`, `std::optional`, and `std::variant`	
	Smart Pointers			
	Hash Tables			
	`std::array`			
ISO/IEC 14882:1998	ISO/IEC 14882:2011	ISO/IEC 14882:2014	ISO/IEC 14882:2017	ISO/IEC 14882:2020

Fixes on C++98: C++03: ISO/IEC 14882:2003, 2003
Latest Version as of Sep-21: C++20: ISO/IEC 14882:2020, 2020

Modern C++ Features: C++11

Major C++11 Features: Core Language Features/1

- `auto` and `decltype` [Module 46]
- trailing (suffix) return type [Module 46]
- list initialization (`initializer_list`), `std::initializer_list` [Module 47]
- uniform initialization: brace-or-equal initializers [Module 47]
- `enum class`: scoped enums [Module 55]
- `constexpr` and literal types [Module 48]
- `noexcept` specifier and operator [Module 48]
- `nullptr` [Module 48]
- `defaulted` and `deleted` functions [Module 54]
- delegating and inherited constructors [Module 54]
- `explicit` conversion [Module 54]
- range-`for` (based on Boost) [Module 47]
- `static_assert` (based on Boost) [Module 48]
- unicode string literals [Module 48]
- raw string literals [Module 48]
- user-defined literals [Module 48]
- `inline namespace` [Module 48]

Major C++11 Features: Core Language Features/2

- **rvalue reference** and **move semantics** [Module 49-51]
  - *move constructor*, *assignment operator*, `std::move`
  - *Perfect forwarding*, `std::forward`
- **lambda expressions** [Module 52-53]
- **concurrency support** [Module 58-59]
  - *threads*, `std::thread`
  - *synchronization*, `std::mutex`, `std::lock`, `std::atomic`, `std::condition_variable`, `std::future`, `std::promise`, `std::async`
  - *thread-local storage*, `thread_local`
- GC interface (removed in C++23)
- `long long`, `char16_t` and `char32_t` [Module 55]
- `final` and override [Module 54]
- type aliases [Module 55]
- variadic templates [Module 55]
- generalized (non-trivial) unions [Module 55]
- generalized PODs (trivial types and standard-layout types) [Module 55]
- attributes [Module 48]
- `alignof` and `alignas` [Module 48]

Major C++11 Features: Library Headers

- `<array>`
- `<atomic>` [Module 58, 59]
- `<cfenv>`
- `<chrono>` [Module 58, 59]
- `<cinttypes>`
- `<condition_variable>` [Module 59]
- `<cstdint>` [Module 55]
- `<cuchar>`
- `<forward_list>`
- `<functional>` [Module 52, 53, 58]
- `<future>` [Module 58, 59]
- `<initializer_list>` [Module 47]
- `<mutex>` [Module 58, 59]
- `<random>` [Module 58, 59]
- `<ratio>`
- `<regex>`
- `<scoped_allocator>`
- `<system_error>`
- `<thread>` [Module 58, 59]
- `<tuple>`
- `<typeindex>`
- `<type_traits>`
- `<unordered_map>`
- `<unordered_set>`

Major C++11 Features: Library Features

- atomic operations library
- `emplace()` and other use of rvalue references throughout all parts of the existing library
- Smart Pointers [Module 56]
  - `std::unique_ptr`, `std::shared_ptr`, `std::weak_ptr`, `std::make_shared` [Module 57]
- `std::move_iterator`
- `std::initializer_list` [Module 47]
- stateful and scoped allocators
- `std::forward_list`
- chrono & ratio library [Module 58, 59]
- Unicode conversion facets
- thread library [Module 58-59]
  - `std::thread`, `std::mutex`, `std::lock`, `std::atomic`, `std::condition_variable`, `std::future`, `std::promise`, `std::async`, `thread_local`
- `std::function` [Module 52, 53, 58]
- `std::bind` [Module 58]
- `std::exception_ptr`
- `std::error_code` and `std::error_condition`
- iterator improvements: `std::begin`, `std::end`, `std::next`, `std::prev` [Module 03, 05, 43-45]
- Unicode conversion functions

Major C++11 Features: Library Features

- New Algorithms:
  - `std::all_of`, `std::any_of`, `std::none_of`,
  - `std::find_if_not`, `std::copy_if`, `std::copy_n`,
  - `std::move`, `std::move_backward`,
  - `std::random_shuffle`, `std::shuffle`,
  - `std::is_partitioned`, `std::partition_copy`, `std::partition_point`,
  - `std::is_sorted`, `std::is_sorted_until`, `std::is_heap`, `std::is_heap_until`,
  - `std::minmax`, `std::minmax_element`,
  - `std::is_permutation`,
  - `std::iota`,
  - `std::uninitialized_copy_n`

Modern C++ Features: C++14

Major C++14 Features: Language

- Binary literals [**Module 48**]
- Generalized return type deduction [**Module 51**]
- `decltype(auto)` [**Module 46**]
- Generalized lambda captures [**Module 53**]
- Generic lambdas [**Module 53**]
- Variable templates [**Module 55**]
- Extended `constexpr` [**Module 48**]
- The [[`deprecated`]] attribute [**Module 48**]
- Digit separators [**Module 48**]

Major C++14 Features: Library

- Shared locking
- User-defined literals for `std::` types [**Module 48**]
- `make_unique` [**Module 57**]
- Type transformation `_t` aliases

Major Deprecated Features of C++98 / C++03

- String literal constant is no longer allowed to be assigned to a `char*`. If you need to assign and initialize a `char*` with a string literal constant, use `const char*` or `auto`
- C++98 exception description, `unexpected_handler`, `set_unexpected()` and other related features are deprecated and should use `noexcept` [Module 48]
- `auto_ptr` is deprecated and `unique_ptr` should be used [Module 57]
- `register` keyword is deprecated and if used there is no practical meaning
- `operator++` for `bool` type is deprecated
- If a class has a destructor, the properties for which it generates copy constructors and copy assignment operators are deprecated [Module 51]
- C language style type conversion using (`convert_type`) before variables is deprecated, and `static_cast`, `reinterpret_cast`, `const_cast` should be used for type conversion
- Some of the C standard libraries that can be used are deprecated in the C++17, such as `<ccomplex>`, `<cstdalign>`, `<cstdbool>` and `<ctgmath>`
- Parameter binding, `export` are deprecated for `std::bind`, `std::function` [Module 58, 53]
- . . . and many more

Note: *Deprecation is not completely unusable, it implies that features will disappear from future standards and should be avoided. But, the deprecated features are still part of the standard library, and most of the features are actually* **permanently** *reserved for compatibility reasons*

Key Take-back

Key Take-back

What have we learnt?

- C++ is multi-paradigm
  - Procedural: Better C
  - Object-Oriented: Encapsulation, Inheritance, and Polymorphism
  - Generic: Templates
- Reuse is Key
  - Library functions
  - Function Overloading (Static Polymorphism)
  - Inheritance & Dynamic Polymorphism
  - Templates
  - STL Containers and Algorithms
- Designing good data types is a key for good programming in C++
- While programming in C++, we should keep an eye on:
  - Efficiency
  - Safety
  - Clarity
- Move with the standards: **C++03** → **C++11** → **C++14** → **C++17** → **C++20** → **C++23**

Do not write C-style programs using C++ compiler

Important References

- C++ reference, cppreference
- CPlusPlus, cplusplus
- An Overview of the New C++ (C++11/14), Training Courses; Effective Modern C++, 2015; Effective C++, 3^{rd} Ed., 2005 and More Effective C++, 1^{st} Ed., 1996; Effective STL, 1^{st} Ed., 2001, Scott Meyers
- Bjarne Stroustrup's FAQ; C++11 - the new ISO C++ standard, stroustrup
- Andrei Alexandrescu: Creator of D; Modern C++ Design, Andrei Alexandrescu, 2001; C++ Coding Standards, 1^{st} Ed., Herb Sutter and Andrei Alexandrescu, 2004; The D Programming Language, Andrei Alexandrescu, 2010
- Herb Sutter: Sutter's Mill: Chair of ISO C++ standards committee for over a decade; Exceptional C++, 1999; More Exceptional C++, 2001 by Herb Sutter
- C++ Templates, 2^{nd} Ed., D. Vandevoorde, N. M. Josuttis, and D. Gregor, 2017
- The C++ Standard Library: A Tutorial and Reference, 2^{nd} Ed., Nicolai M. Josuttis, 2012
- Google C++ Style Guide
- Modern C++ Tutorial: C++11/14/17/20 On the Fly, Changkun Ou, O'Reilly, 2022

Key Take-back: Prepare for Examination

Prepare for Examination

- Watch the Videos
- Revise the Assignments and Solutions
- Practice lots and lots of coding with every feature
- Design and implement complete data types – Complex, Fraction, Vector, Matrix, Polynomial etc. using copy / move semantics
- Use λ's and try simple multi-threaded programming

Road Forward

Road Forward

- Learn the topics not covered
- Breathe programming – regularly code and implement systems
- Read lots and lots of programs by good coders
- Learn Python / Java
- Study **Object Oriented Analysis and Design**
- Study **Unified Modeling Language**
- Study **Software Engineering**
- Study **Books and References** mentioned

Module Summary

- Course on Modern C++ concluded