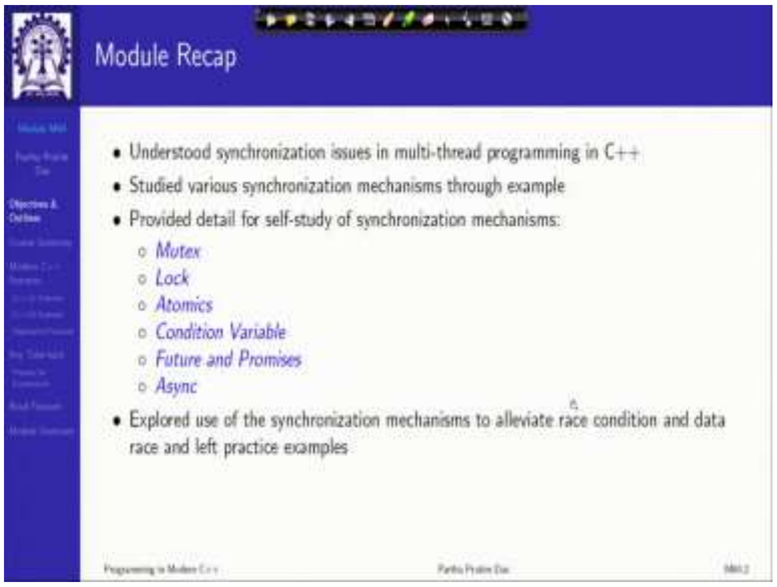
Programming in Modern C++
Professor Parthe Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture 60
Closing Comments

Welcome to Programming in Modern C++ we are in week 12. The last week and we are going to discuss the module 60, the last module of the course.

(Refer Slide Time: 00:39)



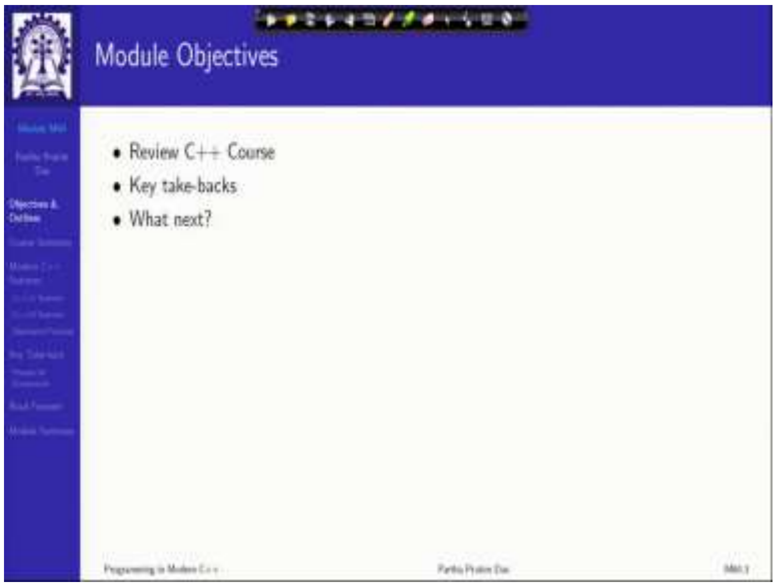
Module Recap

- 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

Programming in Modern C++ Partha Pratim Das M60.2

In the previous module 59, we concluded about the concurrency support, we understand different synchronization issues in multi threaded programs through examples, and left a lot of stuff for your self study. To get a better grasp on the concurrency support in C++.

(Refer Slide Time: 01:01)

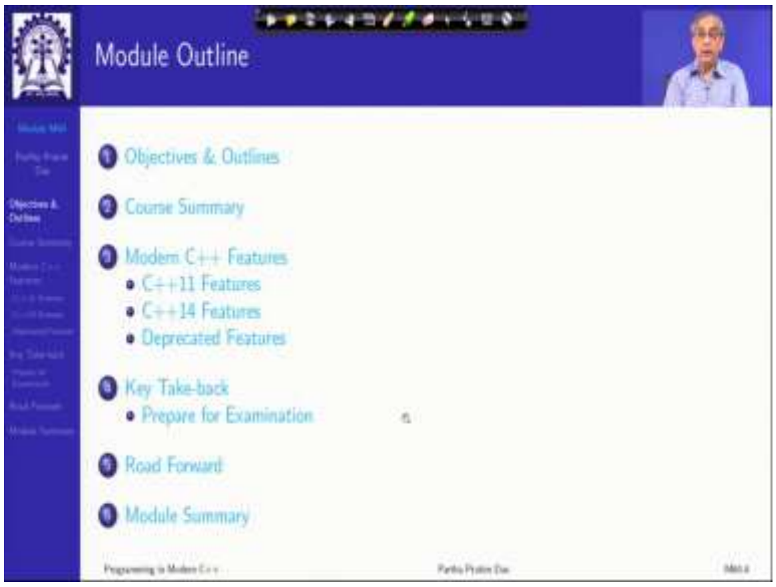


Module Objectives

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

In this module, we conclude. So, we take a quick review of the course, try to highlight some of the key take backs and a little bit of what is next, what can you do next.

(Refer Slide Time: 01:16)

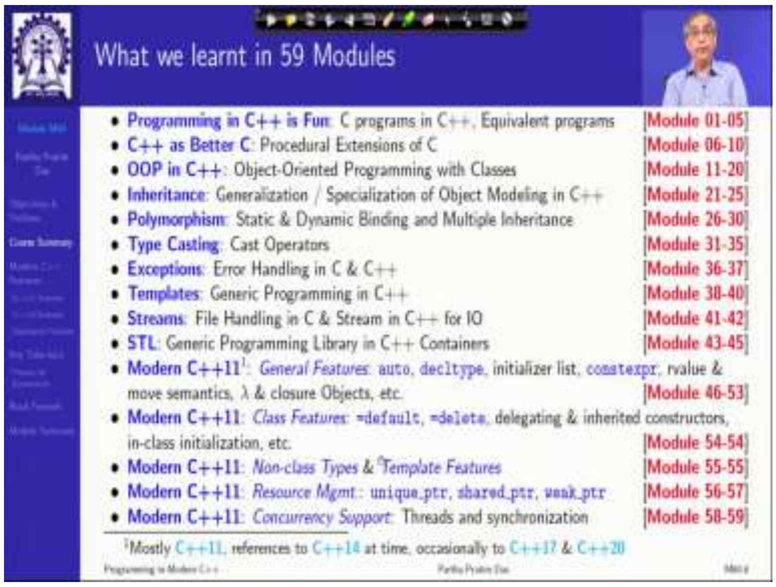


Module Outline

1. Objectives & Outlines
2. Course Summary
3. Modern C++ Features
   - C++11 Features
   - C++14 Features
   - Deprecated Features
4. Key Take-back
   - Prepare for Examination
5. Road Forward
6. Module Summary

Course Summary

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++11**[1]: *General Features*: `auto`, `decltype`, initializer list, `constexpr`, rvalue & move semantics, λ & closure Objects, etc. [Module 46-53]
- **Modern C++11**: *Class Features*: `=default`, `=delete`, delegating & inherited constructors, in-class initialization, etc. [Module 54-54]
- **Modern C++11**: *Non-class Types* & *Template Features* [Module 55-55]
- **Modern C++11**: *Resource Mgmt.*: `unique_ptr`, `shared_ptr`, `weak_ptr` [Module 56-57]
- **Modern C++11**: *Concurrency Support*: Threads and synchronization [Module 58-59]

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

So, this is the outline. So, first, let us just talk about what we have done in the course, what we could afford to do, the Course Summary. We have 59 modules that we have gone through. So, and this is the broad classification of topics, very top level classification of topics to go through in terms of learning first learning C++, then learning Modern C++. So, the first 45 modules spanning 9 weeks was dedicated to learning C++ and C++98 or C++03.

Starting from the fact that programming in C++ is fun. Writing equivalent C++ programs corresponding to C programs, then we looked at the procedural extensions C++ as a better C. Spent 2 weeks on Object oriented design and programming with classes in C++ which is the meet of the initial discussion, then extended for inheritance generalization specialization of object modeling, which leads to different kinds of polymorphism, static binding, dynamic binding, multiple inheritance and so on.

Other features came in, in terms of cast operators to manage the data types in the proper context. And as we saw later on, in Modern C++, the importance of cast operators are somewhat going down because you try to manage them more automatically. Then exceptions are very important for understanding the errors and handling, we have done a comparison between C and C++ styles.

Then templates, which are program generators both at a function level as well as at a class level and key for generic programming streams for management of IO and STL. The generic programming library in C++, particularly C++ containers, is what we have discussed of course, up to this point, we are focused on C++98, C++03 only.

And then the remaining 15 modules or 14 modules excluding the current one, we are focused on the modern C++, which is C++11, 14.we have not gone in beyond that, because that kind of already C++11 itself is too huge and is empowering enough for a programmer to learn the new things on their own.

So, in terms of modern C++11 we have discussed variety of general features, type related features, initializers, const expression and so on. But very importantly, very importantly, the rvalue semantics and the move semantics, this is one of the key idea, which makes lot of other features possible, without that so like in C++03. If you have not understood constructors, you will not be able to proceed you have understood now.

Similarly, in C++11 onwards if you have not understood rvalue semantics and the move semantics, you will not be able to proceed at all. It is not only about, you know optimizing copy, but it goes into, spreads its tentacles. So, to say in so many different features that becomes very critical.

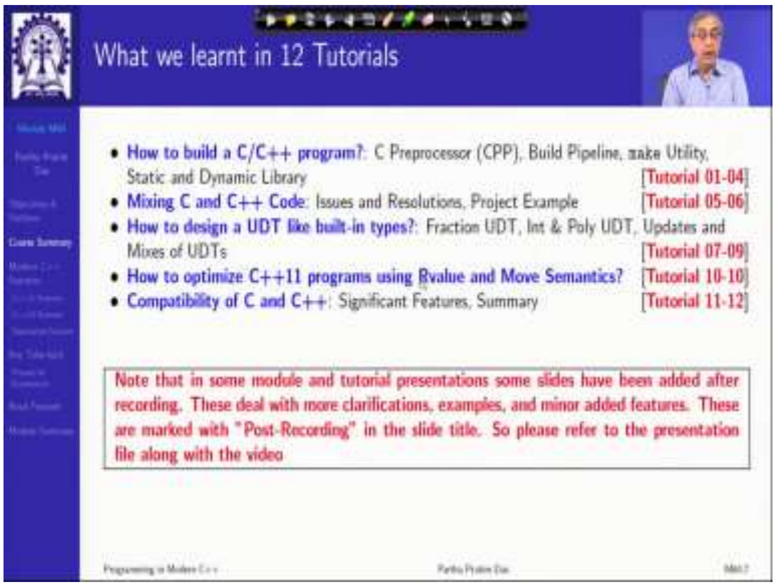
Lambda functions and, or that is functions without name anonymous functions and closure objects are very handy and very useful again and has a wide use, then we have looked at multiple of class features, most of them are convenience features, I would say, they are not, part breaking features like rvalue semantics or closure object and things like that, but, they really make your programming easier in C++11 onwards.

We have looked at a large number of non class type and template features like variadic template, template variable and so on so, forth. Very specific and utilitarian is the resource management. So, this is another area where you must focus very well resource management was there earlier in C++03 also, there was a auto_ptr. But, that had lot of problems and it was not very useful in that way. So, people used to write their own smart pointers, different companies had their own smart pointers, all these have now been standardized in terms of just 3 smart pointers unique pointer for exclusive ownership, shared pointer for shared ownership and weak pointers to solve the circularity problem.

So, this is something again another very, very important aspect that you must master and finally, as we have just concluded on the concurrency support, which takes you to a different dimension in terms of using C++11 and onwards.

So, most of this discussion is around C++11. We have made some references to C++14 which is a key extension but not a very large one. And very occasionally, we have made references to C++17 or C++20.

(Refer Slide Time: 07:26)



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

We also have provided 12 tutorials on 5 broad topics, which I consider that to be a good programmer in C++, Modern C++, it is just not enough to know the language or programming you need to be a master of how to build the programs, what is the preprocessor doing, what is the build pipeline, what is the make utility, how to make libraries, static libraries, dynamic libraries and so on. So, we have discussed these in 4 tutorials.

Mixing of C and C++ code is a major industry issue. So, we have discussed about what are the issues they are in and what are the resolutions with project example. Core off using C++ in general and C++11 onwards in particular is to be able to design user defined data types like built in types, C++03 could do that, but C++11 makes it even more compact even more like the built in types and so, on.

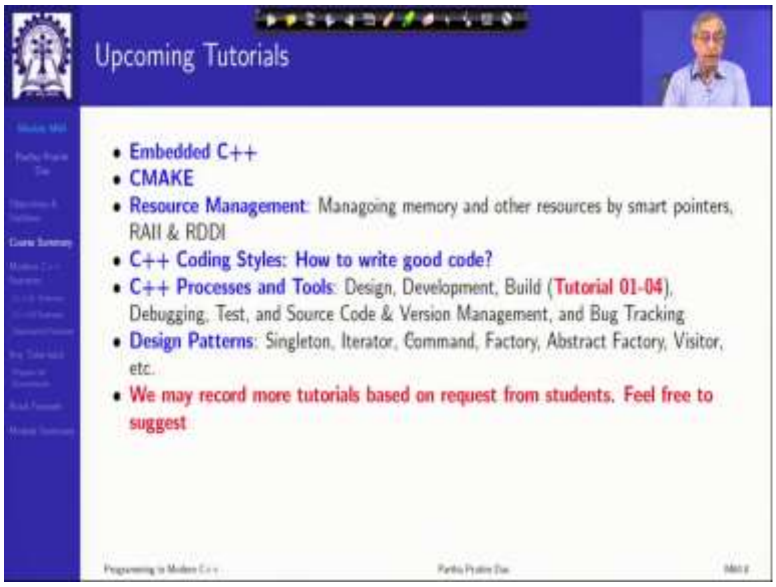
So, this set of 3 tutorials focus on designing, discussing the design of variety of UDTs like fraction UDT, int fixed size int UDT, polynomial UDT mixing of those and so on, which should give you a good idea, they may not be very directly utilitarian, but this will give you a very good idea about how to make types in C++ and use them effectively.

Certainly, 1 tutorial is focused again on talking at depth about using rvalue and move semantics because that is as I said, it is very very important. Compatibility of the 2 languages which is an outcome of the mixing experience, you can say, compatibility of C and C++ has a lot of issues. So, those are discussed in 2 tutorials. So, these are more like engineering aspects of the language.

At this point, I will also note that, I keep on working with the course all the time. So, even after recording certain ideas come to my mind or I read up something, interesting example and so on. So, I even after recording and keep on I am putting those in the presentation slides both in the modules as well as in the tutorial to remind you that this is not a there is no disconnect, those slides have marked as post recording at the top.

So that, you do not get confused, but I will always suggest that besides referring to the video do refer to the presentation files alongside because you will get, in some cases, you will get more clarification more examples, maybe minor added features or use models are discussed in this way.

(Refer Slide Time: 10:38)



Upcoming Tutorials

- Embedded C++
- CMAKE
- Resource Management: Managing 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

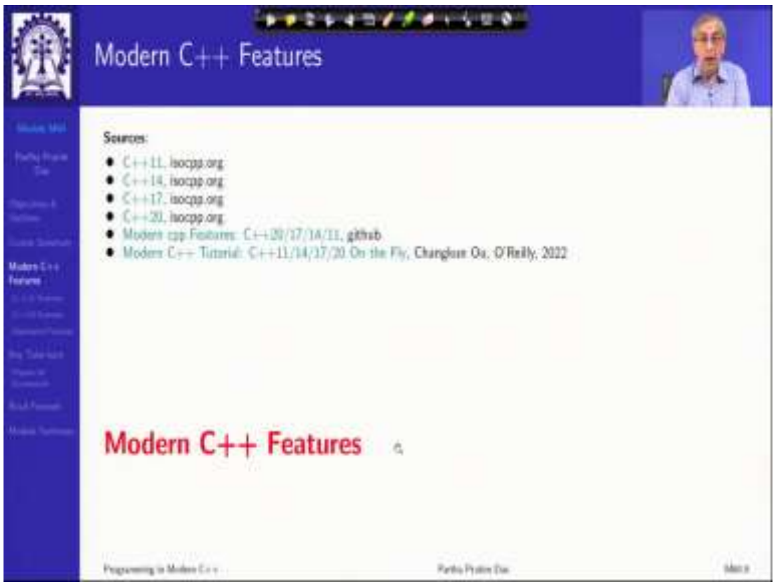
Now, beyond the tutorial, 12 tutorials that I have already recorded, I planed to record some more. The first 2 have been requests have come from the live sessions only embedded C++ and CMAKE, which I did not get time till now to record, but I will record and make them available. Then a set of tutorials on resource management because that I think is very, very important. And several of the engineering issues like C++ coding style, how to write good code, and what is, what does this coding style mean, and what kind of coding style industry typically recommends, and so on so forth.

And obviously tutorials on C++ processes and tools. So, you know, it just not making a program, you have to make a software. So, it is a task that goes on and on and on for days, months, involving the customer fixing bugs, doing enhancement, redeployment, so on so forth. So, design development, build, debugging, test, source code and version management, bug tracking, lots of things need to be done.

I mean, of course, these are not specific to C++, but you need to know, what are those processes, and what are the tools that are involved. In terms of doing those processes of that only for build, I, we have covered in tutorial 1, 2, 4, the build part, but not the other tool. So, I, intend to record a couple of tutorials around that so that you can use them, where as you as you get into the practice.

And on the design side, there is a concept called design patterns in a certain types of designs are repeatedly used. And so I like to have 1 or 2 tutorials on that. And any suggestions I get further, I will certainly our, it may take a little bit of time for me to record, but I will certainly record the corresponding tutorial and make available to you even beyond your course completion and examination, so keep a watch.

(Refer Slide Time: 12:49)



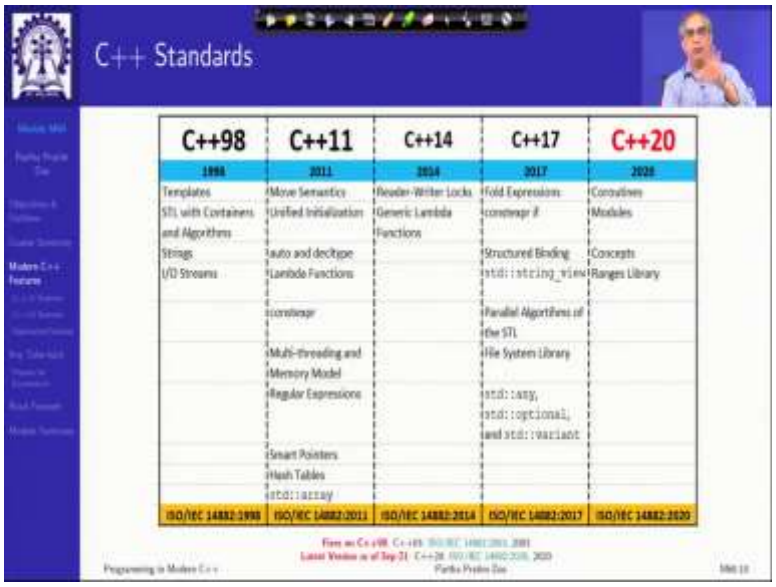
Modern C++ Features

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

Programming in Modern C++ Partha Pratim Das



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	Unified Initialization	Generic Lambda	constexpr if	Modules
and Algorithms		Functions		
Strings	auto and decltype		Structured Binding	Concepts
I/O Streams	Lambda Functions		`std::string_view`	Ranges Library
	constexpr		Parallel Algorithms of	
			the STL	
	Multi-threading and		File System Library	
	Memory Model			
	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

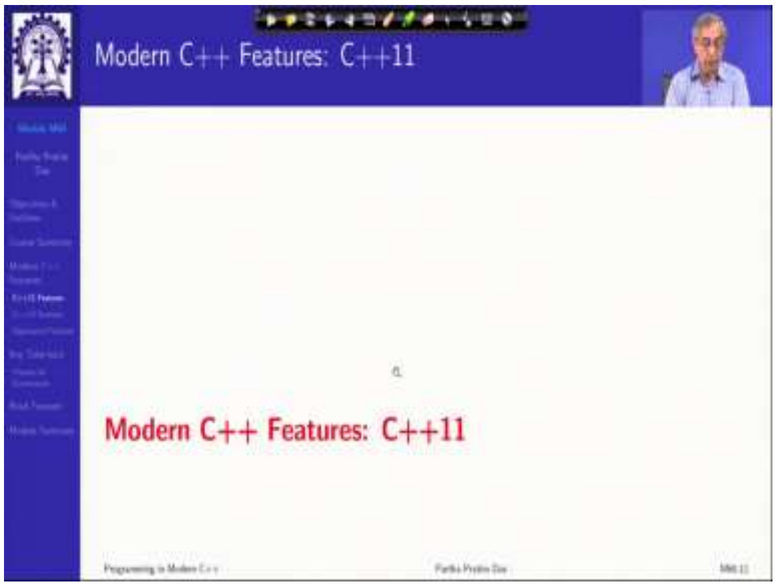
First as C++98. C++03: ISO/IEC 14882:2003, 2003
Latest Version as of Sep 21: C++20: ISO/IEC 14882:2020, 2020

Programming in Modern C++ Partha Pratim Das

Now, that was in terms of what we have done overall, if we look into the features, and this specifically, and what I am meaning by features is the Modern C++ because that is the evolving part, the C++03, 98 is pretty stabilized. So, all features of that are already covered in in the in the course as well as available at multiple places.

But what has happened beyond that, starting from C++11 to 20 as it goes on, particularly C++11 features.

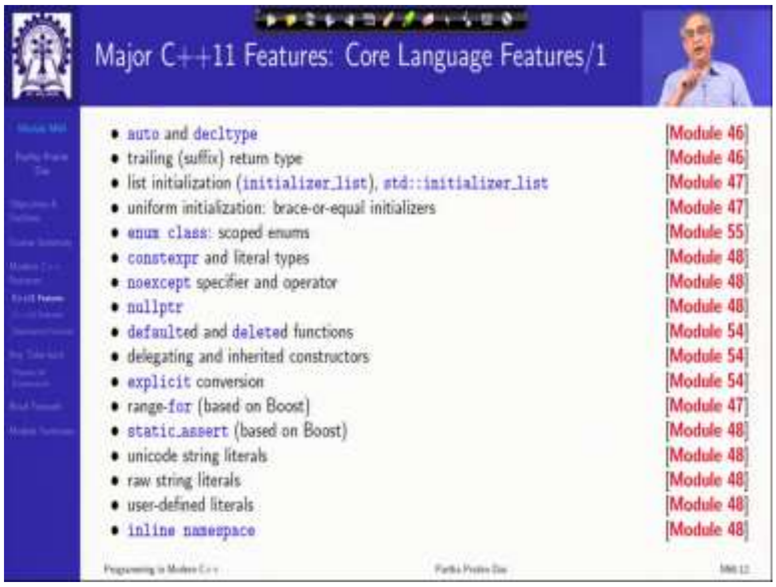
(Refer Slide Time: 13:26)



Modern C++ Features: C++11

Modern C++ Features: C++11

Programming in Modern C++ Partha Pratim Das M48.11



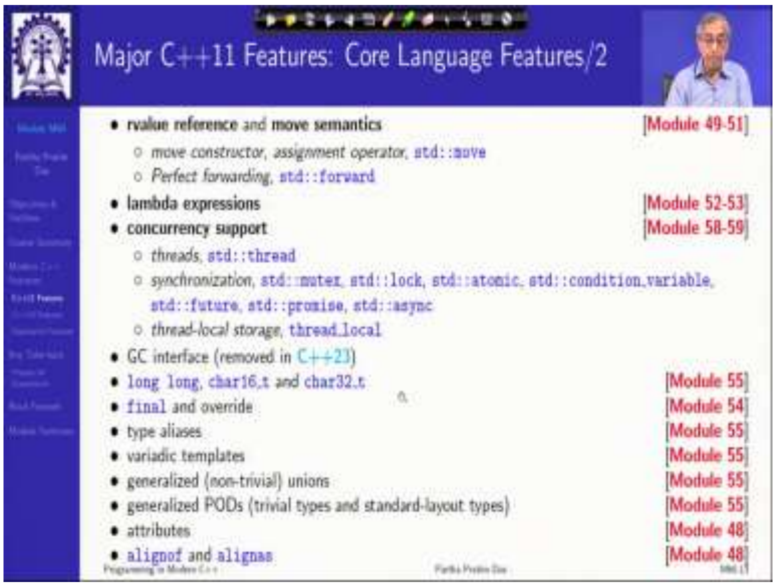
Major C++11 Features: Core Language Features/1

• `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]
• `default`ed and `delete`d 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]

Programming in Modern C++ Partha Pratim Das M48.12

So, here I have given you a map of the major C++11 features and where all you can expect the module where you can expect to find it, that just a cross reference kind of so that it makes it easier for you to, if you are trying to look for information, where is null pointer then you do not have to keep on searching you can just go null pointer, it is in module 48, open module 48. You will get that, open that video, you will get that.

(Refer Slide Time: 13:57)

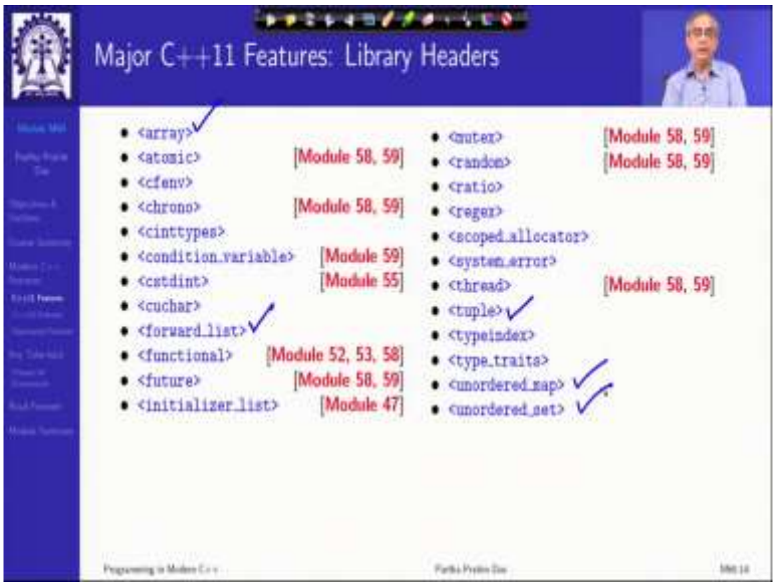


Major C++11 Features: Core Language Features/2

- 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]

That list continues naturally rvalue reference move semantics sub-multiple, Lambda expressions sub-multiple, conferences support has multiple modules. And then there are some more other features which I covered.

(Refer Slide Time: 14:13)



Major C++11 Features: Library Headers

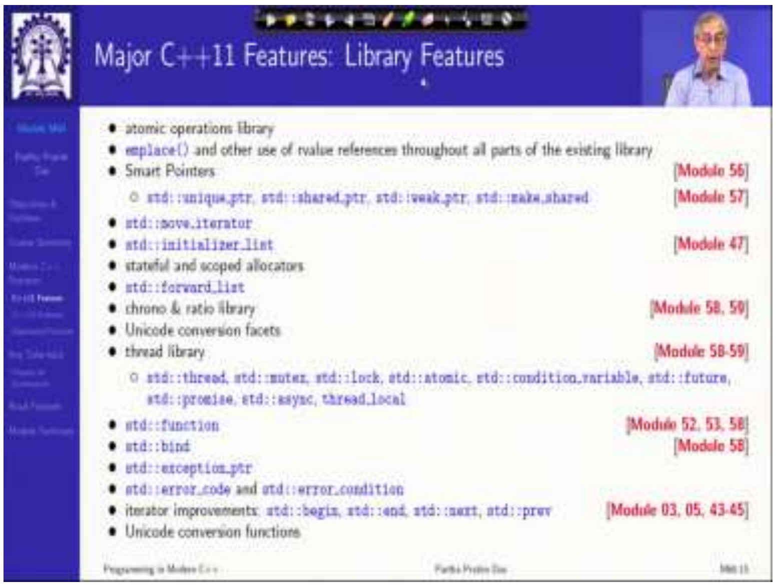
- <array> ✓
- <atomic> [Module 58, 59]
- <cfenv>
- <chrono> [Module 58, 59]
- <cinttypes>
- <condition_variable> [Module 59]
- <cstdint> [Module 55]
- <cuchar>
- <forward_list> ✓
- <functional> [Module 52, 53, 58]
- <future> [Module 58, 59]
- <initializer_list> [Module 47]
- <mutex> [Module 58, 59]
-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

So, this was in terms of the core language features, I had talked of several library headers also I mean, if you open module 46, when we started with the modern part you will see that we had put this list as what we intend to cover.

So, of these obviously we have not been able to cover everything but those library components which we have covered I have mentioned, their names, I mean module names where there. Important omissions are array, which is a fixed size array, which is a container, new container, forward_list, which is another new container.

These we did not get time and this should have been and tuple, tuple also, these 3 should have been covered, but I did not get time to do that maybe, we will do a tutorial on containers particularly and also talk about them. So, this is about but most of the other major these are also containers, but they are very similar to map and set that you have the ordered map and ordered set that you have. But for the rest we have shown what is the, what are the modules where you can find this.

(Refer Slide Time: 15:33)



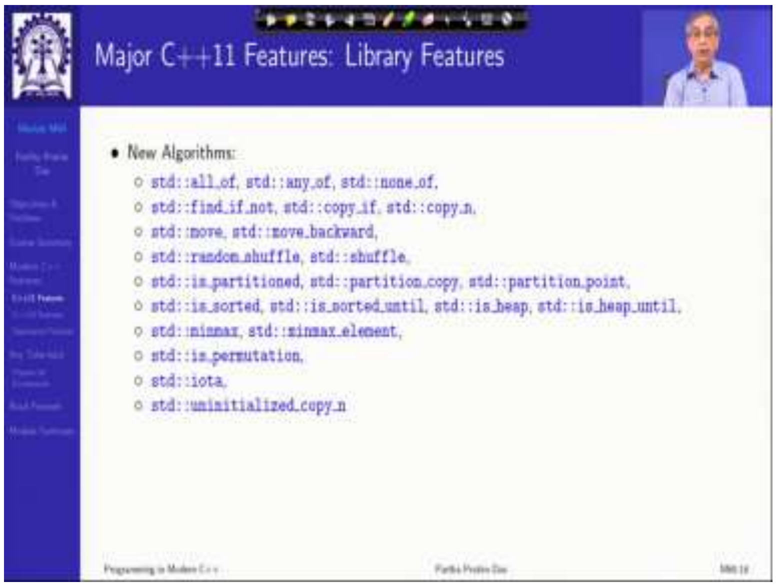
Major C++11 Features: Library Features

- 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

Programming in Modern C++ Partha Pratim Das M46.15

Looking in terms, I mean this into a level of different glasses through major library features like smart pointers, where we can find the major smart pointers or say the initializer list or the thread library discussion of the std::function, or std::bind, and so on that mappings are given here. So, that will make it easier for you to locate again.

(Refer Slide Time: 16:01)



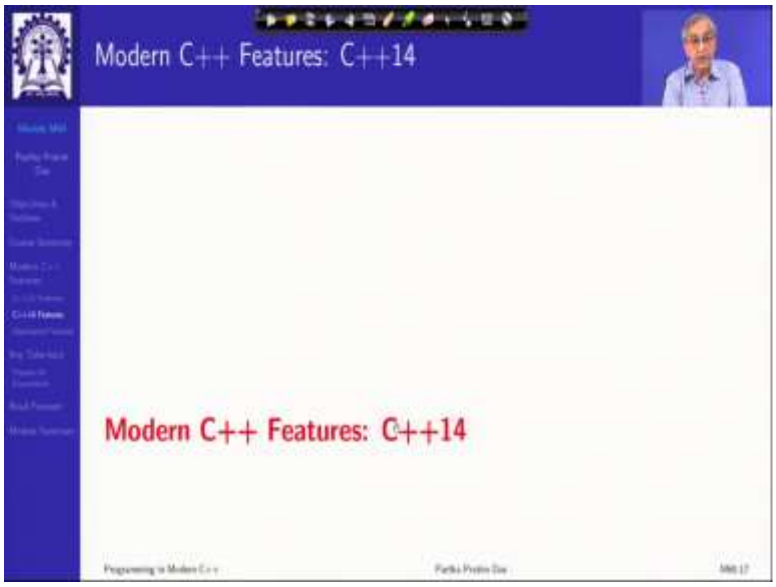
Major C++11 Features: Library Features

- 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

Programming in Modern C++ Partha Pratim Das M46.16

There are new several new algorithms added, some of these we have discussed but not exhaustively and in the algorithms part you will find that.

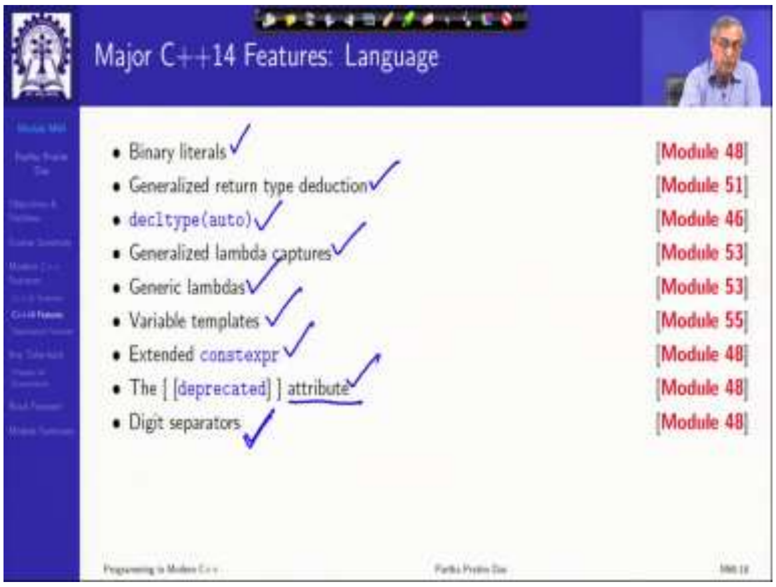
(Refer Slide Time: 16:12)



Modern C++ Features: C++14

Modern C++ Features: C++14

Programming in Modern C++ Partha Pratim Das M46.17



Major C++14 Features: Language

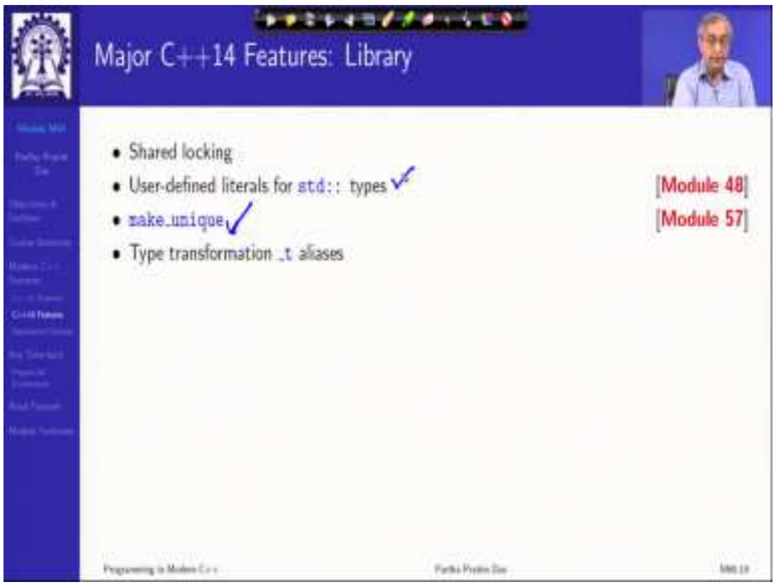
- Binary literals ✓ [Module 48]
- Generalized return type deduction ✓ [Module 51]
- `decltype(auto)` ✓ [Module 46]
- Generalized lambda captures ✓ [Module 53]
- Generic lambdas ✓ [Module 53]
- Variable templates ✓ [Module 55]
- Extended `constexpr` ✓ [Module 48]
- The [[deprecated]] attribute ✓ [Module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

Programming in Modern C++ Partha Pratim Das M46.18

Coming to C++14 There are few major features in the language and most of them, we have covered and most of them are not independent like this one, this one, this one, this one, this one, this one, these are not independent features, these are improvements on C++11 features therefore, while discussing the C++11 feature itself, we have included the discussion of the C++14 features.

So, the corresponding modules you will find, only these are there are small features like in fact attribute also is an extension of a C++11 feature the binary literal, digit separator, very minor features these are independent, but most of C++11 features have been given a glimpse to.

(Refer Slide Time: 17:13)



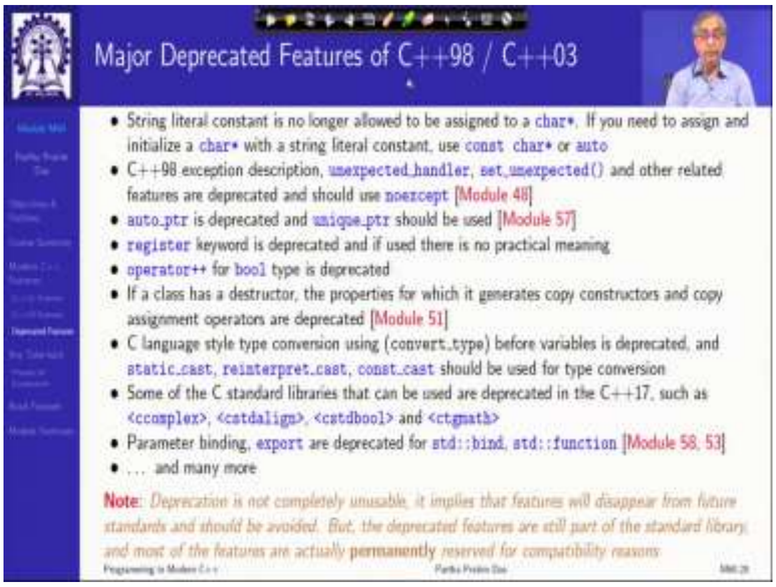
Major C++14 Features: Library

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

Programming in Modern C++ Partha Pratim Das M46.19

In terms of the C++11 library features. The most important feature is certainly make_unique which is used for unique_ptr which we have covered well and we have covered the std:: types or the the other 2 have not been covered.

(Refer Slide Time: 17:36)



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*

Programming in Modern C++ Partha Pratim Das MM.29

I have not listed beyond this, in terms of C++17 or C++20 features, because they are, they would have taken a lot more time and you have to digest this itself which is big. The another list which is incomplete, but I just wanted to sensitize you on is as we are looking at additions to C++98, 03 from C++11 onwards, there are deprecations also, like we used to write string as char*, in C++11 onwards you are not allowed to do that, you have to either write it as a string literal has to be a const char* or auto.

You have auto_ptr is deprecated we have already told that, you have unique_ptr. The increment operator for bool type is deprecated because it has no sense, because now it bool is truly a type it is not a, an integer interpreted.

register keyword for storage specifier, I mean you can use it but it has no effect anymore. Then we you have deprecation of C style cast conversion you have the application of other forms of parameter binding like export and std::bind and std::functions to be used and so on.

So, not all of them, we have explicitly discussed those which we have I have mentioned the modules, but it is very important to note that, if you get a warning from the compiler usually on deprecated features you will get a warning, deprecated features does not mean that they are not usable. But it means that this feature will disappear soon from the future standards and should be avoided. But the deprecated feature will still be a part of the standard library for backward compatibility reasons, but it is not at all advisable to use them.

So, compilers usually give errors or not errors, warnings on that, so make sure that you check on those warnings and if you get such go back and check if you are using deprecated feature then move on use the right feature.

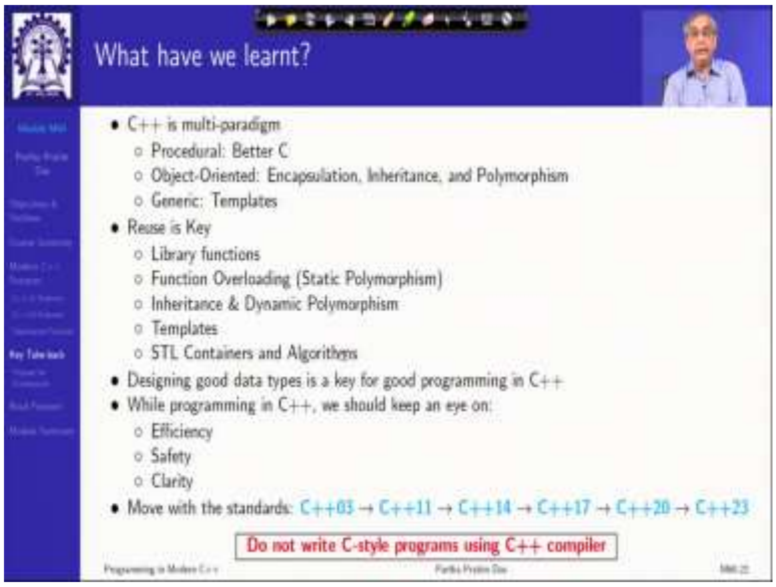
(Refer Slide Time: 20:06)

Key Take-back

Key Take-back

So, this was kind of language wise, this is what we have been able to cover. So, I talk from the perspective of the module, weekly structure tutorials and so on. And from the language side and what is a mapping.

(Refer Slide Time: 20:22)



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

If we talk about key takeback, obviously, C++ is a multi paradigm language is what you have lot more strongly learned procedural, it is a better C object oriented and generic. And if you now, look back, the generic part is very, very heavily strengthened by C++11.

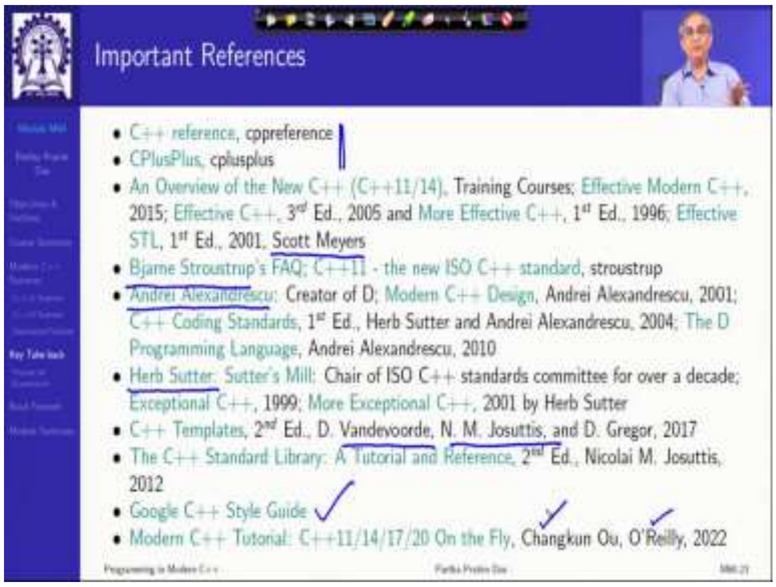
And reuse remains to be the key. So, macros also offered reuse but kind of, in today's time, you will say that, well, it is very, very difficult to find a use case where you have to use a macro. So, you probably are not using it, then you have library functions, function overloading, static polymorphism, dynamic polymorphism, you have templates, you have containers, algorithms.

So, C++11, promotes safety and reuse to a great great extent, if you have understood that learnt that internalize that, then that will be the greatest learning that you have. Naturally, designing good datatype safe type is very, very important for C++. And while programming in C++, you must keep an eye on efficiency because that is why the language exist. And from one version to the other, language apparently is making becoming bigger or at least the library is becoming bigger, but the efficiency is if not improving, at least is not degrading.

That is the core focus. Safety is obviously improving, I mean C++11 is lot more typesafe than C++03 and so on so forth. Clarity, this is a major focus on the clarity that it should be easy to read, intuitive and easy to work with.

So, the final take back is it is an evolving language, it is a live language. So, do not think that your learning ends by doing this course, which has gone up to C++11 mostly, but keep learning, keep moving with the standard C++03 to 11 to 14 to 17 to 20 and 23 is around the corner, it will, next year it will be there. So, try to always learn what is there.

(Refer Slide Time: 22:51)



Important References

- C++ reference, cppreference
- CPlusPlus, cplusplus
- An Overview of the New C++ (C++11/14), Training Courses; Effective Modern C++, 2015; Effective C++, 3rd Ed., 2005 and More Effective C++, 1st Ed., 1996; Effective STL, 1st 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, 1st 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, 2nd Ed., D. Vandevoorde, N. M. Josuttis, and D. Gregor, 2017
- The C++ Standard Library: A Tutorial and Reference, 2nd 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

Over this course, in different modules, different places, I have given plethora of references, those that can always be referred, but here is a 1 slide summary of important references. If you have to be a master of C++, C++11, 14 onwards, you can rely on this, these references. And these are typically, kind of private sites which give you a good idea about different features.

But, I would say that, like when you when you learn literature, you say you read the great authors, you read Shakespeare, you read Wordsworth, you read Milton, so on so forth, or Salman Rushdie at a later point of time and so on.

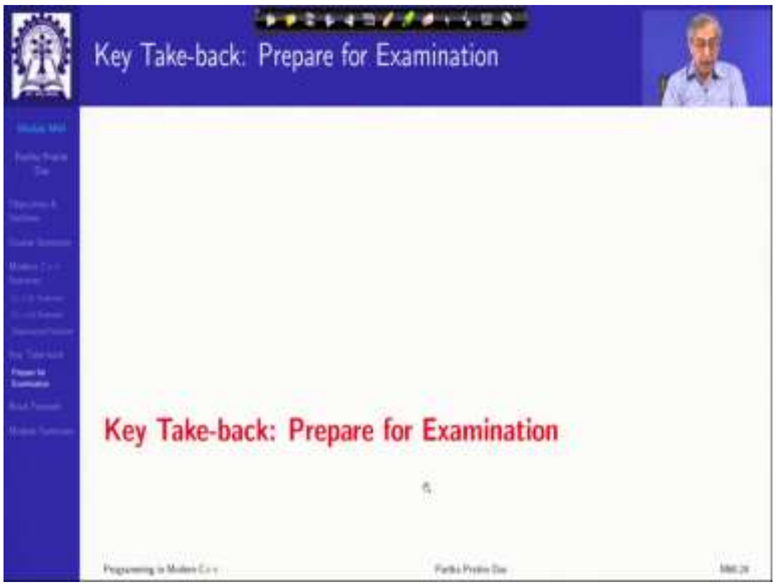
Similarly, here, you must rely on the great authors, Scott Meyers, the creator Stroustrup's, Andrei Alexandrascu, Herb Sutter. Here, I have tried to organize them according to those authors. These are, at least in my view, are the 4 great authors in the learning of C++ content from various aspects.

So, their key works, I have listed here, then. These are also very famous author Josuttis, Vandevoorde and so on. An excellent style guide is available from Google. And in the O'Reilly, there is a nice C plus, modern C++ tutorial which talks about all these 4 dialects, what they have added. It is about an 80 page book, not difficult, very, written in a very crisp manner. And Changkun Ou has been kind to put this entire book in Creative Commons license. So, in this link, you do not have to buy the book in this link, you can go and download that book.

So, these are some of the important references, besides the ISO standard and so on. I did not mention the standard here because you do not really learn, you can not really learn by reading the standard.

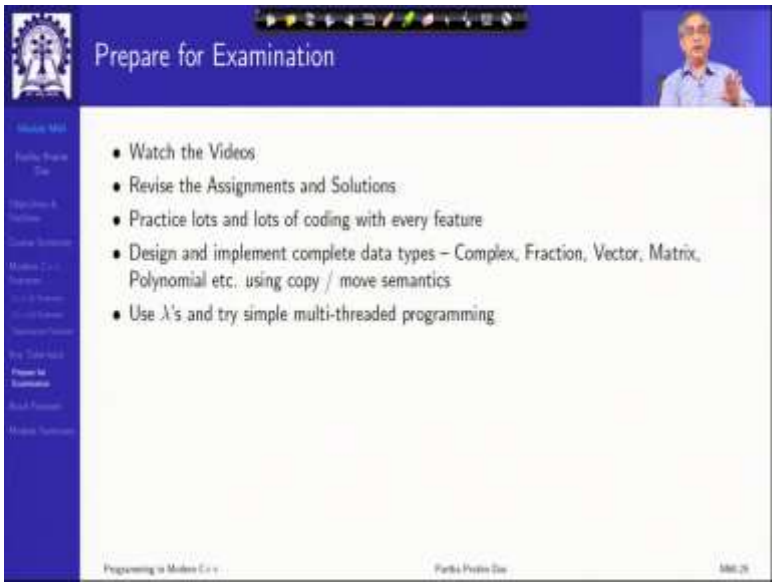
So, for Stroustrup, also, I have not actually mentioned the book, the Bible book. But here I am talking about practical references which a programmer as an engineer, the developer, will need to look up and quickly get clarified on different aspects of the knowledge related to the language and the programming and the usage.

(Refer Slide Time: 25:47)



Key Take-back: Prepare for Examination

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

Regarding your examination naturally watch the videos carefully revise the assignments and solutions, because the final question paper is going to be exactly in the same line. Some will be recalled questions from the module, some will be, recall questions with little modification from the assignments and some will be new questions. Practice lots of lots of coding with every feature, because that is the only way you can learn.

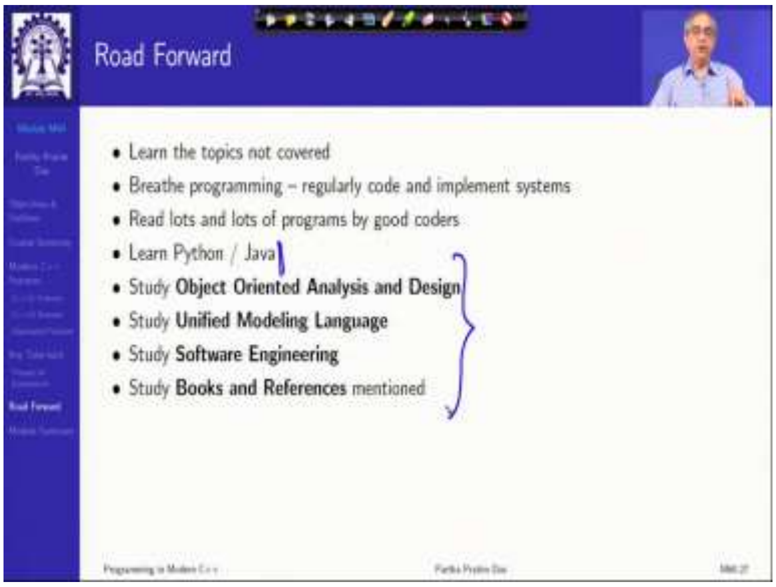
Design, try to design and implement complete data types, we have tutorials on that as well. And very specifically, focus on well Resource Management, lambdas, simple multi threading, and so on. These are some of the key things that you should not get challenged in the examination.

Of course, given the entire course, there will not be a lot of questions on this, but there will be some key questions on these areas as well.

(Refer Slide Time: 26:48)

Road Forward

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

Well, many of you ask what next, you know, that depends on what you want to do. Obviously, learn the topics not covered, there will always be many there dialects, which we have not covered as I said, and as I say that breath programming regularly code and implement systems, the only way to keep going and keep growing.

Try to read lots of lots of programs by good coders from jet hubs and other sources. I would advise that to learn C++ better and realize also learn Python and Java when you get time, it is not good to be a one language developer never. Not on all not in terms of your job opportunities, but in terms of your core technical skills all as such.

And then these are these are some of the, you know, future. Some of the subjects that are related, which you will benefit from, of course algorithms and data structure I have not written here because that was to be a prerequisite to doing this course. But object oriented analysis and design NPTEL has a course on that as well.

Unified Modeling Language, which is mostly covered in that object oriented analysis and design course, there are courses on software engineering, that you must, it is always very important. Because if you can just write a program not many companies would be heavily interested in you, they will be interested.

But if you know a little bit of software engineering how processes go how dynamics of software creation work, they will be more interested. And the more they get interested, you get a better package. That is the sole idea. And of course, to go forward, study the books and references that I have just mentioned in the summary.

(Refer Slide Time: 28:54)

Module Summary

- Course on Modern C++ concluded

So, that is all that I had to share with you in these concluding remarks. So, the course on modern C++ is concluded, but will continue to create tutorials and maybe additional material for you and putting on the NPTEL site.

So, even in the future, you can come there and download and see those videos and make use of the, I mean if you can make use of this knowledge and really get a good career path forward. That would be the best reward for all of us who have worked so hard on this course. All the very best, thank you very much for being with us in this course.