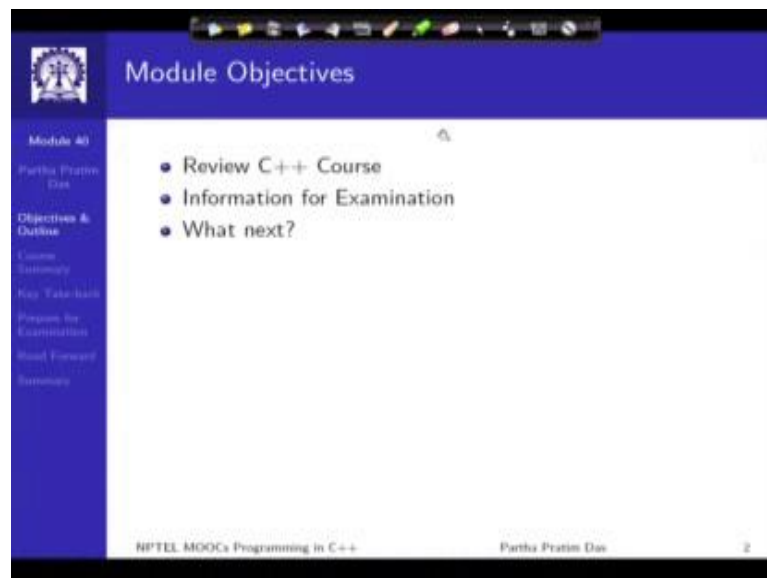


**Programming in C++**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 56**  
**Closing Comments**

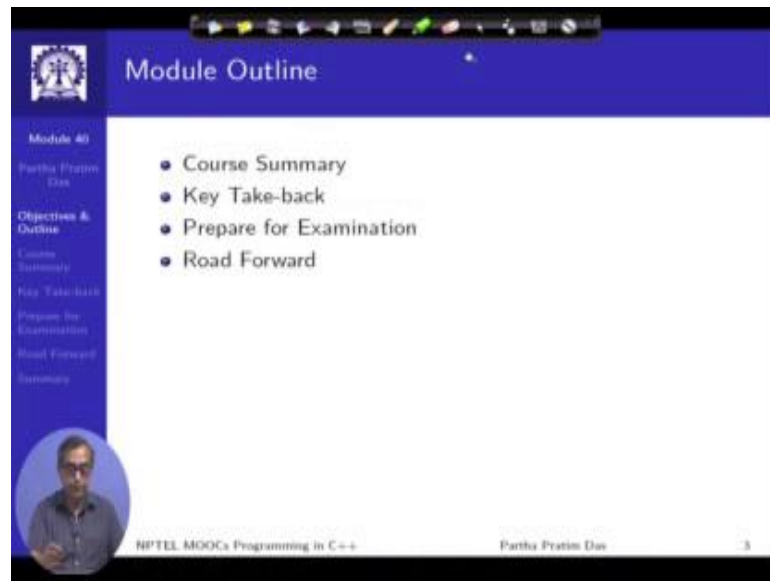
Welcome to module 40 of Programming in C++. As you know, this is the last module of this current course. So, we are not going to introduce any new material in this module. I would rather summarize the overall course, and try to make some closing remarks for you to move forward.

(Refer slide Time: 00:51)



So, the objective of this module would be to review quickly what we have done, what we have not done in the C++ course, in the 8 weeks. And I would like to put in few points regarding how we should be preparing for your examination, and what should be your next course of action beyond this course.

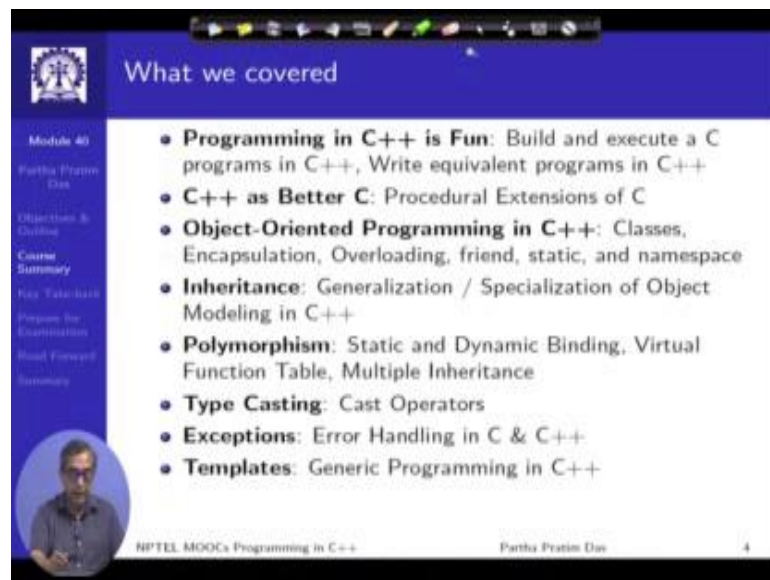
(Refer slide Time: 01:21)



The slide is titled "Module Outline" and features a blue header with the NPTEL logo on the left. A vertical sidebar on the left contains a navigation menu with the following items: "Module 40", "Partha Pratim Das", "Objectives & Outline", "Course Summary", "Key Take-back", "Prepare for Examination", "Road Forward", and "Summary". The main content area lists four bullet points: "Course Summary", "Key Take-back", "Prepare for Examination", and "Road Forward". A circular inset image of the speaker, Partha Pratim Das, is located in the bottom-left corner. The footer includes "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and the slide number "3".

So, this is the module outline and we get started here.

(Refer slide Time: 01:26)



The slide is titled "What we covered" and features a blue header with the NPTEL logo on the left. A vertical sidebar on the left contains a navigation menu with the following items: "Module 40", "Partha Pratim Das", "Objectives & Outline", "Course Summary", "Key Take-back", "Prepare for Examination", "Road Forward", and "Summary". The main content area lists seven bullet points: "Programming in C++ is Fun: Build and execute a C programs in C++, Write equivalent programs in C++", "C++ as Better C: Procedural Extensions of C", "Object-Oriented Programming in C++: Classes, Encapsulation, Overloading, friend, static, and namespace", "Inheritance: Generalization / Specialization of Object Modeling in C++", "Polymorphism: Static and Dynamic Binding, Virtual Function Table, Multiple Inheritance", "Type Casting: Cast Operators", "Exceptions: Error Handling in C & C++", and "Templates: Generic Programming in C++". A circular inset image of the speaker, Partha Pratim Das, is located in the bottom-left corner. The footer includes "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and the slide number "4".

So, the fun really starts happening when we start coding, the same concepts as in the C program in terms of C++ style. And start using different stls, and all other different futures and writing equivalent program. And through this, we really look into how C++

can improve the overall programming experience. With this foundation, what we covered is what we called C++ as better C that is the procedural features of C are extended in C++ to provide a better procedural programming language. This is required from two perspectives; one is just to make things better to work with, and second the most of these features the procedural extensions are critical to support the object-oriented features.

So, if we can just recall we have taken a look into the const-ness of values and variables which is critical in terms of a getting read of manifest constants being able to maintain better type all through the C++. And also in terms of introducing variety of notions of const-ness later on in terms of member function and so on. As you know by now is the const-ness turned out to be such an important improvement in the C programming that C language standard actually went forward, and have taken const as a part of C standard back way from C++.

Besides that we saw a whole lot of other features as better procedural features. We are talked about reference parameter, we talked about overloading. And very significantly completely new operators for memory management, the operator new and operator delete, the array new, array delete and so on. So, combined with all these, we have a procedural extension of a C, these features do not inherently have any object oriented concept, but these make us C kind of procedural coding in C++ much better.

With that foundation of better procedural support, we moved onto discussing object-oriented programming or op in C++. Certainly this was done at a detailed length we talked about classes, their members, the access specifiers, constructors, destructors, lifetime and all of these which basically all these come under the generic encapsulation paradigm of object-oriented programming. Where you encapsulate in two ways, one you encapsulate by packaging multiple data members in terms of an object, and only try to provide a set of behaviour, set of member functions to that. And with a help of visibility, the access specifiers, you make sure as to which part of the object is visible accessible to which members or which agents.

On top of that in support of object-oriented programming, we introduced a whole lot of very important features like overloading of a member functions. We talked about friend

member function, which provide a somewhat a different kind of visibility access visibility than the public or private visibility. We talked about static data members, and member functions. We talked about namespaces. Again namespace in that way may not be consider a strictly object-oriented features, but as it turns out that one we when we have to introduce classes they themselves introduce some kind of a namespace. So, a namespace is a logical consequence and certainly it gives a much better source organization code, organization options in terms of C++ programming.

So, with all these together, the main point at up to this part of the course has been that we can with the use of the object oriented features and now empowered to define a data types ourselves. So, the main achievement up to this point up to this point in the course where we just to object-oriented encapsulation overloading kind of features is that we can now built our own user-defined data type. And this data types could be as complete in the functionality as the built in data types like int, char, double and so on.

And we saw how all these different features of overloading, and friend functions and operator overloading specifically can be used to really have data types, which can be used to write expressions. We have consistently try to use complex data type as an example, of course, you can extend it you can do similar things for other data types as well like fraction like vector matrix and so on.

And built your own required data type, wherever you want and there is no end to what you can achieve in terms of this data types. And certainly while building the data type, there are certain point to keep in mind which I would specifically like to remind you off is a fact that while you bring a data type. And for that, you are overloading the operators; try to keep the semantics of the operator as close to the semantics of the operator in built in types. And do not make it very different, for example, if you are defining a set data type, and you are overloading the plus operator, then it would be intuitively much better to associate the plus operator with union of sets than intersection of sets or say the difference of sets and so on.

At the same time, in terms of the signature, it should remain as close to the built in data type operators as possible in terms of whether it returns the result through value, or

through a reference, or through a constant reference and so on. And we have discussed at length in terms of what difference nuances mean. Other notable points here is a detailed or detailed discussion on copy semantics or detailed discussion on object lifetime and or the variety of free functions that the compiler provide you to support ease of programming.

Moving on next, what you took up you forms another core aspect of object-oriented programming called inheritance or generalization specialization hierarchy. And though this course did not have enough opportunity to discuss object-orientation per say as a design paradigm, but still we have been able to see that in terms of real life world, there is a whole lot of situations where the specialization generalization between the problem domain within the problem domain between different concepts is just a natural phenomenon.

And in terms of the inheritance features, in terms of being able to derive a class from a base class, and in that process override the base class member functions with new member function and overload them if required and reuse them if you just inherit and so on, we can provide a whole lot of complex semantics to the natural world in a very organized and aligned manner. And this probably is inheritance probably is one of the key features of C++; particularly of object oriented aspects of C++ coupled with encapsulation, overloading. And inheritance provide you the core frame work in which the object-orientation can be very extensively modelled and programmed and manipulated in terms of the C++ language.

Next, what do we looked at is what is more commonly called polymorphism. Though the use of the word polymorphism will caution you has a very variety of different meaning in C++, but here what we meant by polymorphism is primarily dynamic binding. This is a very different kind of a feature that we took a look into where the type of a pointer the static type of a pointer or the static type of a reference alone does not decide the particular binding of a member function.

The type of the member or the specific member function being bound with a pointer dreferencing or a reference access is dependent on the actual object being pointed to at

the runtime that is a dynamic type of the pointer, dynamic type of the reference. And that has given us the ability to really build a big class hierarchy with this common base class and with a capability of virtual functions which can do runtime delegation of methods to appropriate runtime objects, the pure virtual function, abstract base class those are very key concepts that were introduced in this part. And we specifically spent time to also understand, how this virtual function mechanism work in terms of the virtual function table and how does it ramifying in terms of the multiple inheritance.

So, these all together form a very strong object-oriented foundation to the C++ language. And then in the next two parts, so we spent substantial amount of time in looking into two features, the type casting and exceptions, the features that become critical not only to support better object-oriented programming, but also to overall improve the experience of programming. So, in type casting, we talked about the possible conversion of objects at the runtime, at the static time based on their static type or their dynamic type and so on.

That is what we saw a whole lot of rules, there the implicit casting rules, the C style casting, what are the issues of the C style casting, and why, particularly cast operators the four-cast operators of which the static cast is certainly the most widely used. The most possible kind of static time casting can be done in terms of this static cast or the const cast, which applies to C v qualification or the dynamic cast which applies to the runtime casting particularly for down casting purposes.

So, you can see that the cast operators basically provide whole lot of different semantics based on the different context of a casting. And as you get more and more experience in C++, you will discover for yourself that one particularly since now you also know template that actually the cast operators are nothing but template definitions. So, In fact, it is not only that these are the four-cast operators that you have, but it is possible that you can define your own cast operator and define semantics for that.

So, casting or changing the type of an object or basically using an object of one type in the context of another is a very critical features, which is required for supporting strong object orientation, because a strong object orientation has meant see into a very strongly

type language. And therefore, there is often a context of using an object of a certain type in the context of another type. So, in that reference the type casting discussions and type casting would prove to be very useful for you I believe.

Exceptions are from a very different flavour, they for the first time address the question of error situation, exception situations and their handling in the comprehensive manner in terms of the language. Now, we all know that if we write software then the software might not work. So, error handling or being able to debug the programs efficiently is a part and parcel of developers own life, is a part and parcel of a program life cycle.

But unfortunately the C language does not provide in a built-in support in the language for doing handling all these you know unhappy paths, all these error paths and a synchronous, asynchronous the error introduce you to logical reasons the error introduce you to system configurations and all that. So, we took a very detailed look in terms of what C provides, what is the lacunae they are in and in view of that we try to understand the basic semantics of try throw catch exception clauses and how does the C++ provide that.

And again exceptions are a mechanism which really completes the whole story of object oriented programming much easier, because exception really make sure that it is no more necessary that you keep talking about doing certain operations and then tracking whether the operations has been correct or not. That if you provide the correct context of try clauses and put appropriate catch handlers, then you can write the whole code without really thinking of the exception situations and fill in the catch handlers for the different exception situations in a completely separate code base, so that being a much better clarity to that.

And at the end, of course, we could spend very little time compared to the depth of this particular topic. We just spent two modules, but this by itself could be become a 10 module course is the factor of templates. Templates are completely different concepts in terms of C and C++, because they are they are certain mechanisms by which you can write a code where one or more variables or parameter or class types are not known at the time of writing the code or at the time of compiling the code. The type could be

specified subsequently when you actually use that function, when you actually use that a particular class and we saw examples of this through simple max, swap kind of function and stack kind of data structure. So, templates give a different kind of polymorphism.

So, if you just look at then templates are a can be can be looked at in multiple different aspects one is it could look at it from the aspect of polymorphism, so overloading gives us a certain kind polymorphism which is called the static polymorphism. The dynamic binding or what we have 'return' here as polymorphism is a dynamic polymorphism, which is basically what happens in a class hierarchy. And template gives you another kind of polymorphism because that is a template polymorphism, because here again you have a single form of a function written in terms of the template of the function or single form of a class, but you can use it for variety of different purposes.

So, when you have a template function instantiation say an implicit instantiation then there is again the same question of binding which we had to answer in case of overloading which we had to answer in case of dynamic polymorphism we again have to answer that. So, that is a very different kind of features that comes in templates are code generated they actually generate code and then compiles that code.

So, if you are able to design templated code in a crisp efficient manner then your basic effort in programming, your basic effort in debugging gets substantially reduces because one code can be used in not only in the multiple contrast of type by throughs overloading, but you could keep on using it on also for the types that will come in future. So, templates are primarily introduced in C++ for this purpose of generic programming or template meta programming. So, if we look into from that aspect then we will actually see that C++ is a combination of three major paradigms of programming which is procedural, because it includes the whole of C and the better C. So, that gives you a complete set of features and lot of powers in the procedural terms. So, it is always very efficient to write algorithms in C++.

It is known for its object oriented features, so it is called an object oriented programming language as well. So, it strongly supports the object oriented paradigm though it as some



lacunae in that and those may be beyond the discussion of this course. But certainly it cannot do few fundamental things of object orientation like reflection, but what is important is it also supports a third paradigm which is called a generic programming paradigm, which is a code generation paradigm where you could write type parametrized codes and generate code based on instantiation. So, you can see that C++ basically develt on three different paradigms and therefore, it is very rightly called a multiple paradigm language.

Of course, if you refer back to C, the earlier version of C++, older versions of C++, then you might feel that it is just procedural and then object oriented, but over and over the years, the template features the generic programming features in C++ have been really gaining in strength. And though we have been using primarily and our discussion, we have been talking about C++ 99 which is a 17 year old standard which was a marginally revised in C++ 03, which means the 2003 standard. But subsequently, we have had significant progress in terms of C++ standard. We have two new standards now; one is called C++ 11 which was finally, actually released in 2012; and we have another standard, the most recent standard is C++ 14 which was released last year in 2015.

And we have not discussed as upon any of these features from C++ 11 or C++ 14 which give you a whole lot of very strong additional parameter to strengthen all of these paradigms primarily the object oriented paradigm, and the generic programming paradigm. And brings it lot of strong concepts into the existing features and still maintains complete backward compatibility with the earlier C++ 03 language. So, this is a basically what I would like to just point out that the whole thing that we have covered is just kind of the small part of what is C++ today, and the way it can actually benefit you.

(Refer slide Time: 22:55)

What we did not cover?

- **Functors:** Function Objects
- **STL:** Standard Template Library of C++
- **Resource Management:** Smart Pointers, Memory Handling
- **C++ Coding Styles:** How to write good code?
- **Design Patterns:** Reusable Designs
- **Mixing C and C++:** Smart Pointers, Memory Handling
- **Source Code Management:** How to organize files, libraries?
- **C++ Tools:** Analysis, Version Control etc.
- ...

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

So, having said that I would also like to point out a few features; now what we have not covered is very difficult to list out because C++ is actually so huge that it is very difficult to say what we have not done. It will not fit into one slide or couple of slides, but here I have just highlighted some of the aspects those are very, very important and you should going forward take initiative to learn them better. So, one certainly is some of the typical programming styles that have emerged based on C++, one is called functors.

Functors are very interesting design, they are called function objects. So, we know functions are what can be called and objects are what can be instantiated. But we have new concepts here where it will say that a function can be instantiated, it could be an object it could have a state. So, it simple a function calls is basically shown in terms of the function operator which is like this. So, a functor object is nothing but a class definition or an object where this function call operator itself is overloaded. So, it is the does not is not really as simple as I am saying, but it is almost that. And once you can do that you can have a lot of benefits.

And the next thing that we have not significantly done accept for sparingly using certain data structures like vector and list and stack is the standard template library which is significantly based on the concepts of functor. And it is very critical that you explore and

a slowly get more and more familiar with this standard template library, so that your power in being able to write good C++ code also improves. Couple of other very you know important aspects which we did not get time for is resource management particularly concepts of smart pointer memory handling and so on. C++ coding style is to how you should be writing codes in C++ certainly not the C way. Design patterns, which tell you, what are the reusable designs? Many of the practical systems will need you to make C with C++ may be there is an existing code base of C you cannot throw all of that away and write C++.

So, you as you write certain parts in C++ you continue to have the remaining parts in C. So, how you make them is a big question. Management of source code C++ has given features like as you saw namespace and export and all that. And with that you can do a much better C++ source code management and certainly for engineering purposes you need to learn a whole lot of C++ tool. So, there is the list is much, much longer. I have not talked about in this any features which C++ 11 or C++ 14 have introduced these are all aspects of C++ 03, but it will be good overtime, if you can really learn them.

(Refer slide Time: 26:12)

**What have we learnt?**

- C++ is multi-paradigm
  - Procedural: Better C
  - Object-Oriented: Encapsulation, Inheritance, and Polymorphism
  - Generic: Templates
- Reuse is Key
  - Macros
  - Library functions
  - Function Overloading (Static Polymorphism)
  - Inheritance & Dynamic Polymorphism
  - Templates & STL
  - Design Patterns
- 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

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

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

So, if we try to summarize us to what we have learned. We have learnt the first is the fact that C++ is a multiple paradigm language it is procedural, it is object oriented and it is

generic. So, in future going forward from tomorrow as you do C++ programming, always try to identify which paradigm you are working in, and how do they makes. A second aspect is reuse is the key in C++. If you see the whole gamut of features as we have seen then there C supported things like macros and library functions only this is what you had in C.

And then in C++, we have function overloading, the static polymorphism, the dynamic polymorphism other kinds of reuse options and then when we do templates and as you go into STL, you will find a huge amount of reuse here because now, you are writing one max function with a parameterised type, and that will work for not only the built in types, but for all future user defined types that you will come across with.

Design patterns are another aspects of reuse where you not only irrelevant limited to reusing code, but you try to reuse the pair designs as well. Certainly the other aspects that you should have learnt well, and is designing good data types is a key for good programming in C++, because C++ is strongly type and everything that you want to do, you want to make a type for that.

And while programming in C++ you should keep an eye on the efficiency because C++ happens to be the most efficient language generic program I mean general purposes programming language today, even it is more efficient than C. Most experiment show that an equivalent code corresponding to the C code written in the C++ would run anywhere between 50 to 60 to 100 percent faster than the C code. So, efficiency is a key, safety is a key, and certainly all different exceptions and all those add up to the safety features and clarity is a major factor that it should be very clearly understandable.

So, in terms of the take back key take back I would like to like you to ponder over this points and I would finally, make one caution which just pardon me for saying this, but have been dealing with budding C++ programmers for last possibly 20 years. And what I find is a good tendency that the programmers have is actually use the C++ compiler, but write code in C.

I do not mean using the syntax of C, you can still use the syntax of C++ you can still use objects and specialization and all that, but the way you do the design is a C style of design. So, I have regularly try to show you the comparison between C style of solving problem and C++ style of solving the problem. Please refer to those please refer to good code solutions, and make sure that when you use C++ you write the code in the C++ style taking full advantage of the multi paradigm language and do not happen to just write a C++ compiler based code with C style.

(Refer slide Time: 29:39)

The image shows a presentation slide with a blue header and a white content area. The header contains the text 'Prepare for Examination' and a small logo on the left. The content area lists several bullet points: '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.', and 'Study Books, try examples'. Under the 'Study Books' bullet point, there are two sub-bullets: 'The C++ Programming Language by Bjarne Stroustrup' and 'Effective C++ & More Effective C++ by Scott Meyers'. On the left side, there is a vertical sidebar with a blue background and white text, listing 'Module 40', 'Partha Pratim Das', 'Objectives & Outline', 'Course Summary', 'New Table-Of-Contents', 'Prepare for Examination', 'Next Forward', and 'Summary'. At the bottom left, there is a small circular video inset showing a man speaking. At the bottom, there is a footer with 'NPTEL MOOCs Programming In C++' and 'Partha Pratim Das'.

Finally to prepare for your examination these are the routine things; watch the video, revise the assignments and solutions. We will provide explanatory solutions soon, practise lots of lots of coding that is a key of learning this. And certainly design and implement complete data types, I mean we have done complex, these are some of the sample data types which I can where glimpses we have done, but you can practise by doing this data types at length. And if you need to refer to books these are the couple of books that I will recommend you to follow.

(Refer slide Time: 30:17)



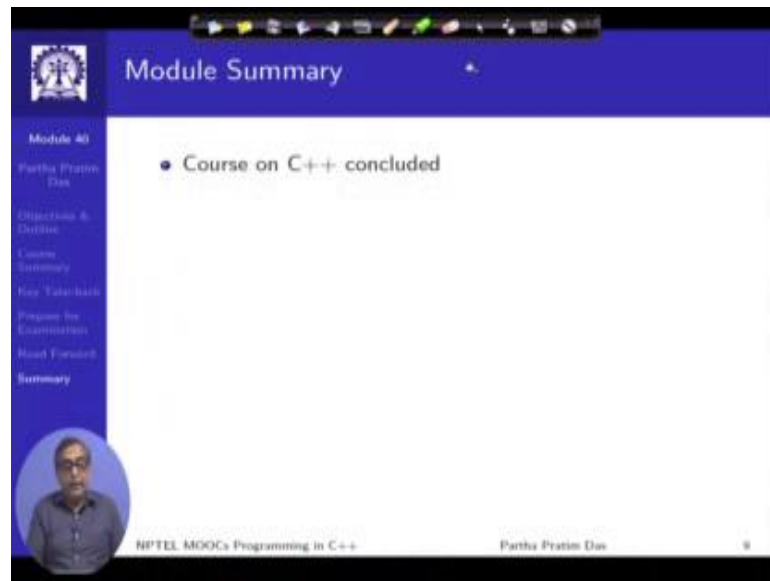
**Road Forward**

- 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
  - The C++ Programming Language by Bjarne Stroustrup
  - Effective C++ & More Effective C++ by Scott Meyers
  - Exceptional C++ & More Exceptional C++ by Herb Sutter
  - Modern C++ Design by Andrei Alexandrescu
  - Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, & John Vlissides
  - Learning UML 2.0 – A Pragmatic Introduction to UML by Russ Miles & Kim Hamilton (O'Reilly)

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

Going forward, beyond this course, I could tell you lot of things that you could do, you could learn the topics that were covered. But the core things you must breathe programming regularly code and implement systems, read lots of code that these two are very, very important to learn C++. And beyond that these are all the futures studies that you can do. It is good to learn other object-oriented languages to understand one language better. So, learn python, learn java. If you get an opportunity and go through object-oriented analysis and design UML for modelling the system software engineering and here are some of the very good books which deal with C++ or some of this related topics.

(Refer slide Time: 31:03)



So, with that, I would to close and summarize simply saying that the course on C++ is concluded. Wish you all the best for your examination and beyond that to become a very efficient, proficient and prolific programmer in C++.