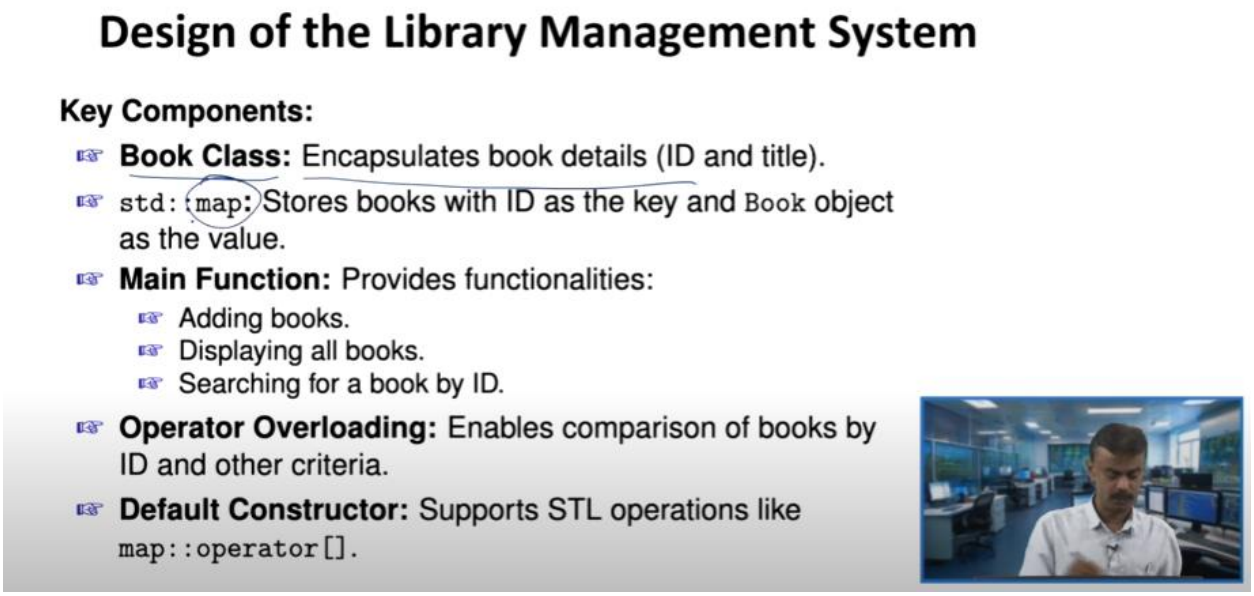
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture45

Lecture 45: Case Studies - Library Management System, Real-Time Stock Tracker

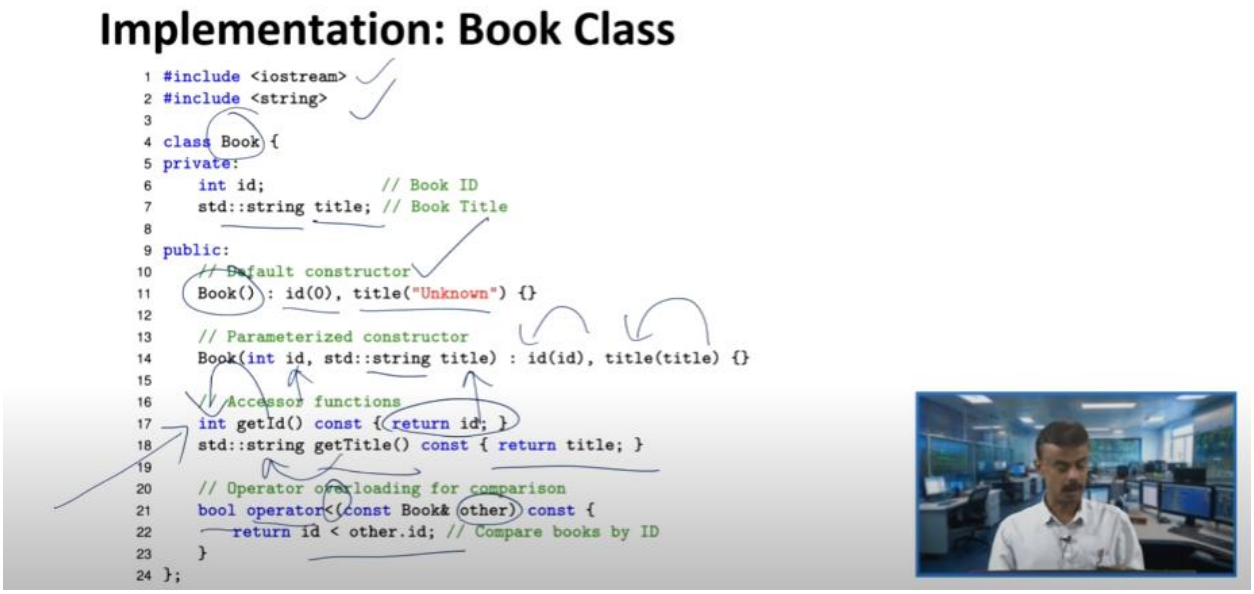


Welcome to lecture 45 on the standard template library, which we call STL, right? So, we have studied almost all the concepts in STL. So now, what we can do is take two case studies, right? So, one case study is about a library management system, right? So, the objective of this case study is to design a system to manage books in a library using STL, right?

So, we used to manage books with unique IDs and titles. So, what we can do here is use a map, right? So, whatever we have studied, In this STL, we use a map. So, the question is to use a map to store book details efficiently, all right.

Then, demonstrate encapsulation. So, whatever object-oriented concepts we have studied, we are going to utilize encapsulation using a book class. Implement operator overloading for comparisons, right. It will be a very good case study, right. We are going to create a class with the help of encapsulation.

And we are going to use operator overloading. And more importantly, whatever we studied very recently—map, right? Because of STL, right. So, we are going to use this map to store book details efficiently. So, let us consider the key components. So here, you have the book class, right.



This encapsulates book details: ID and title. And more recently, we studied the map. So, here we are going to use a map. So, which will be storing books with an ID. That is the integer value.

And this is the key. And the book object—the title of the book, right? It is a string as the value. So, the main functions are adding the books. So, these are all the functionalities. Displaying all the books and searching for a book.

So, these are all the main functionalities of the library: adding a book, displaying all the books, and searching for a book, right? So, these are all the main functions. So, operator overloading. So, this is useful to compare the books, right? So, either by ID or some other criteria.

So, we may use less than or equal, right? Some operators we are going to overload that we will see. And then we use a default constructor. Right. So, this will be useful. And, in fact, it is supporting STL operations like operator.

So, let us consider how we are going to implement this. So, let us consider the book class. Right. I include iostream, string, usual program. Right.

So let us have class book. Class book has two private member data. One is ID. Right. Whose data type is integer.

Another one is title. whose data type is string and here you have in the public you have a constructor which is a default constructor and id is assigned to 0 you are initializing id equal to 0 and title is unknown right title is unknown right that is about your default constructor next one is the parametrized constructor so the parametrized constructor id int right int id and you have title which is string right and then you are assigning id which is equal to id and title is equal to title right.

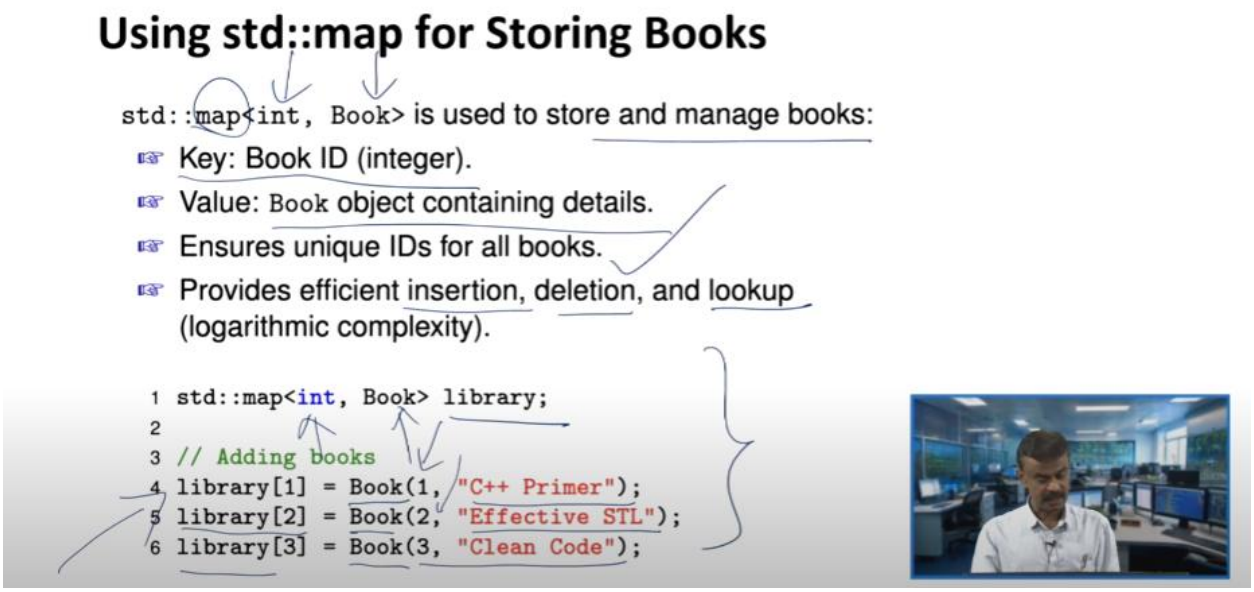
So, this is a parameterized constructor. Then you have accessor functions right get and set kind of right. So, here in this case get int get id right. So, that will return id that will return id. and here get title so the getId whose return type is integer and getTitle member function whose return type is string right.

so this is returning title and here you have operator overloading right so the operator overloading you use less than for operator overloading you use return type bool and you have operator right and then other object is other it is a reference whose class is book, right. So, here it will return 1 if id is less than other dot id, right. If this inequality is being satisfied, it will return 1, otherwise it will return 0.

So, that means you are comparing books by id, right. So, this is the class. If I am asking you to encapsulate this, you write, right, with default constructor, with parameterized constructor, with get and set functions. You call it as accessor functions because I can able to access ID and the title from outside, right. So, because your getId and getTitle are public, right.

So, these are all the public member functions. So, I can able to access. In fact, these functions can able to access ID and title, right. So, these all we know. And then you have the operator overloading.

So I encapsulate the complete class book over here with the help of default constructor, parameters constructor, accessor functions and then operator overloading. So now when I use map for storing books. So we will see how we are going to do this. So here we use map. So the map contains integer and book.



right you have one integer variable and object book you can have we have seen several examples that the map right having more than one variable right so here we are using variable and object object is book right so this is used to store and manage books and then here you have key book id right so key which is nothing but book id which is an integer and then the value book object containing details may be name of the book All right.

Name of the book. So whenever I am having the book, I mean, you can find it in the library. You will have the unique IDs. All right. You will have the unique IDs.

All right. So, we have to ensure that. And, in fact, the map ensures. Right. And it provides efficient insertion, deletion, and lookup.

Right. So, we are going to make sure of that. So, this map provides efficient insertion, deletion, and lookup. Right. So, these are all the reasons we use a map.

All right. So, for storing books. So, let us see how you can do this. All right. So, if you recall, we studied an example: Alice, Bob, Charlie.

Right. So these are all the examples we had seen in the previous classes. Right. So here map, map contains in book and then I have an object library. Right.

So the library comes under the class map. Map is having integer variable and book object. So now I can add books like this library of one. All right. So you have book object.

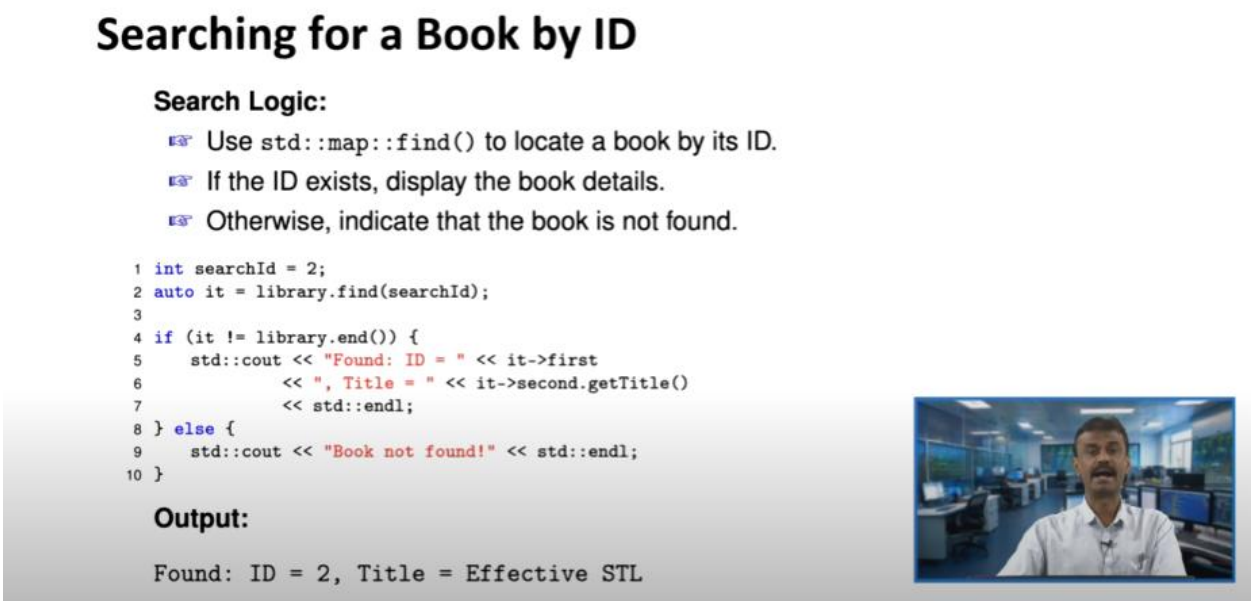
Correct. So you are passing one and C++ primer. And the next one is library two. Right. You have book object.

You are passing two effective STLs. Right. And a library of three object books. Passing three and clean code. Right.

So, like this, I can form the map. Right. I can write the map like this. So, whereas when I am using the map, these are all storing the information about the books. So, I can do it with the help of a map like this.

So, now adding books and displaying the library. So, what can you do, right? So, again, I am using a map. Under the map, I have the operator, right? So, the operator, I am going to overload this, the array symbol, to add key-value pairs.

Right. So, to add key-value pairs and then display the book using a range-based for loop. The range-based for loop is useful to display the books. So, here you go. So, the library we have already used.



So, here, auto, the reference of entry and library, both are the same, and the cout, right. So, here in the previous case, we have this as the library, right. Library 1, 2, and 3. So, now they are going to map to entry, right, and then This is a range-based for loop.

I am going to display the ID and title, right. So, the library has an ID and title. If you look at line numbers 4, 5, and 6. So, now the library's base address is copied to entry. So, the entry will also contain all the information, and when you are printing, that means we had seen a similar example, if you look, right.

First and second age, right and the person. Alice 30, Bob 35, if you recall, right. So, in a similar way, entry dot first and entry dot second dot get title, right. So, these when you print, right, entry dot first. So, we know first will be nothing but the number 1 that you are passing or 2 you are passing.

So, that will be displayed, right. ID colon is outside, this will be displayed and entry first is a number. And then title. So, under title second dot getTitle. So, getTitle we have here line number 18 right.

So, that is returning title right. So, here you go title colon we are printing. So, that will be there. So, it will print C++ primer or effective STL right with ID 2. So, all these information will be printed right.

So, with the help of a range-based for loop. So, we can do this correctly. So, now we are searching for a book by ID. Alright. So, we have.

Right. Search logic. So, what are we going to do? We are going to call find. Right.

We use find. Just now we have seen. In fact, last lecture we have seen. Find. Right.

So, the find function which will locate a book by its ID. In this case study. Right. If the particular ID exists. So, that will display the book details.

Alright. So, that will display the book details. Otherwise. So, this is indicating that the book is not found all right. So, let us see how this works.

So, here you have int search id equal to 2 right int search id equal to 2. So, here the find function we had seen right if you recall the find function. Right. So that will be returning the reference. Right.

I mean, in fact, one of the examples we have seen that is returning the reference. So here find search ID. Your search ID is 2. You are passing 2. So the library is invoking this find.

So which is returning the reference. Right. So now when I have this library, you can have. So library 1, library 2, library 3, that means library 1 is the begin, library 3 is the end. Again, you recall the begin and end concept that we had seen.

You recall the begin and end concept that we had seen. So in that case, here you go. So if it not equal to library end, so it is returning. So whether the search, so I have 1, 2, 3. If you recall, I have 1, 2, 3.

So, search id is true. So, this location will be returned. So, this is not the end. If you recall, this is not the end. Correct.

So, this is a begin. This is the end. I will put b and e. Right. So, now, if you look it not equal to library n true. So it goes here, found ID, it arrow first, right?

it arrow first will print the ID, right? Found ID, IT arrow first. So it will print 2, 2 will be printed. And then you have title, title equal to, right? So that will be printed.

And here you have IT arrow second dot get title, right? The second one. So what is the second one? So here you go. We have effective STL, right?

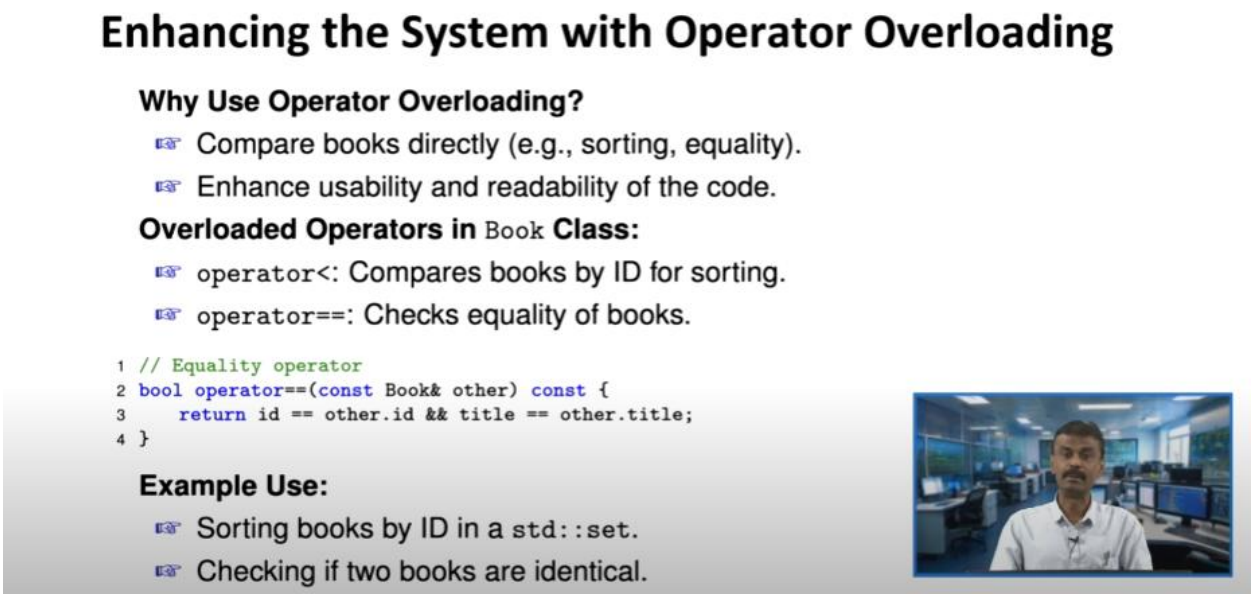
And then your getTitle. So title will be returned. So when it is returning, right, so you will get effective STL, right, so you will get effective STL, right, so STD endl, fine, so it will be new line, if suppose, right, they are matching, right, IT and library end is matching, so after 3, right, so then you will go to the else part,

then you can have book not found, but in this case search ID is 2, so they are not begin or end, All right. So therefore, you are getting the output effective STL. And also, right, we were talking about operator overloading. Right.

So why do we use operator overloading? So here, you have to compare the books. Right. You may have the same copy. Right.

Let us say 10 copies. All are the same book. Right. So whether all these books are equal or the books are in sorted order in some way. Right.

Either with the help of ID or with the help of authors. Right. First author's first name alphabetically. Right. So somehow, it is being sorted, or the equal number of books are kept right in the system.



I mean to say. right. So, in that case this operator overloading is useful to compare books directly right and also this is useful for enhancing the usability and readability of the code right. So, this will be useful the operator overloading right. So, this will be useful for the usability and readability of the code.

So, let us say right consider what are all the operators that you are going to use right or what are all the operators that you are going to overload So, here in this case, let us say less than or double equal to, right. So, less than, it is comparing books by ID for sorting, right. So, you have an ID and then, so whenever I have a book, I may have some numbers 107, 108, 109 and then, so I can compare. So, whichever is sorted, right, whichever is the minimum, I mean less than certain value.

For example, I am searching for 108 books. I got 107. So, then I can move further right. So, for that it will be useful and similarly operator equal to right. So, that is useful for checking the equality of books right.

So, here we use let us say one is less than operator and another one is let us say equal to operator where the two books are equal comparing right. So, operator double equal

to book address of other. So, this is a reference. object other right. So now I am comparing all right.

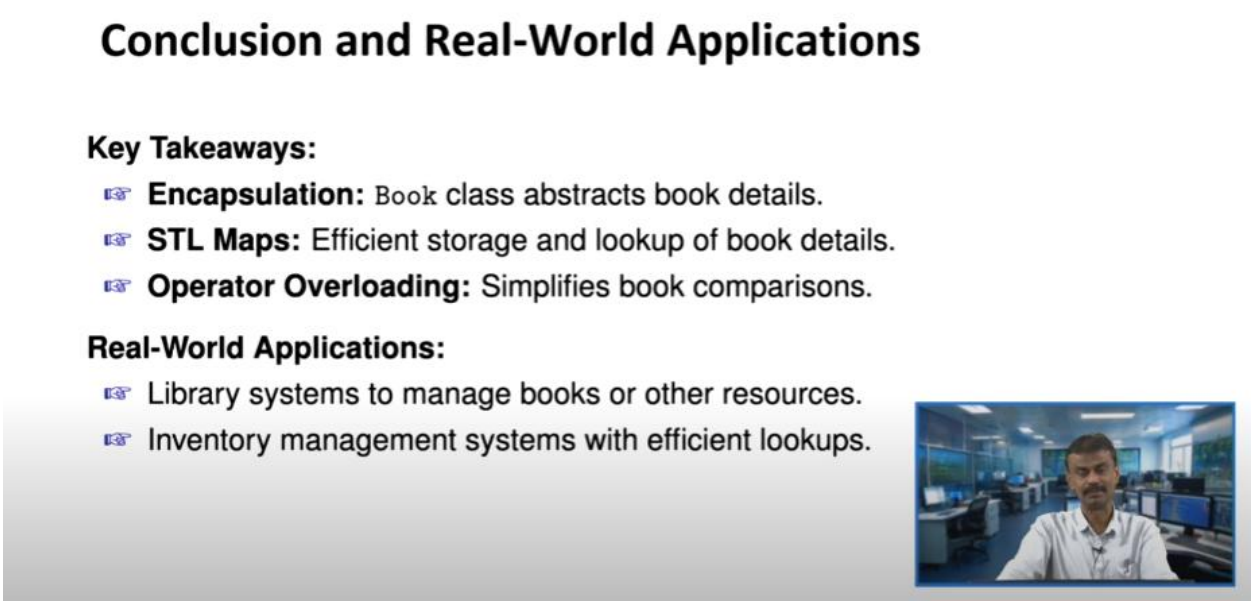
So this will return the Boolean value id equal to other id. Let us say two books you are comparing. Are the id same right. You can get true. Title same true right.

So if they are true return the value one right. So books are of same category or same author right. In fact in this case same title same id right. So it will return one. Otherwise if it is a different book

Naturally, the ID will be different, the title will be different, or assume the title is the same, but the ID is different. Right, or two different authors can have the same title, correct. So, we know the truth table, and based on that, it will return either 1 or 0, right. So, for that, we can use operator overloading, right. So, sorting books by ID, we can also use a set, right.

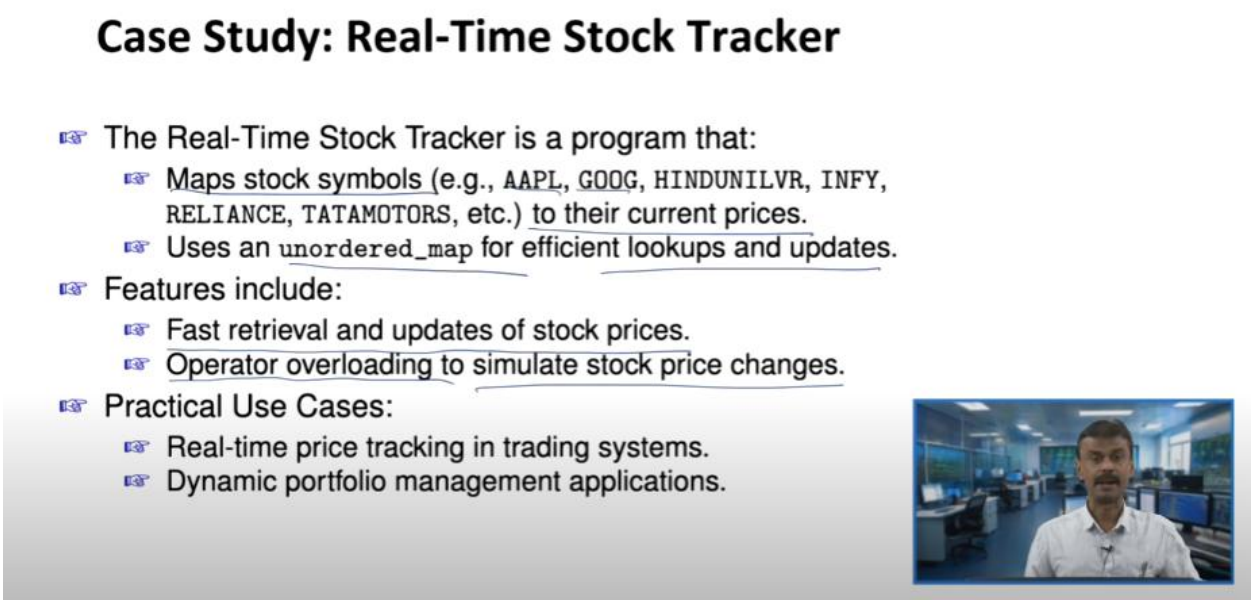
So, checking whether the two books are identical or not. So, we have talked about sets, maps, everything. So, whatever STL we have studied, in the past few classes. So, we can make use of these in this case study, all right.

So, here the key takeaways are, right. So, it is a real-world application. So, encapsulation. So, I initially put the class and then several STL functions, right. So, a particular STL class, right, several libraries of STL.



So, maps have been used. So, which is for efficient storage and lookup of book details, operator overloading, right. So, either less than or double equal to, right. So, for the comparison of books, operator overloading was useful, right. So, here you can have the real-world application.

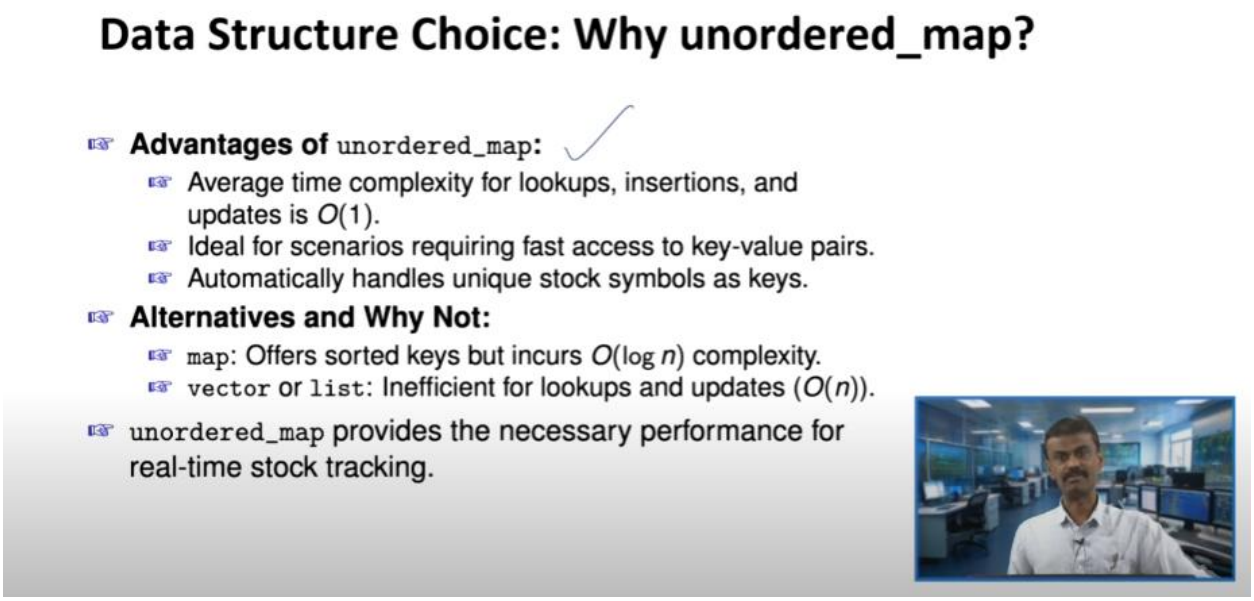
So, when you are using all these concepts, particularly the STL, right? I am focusing on STL because of this chapter, right. The library system in this particular example, so we can see how to manage books, right? Or any other resources, right. And similarly, in this line, you have an inventory management system with efficient lookups, right. So, similar kinds of problems you can think about, right. So, these are all the real-world applications where you can use the Standard Template Libraries, so this is one case study.



So, we can see one more case study: a real-time stock tracker, right. So, a very interesting problem because everyone is into stocks nowadays, correct. So, if I have to have, right, a real-time stock tracker, right. So, for example, you can map stock symbols, right: Apple, Google, Hindustan Unilever, Infosys, Reliance, Tata Motors.

So, right. So, these are all some of the well-known symbols. So, those who are into the stock, I mean, they know that. Right. So, here, assume that they are in the current prices, to the current prices.

So, now, let us use an unordered map. Few lectures back, we had talked about the unordered map. Right. So, we use this for efficient lookups and updates. What exactly the current and then what are the updates?



Next, the features include, right, the fast retrieval and updates of stock prices, right? Also, we use operator overloading like a previous case study, operator overloading to simulate stock price changes, right? So, we use operator overloading. And practical use cases is real-time price tracking in trading systems, right? And apart from this real-time stock, so these are all the practical use cases, real-time price tracking in trading systems, dynamic portfolio management applications, right?

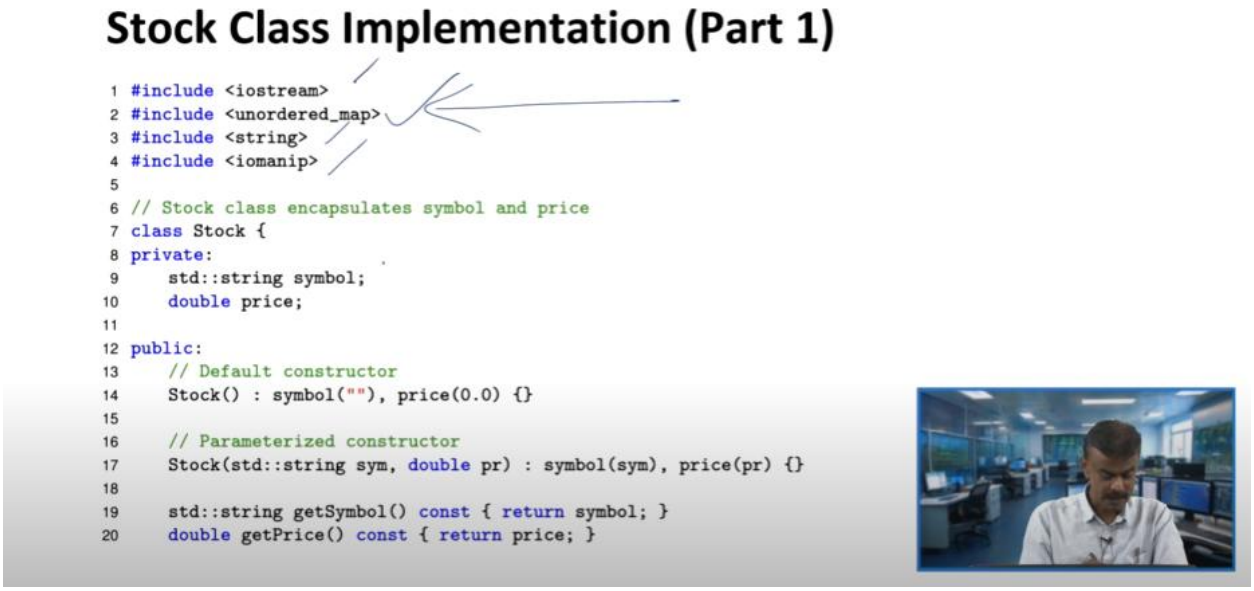
So these are all the practical use cases, right? So now you may ask the question: why have you chosen this unordered map, right? Why have you chosen this particular data structure to solve the problem of the real-time stock tracker, right? So the reason is the advantage of unordered map, right? So suppose I want to have the lookups, right?

So when I have the table, I want to have certain lookups or I want to insert certain data, or I want to do some updates, I can do it in constant time, denoted as O(1), all right? Constant time. If there are n number of stocks, right, and n is a very large number, so in constant time, k, k is very strictly less than n, right? n may be a hundred thousand, and k may be, let us say, four or five steps. You can find, right? So you can make a lookup

or if you want to insert, you can do it in O(1) time. So that is the main advantage of using unordered map, O(1), right? And unordered map is ideal for scenarios, right, where you may require fast access to key-value pairs, right? So you have the pair, key-value pair. I want to have fast access, so already I told it is an O(1) algorithm. So that is the reason we are choosing this unordered map. And it also automatically handles unique stock symbols as keys, right? And alternatives are map.

so you can choose map the complexity is order login right so if you have n number of data So approximately log n steps require if you use map all right or you can use vector or list. So vector or list if you use the complexity is n right. Suppose I want to do for the lookups or insertions or updates. So I have to traverse the complete list right or complete vector right order n. So constant step is better.

So therefore we are choosing unordered map right. So, unordered map. So, which is providing the necessary performance for the real time stock tracking right. So, when I am comparing order 1, order log n and order n right.



So, I will go for this order 1 algorithm that means the constant time I can do right finding lookups or insertions or updation right. So, that is the reason we are going for unordered map. So what are all the components? Stock class, right? So you can have the class, right, which encapsulates stock symbol and price.

And we use operator overloading to simulate price updates, right? We have already defined why we are using an unordered map. So this is mapping stock symbols to stock objects. And also, this enables fast lookups and updates. The main functionality is to add stocks to the tracker, update the prices dynamically, and display the current stock prices.

Right. So these are all the main functionalities. So now we'll quickly go into the program. So here, we have used an unordered_map. Right.

So string_find is an old friend. We know that. And input-output manipulation. So these we have done. Right.

Only the STL part. So, we are using here unordered underscore map. In fact, we have done with the example, right? So, let us say class stock. I want to have the encapsulation.

So, private member data is symbol, symbol under string, right? And the price in double, right? The price in double. So, you have the default constructor. So, in the default constructor, correct stock and initially symbol is initialized to null character, right?

All right. So, you have one two double quotes that means a null character and price is initialized to 0.0, and here you have parameterized constructor. So, under parameterized constructor, you have symbol and pr, all right, symbol and pr. So, that means two parameters you are going to pass, you are having two arguments over here and sym which will be copied into symbol and pr which will be copied into price, right.

Stock Tracker Implementation (Part 2)

```
21      // Operator overloading to update price
22      Stock& operator+=(double change) {
23          price += change;
24          return *this;
25      }
26  };
27
28  // Unordered map for stock tracker
29  std::unordered_map<std::string, Stock> stockTracker;
30
31  void displayStocks() {
32      std::cout << std::setw(10) << "Symbol" << std::setw(10) << "Price"
              << std::endl;
33      for (const auto& pair : stockTracker) {
34          std::cout << std::setw(10) << pair.second.getSymbol()
35                    << std::setw(10) << pair.second.getPrice() << std::endl;
36      }
37  }
```

So, this is about your parameterized constructor, and here you have the getSymbol function, a member function whose return type is string. Right, whose return type is string, and this is returning symbol, the getter function, right. So, the getter function is returning symbol, and you have getPrice, whose return type is double, and this is returning price, right. So, these are all the getter functions that I can access from outside the class.

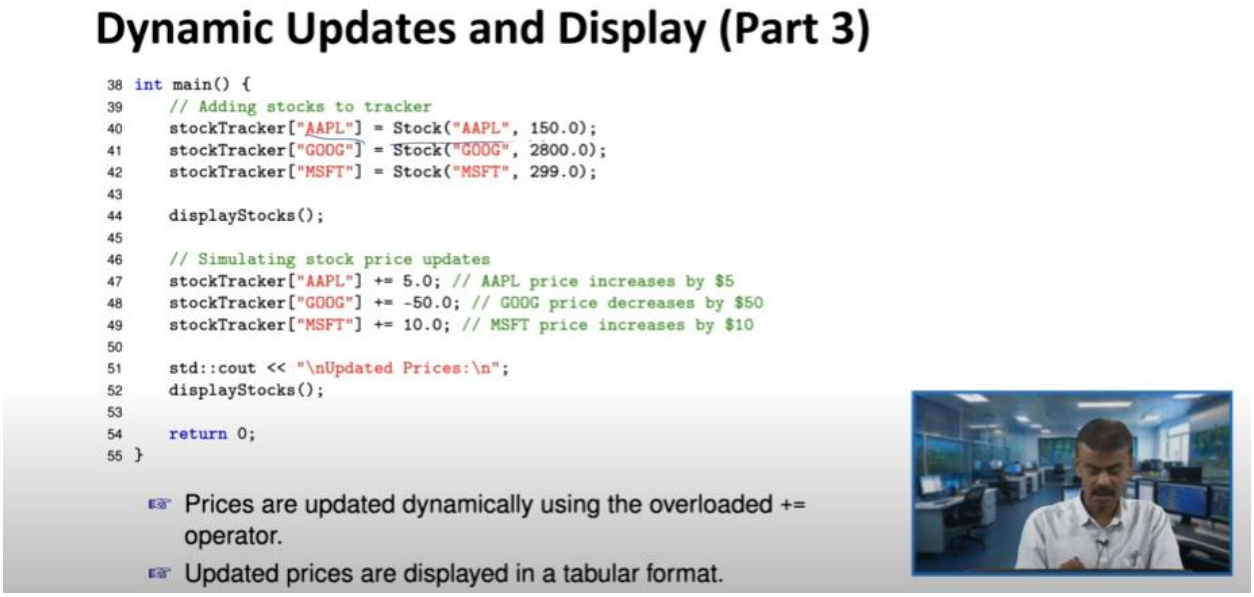
And when I am accessing outside the class, these functions can access the private member data, such as symbol and price. So, now, here we use the operator overloading. So, stock address, right, the reference operator, we use plus equal to, right, operator plus equal to. So, this is going to return the reference variable, or you can say that it will return the pointer, correct. So, operator plus equal to, your parameter is changed.

Right. Your parameter name is change. Right. So, price is equal to price plus change. If there is any change in the price, you are updating.

So this update can be done based on the operator overloading. So price plus equal to change. So price is equal to is equivalent to. Right. Price is equal to price plus change.

Change is double. Price is also double. So we know the meaning now. Right. So plus equal to price is equal to price plus change.

And then return star this. So, we know this is a pointer. Right. And it is returning completely. So, assume that you have changed the price for, let us say, 3, 4 stocks or n stocks.



Right. So now you have an unordered map. Right. Which contains a string and stock, like the book that we had seen in the previous case. Right.

And stockTracker is the object. So, stockTracker is an object under the class unordered_map. So, the unordered_map contains a string and stock. Okay. One string and one stock.

Right. So, stock is the class that we have defined. Therefore, it is expecting one object under stock. Okay. And then you have void displayStocks.

Right. So, that is displaying a simple cout set width w 10. Right. Printing symbol. Again, set width 10, printing price.

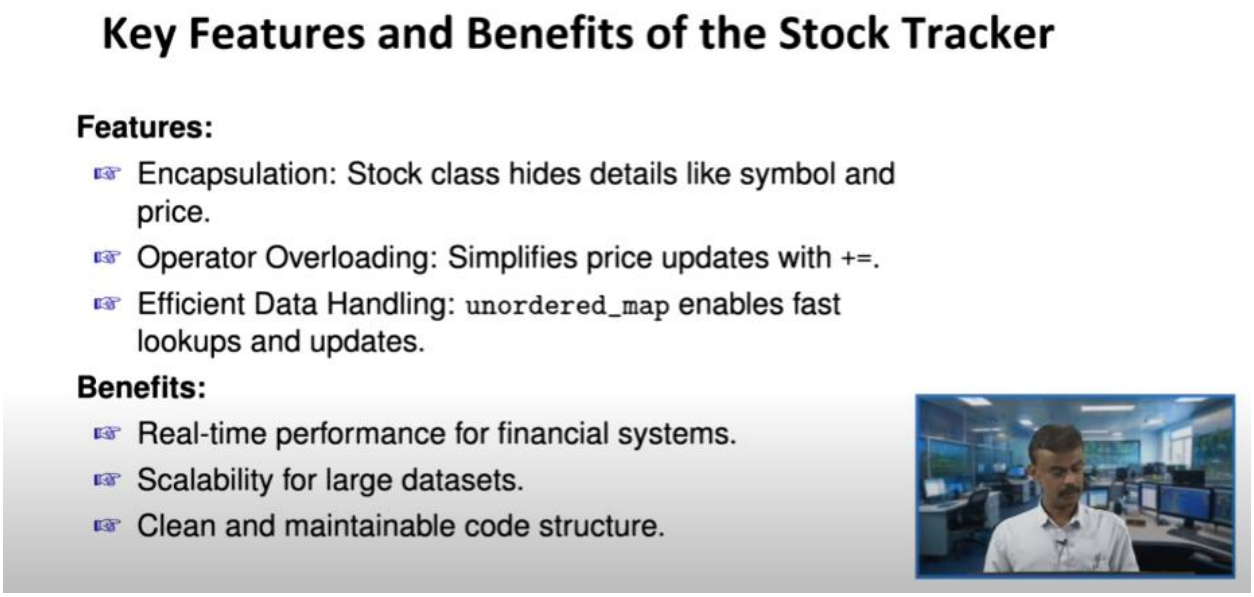
Right. And then, your range-based for loop for constant auto pair. Right. Which is mapping onto stock tracker. That means both the reference variables.

Right. Both base address pair and stock tracker. It is, in fact, the for loop. Right. Range-based for loop.

So the base address of pair and stock tracker both are same now. right and then you are printing pair dot second dot getSymbol right and pair dot second dot getPrice right we will see how we are going to put this symbol and pr right sym and pr i mean to say string and double so here you have here we are printing so we know that like the previous program it is going to call the object pair and the second the second dot symbol

right the second dot getSymbol and here you have getSymbol which is going to return symbol. So, now we will see we are coming to the main program. So, main program you have stock tracker apple right. So, you have an object stock pausing apple and 150.0 right and another one google 2800.0 and msft 299.0 right and then you are updating. Apple you are updating with plus 5.0.

Google minus 50.0. And MSFT plus 10.0. Right. So you are making the update. Right.



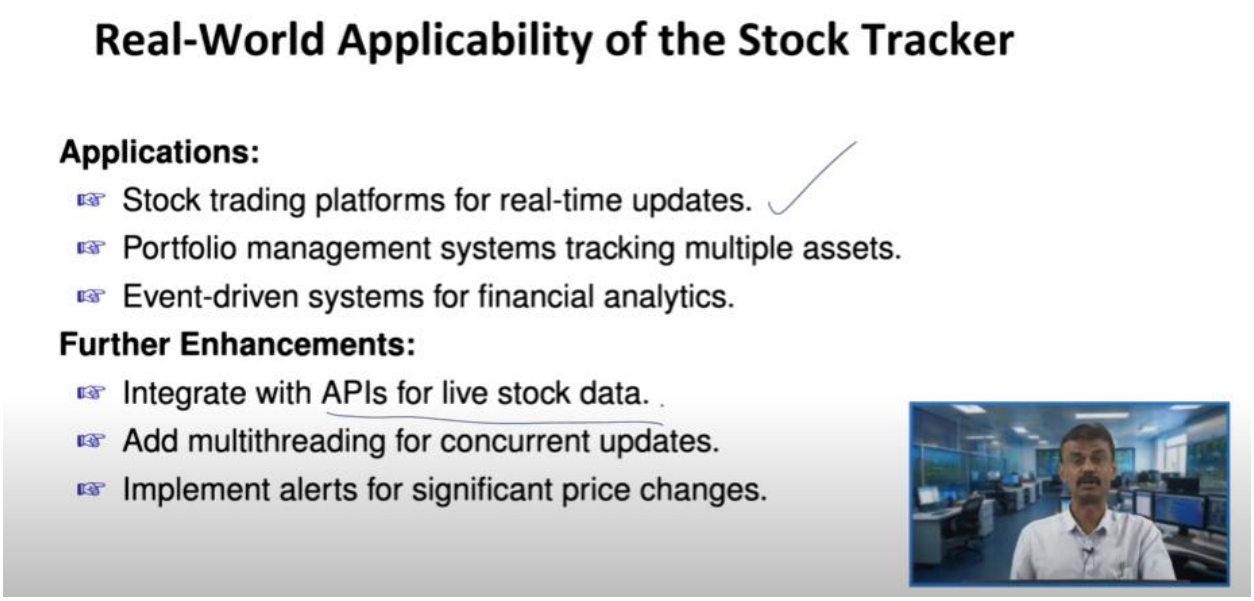
And then the updated price will be displayed when you are calling the display stocks. So, here you go the display stocks that you have written over here. So, everything will be displayed and in fact it is calling the function getSymbol and getPrice. So, when we

run the code you can see the output like this right. So, since we have used the unordered map.

So, you are getting first MSFT and Google and then Apple ok. So, and then there are some updates right. So, here in fact, I did not say MSFT is nothing, but a Microsoft right. So, Microsoft there is a change right 299 plus 10 it becomes 309 right and Google 2750 and Apple right 155 right. So, the prices are getting updated dynamically using the operator overloading plus equal to right

and then you can see the updated prices are displayed as a table form right. So, these are all for example, here STL point of view we have used unordered map right. So, like this you can think of right and you can solve several real world problems right. So, the key features here we use encapsulation, we use operator overloading right

And then we can use the data handling right, the efficient data handling based on the unordered map data structure. So, as we have already seen, for the lookups, updates, and insertions, it is an order-one algorithm. So, that is the reason we have chosen unordered map. So, these are all the important features and the benefits of these STLs, right? So, the real-time performance for these kinds of problems, for example, if you want to deal with financial systems.



Real-time performance for financial systems. Scalability for large databases, right? Or large datasets if you have, all right? How to deal? And clean and maintainable code structure, right?

So, you can have the clean and maintainable code structure. So, if you start using these STLs, right? Along with encapsulation, operator overloading, all right? Make use of the properties of constructors, default or parameterized, right? So, the clean and maintainable code structure will happen when you start using these STLs, all right.

So, these are all the other applications. So, stock trading platforms for real-time updates, portfolio management systems tracking multiple assets, event-driven systems for financial analytics. So, these are all the similar applications you can try to solve the problem with the help of, right? So, whatever we have discussed, right, regarding the stock. And further enhancements you can do.

So, we can integrate this with an API for live stock data, all right. So, that we can think of, and we are going to talk about multithreading and concurrency, right. So, here you can have the further enhancement with multi threading for concurrent updates and also all right you can have the implement alerts for significant price changes right. Price changes, right? So, we have seen some examples, right? Either it is going down or up, so we need the alerts. So, that also you can think of. So, with this, I am concluding this particular chapter. Thank you very much.