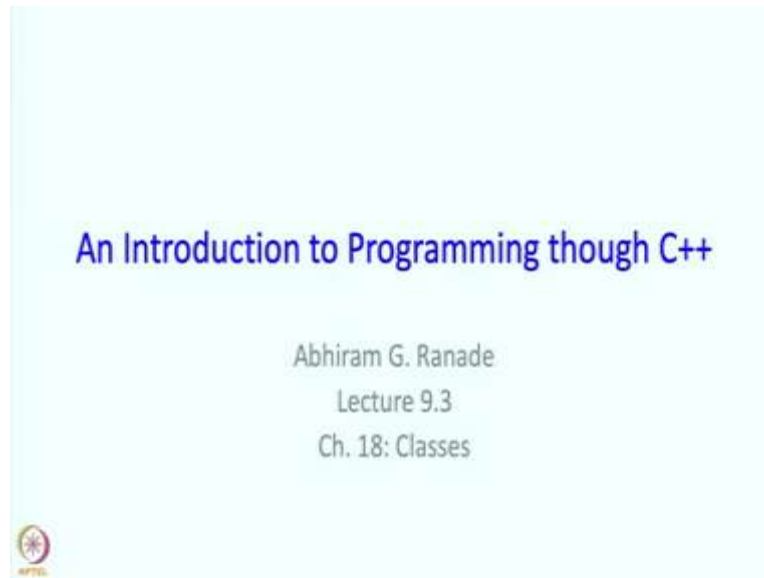


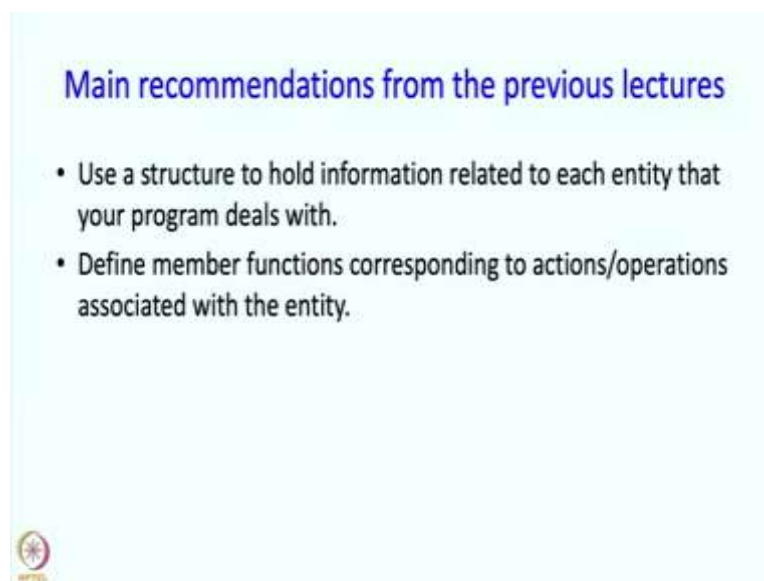
An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 21 Part- 1
Classes
Introduction

(Refer Slide Time: 0:26)



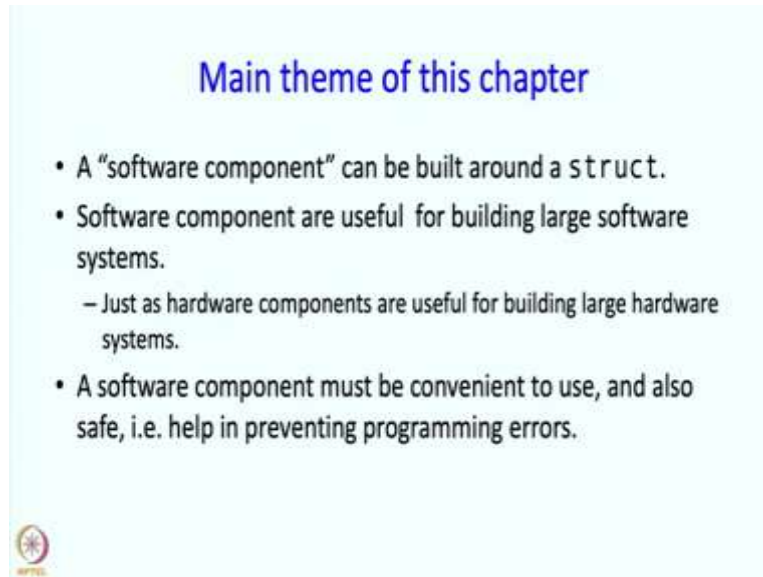
Hello and welcome to the NPTEL course on An introduction to programming through C++. I am Abhiram Ranade and today's lecture is on classes and the reading material for this is chapter 18 of the textbook.

(Refer Slide Time: 0:33)




So, here are the main recommendations from the previous lectures. First, use a struct to hold information related to each entity that your program deals with. Then define member functions corresponding to actions or operations associated with the entity.

(Refer Slide Time: 0:52)



Main theme of this chapter

- A “software component” can be built around a struct.
- Software components are useful for building large software systems.
 - Just as hardware components are useful for building large hardware systems.
- A software component must be convenient to use, and also safe, i.e. help in preventing programming errors.



The main theme of this chapter is to take this idea further in the following sense. So, the goal is to build something like a software component around a structure. Now, software components are useful for building large software systems just as hardware components are useful for building large hardware systems. And a software component must be convenient to use and also safe exactly in the sense that hardware components are convenient and safe. So, for example, they should software component should help prevent programming errors.

(Refer Slide Time: 1:42)



“Packaged Software components”

- Things that you buy from the market are packaged, and made safe to use.
 - Fridge, television : no danger of getting an electric shock.
 - A “control panel” is provided on the device. A user does not have to change capacitor values to change the channel on a television.
- Analogous idea for software:
 - Make functionality associated with a struct available to the user only through member functions (“control panel”)
 - Do not allow the user to directly access the data members inside a struct. (Just as a user cannot touch the circuitry) **The user does not need to know what goes on inside.**
- If you build a better fridge, keep control panel same as the previous model, the user does not need to relearn how to use the new fridge.
 - If you build a better version of the struct, but keep the member function signatures the same, the programs that use the struct need not change.



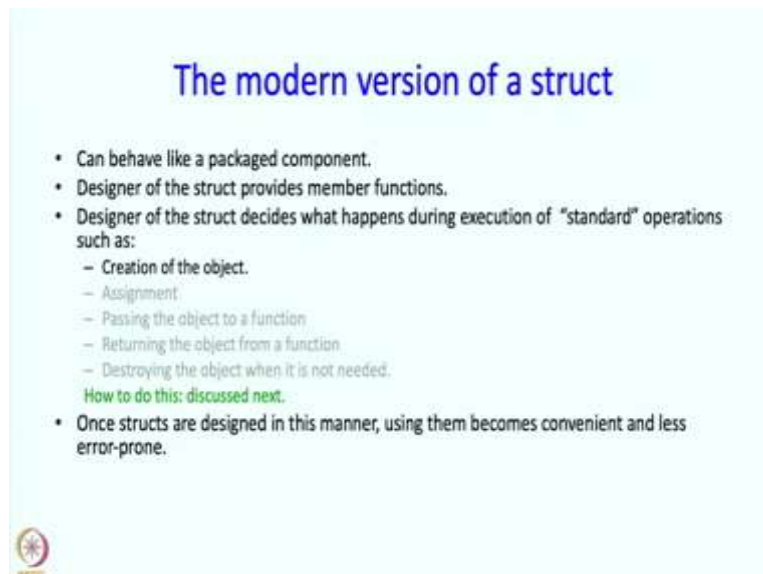
So, we have been talking about this theme for some time now. So, let us just clarify this a little bit more. So things that you buy from the market are packaged and made safe to use. Fridge, television, are some of these things. So, for example, there is no danger of getting an electrical shock. A control panel is provided on the device. A user does not have to change the capacitor values somewhere to change the channels on a television.

The analogous idea for software is that the make the functionality associated with a struct available to the user only through member functions. So, this is kind of analogous of the control panel and do not allow the user to directly access the data members inside a struct, just as a user is not allowed to touch the circuitry inside a television.

And in some sense, the user has no business looking inside right that is the attitude that is that that is the view that is taken. Or the user does not want to know what is going on inside just to keep keep his or her headache low. And I mean keep separation of concerns. The user has lots of things other things to worry about.

So another idea is that if you build a better fridge, you often keep the control panel the same as the previous model. The user does not need to relearn how to use the new fridge. If you build a better version of the struct, but if you keep the member functions signatures the same, then the program that uses the struct does not have to change. So these are the kinds of things that member functions and this kind of discipline will allow us to accomplish.

(Refer Slide Time: 4:01)




The modern version of a struct

- Can behave like a packaged component.
- Designer of the struct provides member functions.
- Designer of the struct decides what happens during execution of "standard" operations such as:
 - Creation of the object.
 - Assignment.
 - Passing the object to a function
 - Returning the object from a function
 - Destroying the object when it is not needed.

How to do this: discussed next.

- Once structs are designed in this manner, using them becomes convenient and less error-prone.



So, the modern version of a struct can behave like a packaged component and the designer of the struct provides the member functions. Actually the designer of the struct can do lots of

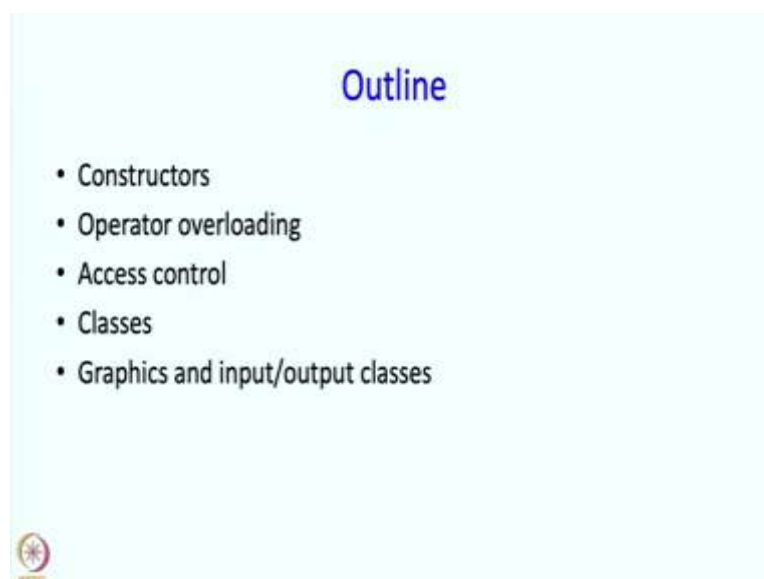
things. So, for example, if I have an ordinary variable, lots of things sort of there are lots of standard operations that can be performed on variables. To begin with, and we have to create that variable so there is the creation of the object. Then I may want to assign, I may want to pass the object to a function and we want to pass the struct or object or whatever, back from a function I want to return it as a result.

And we know that when control leaves a scope, the variables which have been created inside in the activation frame have to be destroyed. So, the modern version of a struct allows you to customize these operations however you want. And that gives the designer a lot of freedom and the freedom can be used to do quite some imaginative things.

Now, in this course, we are not going to talk about all these things at least not immediately. We are going to talk about just the creation. In today's lecture we are going to say, how the designer can decide what happens during creation? And the idea is that once structs are designed in this manner using them becomes convenient and less error-prone.

So, we have been, we have been talking about all these things for some time now. But in some sense, they are going to come to fruition in this lecture. And structs which are endowed with such features are also called objects. So, have been using the term objects but the technical meaning is an object is something is a struct or it is a variable created from a struct but which can have all such features.

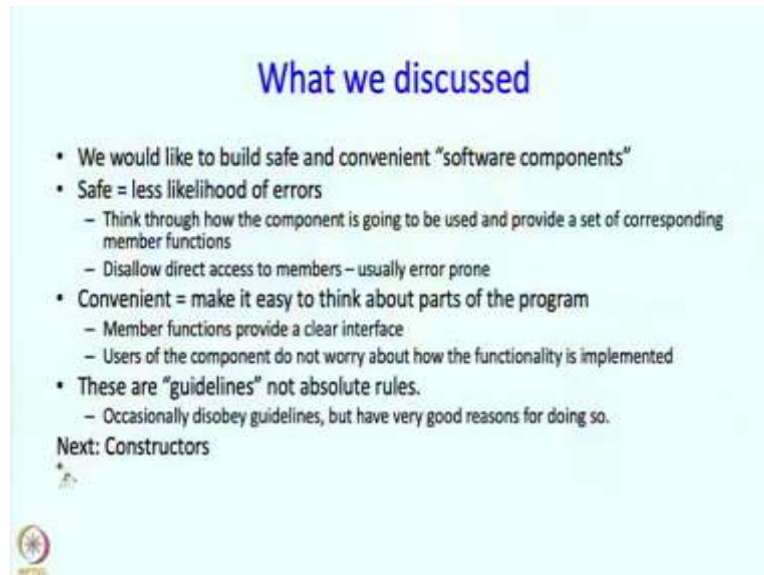
(Refer Slide Time: 6:13)



Alright, so here is an outline of this lecture I am going to talk about something called constructors. I am going to talk about operator overloading, I am going to talk about access

control, then I am going to talk about classes, and then I am going to talk about some special classes for graphics and input and output. So, what have we discussed so far in this segment?

(Refer Slide Time: 6:40)



We have said that you like to build safe and convenient software components, safe means less likelihood of errors. Basically, this means think through how the component is going to be used and provide a set of corresponding member functions. And we will see that we can even disallow direct access to members and the idea behind this is that usually such access is likely to be error-prone.

We want our objects to be convenient in the following sense that they make it easy to think about parts of the program. So, member functions should provide a very clear interface and the users of the components should not worry about how the functionality is implemented. So, they do not, the component should be designed so that they do not need to look inside. So, there should be very very nicely designed control panels so to say. And I should say that these are guidelines and this is how we would like software to redesign. But sometimes it does not work that way and that is okay, once in a while, provided you have good reasons. Alright, so next we will discuss constructors but before that let us take a short break.