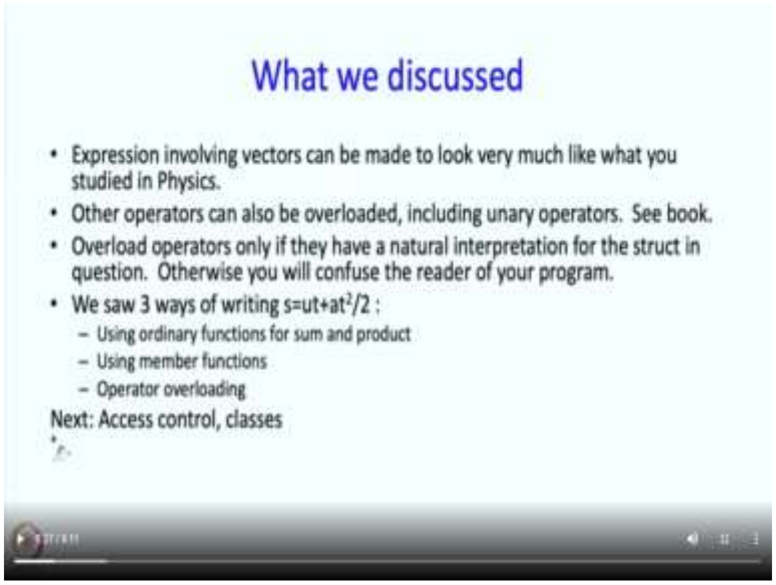
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture No 21 Part 4.
Classes
Access control.

(Refer Slide Time: 00:19)



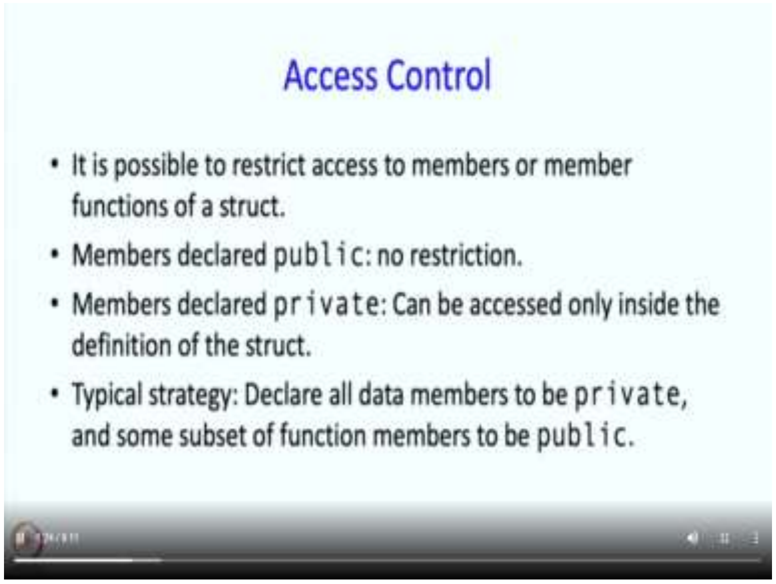
What we discussed

- Expression involving vectors can be made to look very much like what you studied in Physics.
- Other operators can also be overloaded, including unary operators. See book.
- Overload operators only if they have a natural interpretation for the struct in question. Otherwise you will confuse the reader of your program.
- We saw 3 ways of writing $s=ut+at^2/2$:
  - Using ordinary functions for sum and product
  - Using member functions
  - Operator overloading

Next: Access control, classes

Welcome back. In the last segment we discussed operator overloading. In this segment we are going to discuss access control and the notion of classes.

(Refer Slide Time: 00:29)



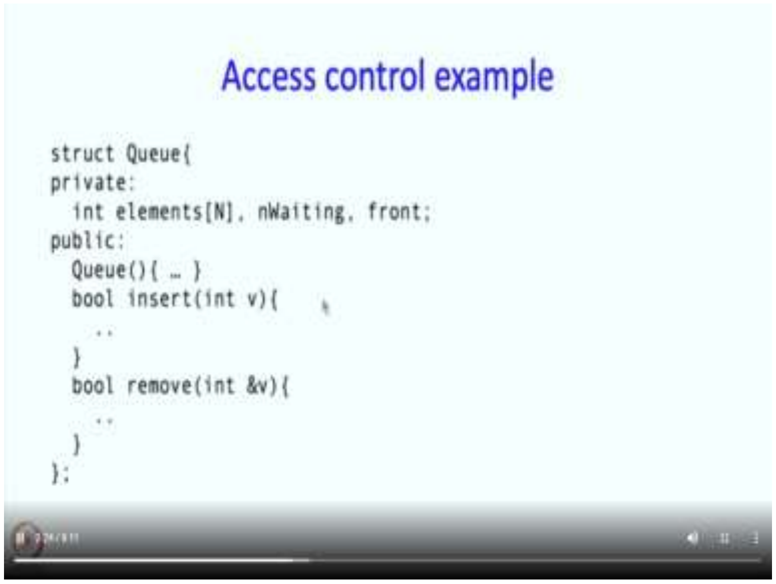
Access Control

- It is possible to restrict access to members or member functions of a struct.
- Members declared `public`: no restriction.
- Members declared `private`: Can be accessed only inside the definition of the struct.
- Typical strategy: Declare all data members to be `private`, and some subset of function members to be `public`.

So it is possible to restrict access to members or member functions of a struct. So there is a way by which we can declare a member to be public which means that there is no restriction. We can declare members to be private which means that they can be accessed only inside the definition of the struct and the same thing for member functions.

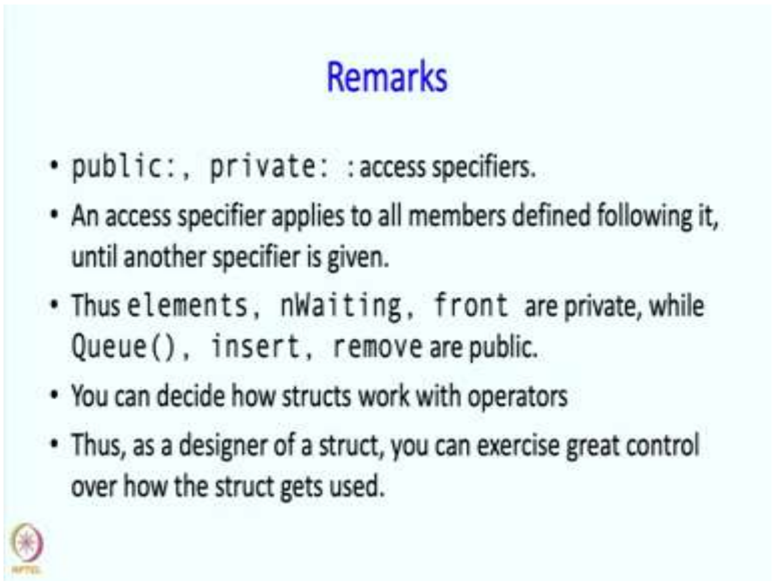
So the typical strategy when we are designing these software components is to declare all data members to be private so that code which is outside of the definition that the structure definition cannot look at the data members and some subset of the functions should be public. So that is sort of like the control panel. So whatever we want the external users to do, we are going to make them public. So here is a simple example.

(Refer Slide Time: 01:52)



Access control example

```
struct Queue{
private:
  int elements[N], nWaiting, front;
public:
  Queue(){ … }
  bool insert(int v){
    ..
  }
  bool remove(int &v){
    ..
  }
};
```



Remarks

- `public:`, `private:` : access specifiers.
- An access specifier applies to all members defined following it, until another specifier is given.
- Thus `elements`, `nWaiting`, `front` are private, while `Queue()`, `insert`, `remove` are public.
- You can decide how structs work with operators
- Thus, as a designer of a struct, you can exercise great control over how the struct gets used.

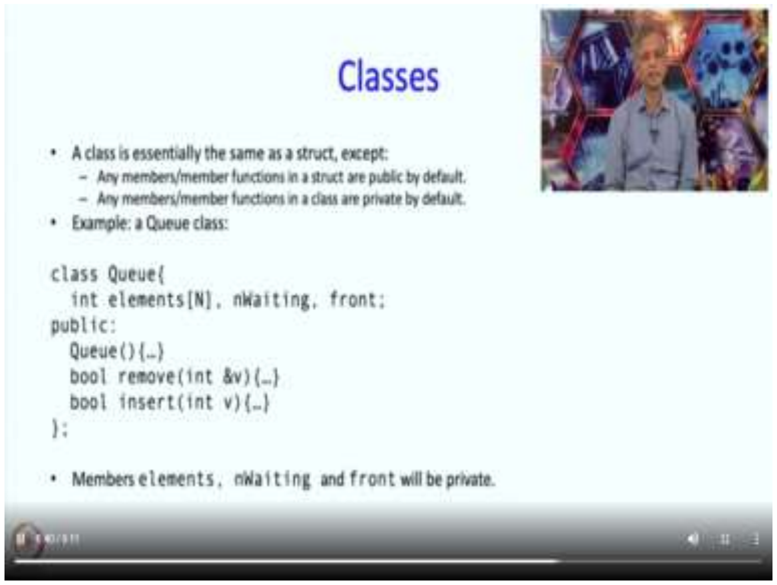
We have this struct Queue and what we are saying over here is these are private. So to say that to declare these to be private, we just have to put private colon before. So this is the so called access specifier and then the rest of these are public or the constructor and the two member functions are public. So we have put this access specifier public colon before it that is it.

So if you have such a Queue then from your main program you would not be able to say Queue.elements. The compiler will say that look you do not so this is a private member or under for you cannot access it. So you would be forced to use just these member functions.

So, public and private are called access specifiers and they are written with that colon afterwards and the idea is simple and access specifier applies to all members defined following it, members and member functions until another specifier is given. So in the previous case, elements nwaiting and front are private. As we said and Queue the constructor insert and remove are public. So these appear after the private access specifier, these appear after the public access specifier. So basically, you can decide how the struct works. I should say with all these using these access specifiers and also operators. I mean if you want an operator to be accessible, then you should make it public. So operator plus or operator star should also be made public if you want external code to be able to access that.

So basically as a designer of a struct, you can exercise great control over how the struct gets used and this might you might think of this as sort of creating barriers for yourself. So in a sense that is true, but as I discussed some time ago, you want to be able to claim that look my code is not touching the members even by mistake. And however, why am I so sure of it? Because the members are private and therefore the data members are private and therefore I know that if somewhere I try to access a data member then the compiler will complain. So that is a good reason for making things keeping things public and private with a well thought out strategy.

(Refer Slide Time: 05:06)



Classes

- A class is essentially the same as a struct, except:
  - Any members/member functions in a struct are public by default.
  - Any members/member functions in a class are private by default.
- Example: a Queue class:

```
class Queue{
  int elements[N], nWaiting, front;
public:
  Queue(){…}
  bool remove(int &v){…}
  bool insert(int v){…}
};
```

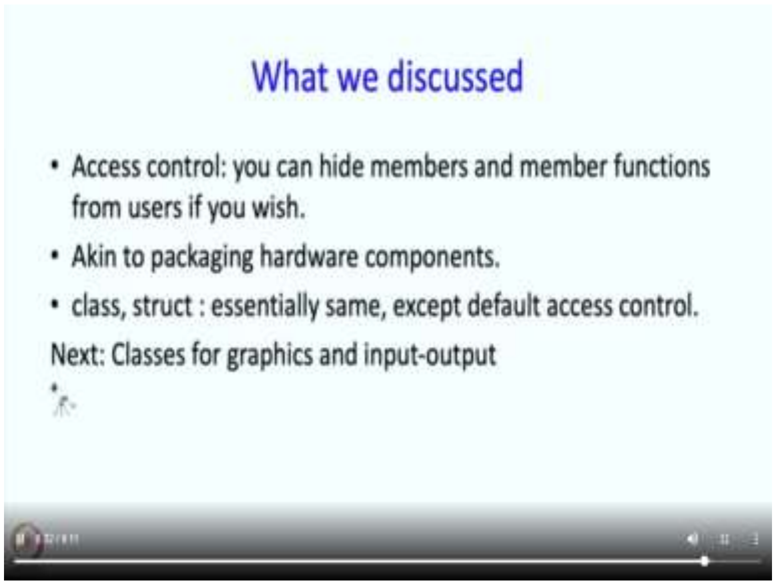
- Members `elements`, `nWaiting` and `front` will be private.

So this brings us to a term classes, a class which you might have heard quite a bit and especially in the context of languages like C++ or Java, and also other modern languages, so what is a class? A class in C++ is really essentially the same as a struct, except for how the defaults work. In a struct, everything is public by default. But of course, you can change it, you can put the access control access control the specifiers. And in a class if you define something as a class then everything is private.

So we will write a Queue class here so if I write this Queue, and I have said nothing over here, I have not said private but because this is a class if no specifier has been given then that is treated as private. So that is why these things would be private. And since there is a public specifier, these things would be public that is it. So that is really the difference between structs and classes and classes are a newer term in the sense that classes are associated with all this philosophy about how to write code.

About making member functions, and making things which are public and private. But it really is the same thing as a class which has been endowed with these member functions. So because C++ has the legacy of C and C has the struct which looks very much like a class, the designers of C++ said that look yes, they are more or less the same but we will keep this minor difference because in C, everything is public and therefore the term struct which is coming from C also has that idea that everything is public except if you change it explicitly.

(Refer Slide Time: 07:50)



What we discussed

- Access control: you can hide members and member functions from users if you wish.
- Akin to packaging hardware components.
- class, struct : essentially same, except default access control.

Next: Classes for graphics and input-output

Alright, so what have we discussed? We have discussed access control and this allows you to hide members and member functions from users if you wish and this is packaging, this is like packaging hardware components. So you do not want the people to see the wiring inside. It is not safe, you may get a shock or something like that or in the case of software, those are parts of the implementation and you are and you may not know enough to handle those parts. Then we said that classes and structs are essentially the same except for the defaults. Next, we are going to talk about classes for graphics and input output, but before that, let us take a short break.