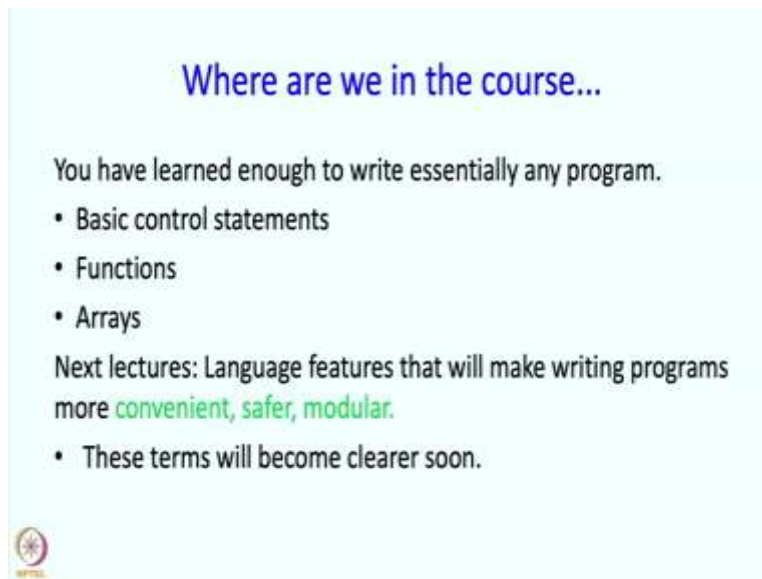


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture No. 19 Part – 1
Structures
Definition and instantiation

Hello and welcome to the NPTEL course on an introduction to programming through C++. I am Abhiram Ranade and in today's lecture sequence I am going to be talking about structures. So, let us take a look at where we are in the course.

(Refer Slide Time: 0:32)




Where are we in the course...

You have learned enough to write essentially any program.

- Basic control statements
- Functions
- Arrays

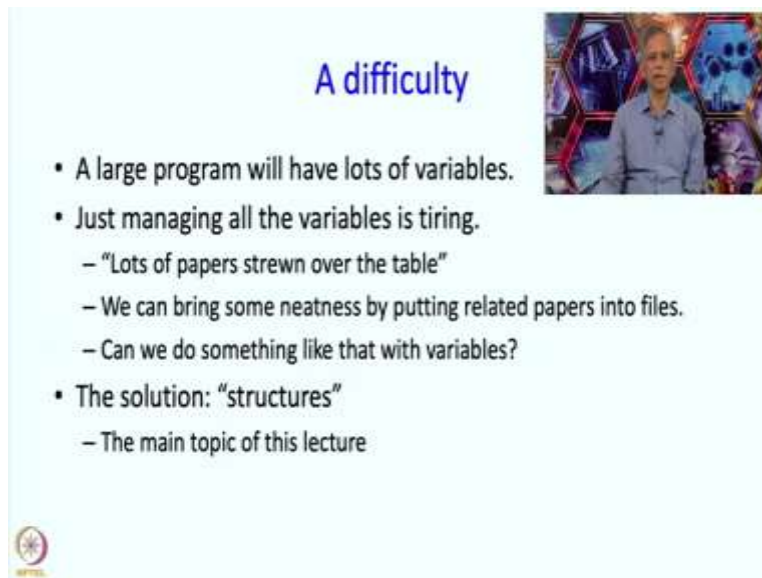
Next lectures: Language features that will make writing programs more **convenient, safer, modular.**

- These terms will become clearer soon.




So, you have really learned enough to write essentially any program, so you have learned basic control statements, you have learned functions, you have learned arrays and in this lecture, we are going to learn language features that will make writing programs more convenient, safer and modular. These terms may not mean anything to you right now, but they will become clearer as we proceed.

(Refer Slide Time: 1:04)



A difficulty

- A large program will have lots of variables.
- Just managing all the variables is tiring.
 - “Lots of papers strewn over the table”
 - We can bring some neatness by putting related papers into files.
 - Can we do something like that with variables?
- The solution: “structures”
 - The main topic of this lecture



So, let me begin with a difficulty. So, if you are writing a large program then it will have lots of variables. And just managing all the variables is often a bit tiring. It is like you have lots of papers strewn over the table and you do not know how many papers you have, which is what you called, which variable, all such things. And on a table we can bring some neatness by putting related papers into files.


So, that way we have groups and we know what we can, we can sort of roughly know what is in which group and that helps us figure out what all we have. So, an interesting question is could we do something like that with variables? And that, in fact, is the solution that structures gives us and that is the main topic of this lecture.

(Refer Slide Time: 2:06)

Structures – high level idea

Most entities we deal with in programming have lots of attributes.

- If our program is about simulating movement of stars
 - Each star has a position, velocity, mass, ...
- If our program is about managing books in a library
 - Each book has author, library number, who has borrowed it, ...
- **Key idea:** collect together all information about an entity into a group/supervariable = structure




So, here are the high level ideas. Most entities we deal with in programming have lots of attributes. If our program is about simulating movement of stars, then each star has a position, velocity, mass. If our program is about managing books in a library, then each book has an author, library number, who has borrowed it, of course, the title, and the key idea in structures is to collect together all information about an entity into a group or you might even think of it as a super variable and this super variable is called a structure.

(Refer Slide Time: 2:56)

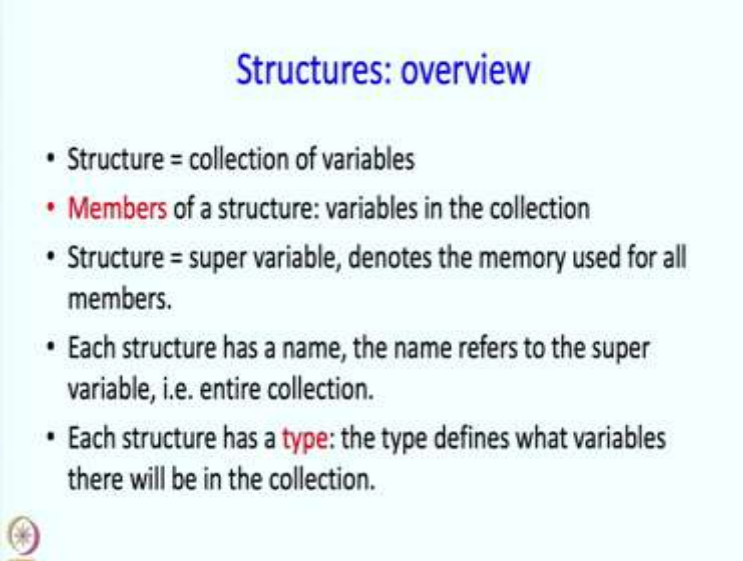
Outline for this lecture

- “Structure”
 - Basic facility provided in C++ to conveniently gather together information associated with an entity.
 - Inherited from the C language
- Operations on structures
- Examples.




So, here is the outline for this lecture. So, we are going to start by talking about structures. So, we will talk about the basic facilities that C++ provides to define structures, and by the way the structures are inherited from the C language. We will also talk about operations that can be performed on structures, and also lots of examples.

(Refer Slide Time: 3:30)



Structures: overview

- Structure = collection of variables
- **Members** of a structure: variables in the collection
- Structure = super variable, denotes the memory used for all members.
- Each structure has a name, the name refers to the super variable, i.e. entire collection.
- Each structure has a **type**: the type defines what variables there will be in the collection.




So, a very high level overview, a structure is nothing but a collection of variables and the term member is also used. The members of a structure are the variables in the collection. So, a structure can be thought of as a super variable and it denotes all the memory used for all the members. Each structure has a name. The name refers to the super variable or in other words the name refers to the region of memory in which all the members are present. So, it refers to an entire collection. Each structure also has a type and that type defines what variables there will be in the collection.

(Refer Slide Time: 4:14)

Structure types

- You can define a structure type for each type of entity that you want to represent on the computer.
 - “Programmer defined type”
- Example: To represent books, you can define a Book structure type.
- When you define a structure type, you must say what variables each structure of that type will contain.
- Example: In a structure to represent books, you may wish to have variables to store the name of the book, its price, ...



So, let us talk about structure types first. You can define a structure type for each type of entity that you want to represent on the computer. And these types are what might be called programmer defined types. For example, to represent books, you can define a Book structure type. When you define a structure type, you must say what variables each structure of that type will contain. So, continuing the example, if you define a structure or structure type to represent books, you may wish to have variables to store the name of the book, its price and other attributes.


(Refer Slide Time: 4:58)

Defining a structure type

- General form

```
struct structure-type{
    member1-type member1-name;
    member2-type member2-name;
    ...
}; // Don't forget the semicolon!
```
- Example

```
struct Book{
    char title[50];
    double price;
};
```
- A structure-type is a **user-defined data type**, just as `int`, `char`, `double` are primitive data types.
- Structure-type and member names can be any identifiers.



So, here is how you define a structure type. The general form is the keyword struct followed by the name of the structure type that you are trying, that you are defining. Then there is the, it consists of several variables or members. So, for each member you first have to specify the type, the type of that member and then you have to give it a name and that is it. So, it ends with a brace and usually braces do not have semicolons as we have seen them so far, but here there is a semicolon, and here is an example of a specific structure type that we might define consistent with this general form.


So, here is a type for storing data about books. So, we are going to call that structure type Book capital B and then it will have members, 1 member is a title which is a character array. So, members can be arrays or members can be just plain variables, and then there is another member which is price and it is of type double. You could have more, but let us say we just have these 2. So, as I said earlier a, a structure type is user defined data type and it is really similar to int, char, double which are the primitive data types and the structure type and member names can be any identifiers.

(Refer Slide Time: 6:38)

Creating structures of a type defined earlier

- To create a structure of structure type Book, just write:
Book p, q;
- This creates two structures: p, q of type Book.
- Each created structure has all members defined in structure type definition.
- Member x of structure y can be accessed by writing y . x

```
p.price = 399;  
// stores 399 into p.price.  
cout << p.title;  
// prints the name of the book p
```



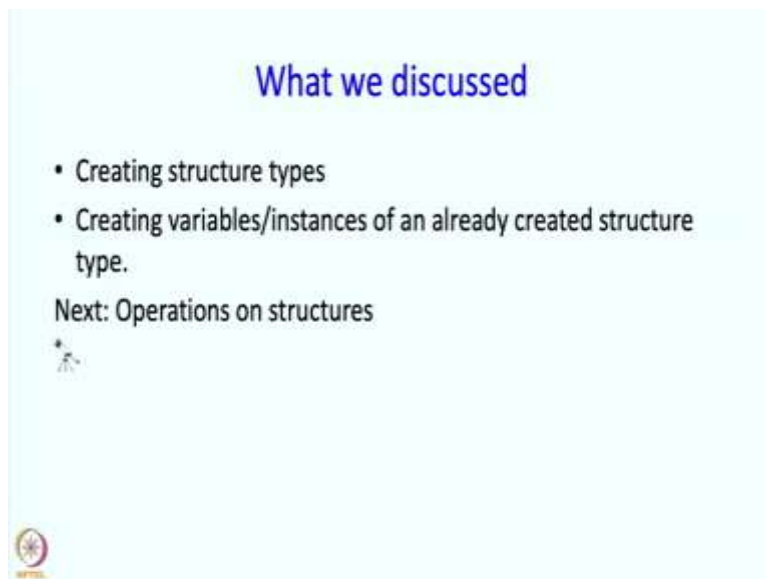
So, how do you create a structure of a type that you have defined earlier? So, we just defined a structure type book. So, if you want to create a structure of that type, we just simply write Book p, q. So, this creates two structures p, q of type Book. Each created structure has all members defined in the structure type definition. So, at this point we have actually allocated memory.

When we defined a structure type we did not allocate memory, we just said that look here is sort of a format using which you can create structures and structures actually cause variables to be created and memory to be allocated.

Now, member x of structure y can be accessed by writing 'y.x'. So, since we have a structure p and on the last slide we said that structure type book has members title and price. We can get to the price member of p by writing p.price and this statement is storing 399 into that member. Now, we can print it if you wish.

So, if you write cout<<p.title or cout<<p.price, whatever the contents of these members or these member variables are, will be printed. So, c a s o p dot title is presumably the name of the book represented by the structure p and so that will get printed. Of course, we will have to assign it first and so once it is assigned we can print it out in this manner.

(Refer Slide Time: 8:29)



Alright, so what have you discussed? We have discussed how to create structure types and then we discussed creating variables and instances of an already created structure type. Next, we are going to discuss operations on structures, but let us take a quick break.