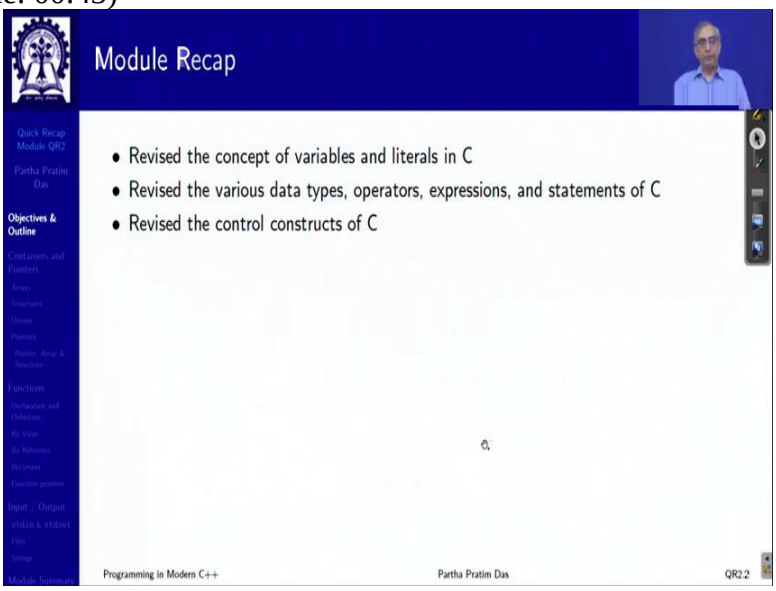
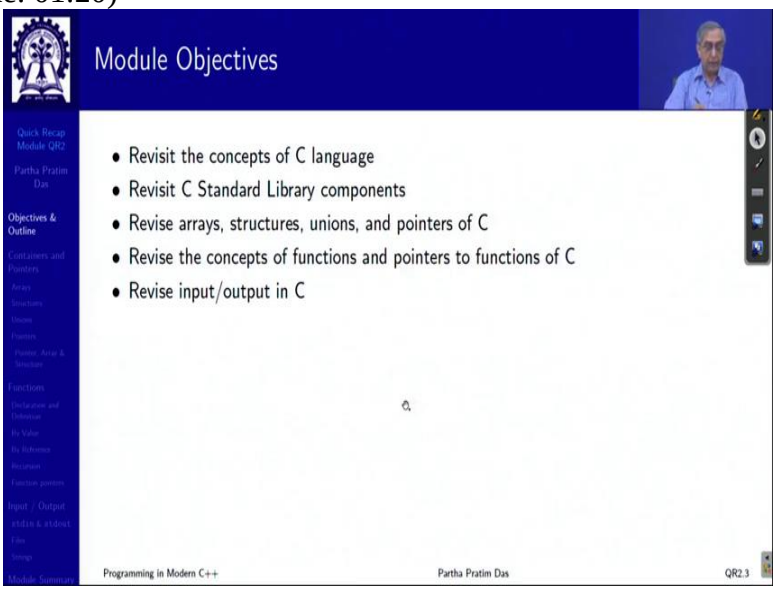
Programming in Modern C++
Professor Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 03
Recap of C/2

(Refer Slide Time: 00:43)



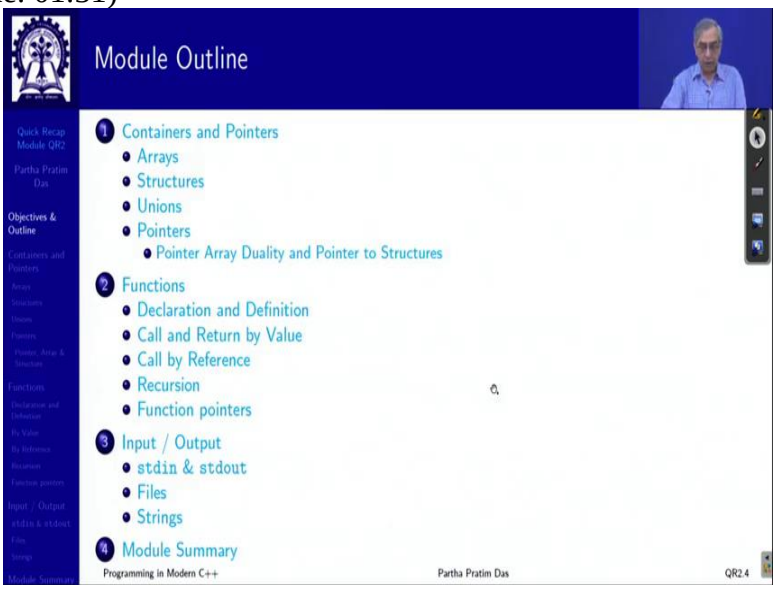
Welcome to Programming in Modern C++, we have discussed a quick recap module, module 1 earlier taking around through the different aspects of C language, which should be good for you to, really check on your knowledge level. So, that you will be able to take best when the course actually starts, because we will make use of C very heavily in discussing C++ and modern C++ subsequently. So, in the first QR1 module, we talked about concepts of variables literals, data types, operators, expression statements and so on, particularly the control constructs as well.

(Refer Slide Time: 01:20)



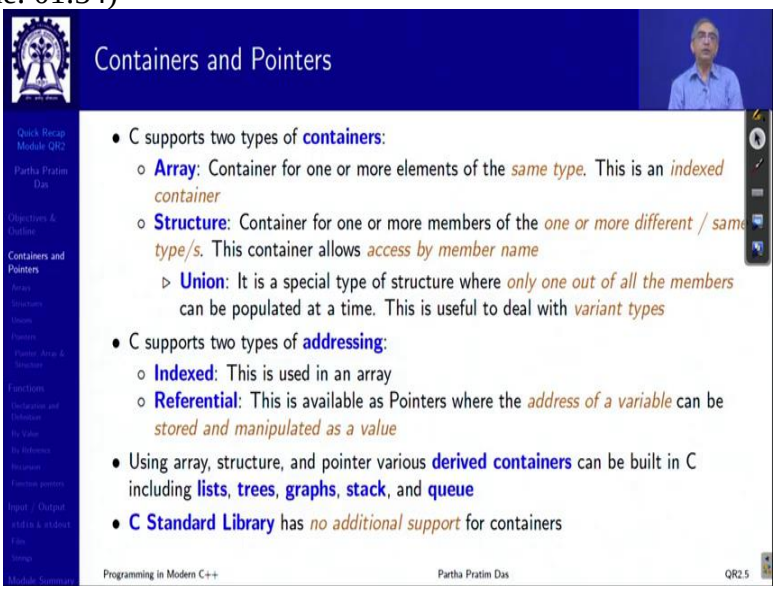
So, I will take you a quick round of the remaining part of the language. And a little bit of the standard library as well.

(Refer Slide Time: 01:31)



So, this is this is the outline.

(Refer Slide Time: 01:34)



So, after the variables which can store a single value, we all realize that we need certain collections, which often referred to as data structure that we need containers, which can keep a collection of different values together. Now, if those values are of the same type, we typically put it in a container array. If there are of one or more different types or same types also, we can put them into a container called structure.

Array is basically known for its indexed property. That is in array, I can have any number of elements and every element is accessed by the index of that element or the position of that

element in the array. In a structure, the elements are accessed by their name. So, this is another difference that we must always keep in mind, then, we call structure a struct in, in C as well as C++.

Then there is a special type of structure called union, where only one out of all the members can be populated at a given point of time. If there are three members, one is of type int, one is of type double, one is of type care at any point of time either it can have only the int value or only the double value or only the care value, I cannot have more than one. So, this was provided in C to deal with variant type.

I mean, if I, if I want to define something which I do not know, what type it will be, it could be integer, it could be float, it could be character or anything else, then how do I specify that in the language, so, that is why this is called a union of this. These are, these are C is meant for low level programming also as you know. In a lot of systems programming, like huge part of Linux is written in C.

So, in that you need to do a lot of these for example, you are listening to a port and a data comes, you do not know what kind of data packet will come there could be you know three, four, five different types of data packets. So, should you be listening to an integer, should you be listening to a double, you do not know. So, once the packet comes, then you have to put it to a proper container.

So, you cannot have five different types of containers. So, by union you are kind of giving you a mechanism, where you can put it if it is any one of the types which you have unioned. So, besides the containers you need to do addressing, you need to, get the address of different variables, different locations to be able to use them. So, C supports two types of addressing. One is indexed, which is used in the array.

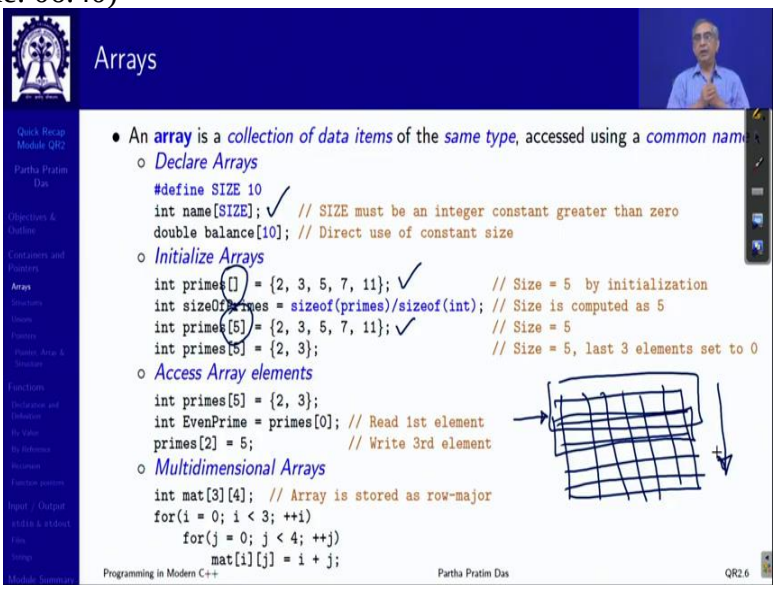
And other is referential, which is known as pointer. That is address of a variable can be stored and manipulated as a value. So, this is something which is, which is a very strong area of C which is a very risky area of C. I mean C is in the news always because of its referential feature, but you must be very thorough in terms of using, defining, manipulating the pointers because they really are the strength as well as the often are the main source of pain, for a programmer.

So, using the array structure and pointers various derived containers can be built. For example, I can build a stack, I can build a list, singly linked lists, doubly linked list, I can

build trees, binary trees, queue, graph, all sorts of things. And C standard library has no additional support for containers. This, that is it does not give you any additional container like there is no C standard library header which has a stack available.

There are, there are solid reasons of why C cannot, but it is not there. And this is I mean, I highlight this point as a reminder to you because with that, you will, you will see that in C++, we will have lots of support for containers in the standard library as well. But while you are recapitulating C just this is all that you have for containers and pointers.

(Refer Slide Time: 06:40)

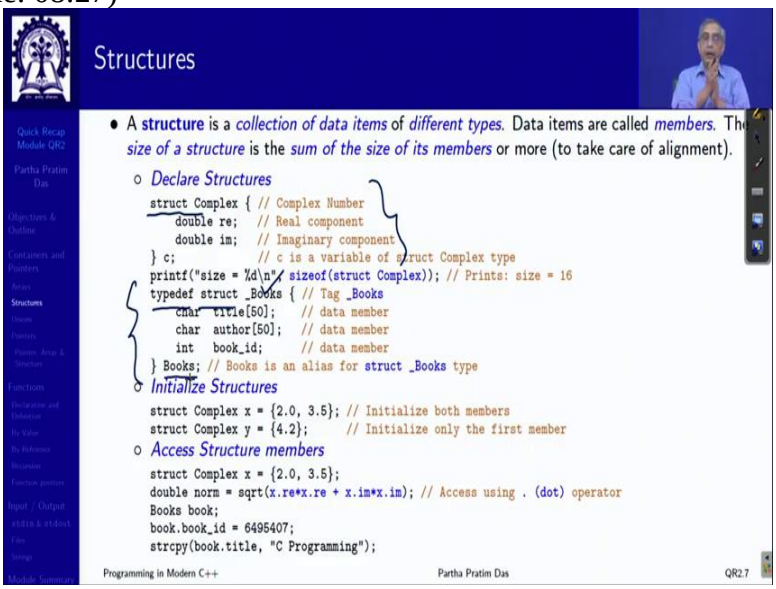


So, quick on array, this is how you declare it always needs a size, I mean number of elements to be specified. In most cases when we define arrays or actually when we define arrays, we must always provide that, we can initialize arrays in in different ways. And without defining the number of elements or with defining the number of elements, I have in the comment, I have given reminders on what do they mean.

Then once it has been accessed, then for the range of indices that exists between 0 and the number of elements minus 1, I can use that index and access that element to read or write. Arrays could be multi-dimensional, which means that it is, it could be arrays, of arrays. So, it is considered row major. So, C uses row major. So, if I have this, then this will be considered as an array.

The next this is considered as an array and then as if you have another array of these arrays. The advantage of doing this is the fact that it can, this idea can be extended to any number of dimensions and you do not need a separate support for that. And multi-dimensional array is a very strong feature to help doing a lot of different things.

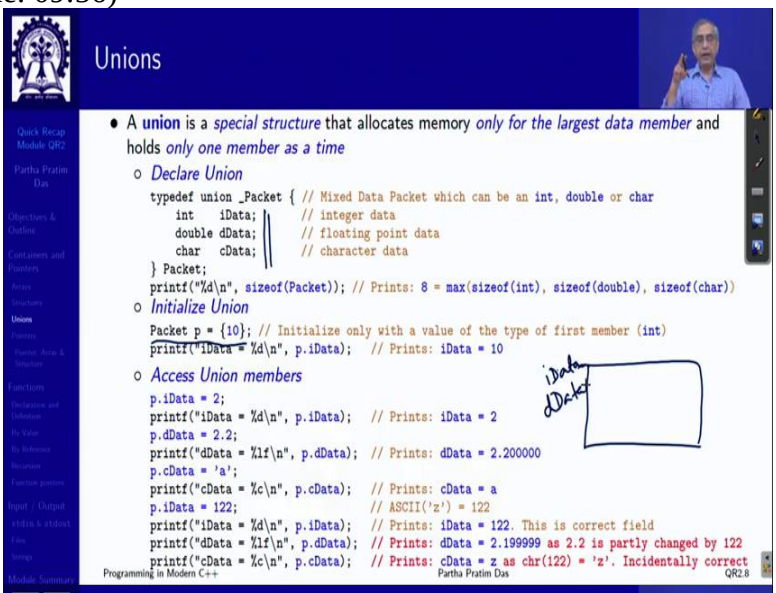
(Refer Slide Time: 08:27)

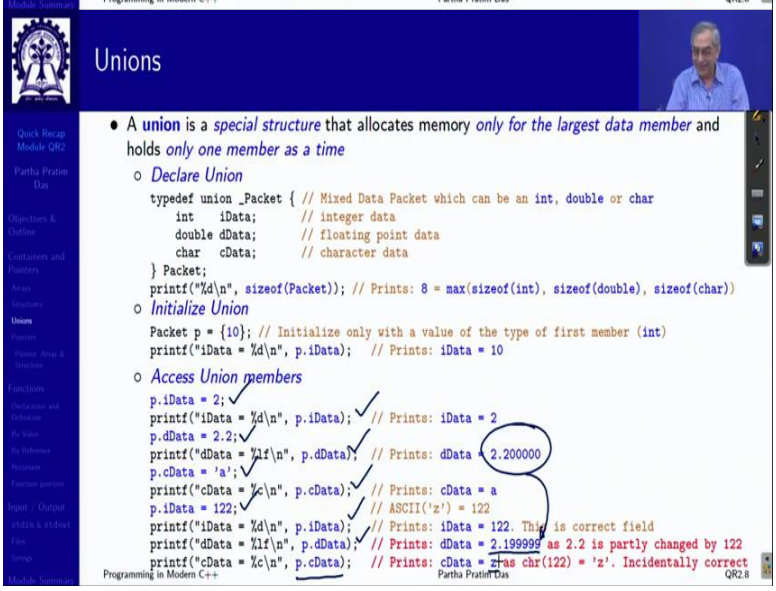


Structures, the other container this is a typical declaration you will have components. And each component will be given a name, you start with the struct keyword give the tag name which is called, basically it is called a tag name. And therefore, whenever you have to use, you have to refer to this as struct complex. Otherwise, we can use the aliasing feature the type def which is which can give a different name to a type.

So, here you have struct the aliasing name is underscore, your tag name is underscore books. Whereas I type def alias it the struct underscore books i l a s just books. So, then I would not need to say struct underscore books everywhere I can just say books. Syntactic convenience, does not, at least the level of C it does not give you any other specific benefits. I can initialize structures as I initialize arrays. And I can access the structure components by their name using the dot notation. You must be all familiar with that, just close your eyes and remind yourself that this is what you have.

(Refer Slide Time: 09:56)





Union is extremely similar. In terms of declaration, all that is different is in case of struct you are saying, union. But the main difference comes in the fact that if I have three fields, as I said, only one of them could be there. So, when you initialize union, even though it has three fields, he will initialize it with only one value, because only one value can be there. And here I am showing initialization, which is with the integer literal 10.

Now, the question is, since union keeps only one value, it has only one location. That location could keep, could be called i data. It, the same location is called the d data and so on. So, it is up to the programmer to remember what kind of data she has kept. If you keep one kind of data and try to use it in a, for a different kind, then you may have unexpected results.

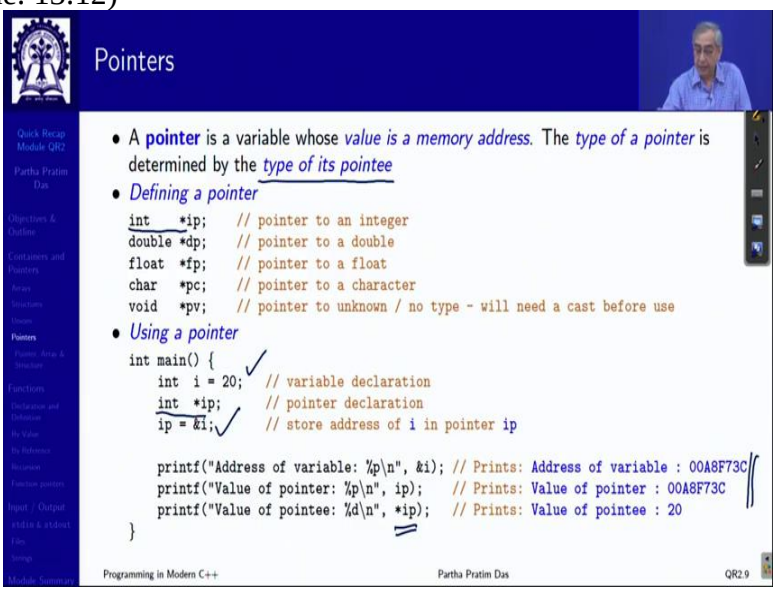
So, in the code below, I have tried to show that if you put an integer and use that field, you have put 2 in iData and you are using iData, it is fine. Put 2 in dData using dData, it is fine,

but 2 in cData, using cData it is fine. But let us say you have put 2 in iData, 122 to iData you're using iData this is fine. But suppose you put 122 in dData, you are accessing something else.

Interesting, you get some 2.199 something, because what happened? The double is the biggest. So, it fills all the bits, when you put in i data, you are just changing a part of that, rest of it is the earlier double value which existed. Only some part as if it got corrupted. So, it changes, this 2.2 into something a little different. But I mean do not go by this. It can change to anything; else, it depends on the system depends on the values.

But interestingly, if you, if you try to read the same thing with C data, corrected data, then it will print z, why? Because z is the as common 122 is the ASCII code of z. So, it happens to print. So, please be careful that always use the field that you have, you know that you have populated.

(Refer Slide Time: 13:12)

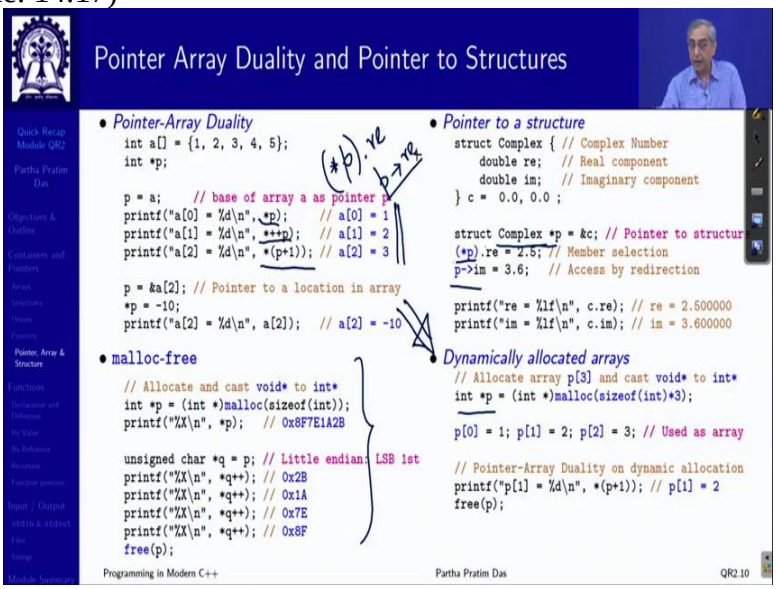


Pointers, the most powerful and as I said the most risky, which is a variable which gives a value which value is a memory address. And the type of a pointer is the decided by the type of the pointee. So, and point int star is an integer pointer that is if you go there, you will find an integer. Whereas the double star is a double pointer, because if you go there, you will find a double value.

So, as you know, you can always take the address of a variable and put it there, here are examples of what simple things you can do with reading the value, dereferencing the value at any, if you have a pointer, you can put a star in front of it so that it actually goes there and

finds that value. So, ip is defined as an integer pointer. So, when I do *ip, I get an integer. So, that is a basic property of the pointer as we all know.

(Refer Slide Time: 14:17)



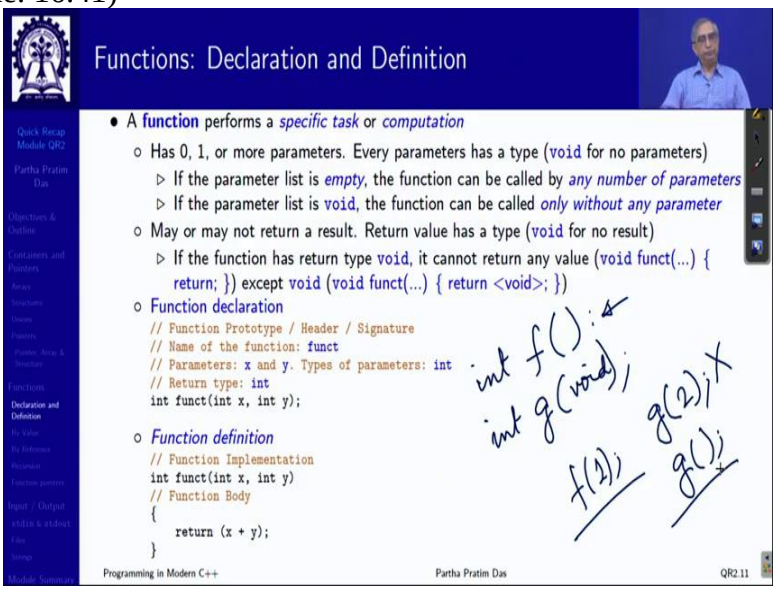
Now, the main power of pointer comes from the duality and ability to navigation, duality with array, ability to navigation that if I have a pointer, I can treat it as an array. If I have an array, I can treat the base address of that, that the name of the array as a pointer. And this duality helps in terms of managing dynamically allocated arrays. Otherwise, we would have been in deep difficulty if we do not know the size of an array.

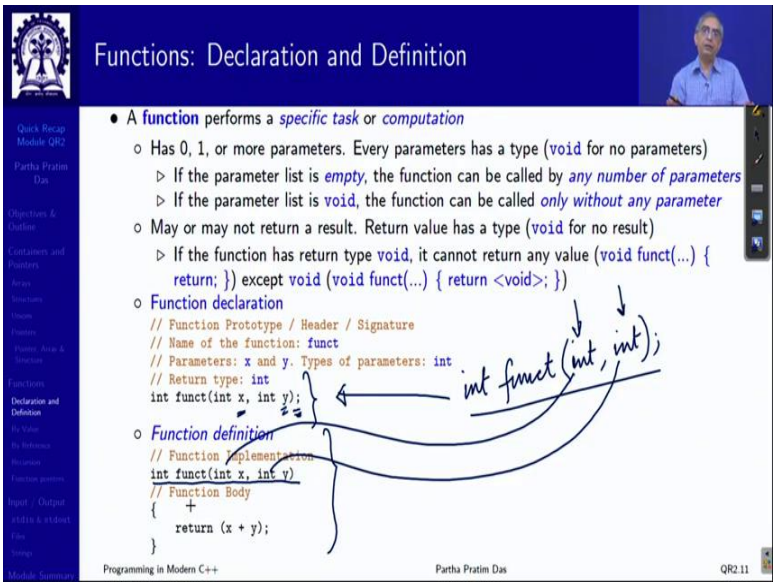
At the time of writing the program, how do we write that array? I cannot put the number of elements. So, how do I do it? So, I can dynamically allocate using malloc. And then malloc gives me actually not an array, it gives me just a chunk of memory. Which, I am thinking is an area of say integers. I think it is an array 0.5. So, I think as if p is the base of that array.

So, this interchangeability of, computing on the pointer address value and the actual array notation, this interchangeability is a very strong power to see which C programmers amply enjoy and use. Naturally, pointers can do any form of dynamic allocation by malloc and it has to be freed you know, know that. You can also have pointers being, using I mean dereferencing a particular location.

So, I can say I have a pointer to a structure, so, *p I have a pointer to a structure complex structure. So, star p is a complex structure. Then I want to go to re. So, I say *p.re and a shorthand for that, *p.re is p pointer re. So, often it is it is easier to read, easier to write and so, on. So, these are, these are the basic pointer you know recaps you must have.

(Refer Slide Time: 16:41)





What certainly is most important is a function, which does a specific task or computation. So, I said that, if the function usually is a, is an expression because it returns a value it does a computation and returns a value. But it could be returned, its return type could be void in which case it is just performing some tasks, it is not, it is just a statement by itself. So, a function can have 0 parameters, no parameter, one parameter, more parameters.

Every parameter has a type. Now, you can also instead of 0 parameter, you can also write void in place of that without any parameter name, that also means that the function does not take any parameter. It is a little bit of an old style it is still supported, but normally people do not use it. The subtle differences if the parameter list is empty, that is say if the parameter list is you have said is say int f void say these are the two signatures.

If the parameter list is empty then in C actually you can call this function by any number of parameters, that is you can call it as f1, it is. But if the parameter is void, then this is an error. You will only be able to call it as without any parameters. So, this is the subtle difference of a writing a void in terms of the parameter. Otherwise, you will not, obviously if you want to say parameter you cannot say void because it just void is no type as you know.

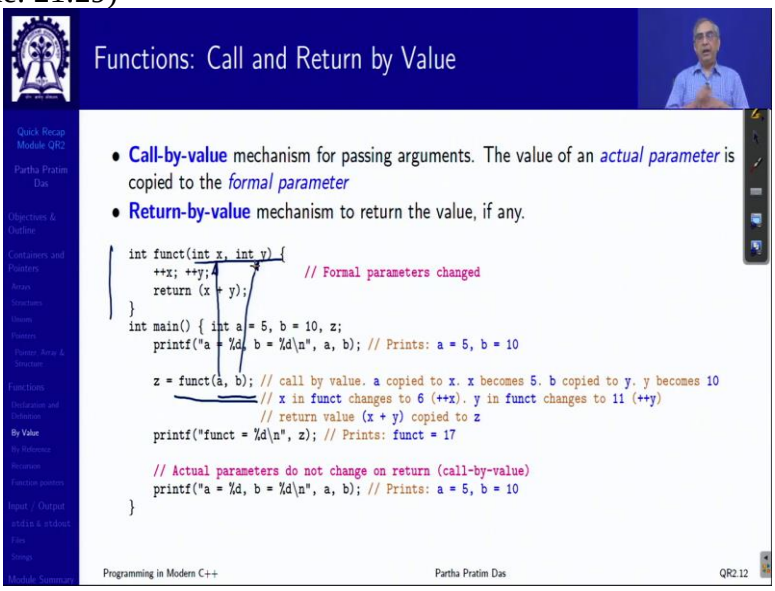
So, and in terms of return you understand, that if the function returns void then it is not returning any value at all. So, we talk about function declarations, which has a return type function name, list of parameters with type and the parameter formal parameter name. In the declaration if you, if I just terminate it with a semicolon, which we refer to by different names, some call it declaration, some call it function header.

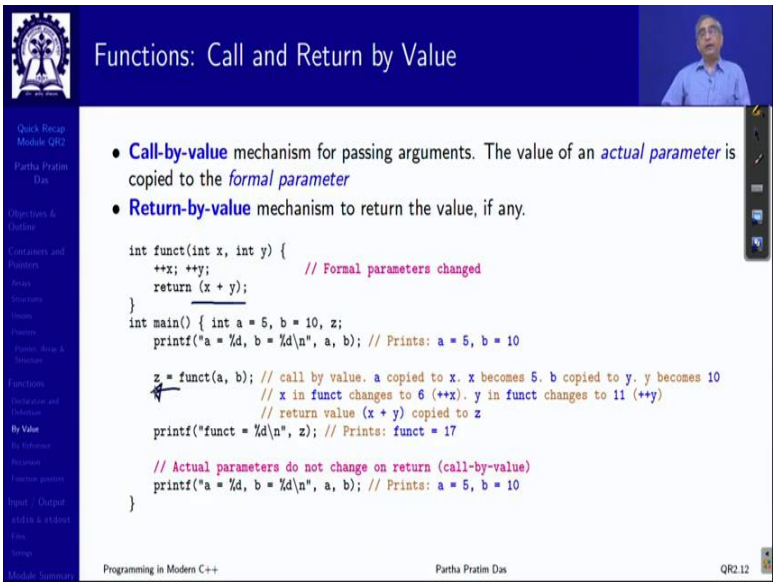
At times, we call it the function signature any of these, then what you are saying that how to call this function, what are the parameter types to pass and what is the return type to expect, what is the name of the function but we are not saying what competition it is doing, which is a part of the function definition. Now, when you write the signature, it is important to note that writing the formal parameter name, as a part of the signature is optional.

I can just write it like this. Since there is no body, I am not going to use x or y, that I write here. So, all the information that I want to give is the first parameter is in second parameter is in. And when I give a body to this function, implement this function, this and this must match. For the given function name, the parameter types from left to right between the signature and the function definition must match.

There will be a lot of stories discussed as a part of the course, when we go to C++. Because in C++, this rule will not hold because you have overloading. In C, you do not have that every function is global has a unique name all across the project, and has a unique set of parameter types going from left to right, and the corresponding retracting. So, this is in gist, rest of the function body is just you know, another program code.

(Refer Slide Time: 21:29)



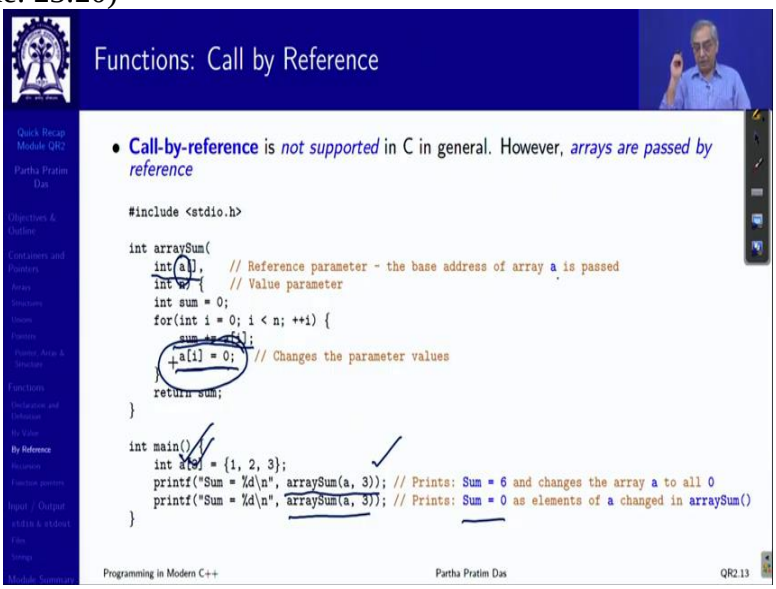


Now, in C functions are called by value, that is actual parameter. If this is the function, and this is it is called, then these are formal parameters x, y, and a, b are actual parameters. So, when I make this call value of a is copied to x, value of b is copied to y, which means x and y have get locations which are different from a and b. And they just get a copy of the value. Similarly, when you return from the function, it is also by value.

For example, you are returning here, you are returning x + y. So, what is x + y? x plus y is a value, it is not a variable. So, what will you return, we will just return the value of this. So, you just put that value into some temporary and put it here to be placed onto z. So, that is just that value is being copied. It is an expression, x + y is an expression it has a value. So, that value will be copied. So, it is called return by value.

And C only supports this, in C++, we will add a lot of stories onto it. Now, one good thing about call by value is that if you make changes within the function, then the actual parameters will never get affected. So, you have that safety and that has consequent pain also, like it is very difficult to write a swap function in C.

(Refer Slide Time: 23:20)



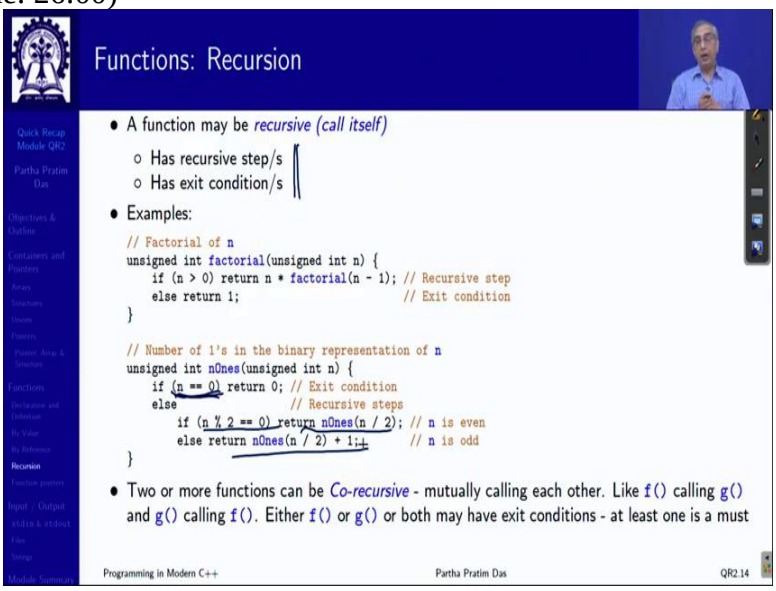
Now, there is a call by reference mechanism, which C++ heavily use, where you do not copy the data, but rather you pass the address of the actual parameter. It is, it is not exactly a pointer. It is a different concept and we will talk about it at length during C++, but this is called a reference, where without copying the data, you just give make the actual parameter and the formal parameter refer to the same memory location.

Naturally, this has different consequences, in C you do not have this mechanism except when you are using an array. Array is huge, so, by doing copying for the array would have been very expensive. So, what C has provided, C does and does not talk about it very clearly in every text is arrays in C are actually passed by reference, which mean that if you pass an array, as I am doing here.

Passing an array in array sum, so this is mine array a, as I pass that my formal parameter and the actual parameter are referring to the same memory location. So, if I make changes, like I am making changes here, after reading its value putting to the sum, for some reason I am making it to 0. And that change will happen in the actual parameter. Therefore, you see that if you call array sum for the first time, you get the correct result 6, 1 + 2 + 3. You are adding them.

But if you call it the second time, you get a 0. Why? Because all elements have been made to 0. So, this is this is a simple test to show you if, if it were actually a call by value, then this would not have happened, because anything that you do inside the function would have been done in a different location and would not have affected your actual parameter. But here, that is not the case for array. So, that is a special case and we will obviously have variety of a wide range of discussions on call by reference in general.

(Refer Slide Time: 26:00)



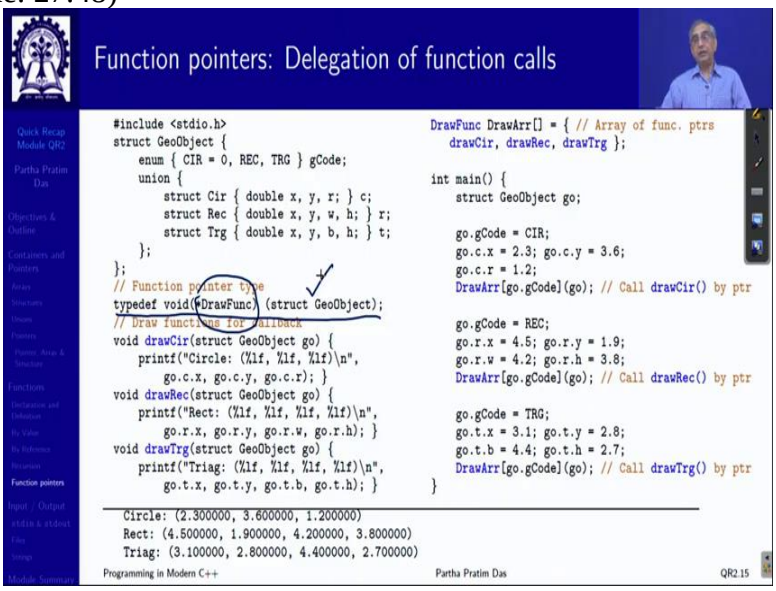
But the C feature I wanted to. Now, when you revise your functions, make sure that you revise recursion. Because even though it is an algorithm concept, but having it clear is very, very important that the recursion will always need the recursive step and the exit condition. Like it, this is the exit condition and this is, these are the recursive steps. So, these are just, fun example, I mean factorial is obviously overused example.

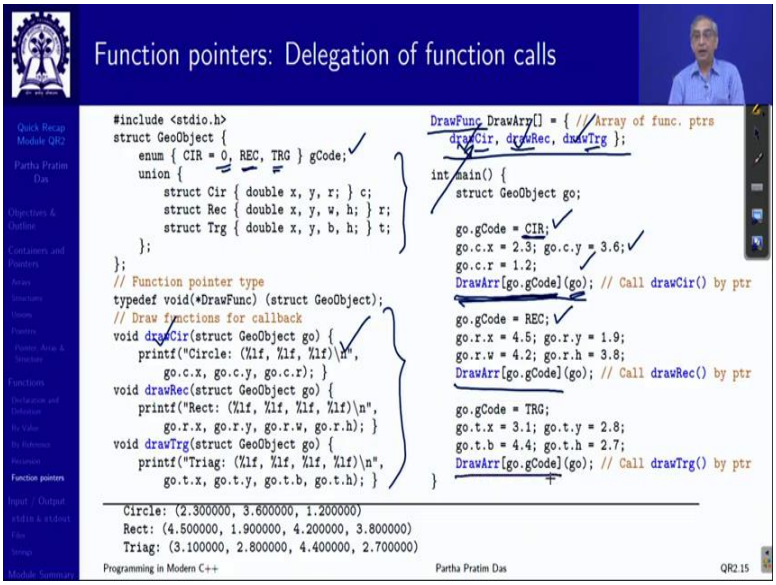
And this is the other one is a fun example, which takes an integer and counts the number of digits, number of ones that exist in that number. It is done recursively. So, you keep on calling the function. And also keep in mind rather I mean, you need recursive functions, because they are one of the strongest functional competing mechanisms that you have in the language or in the algorithm.

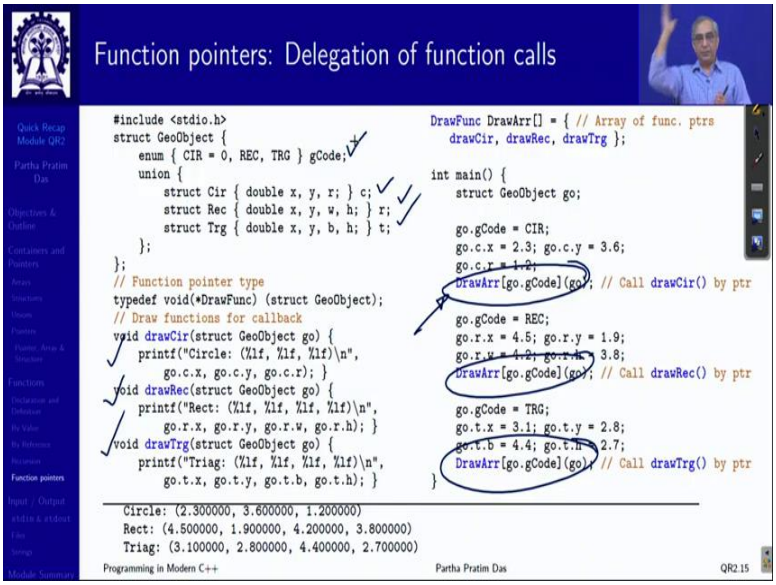
And please remember that two or more functions can be co recursive. That is, if I have f and g two functions, f can call g and g can call f. Naturally, there has to be at least one exit condition, either in f or in g, it could be in both also, but at least one of them must have exit condition. Otherwise, this process of f and g will not ever end. It makes it a little bit more

complex, but many a times co recursive routines become very useful in terms of realizing variety of algorithms.

(Refer Slide Time: 27:48)







So, you last but not the least functions can be written as pointers, or rather functions can be pointed to. So, you this is the typical way you use typedef to do this. That you are saying that I have a function whose name is star draw func. So, it is actually not a function, it is a function variable or it is a function pointer, which will point to a function I can use. Which takes a struct geo object and returns nothing.

So, if you read through this carefully, you will see that here is a union of, of different geometric objects. Here is an enum which actually remembers which type of geometric object has been stored here, whether it is a circle, a rectangle or a triangle. And then we have some draw code, this is not a real drawing code, rather, in the name of drawing circle, I am just printing the values of the circle and so on so forth.

Now, what if I want to write a single code which can print or draw for any of the object types. So, what I do is I have three functions. So, I make an array of draw func type, what is draw func type? It is a function pointer, it is a pointer so I can make an array of that and make these three the three elements of the array, which are the three functions. Now see the beauty. If I set a circle by giving a gCode and giving its values, and I call this, what is gCode?

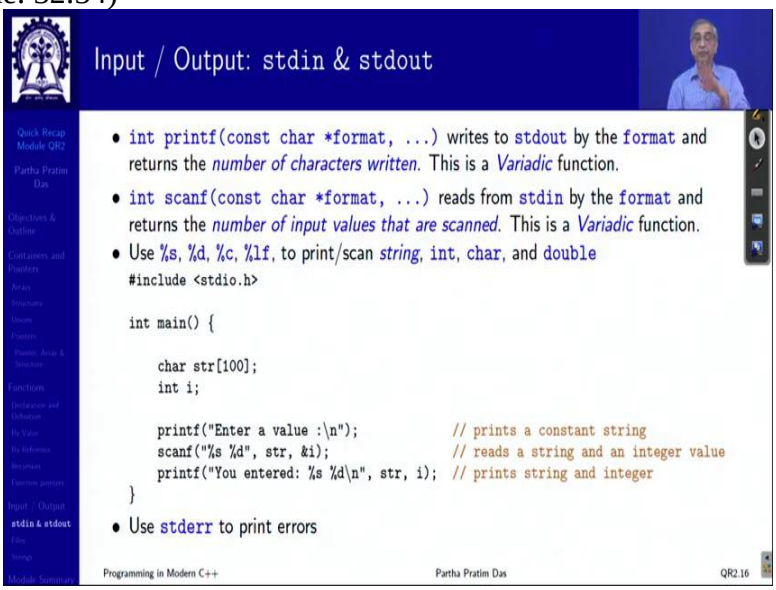
gCode is CIR. What is CIR? CIR is 0, what is DrawArr[0]? It is this function, which is a separate function. So, when I do this call, so, this entire thing is a function, which is actually the draw circle function. And I pass the geo structure to it. So, this gets called, it is done little carefully by design that I have put the codes in a way which and rather I have put the function pointers in a way which matches the order of the code.

So, when I do REC it is one. So, this actually means the drawing function, this actually means the drawRec function. This actually means for TRG, the drawTrg function. But the beauty of the whole thing is, all of these codes, all of these calls are same, and they do not really need to know the name of the function I am calling. So, it is possible that I can have a big collection of geometric objects as an array of these unions.

And I can just iterate over them in a for loop using just this one call which will automatically take the right function to be called. This process by which you change the function called based on the type of the object, these three different types of objects, here of course in C it is not being guided by the type of the object. I am specifically having to maintain a code to do that.

But this whole process the which is where you select a function based on certain hyper parameter or based on certain in marker for types is known as delegation. That is, you want to do a job, there are three agents, draw circle, draw rectangle, draw triangle, you have to decide which one you are delegating it to. And in this technique, you can write a very compact code for that. C++ provides a huge support for this, which can make things much simpler. But to be able to understand that well, you must understand this function delegation through function pointers.

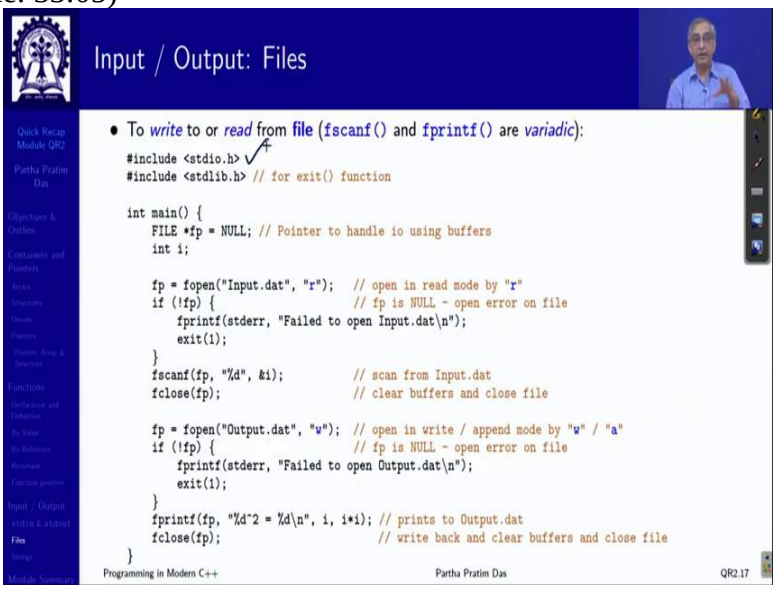
(Refer Slide Time: 32:34)



Rest of it is too elementary, you have input output, stdin, stdout, printf, scanf both of them are variadic functions that is they must have this format parameter. And then any number of parameters in the format string will have different format codes to decide what kind of

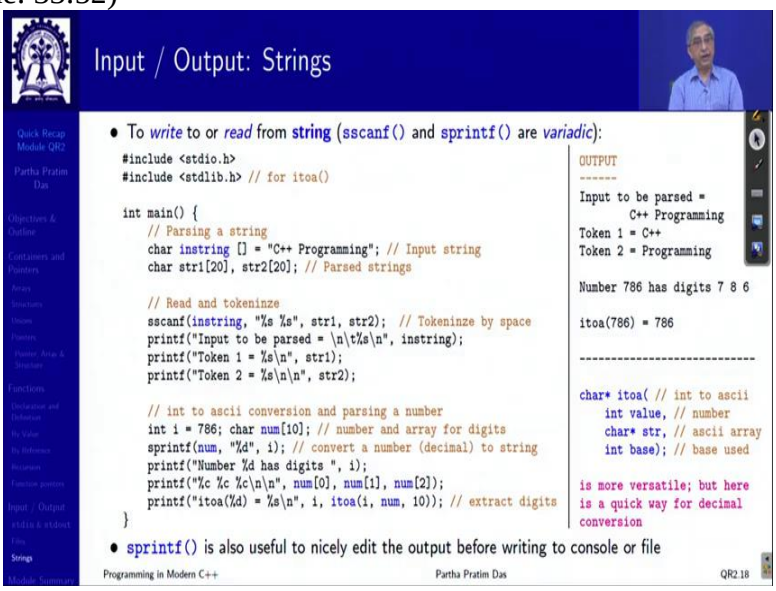
variable you have given corresponding to that. With, in addition, you have an std err to print error messages.

(Refer Slide Time: 33:05)



You can do input output on files. If you are not on top of this, just recapture it. There is a there is a in stdlib, there is a file structure I am sorry, in std, this is not in std lib. In stdio.h, there is a file structure, which you need to create a pointer of, and then you can open a file, you can print to a file, you can close a file, you can read from the file all of this corresponding to the usual console operations, what can be done, it is very powerful. And we will continue to have that in a different way, in terms of C++.,

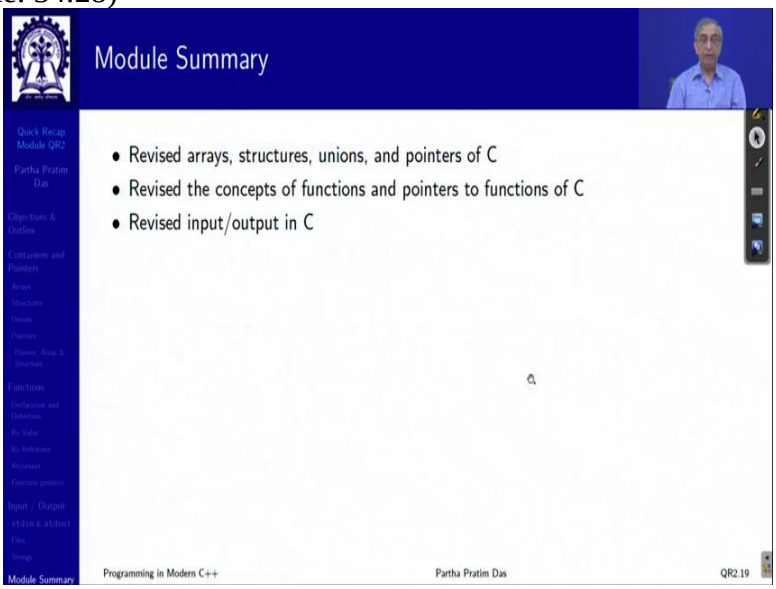
(Refer Slide Time: 33:52)



Similarly, you can do a lot of things with strings in the input output mode also. So, as you have printf for file, you have fprintf, for string you have sprint, scanf, fscanf, sscanf. And

here I have given some small examples to show why, thinking of string as, input output could be useful. Go through them, it will give you better insight.

(Refer Slide Time: 34:28)



So, this concludes my first round of quick recap on C. We have talked about the remaining major features. And please go through this to make sure that you do understand them, not only know but you understand and we are able to use them in the programs. That will hugely benefit your speed in following the actual module lectures on programming in modern C++. Thank you all very much and looking forward to seeing you in the course.