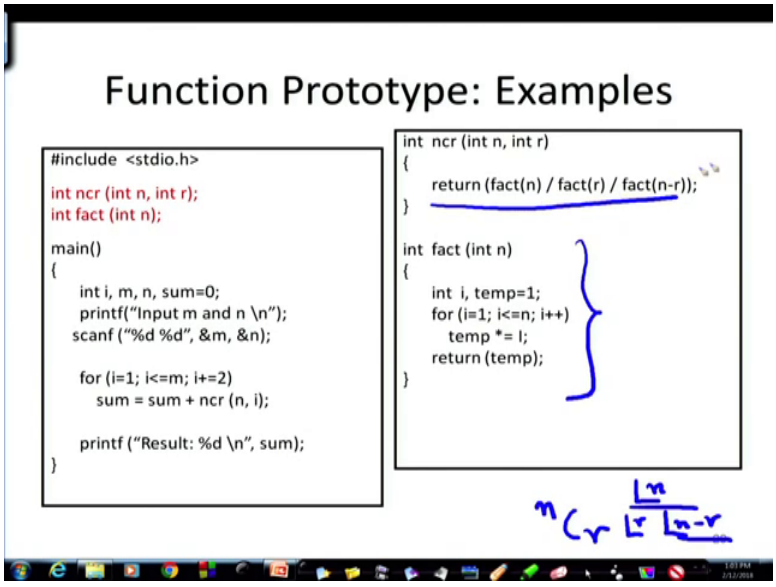
Problem Solving through Programming in C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 39
Parameter Passing in Function Revision

So, in the last lecture, we have discussed about function prototype. Before moving to some other topic let us look at an example of function prototype and so that we can understand it better.

(Refer Slide Time: 00:26)



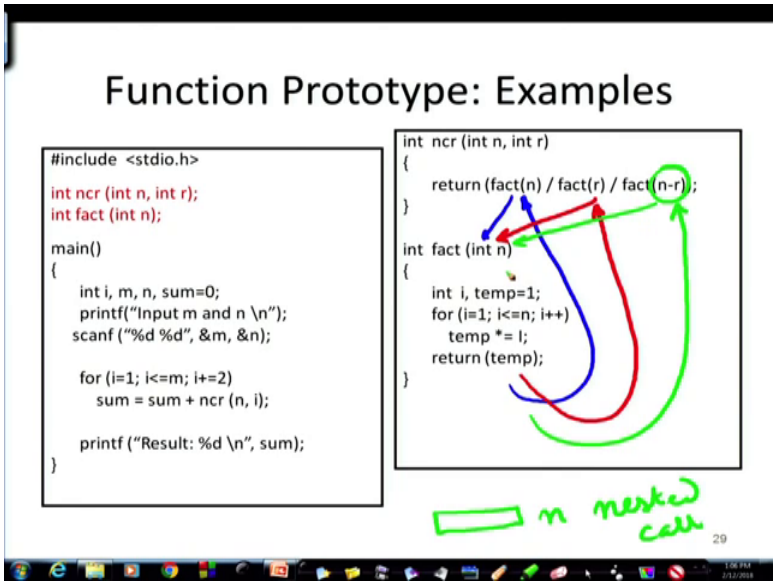
Here you see here is a program were I have got a function n c r, n choose r and another function fact. Now, the functions are actually little later here you see n choose r is like this standard n choose r that you do. Here that is a factorial n by factorial r, so factorial n divided by factorial r into factorial n minus r that we know from our school right. So, that is one function that is one function.

The other function is the factorial. So, here is another function, this is one function this another function. Now, look at the beauty of this function this function is in a simple one return statement I have written everything. So, here you see the body of the function is merely a computation of an expression fact 10 divided by fact r divided by n, fact n minus r and this will be computed and that will be returned as an integer. Now, what will be the input the input will be n and the r both of them are integers.

Another point to note here is this function itself this function itself when is running first say I get into this function and I tried to compute the return value, first I find fact n and from here I am calling another function that is the factorial function and the factorial function is taking only one input one integer n and based on that it is computing the factorial. Computing the factorial you must be remembering by now, or knowing by now, that is I am starting with 1 and 1 times 2 times 3 etcetera I am going on doing and that is what is being done here temp is 1 and temp is equal to temp times i, this is wrong this should be small i, temp is temp times i, whatever the i value is, 1 times 1, 1 times 2, 1 times 3 it goes on till i reaches n and then i return temp.

So, the point to that you know, this will this thing you know. So, what is being done here is when I am computing this function then I come to this I call this function return temp, I return here and then I move here and again I find again I call to this fact again, this time with the different parameter r and so again fact r is computed after this is computed second time when I return back here. And then I again call this fact using n minus r, so just to make the things clear. So, the first time I am calling this from here and returning back here, second time I am calling this from here the same thing and returning back here and third time I am calling from here and returning back here.

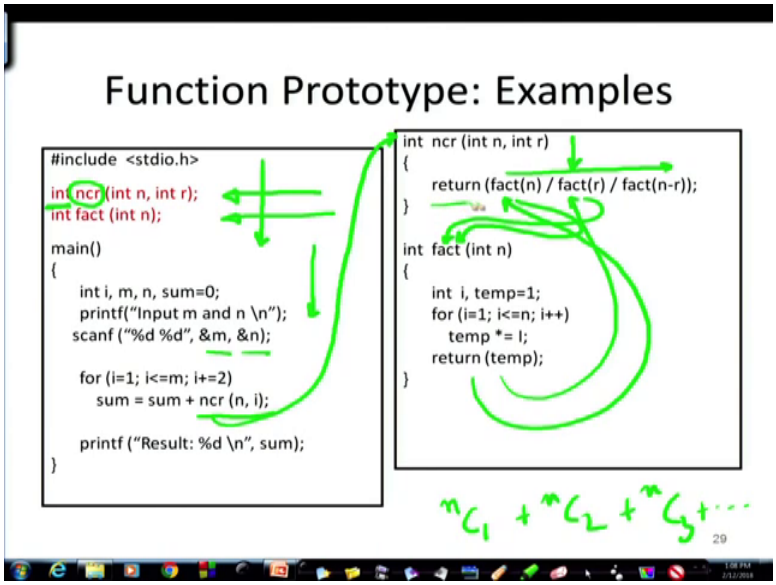
(Refer Slide Time: 04:29)



Now, for the third call I am, so, here is one n and that this n and here what I am passing I am passing n minus r, I am first computing this and that is being passed here and that is

being computed. So, here you see it is an example of nested call this was called from the main and this in turns call I mean call another function and in this way it goes on. That is about the beauty of these two functions. But now, here is my main, main function here is my main function. So, the compilers comes from in this way and recognises at this point that ok, n c r is a function how does it know it looks like int. But how does it recognise that it is a function? It recognises n c r to a function because of this parameter argument list.

(Refer Slide Time: 05:40)

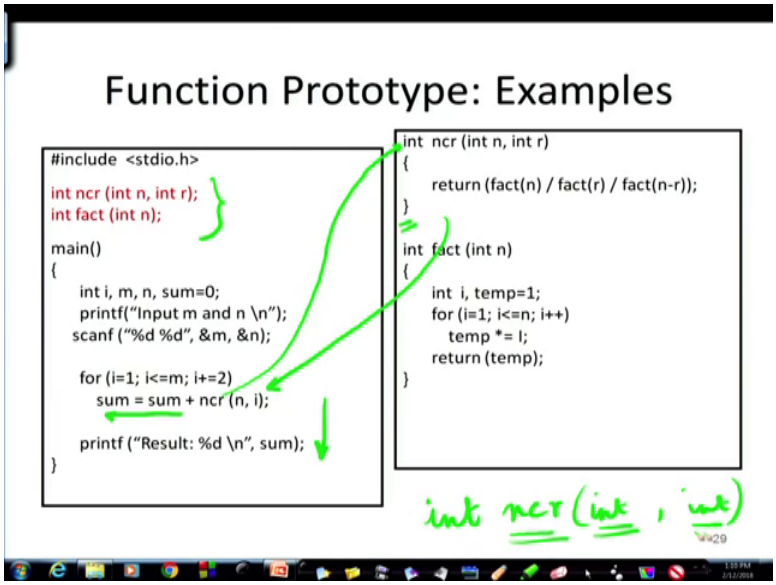


Here it is said int r there are two integers coming in as parameters next it understands that fact is also a function. So, its expecting to encounter n c r and facts somewhere down the line and proceeds here it takes m and n, reads m and n, then it is in a loop where it is calling n c r. So, it is series you can understand n c r is computing some with some value i initially 1. So, it is basically n c 1 plus n c 2 plus n c 3 plus etcetera it will go up to m all right. So, that is what is being computed here and so from here a call is made to this.

And as we have seen earlier while this is being executed this one is calling here this is returning back here then again this one. Now, we are making a journey in this line. So, this one is calling this and we are returning back here as we have seen just now. Now ultimately when this entire thing is computed then we will come to this, last bracket here sorry I should not this bracket here; that means, it is a end of n c r. So, I had called n c r

from here. So, I will return back here and will whatever n c r value is that will be added to sum and that will go to this sum and then will proceed in this way I hope this clear.

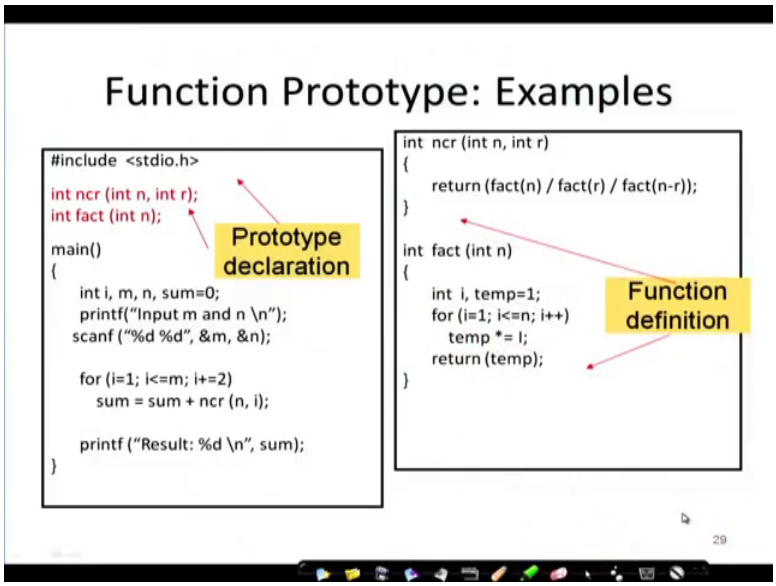
(Refer Slide Time: 07:42)



Now, here I have not written n c r. So, I could have I could have written this earlier in that case this prototype would not be needed, but since I decided that I will be writing it later. So, I decided to just introduce them as prototypes here. Now, you should be very careful about the syntax of the, of writing the prototypes it should have the function type just like a function name and the parameters.

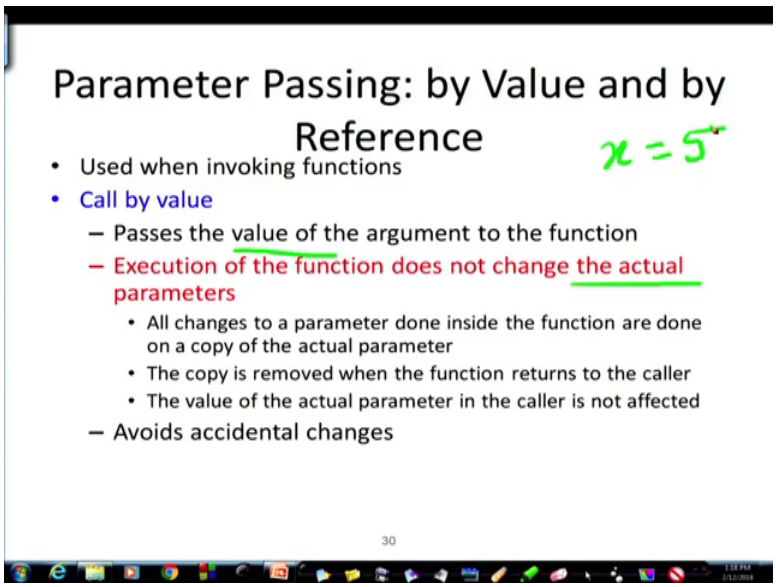
Now, you will see later that it is also possible that I could have written something like int n c r, int comma int that will also mean that this n c r takes two integers as (Refer Time: 08:45) I mean arguments, that will also that is also allowed, but it is nice to write this in this way.

(Refer Slide Time: 09:06)



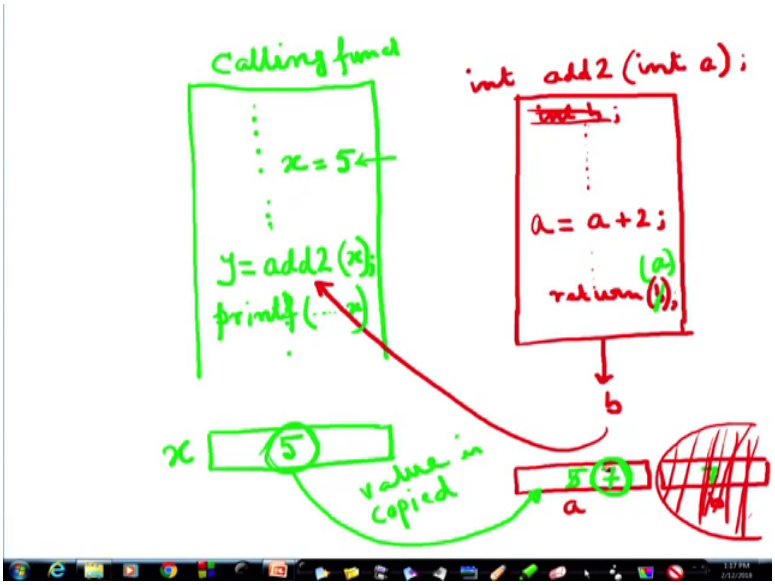
Now, next will move to a very important concept; let us quickly have a look at this the prototype declaration and the function definitions are actually here.

(Refer Slide Time: 09:10)



Now, we are moving to a very important concept of passing the parameters, how do we pass the parameters from the calling function to the called function. There are two distinct ways in which it can be done one is calling by value another is calling by reference. So, right now, let us think of calling by value all right. Now, let us try to understand it in a simple way.

(Refer Slide Time: 09:57)



Say, here is my main function or the calling function let me not call it main function as we have seen they can be nested functions. So, this is the calling function. Now, inside the calling function I have got some variable x and which may have the value says somewhere 5 it is an integer. Now, somewhere here I am calling a function let us call a simple function decrement or no decrement add 2 or say add 2; that means, whatever is a value we will add 2 to that simple thing. So, here I say y is add 2 x and then semicolon and I go on. And here is my function add 2 int a all right and add 2 is also type integer.

Now, so a we know is a local variable to the function. So, here I will take. So, what will happen? Suppose it was 5, this will result in a will get 5 and here may be here I declare another variable say b although it was not necessary, but I am just saying b is a plus 2 because its task is to add 2 and then I say return b. So, this b which is an integer is being returned. So, b which will be in this case it was 5 it will be going back here. I hope this is clear.

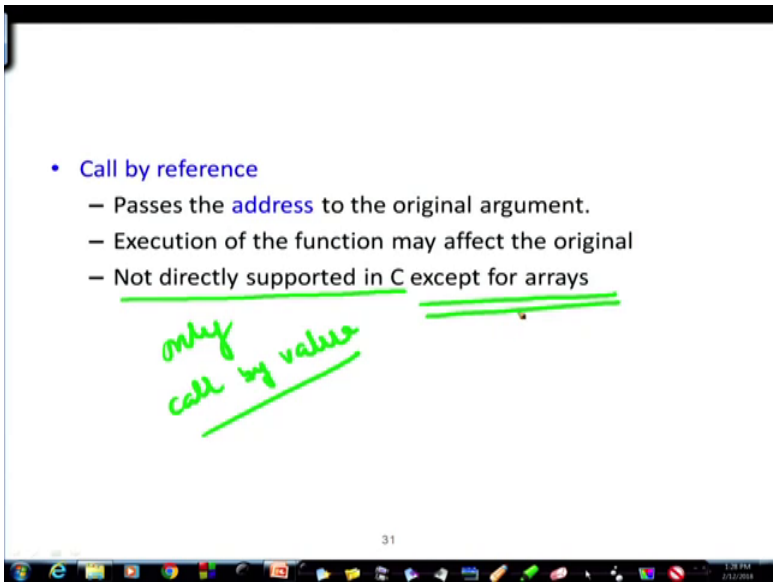
Now, when you know the scope of variables that b is a variable or a or b whatever the life of those or the validity of those are restricted only during the life of this function. So, but; however, in the case of a looking from the point of view of compiler I have got a variable a and I have got another variable b and here I have another variable x. Now, this variable x was having the value 5. So, here in x I had 5 by this statement, this statement made it 5. Now, since x is 5 and this one is expecting the value a, now, this 5 will be

copied in a; what is being copied? Not x, but the value of x. So, this one will be will be getting 5 right. So, the copy value is being copied, the value is copied in this variable 5. Now, it takes b was something, but here. So, b becomes 5 plus 2 b becomes 7, 7 comes out.

Now, suppose I change this program a little bit. I erase this b and make it sorry I make it a. So, this b is no longer there this is not required all right this variable has not been defined I say that this line is also not there, int a and I just do a plus, a assigned a plus 2. So, now, when it was called add x then from here from here this x was copied here and a will be incremented to 7. So, this will be incremented to 7 and obviously, this will be return a not return b because b is not there anymore, so 7, that 7 will be returned here. But you see if here now, 7 has been returned and 7 will be y. Now, suppose if I say I am not following the syntax I am just saying because there is no space here printf something x. What would be printed? For x what would be printed? 5 will be printed because x is still 5 x has not changed. I have simply copied the value to the argument variable and have played with that changed it whatever I wanted to do I have done.

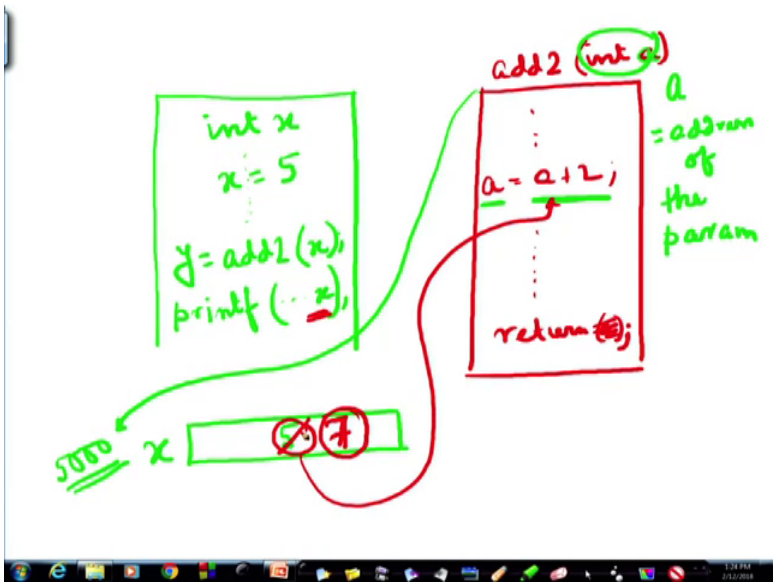
So, take a little time to understand this. This means that the variable that I have in the calling function the value of that will be copied to the argument of the called function. So, there are several advantage is to this one is, so it passes the value of the argument execution of the function does not change the actual parameter like the actual parameter was x which was 5, it remains as 5 although the function added two to that and it came back all changes to a parameter done inside the function are done on a copy of the actual parameter not the original parameter. The copy is removed when the function returns to the caller that entire variable location that was given for the variable a in our example is returned back to the pool of memory locations. The value of the actual parameters in the caller is not affected. Consequently it also saves us from some accidental changes programming that can come copying due to programming errors.

(Refer Slide Time: 17:59)



On the other hand the other thing that is another type of parameter passing is known as call by reference, call by reference. Here we are not copying we are not copying the variable, we are passing the address of the original argument.

(Refer Slide Time: 18:35)



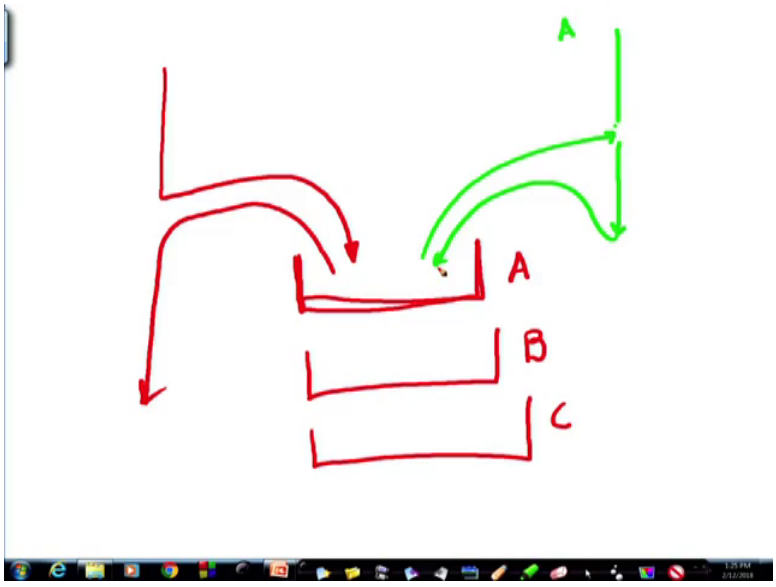
So, let us take the earlier example again if I had my main function here and I had similarly int x and here y was add 2 x here printf something x, and here I had that function add two int a and here in the body I did a assign a plus 2 and return a, return a. Now, here is a variable x and that has got an address say that address is whatever 5000.

So, now, its value here somewhere x was 5. So, its value is 5. So, here what I am in the case of reference I am not copying the value of 5 to a instead in the parameter passing I will write it in a different way not in this way which I will discuss later.

If I assume that this one a, a is not taking the value, but the address of the parameter, address of the parameter. Means what? Means that this time this a here I am passing not the value 5, but I am just simply saying that whatever data you want to work with that is the data is in this location 5000. And it excepts that reference that the data that I am working on my a is actually staying 5000. So, what it does? It takes the when it computes a assigned a plus 2 this is not looking nice. So, let me write it a, a assigned a plus 2 it. Now, knows that it is not the value a is here I have to get the value from this location. So, it gets the value from this location, but no other variable location has been allocated because I know where there is only one common place, only one common place you have done given something here and you haves told me where you have kept the data and I am also working in that vessel itself.

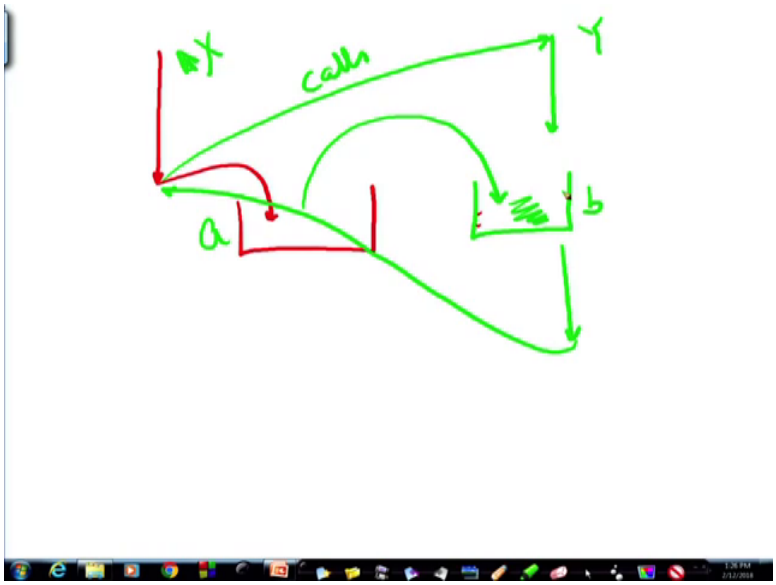
So, what is happening here is this 5 will be changed to 7, here only and return I need not return a it was just return would be sufficient. So, changes have been done here. So, in this case when I come to this printf x what will happen what will be printed 7 will be printed because the actual data has changed here. Let me show it you another example.

(Refer Slide Time: 23:03)



Here is my calling function and I think this vessel analogy will be fine. Now, it says that here is the data where I have poured it only tells you where it has poured, the name of the vessel, suppose the name of the vessel is a there are many other vessels A B C and only this vessel name has been passed and when this one is doing something it knows the vessel name A. So, it comes here and takes the data from here does something and returns the data from here and here this program when it is computing when it returns and then ultimately it returns there and while it returns it takes this value this value the change value and continues. So, in the case of calling by reference we are not copying the value. On the other hand what would have happen in the case of call by value using the same vessel analogy?

(Refer Slide Time: 24:15)



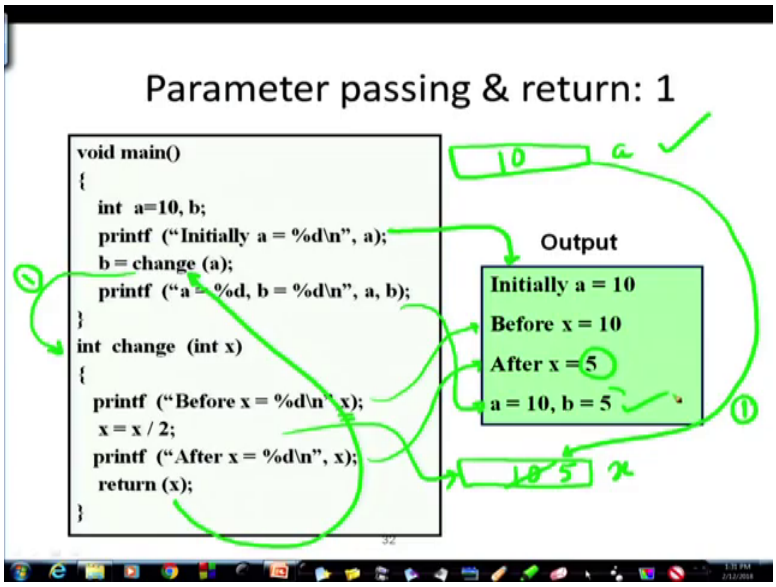
Say, this program was running the value was in a vessel the value was there in a vessel, but sorry, but my function do not bother about the inter change of colours; my function was also having its own vessel and when the function has been called the main function calls this a calls or say x calls y right, then also x copies y, copies the value of the variable suppose that was a that is copied in the vessel that is belonging to the function it may be b. So, this one does whatever it does here and returns this value over here. So, this one is not disturbed whatever changes are being done are, done here.

So, that is a very fundamental concept in parameter passing. There are two types of parameter passing one is call by value another is call by reference. Now so, here you see

execution of the function may affect the original because I am sharing the same vessel. Now, this in C, in C we actually we are in C we actually carry out only call by value, only call by value except for the case of arrays except for the case of arrays there is a reason for that you will understand and for arrays we are not passing the values we are passing by reference otherwise its always call by value.

So, let us have a look at some of the examples here, first example.

(Refer Slide Time: 27:02)



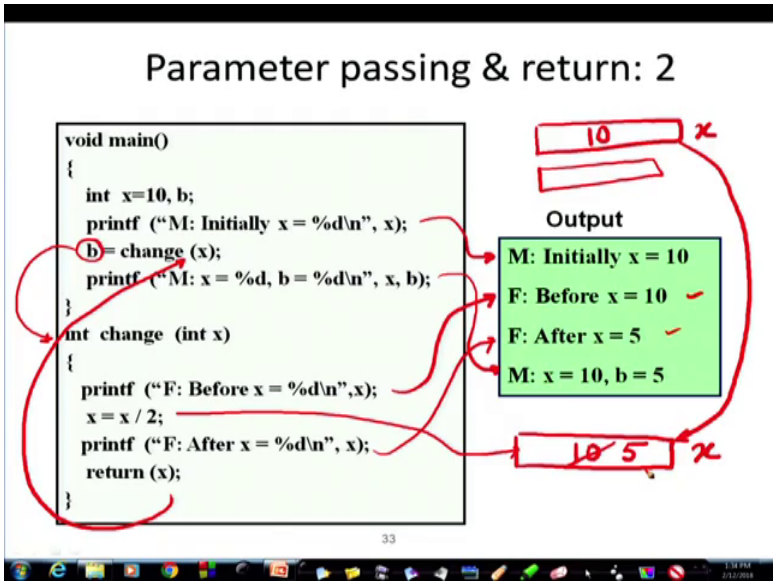
We have got the main let us see what is happening, a has been initialised to 10 and b is not initialised, printf initially a, a. So, what will be printed? Look at this, this will print first line initially a equals the value of a which is 10. So, here, here there is a variable a which has got the value 10. Then b is being assigned change a what is change a change a is a function.

So, one mistake is here I should have declared this change prototype here; however, I should have the function prototype should have been defined earlier. Now, I come here printf before in the; what does the function do prints before x x. So, before, so this point before x equal to x whatever x was, it has got the copy of that x, it has got the copy of that x. So, this it was a, a and that has been copied here for its own x, x is also 10 copied because when it called you actually copied this. When this call was made, when this call was made first then before that, before that it was also copied here and then only this was done. So, it was printed then x divided by 2. So, x becomes 5 by this statement done.

So, what is being printed? After here this line is being printed here after x equals x. So, what will be printed? I am still inside the function please remember I am still inside the function. So, 5 will be printed and then I return return x. So, what is being returned? 5 is being returned where it is being returned here, x has been a has been changed and that is going to b printf a assigned. So, this printf is this printf a sorry a equals here it I am printing a, a is a, so a is not changed. So, it is being printed as 10 and b is being printed as 5.

So, you see a has not been changed the reflection of the change has been reflected in this function and is being assigned to b, clear. Now, let us take another example and you will yourself try to look at this example and a little change has been done let us all together try to follow this example. So, a starting again a mistake is I should have declared the function prototype here. So, those things I have not shown here so, but you should do it.

(Refer Slide Time: 30:59)

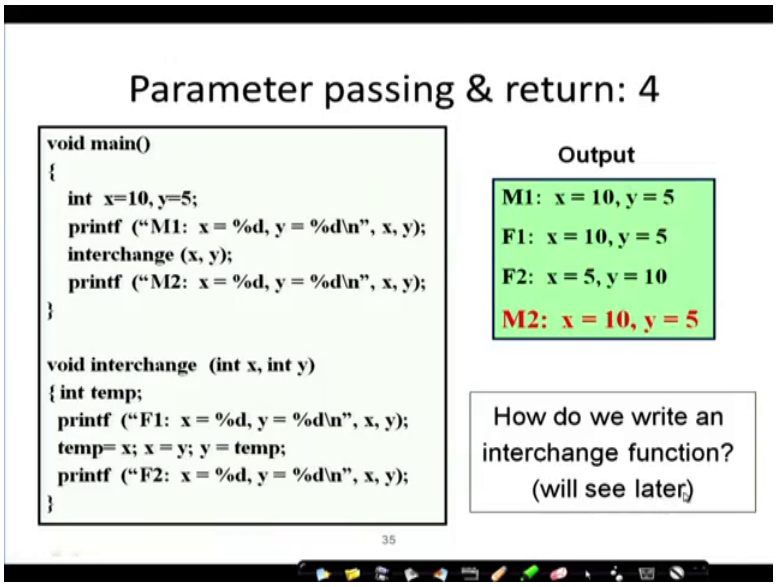


Now, let us see here. Again int x equals 10. So, x is a variable to the main function x is having 10 printf M here main function, printing initially x is 10, there is calling the function. So, here the function is being called. And whenever this function is being called there is a local to the function there is an argument x where this 10 is being copied.

Now, printf here, this printf in the function it prints before x was x before changing. So, x was x. So, what will be printed? 10 will be printed. Then I change x here. So, at this point x is becoming 5, all right. Then I am saying print after the change this is this

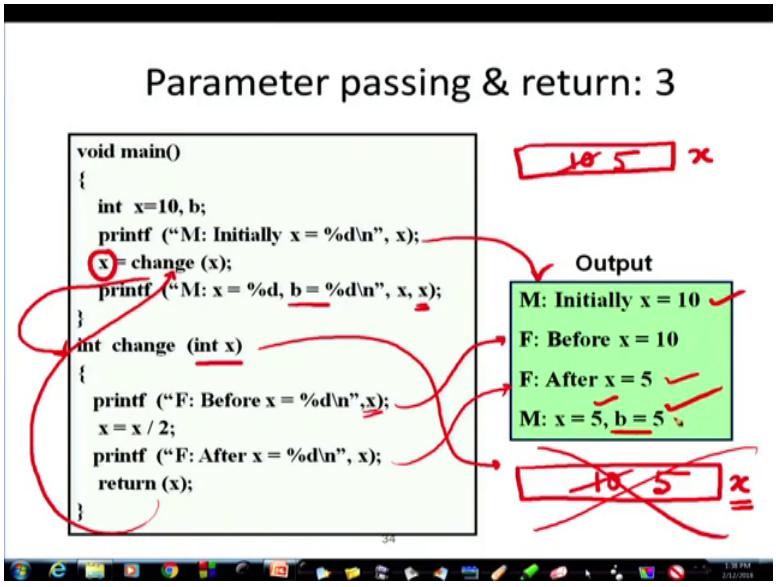
printout after that change x is what it has been changed 5 and then I return to the main function with the change value of x, so b gets 5. Now, I am printing in the main function x is 10 and b is 5. So, it has been changed that is being reflected in this printf and b is also 5 b was here, b is in the main function. So, that b has been assigned after this change and that is also 5. Now, the distinction I think will be clear. So, so these are the two cases that we have shown.

(Refer Slide Time: 33:38)



So, now, here let us see this is another example.

(Refer Slide Time: 33:49)



Here the slight change that has been done is that the change value I am keeping in x. Now, this was x, and x was 10. Now, note x was 10. So, initially x is 10 fine, from this line. Now, I have called change x, so it is coming here. But its parameter is also x, but that really does not matter. These x from here I will create another x. This who is the owner of this x, the owner of this x is only this function as long as this function is running these x has got a meaning it is existing other after that it is not there. But since I called it the value 10 was copied in this, but you see these two are two different memory locations. Consequently what is happening? When I print it here before x was 10. Now, see which x is being printed? This x is being printed because this x is not known to this function this function only knows its own x then x is changed to 5 that is also done to be done locally here and is being said here after that the x is 5 because this x is known then I return x. So, 5 is returned and 5 is assigned to x.

Now, whenever I have gone out of this function this x is no longer existing vanishes; that means, the compiler returns it to the memory pool now. So, which x? Is this x? This x. So, this value that is 5 that is being changed will come here. Now, here I am doing printf x x; that means, both b is, so x will be printed as 5 and b equals x. So, x is also 5. Here you see that the same value is the variable is x. So, both will be 5.

So, I hope you could understand this difference. Let us take the another example or we will come back to this in the next class and we will start with a new example and continue with parameter passing. So, what we learnt in today's lecture is a very important concept of call by value and call by reference and we could see that in C in general call by value is adopted except for arrays and what is call by value call by value means the calling function copies the value of the parameter to the variable corresponding to the argument. Whatever change is done is done locally inside that argument and when is returned it goes back to the main function, the value goes back to the main function. Whereas, in case of call by reference the there is only one vessel one variable and I am not creating another variable, I am simply passing the address of that variable to the called function and the called function changes in that particular variable and whatever changes are happening they are reflected in the main function, calling function as well.

We will continue with this.