Problem Solving through Programming In C

Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

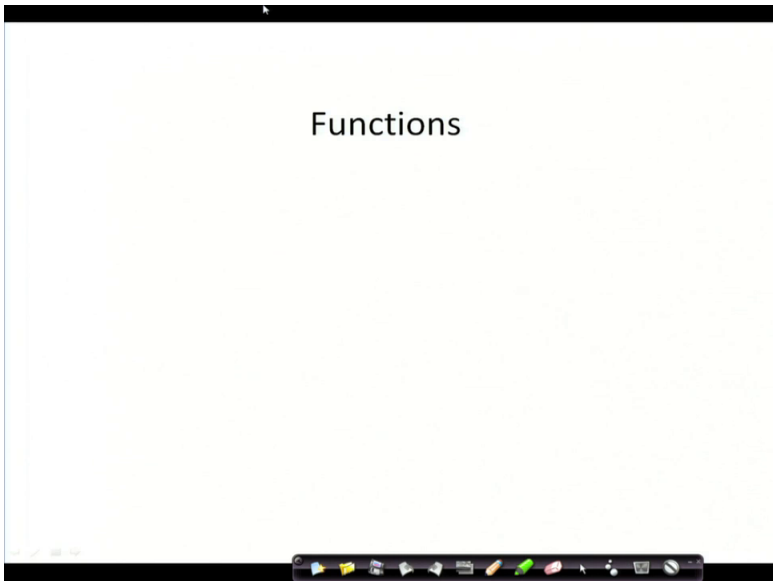
Lecture – 35
Introducing Functions

In the earlier lectures, we had discussed about one dimensional array as well as 2 dimensional array, we have seen how one dimensional array can be stored, read and printed and as well as we saw how a linear search can be applied over one dimensional array.

Similarly we have seen for 2 dimensional arrays how it can be read, it can be stored, it is stored in row major form and how it can be printed. Now today we will start discussing on a very important and vital component of programming and also (Refer Time: 01:06) of the C language that is functions. The concept of functions is very general and we will have to first look at it from the general angle why it is required, what is a function?

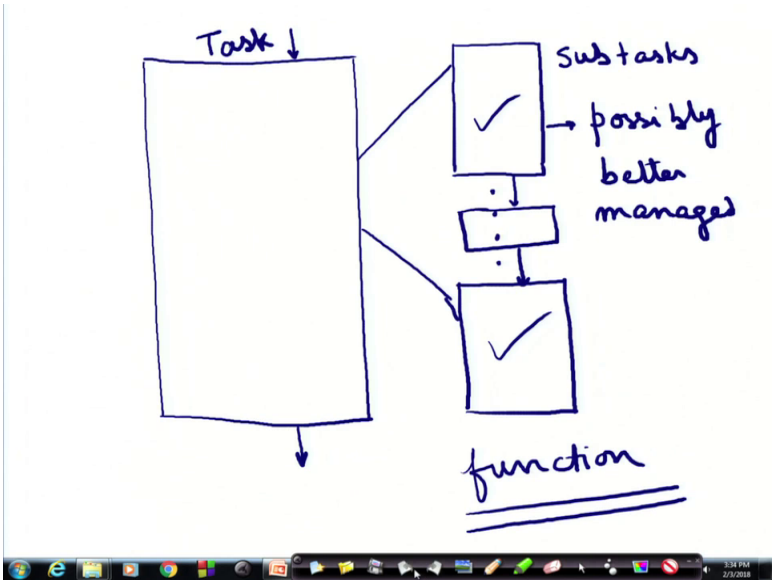
And then we can look at some of the details of this the implementation of this concept in the C language. So, first let us start with functions.

(Refer Slide Time: 01:39)



So, we will we will start to visualise a task.

(Refer Slide Time: 01:48)



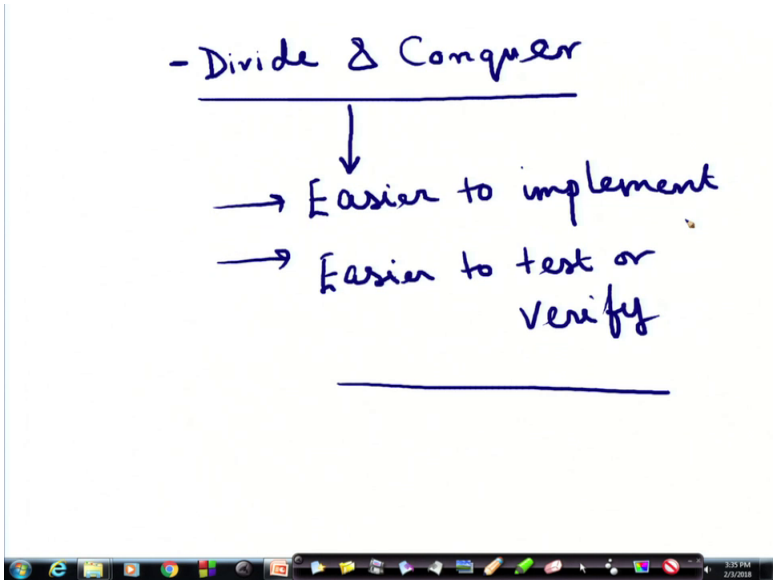
Suppose I have got a big task alright which I have to implement by programming. Now any task this is one task this task may be too complicated. And for any human programmer or any human being to solve this task from the beginning till end in one shot may be difficult. Often what we do we break up this task into a set of smaller sub tasks, different sub tasks? And each of this sub tasks are smaller enough, small enough. So, that we can manage it better each of these are possibly better managed, alright.

We can say if you think of a program segment a part of a program. If the task is broken big task is broken down into a small task a number of small tasks, then it is easier to write the program for each of these smaller tasks. And each of these tasks can be independently tested for it is correctness, whether it is working properly or not each of these can be separately tested.

And then if all these are connected together if all these are connected together one after another and then I can ultimately get this enter task done. So, instead of addressing the whole task into execute the whole task, or implement the whole task, as a single task it is often advisable to break it down into a number of subtasks.

Now let us talk about programming, now if we want to know what is a when we say the term function, if we take the English meaning of that function is means doing something. So, it is a also a task therefore, this entire function can be broken down into smaller functions and each of this functions can be implemented independently.

(Refer Slide Time: 04:50)

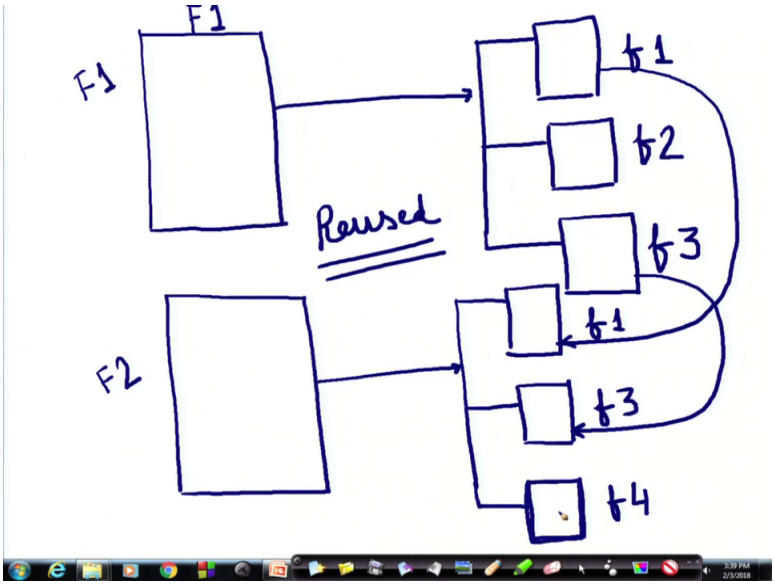


So, the advantage that we get by this means is that we can divide the problem into smaller parts and thereby we can conquer the small, we can conquer the smaller sub segments and in the process, we in the process we achieve the complete task that is the first thing. So, by this it is easier to implement and also easier to test or verify.

We can check whether each of these components are working correctly or not. Now when we find that all the components are working correctly and if they are connected together correctly, then we will have the entire problem solved correctly that is the first advantage of writing a complex program broken down into smaller functions.

Now the other advantage is suppose I had 2 implement one task.

(Refer Slide Time: 06:25)



And in order to implement this big task, I had to write a number of functions smaller functions. Say 3 smaller functions together implement this big function alright. So, let us call them f 1 function 1, f 2 for function 2 and f 3 for function 3.

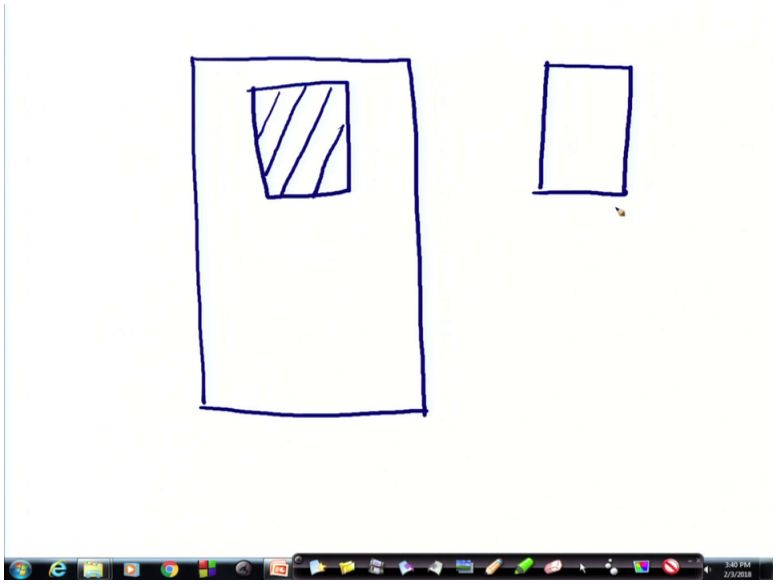
Now suppose I have now this problem has been solved by solving each of these 3 functions, each of these 3 functions have been written and tested and consequently they have been connected together and we have got the complete function. Now suppose another friend of yours wants to write another task solve another task say this was this was also a task.

So, this is also a function I am writing that as capital F 1 and there is another friend of yours who is trying to complete, another task which we are naming as F 2. Now in order to develop F 2, he finds that I have to solve some problems some sub problems, which are solved by f 1 and another sub problem which is solved by f 3, might be that this sub problem requires f 1, f 3 and say another function which was not written by anybody till now f 4.

Now; obviously, since f 1 has been written and tested he the writer of f 2 did not write f 1 again, because that f 1 this f 1 can be used for this purpose similarly this f 3 can be used for this purpose, and only effort that he has to spend is to write f 4. Thus the functions which are written already right can be reused.

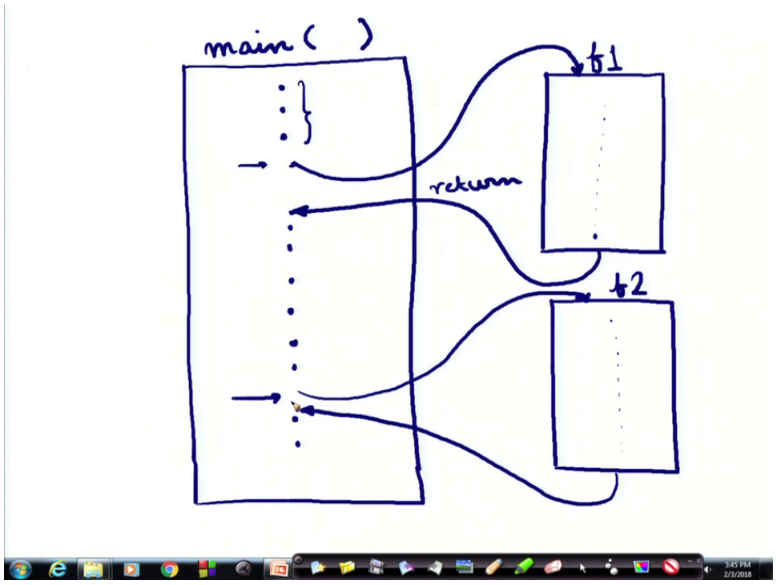
So, this reusability is another very important advantage of writing functions. Now we have said that in order to solve F 1 or F 2 we have to write we will have to independently write and test smallest functions and connect them together. What do I mean by connecting them together let us try to understand that.

(Refer Slide Time: 10:05)



So, say we have a function that is to be done developed and we find that this part has to be this part; this task has to be solved by some function. So, I write a function here for this part; let me do it again.

(Refer Slide Time: 10:39)



This is my whole problem that I have to solve. So, I am solving some things here some steps I am solving. And then I find here there is something that has to be done which is not very easy. So, for that I came at this point and for this I need to write a function or might be there is a function already existing in my library, which has been writ

ten by somebody else, which I can reuse.

Now in that case and also say here after doing that here I can do a couple of simpler tasks and then here I come to another point, which requires it to be independently solved and written. So, I need a function for that and then after that I will do some more simple things and my task will be over say this is a situation.

So, when I come here I will have to I means this program, which is a main tasks main task that I have to solve. So, since we have seen the name main in our description for C earlier, let me also name this to be main and I am just for nothing I am just putting some parenthesis here.

So, main is a main task that I have to do and at this point I am doing something's and here I need the help of this, now remember. That this function whose name is say f f 1 that can be used by my main function or somebody else main function also. Therefore, when I need this to be executed with my data, then I must whatever data I have prepared here that some of that I will have to pass on to this here.

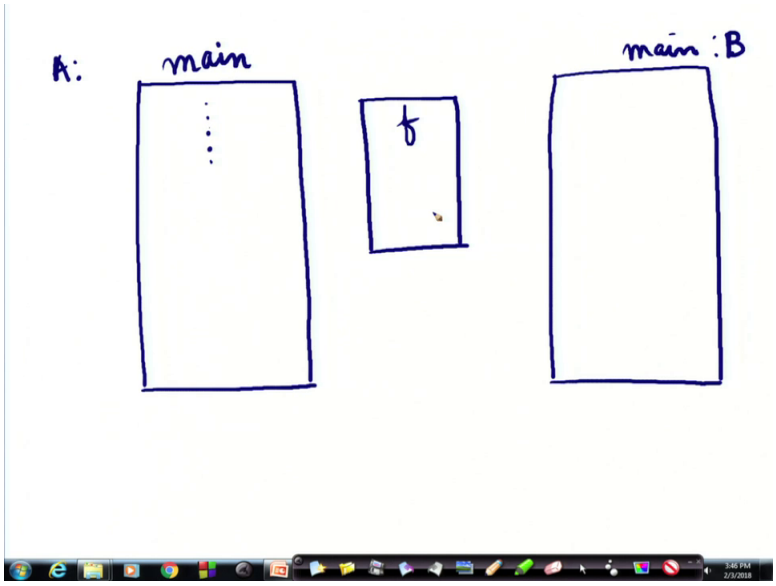
And this will solve this subtask and then I will have to; that means, a this must give back whatever it computed to the main task and the main task will continued doing simple things here. So, you see that there are 2 links; one is going in the function, another is coming out or some data that is being returned by this function. This is being returned by this function. Then I carry out some tasks simpler tasks here and at this point I find now I need help.

So, some part of the data from here will be used by the function f 2 and that has to come to f 2 here and f 2 will carry out the task and will return to the point here. Now about this return you should observe one thing, that the function was called from this point, we call this thing to be a function call function call with some data being passed on to this. And function calls or function invocation and after invocation this function works and it returns where does it return?

It returns to the point in the main task just after the point from where it was called. It was called from here. So, it comes here and then returns to the next point just after the after this call. So, this is a calling point and this a return point.

Similarly for here you see that it is executing and at this point it is being called. And after execution the control is returned into the immediately after the point from where it was called.

(Refer Slide Time: 15:56)

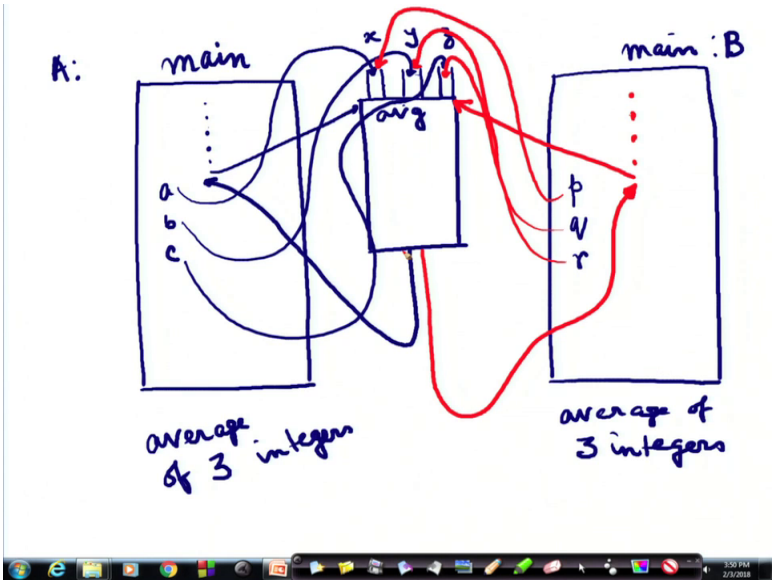


So, this is one important thing to understand the other important thing is that let me draw it again here. I have got my main task and this is my main and here is another main task, some other main task not written by me, but somebody else this is of mister A and this is of mister B for mister B there is he has written another main.

Now mister A or mister B both will require this function f. Now the task the purpose of what this task does is same for example, this task is computes the average of some numbers, 10 numbers, 5 numbers whatever it is or may be floating point numbers, integer numbers, whatever it is it computes the average.

So, let me let me quickly rename it not keeping it wage any longer let me call it, average or for a specific purpose, I want to write it inside the reason will be clear immediately this computes the average.

(Refer Slide Time: 17:21)



Now mister a wants to use this average function for some data for 3 data he wants to compute the average of 3 integers, suppose this one this one computes the average of 3 integers, but the integers can vary right. So, this also wants to find the average of 3 integers to make it simple.

But the in numbers for which A wants to compute the averages different from the numbers that B will compute the average for, but this a v g is common to both therefore, suppose the numbers that some integers say are a b and c and here this person wants to find the average of 3 integers which are p q and r 3 variable names.

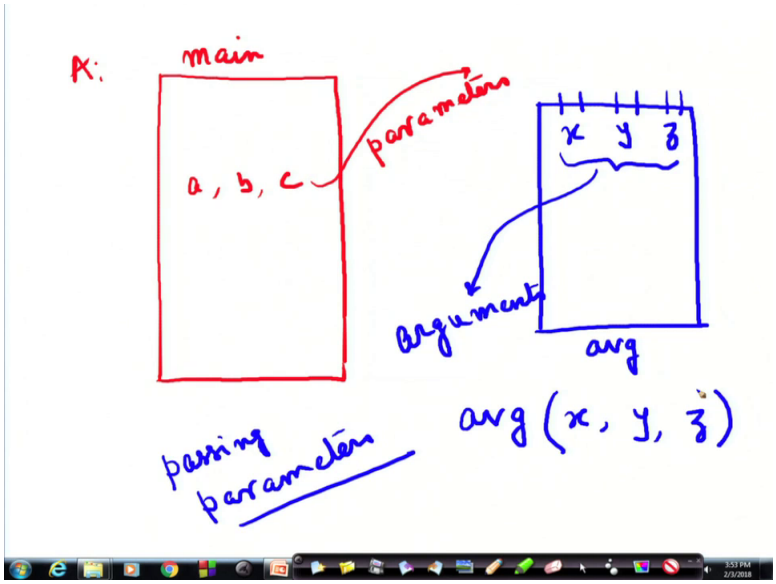
Now this average cannot remember p q r or a b c it will simply just like think of a box with 3 pipes coming in and data will come in through there. So, he just names this average for he names these pipes as x this is y and this is z. So, when this function main wants to compute the average of a b c he must send a through the pipe x b through the pipe y and c through the pipe z and you will get the average computed, and the average will be returned from the point where it was called from this point immediately after that it will go back clear.

Now when let me change the colour, now when the function main B; B wants to compute the average he wants to compute the average of p q r. So, p will be sent to the pipe x, q will be sent to the pipe y, and r will be sent to the pipe z. And the average will be computed suppose it was called from this point, from this point, this average was called.

So, then it will return the to this point with the value of the average with the value of the average whatever is computed here.

Therefore, so if this is clear then let us come to 2 more terms.

(Refer Slide Time: 21:17)

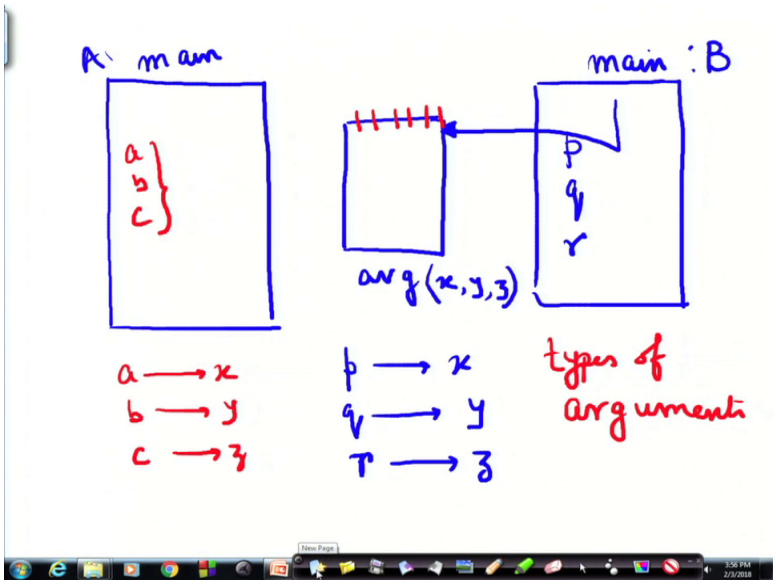


They are that this function main A, A is main was sending a, b, c to the function as parameters we call it as parameters. And this function which is our average function is not biased to a, it has got it knows that it needs 3 inputs and there are 3 input pipes, 3 input positions.

So, these x y z are arguments this we call as arguments alright; that means, this average when it is written it has got 3 input pipes. So, we can write simply like this function name average with 3 parameter arguments x y and z. Now who ever calls it we will have to establish the mapping, between the parameters that it wants to pass the parameters it wants to pass to the through the arguments.

So, the connection that we had shown in the earlier diagram which was where both the programs, were both the programs are accessing it. So, each of these parameters should the parameters should be matching with this arguments. Let us come to this once again say.

(Refer Slide Time: 24:00)

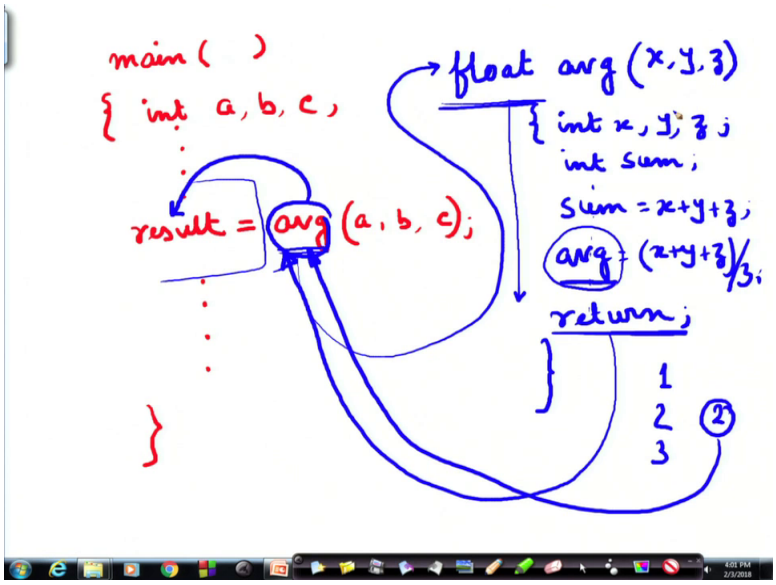


So, here is main of A and here is main of B, I repeat what I was saying till now and this has got p q r now, you must be must have realised now, what are these p q r? When I use this function average, which has got 3 arguments x y z which are nothing, but 3 pipes which are arguments and when I call from some point, when I call this function average then I have to pass these parameters p q r to these arguments x y z.

So, p will go to x q will go to y and z r will go to z, on the other hand when a is calling when a is calling, what is happening, when a is calling then a had the a b c to be passed on to this. So, a goes to x, b goes to y and c goes to z. Now since now if you just think these are 3 pipes, now the pipes are every pipe has got a width.

So, if I want to fit smaller pipe into a bigger pipe that would not fit. So, the width of diameter of both the pipes must fit together, what do I mean by that, what I mean by that is that whatever arguments are whatever are the types of the arguments that must match the types of the parameters or in other words the types of the parameters also must match the types of the arguments. So, that the pipes fit there should not be any mismatch over there. So, let us take little more deeper look here.

(Refer Slide Time: 26:55)



So, suppose I have got a main function here I am not so much bothered about the same types now and the main function is running and here I want to say result is average of a b c and etcetera, etcetera, etcetera, etcetera and we end here. And let us assume that here I have declared int a, b, c both of them are integers. Now we have got the function which is average.

Now, this average of 3 integers can be a float. So, I just write I will explain it a little later float average x, y, z and I can say int x, y, z and whatever I am computing average here and. So, thus this is simple you can have int sum and sum equals x plus y plus z, and average is equal to x plus y plus z by 3. Then I write returned, why do I write returned I will come to that.

Now look at the 2 things a I will talk about this part a little later. Here I am I have declared a b c to be integers. So, in the memory for a b and c 2 bytes or 4 bytes as the system may demand that type of that much memory has been given. Now when I write here a v g; that means, I am invoking or calling this function a v g. And I am passing on a v g requires 3 parameters x, y, z and x, y, z are all integers. And I could have written there is another way of writing it that I could have written here int x int y int z that is also possible, but later we will see.

So, a, b, c there is a correspondence between a, b and c with x, y and z. And they are matching in the type then this computation. So, we come here then follow my blue line this is being computed and ultimately the value of average is computed.

Look that the value is being computed in a v g which is also the name of the function. So, in a way you can say that the name of the function is just like a variable that is holding the value, that is holding the value. So, that average any variable that is holding a value must have a type that is why since average of 3 integers can be a float therefore, we have to assign a type to the name of the function.

Designating what type of value it is returning. So, here we can see that it will compute average and then there is a statement new statement that you are encountering here is returned, where is it returning it is returning to the point from where it is called at is this point.

So, the average, suppose if the values are 1 2 and 3 then the average is 2. So, then 2 goes to this a v g and that a v g is being transferred to result. Now if I do print result here 2 will be printed. So, this is to give you an idea of what is meant by in calling a function invoking a function, what is meant by returning from a function, and what are parameters and arguments.

We will see more of this in the subsequent lectures, but you should also understand why we do this reasons at 2 fold as I said one is to divide or break down a complex problem into manageable sub problems, and the second and test them independently, and the other issue is other advantage big advantage is that once we make a thing a function, which is tested we can keep it for being reused that is a very very important thing.

And this concept is general over different languages like; we have got class etcetera which we can reuse a number of times. And this is this idea the implementations are varying from language to language, but the concept is common and is used in different languages.

Thank you and we will continue with functions in the subsequent lectures.